THE EXPERT'S VOICE®

Pro DNS and BIND

Ronald G. F. Aitchison

Apress®

Pro DNS and BIND

Ron Aitchison

Apress®

Pro DNS and BIND

Copyright © 2005 by Ron Aitchison

ISBN (pbk): 1-59059-494-0

Library of Congress Cataloging-in-Publication data is available upon request.

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jason Gilmore
Technical Reviewer: Brian Wilson
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox, Susannah Pfalzer
Assistant Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Linda Seifert
Indexer: Valerie Perry
Artist: Kinetic Publishing Services, LLC
Interior Designer: Van Winkle Design Group
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The sample files and source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

To my parents, Gordon and Vera Aitchison.
Any good characteristics I possess I owe entirely to their
good genes and a good upbringing. My bad characteristics
I had to work at strenuously on my own.

Contents at a Glance

Contents

About the Author

RONALD (RON) AITCHISON is the President of Zytrax, Inc., a Montreal-based company that specializes in wireless and wire-line IP communications. Zytrax develops its own products as well as undertaking specialized consulting, training, system design, and development for clients. Zytrax is currently developing the Netwidget—a business appliance family aimed at bringing the compelling cost advantages of Open Source to small- and medium-sized companies by offering trivial user installation, and robust and reliable operation combined with high levels of security. Netwidget uses BIND, NSD, DHCP, Apache, Squid, ProFTP, Samba, Courier e-mail, OpenLDAP, and OpenSSL, among many other high-quality packages, and is developed in a mixture of C and Ruby. Zytrax supports its own and customer-hosted DNS, web, e-mail, and LDAP services on a mixed network of Windows, Linux, and, increasingly, FreeBSD systems, and has been an Open Source user since 1998.

Prior to founding Zytrax in 1994, Ron worked in senior roles in development, sales, and marketing in both Europe and the US. He started his computer career in 1973 as a grunt systems programmer developing communications software for mainframes in a nineteenth-century palace outside of Edinburgh, Scotland. His major achievement in those years was, as cofounder of the local micro-club, persuading Intel to ship the UK's second 8086 system for club use ahead of minor competition such as IBM and others. He moved into sales and marketing for a number of years before returning to real—technical—work when he established Zytrax. He was educated in mechanical engineering at the University of Strathclyde in Glasgow, Scotland, a long time ago.

About the Technical Reviewer

BRIAN WILSON is the associate director of technology at the Fisher College of Business at The Ohio State University. He has worked in the IT industry for the last 16 years. Brian specializes in network and firewall design and implementation, and he oversees and directs all technical aspects of the Fisher College of Business. He received a BS in computer science engineering from The Ohio State University.

In his off time, Brian spends time with his wife and three children at their country home. Brian enjoys listening to and watching baseball. Over the past couple of years, Brian has become determined to be a horseman with the acquisition of a new quarter horse.

Acknowledgments

The author would like to gratefully acknowledge the patience and forbearance of a number of individuals during the writing of this book:

The Apress team of Kylie Johnston, Ami Knox, Ellie Fountain, and Susannah Pfalzer, who struggled valiantly to both keep me on track and to force me to write in something vaguely resembling the English language. Brian Wilson, who kept me on the straight and narrow when I threatened to, and frequently did, veer into the dense underbrush. Finally and especially, Jason Gilmore, who foolishly put the idea for this book into my head in the first place. His frequent comments of "Don't understand this sentence/paragraph/section" on my Pulitzer Prize winning streams of prose drove me wild but were, in every case, on the mark, and the subsequent rework contributed to a significant improvement on my, with hindsight, pathetic originals.

One of the sad things about e-mail is that one never meets the individuals who took the time from busy lives to respond to questions and provide insight and information on numerous obscure topics. I would like to thank, in no particular order, Paul Vixie, Olaf Kolkman, Miek Gieben, Jakob Schlyter, and Simon Josefsson.

Finally, I would like to single out Jacco Tunnissen, who runs the superb site `www.dnssec.net`, for his help, contacts, and advice as well as the regular streams of information that he sent me whenever I was chasing down some definitive reference and had exhausted all other avenues.

In spite of all the help, any errors are entirely the responsibility of the author.

Introduction

Every time you get e-mail, every time you access a web page, you use the Domain Name System (DNS). In fact, over 2 billion such requests hit the DNS root-servers alone every day. Every one of those 2 billion requests originate from a DNS that supports a group of local users, and every one of them is finally answered by a DNS server that may support a high-volume commercial web site or a modest, but much loved, family web site. This book is about understanding, configuring, diagnosing, and securing the DNS servers that do the vital work.

Many years ago when I set up my first pair of DNS servers, I wasted my time looking for some practical advice and some sensible description of the theory involved. I found neither. I completed the DNS rite-of-passage—this book was born from that experience.

DNS is a complex subject, but it is also unnecessarily cloaked in mystery and mythology. This book, I hope, is a sensible blend of practical advice and theory. You can treat it as a simple paint-by-numbers guide to everything from a simple caching DNS to the most complex secure DNS (DNSSEC) implementations. But the background information is there for those times when you not only need to know what to do, but you also need to know why you are doing it, and how you can modify the process to meet your unique needs.

Who This Book Is For

This book is about running DNS systems based on BIND 9.3.0—the first stable release that includes support for the latest DNSSEC (DNSSEC.bis) standards and a major functional upgrade from previous BIND 9 releases. If you run or administer a DNS system, are thinking about running a DNS system, need to upgrade to support IPv6 DNS, need to secure a DNS for zone transfer, dynamic update, or other reasons, need to implement DNSSEC, or simply want to understand the DNS system, then this book is designed to provide you with a single point of reference. The book progressively builds up from simple concepts to full security-aware DNSSEC configurations. The various features, parameters, and Resource Records that you will need are all described and in the majority of cases illustrated with one or more examples. The book contains a complete reference on zone files, Resource Records, and BIND's `named.conf` configuration file parameters. Programmers and the insatiably curious will find BIND's Simple Database API, resolver library interfaces, and the gory details of DNS wire-format messages compelling reading.

How This Book Is Structured

This book is about the Domain Name System. Most of the examples used throughout the book are based on the Berkeley Internet Name Domain, universally known as BIND, which is the most widely deployed name server software in current use. BIND version 9.3.0—a major functional upgrade to support the latest DNSSEC standards—was used as the baseline version for all the examples. During the course of writing the book, version 9.3.1—a bug clearance–only version—was released. While the book references 9.3.0 throughout, the majority of, but not all, tests were rerun on the new version—the only difference noted was the change to the configure variable used when building a base version for FreeBSD, which is related to FreeBSD, not BIND. Readers are advised to always obtain and use the latest stable BIND version.

Like most technical books, this is a mixture of descriptive text, reference material, and samples. For those completely unfamiliar with the subject, Part 1 (Chapters 1 to 5) is designed to introduce DNS in a progressive manner and could be read as a classic text on the subject. For those of a hands-on disposition, Part 2 provides an alternative entry point, with the various earlier chapters to be read as needed. Experienced readers would typically head straight for the meat in either Parts 3, 4, or 5, depending on their area of interest. As well as providing help and guidance during your initial endeavors, it is my fervent hope that this book will also provide you with an indispensable reference work for years to come.

Chapter 1, "An Introduction to DNS"

Chapter 1 provides introductory and background material to the DNS as a specific implementation of the general name server concept. The key concepts introduced are the domain name hierarchy, delegation, DNS operational organization, the role of ICANN, and the various components that comprise a DNS system, including zones and zone files. The chapter is for those who are unfamiliar with the topic or the changes that have occurred in the recent past.

Chapter 2, "Zone Files and Resource Records"

Here you are introduced to the basic Resource Records and directives used to construct zone files. An example forward-mapping zone file is introduced that is used throughout the book and illustrates key DNS operational concepts such as resilience and location diversity. Those with little or no knowledge of zone files and their construction will find this chapter a gentle introduction to the topic.

Chapter 3, "DNS Operations"

This chapter describes the basic operation of a DNS system, including queries, referrals, reverse mapping, zone transfers, and dynamic updates. A brief overview of DNS security is presented to familiarize readers with the potential threats posed when running DNS systems. This chapter is intended to give the reader a thorough grounding in the theory and background to these topics.

Chapter 4, "DNS Types"

The text in this chapter breaks down configuring a DNS into a number of types such as master, slave, caching only, forwarding, Stealth, and authoritative only with the objective of giving the reader a set of building blocks from which more complex configurations can be constructed. This chapter will be useful to those unfamiliar with the range of possibilities offered by the DNS and its BIND implementation, including the new `view` clause introduced with the BIND 9 series.

Chapter 5, "DNS and IPv6"

Chapter 5 focuses on IPv6 and the DNS features that support this increasingly widespread protocol. A brief overview of IPv6 address structure and notation is provided for those currently unfamiliar with this topic.

Chapter 6, "Installing BIND"

This chapter covers the installation of BIND on Linux (Fedora Core 2), FreeBSD, and Windows 2000 from binary packages. For those cases where a package is not available, building from a tarball is also described.

Chapter 7, "BIND Type Samples"

The zone and `named.conf` sample files for each of the DNS types introduced in Chapter 4 are provided. While these samples can be used as simple paint-by-number implementations, explanations are included to allow the configurations to be tailored to user requirements.

Chapter 8, "Common DNS Tasks"

A number of standard DNS configurations are described and illustrated with sample files and implementation notes. The items covered include delegation of subdomains, load balancing, fixing sequence errors, delegation of reverse subnets, SPF records, and the use of wildcards.

Chapter 9, "DNS Diagnostics and Tools"

The major utilities supplied with a BIND distribution, including those used for security operations, are covered with multiple use examples. The reader, however, is encouraged—especially with `dig` and `nslookup`—to get out and explore the Internet using these tools. A practical example is used to illustrate to some diagnostics techniques and procedures.

Chapter 10, "DNS Secure Configurations"

DNS security is broken into four parts: administrative security, securing zone transfers, securing dynamic update, and DNSSEC. An overview of general cryptographic processes including symmetric and asymmetric encryption, digital signatures, and MACs, which form the basis of DNS security implementations, is provided for readers unfamiliar with this topic.

Chapter 11, "DNSSEC"

This chapter deals exclusively with the latest DNSSEC.bis security standards and covers both the theory and implementation. Zone signing, chains of trust, Zone Signing Keys and Key Signing Keys, DNSSEC Lookaside Validation (DLV), and key-rollover procedures are all covered with practical examples.

Chapter 12, "BIND Configuration Reference"

As suggested by the title, this is purely a reference section, and it catalogues and describes with one or more examples the clauses and statements used in BIND's `named.conf` file. The chapter is organized in a manner that allows the reader to easily find appropriate statements to control specific BIND behaviors.

Chapter 13, "Zone File Reference"

This is purely a reference section that describes each Resource Record in the current IANA list—normally with one or more examples to illustrate usage.

Chapter 14, "BIND APIs and Resolver Libraries"

Designed more for programmers and designers, you will need a reasonable understanding of C to make sense of this chapter. The new BIND Simple Database API and the original BIND RES library are covered, together with an overview of the current status of DNS-related POSIX interfaces.

Chapter 15, "DNS Messages and Records"

This chapter covers the gory details of DNS wire-format messages and RR formats. A reasonable working knowledge of decimal, hex, and binary notations are required to make sense of the chapter. Essential reading if you are developing DNS applications, when RRs are not supported by your sniffer application or you are insatiably curious about how this stuff works.

Appendix A, "Domain Name Registration"

This appendix is a collection of material, presented in FAQ format, that may help to answer questions about registering domains in a variety of situations.

Appendix B, "DNS RFCs"

This appendix presents a list of RFCs that define the DNS and DNS-related topics.

Additional Material

In addition, the author maintains a web site about the book (`www.netwidget.net/books/apress/dns`) that covers additional material, including links to alternative DNS software, resolver language bindings, and background reading on various topics covered in the book, which may be of use to the reader.

Conventions

The following conventions are used throughout the book:

- The # (hash or pound) symbol is used to denote a command prompt and always precedes a command to be entered. The command to be entered starts after this symbol.
- Lines consisting of four dots (....) in zone and configuration files are used to denote that other lines may or may not be present in these files. The dot sequence should not be entered in the actual files.
- When describing command syntax, the following convention is used throughout:

  `command` `argument [option1]` **`keyword`** `[option2 [optional3] ...]`

  where all items in bold, which include command and keywords, must be entered as is. Optional values are enclosed in square brackets and may be nested. Where repeated options are allowed, a sequence of three dots is used to indicate this.

Contacting the Author

The author may be contacted at `ron.aitchison@netwidget.net`, and he maintains links and other information relating to this book at `www.netwidget.net/books/apress/dns`.

PART 1

Principles and Overview

CHAPTER 1

An Introduction to DNS

The Internet—or any network for that matter—works by allocating a locally or globally unique IP address to every endpoint (host, server, router, interface, etc.). But without the ability to assign some corresponding name to each resource, every time we want to access a resource available on the network, the web site `www.example.com` for instance, it would be necessary to know its *physical IP address*, such as 192.168.34.166. With hundreds of million of hosts and more than 50 million web sites,[1] it's an impossible task—it's also pretty difficult with even a handful of hosts and resources.

To solve this problem, the concept of *name servers* was created in the mid-1970s to enable certain attributes (or properties) of a *named resource*, in this case the IP address of `www.example.com`, to be maintained in a well-known location—the basic idea being that people find it much easier to remember the *name* of something, especially when that name is reasonably descriptive of function, content, or purpose, rather than a numeric address. This chapter introduces basic name server concepts and provides a bit of background regarding the evolution of the Domain Name System from a tool used for managing just a few hundred hosts to a global utility responsible for maintaining smooth operation of the entire modern Internet.

A Brief History of Name Servers

The problem of converting names to physical addresses is as old as computer networking. Even in times long since past, people found it easier to remember they were using a teletype device called "tty2" rather than "port 57 of the MCCU" or whatever the addressing method then in use. Furthermore, administrators wanted the flexibility to reconfigure equipment while leaving users with a consistent way of describing the device they were using. In the preceding example, the user could continue to use "tty2" even if the device had been reconfigured to be on port 23 of the mythical MCCU. Simple configuration files were typically used to perform *address translation.* As networking, rather than simple communications, emerged in the early 1970s, the problem became more acute. IBM's System Network Architecture (SNA), probably the grandfather of networking, contained a rudimentary mainframe database for *name translation* when originally published in 1974. The much-maligned Open Systems Interconnect (OSI) Model, developed by the International Organization for Standardization (ISO—`www.iso.org`), defined *Address/Name Translation* services at the Transport Layer (Layer 4) when initially published in 1978. NetBIOS provided the NetBIOS Name Server (NBNS) when originally defined in 1984, which later morphed into Microsoft's Windows Internet Naming Service (WINS).

[1] `http://news.netcraft.com/archives/web_server_survey.html`

The first ARPANET (the network that morphed into the Internet) RFC, the quaintly named Request For Comments that document and standardize the Internet, on the concept of domain names dates from 1981 (RFC 799), and the definitive specifications for the Internet's Domain Name System as we know it today were published in 1987 (RFC 1034 and RFC 1035).

Name Server Basics

When a name server is present in a network, any host only needs to know the *physical address* of a name server and the *name* of the resource, a web site for example, it wishes to access. Using this information, it can find the address (or any other stored attribute or property) of the resource by interrogating (commonly referred to as *querying*) the name server. Resources can be added, moved, changed, or deleted at a single location, the name server, and new information will be immediately available to every host using this name server. Our name server is simply a specialized database that translates names to properties—typically IP addresses—and vice versa. Name servers both simplify network management and make networks more dynamic and responsive to changes.

Solutions, however, can also generate problems. If our name server is not available, then our host cannot access *any* resource on the network. We have made the name server a critical resource. So we had better have more than one name server in case of failure.

The initial solution to the problem of name server availability was to introduce *Primary* and *Secondary* name servers. If the Primary name server did not respond to a query, the host would retry using the Secondary name server. So critical is the name server that today it is common to see lists of three, four, or more name servers. The terms *Primary* and *Secondary name servers*, and even *Tertiary*, and *Quartiary name servers*, while still widely used, imply priority of access, which works against availability. Not only would such prioritization cause transaction bunching on the Primary name server, degrading overall performance, but in the case where the Primary name server was inoperable, *every* transaction would have to wait for a timeout before retrying with the Secondary, and so on. Most name server software uses some form of randomized, measured response time or round-robin access to the name server list to try and spread loads and decrease response times.

As our network grows, we start to build up a serious number of names in our name server. This gives rise to three new problems:

1. *Organization*: Finding any entry in the database of names becomes increasingly slow as we power through many millions of names looking for the one we want. We need a method to index or organize the names.

2. *Scalability*: If every host is accessing our name servers, the load becomes very high. We need a method to spread the load across a number of name servers.

3. *Management*: With many name records in our database, the management problem becomes increasingly difficult, as multiple administrators attempt to update records at the same time. We need a method to separate (known as *delegating*) the administration of these name (generally known as *resource*) records.

The need to satisfy these three requirements led to the creation and evolution of the Internet's *Domain Name System (DNS)*, discussed in the next section.

The Internet Domain Name System

The Internet's Domain Name System is a specific implementation of the name server concept optimized for the prevailing conditions on the Internet. From our brief history of name servers, we saw that three requisites emerged:

1. The need for a hierarchy of names
2. The need to spread the operational loads on our name servers
3. The need to delegate the administration of our name servers

The Internet DNS elegantly solves all three problems.

■ **Note** The standard RFCs that define the basic DNS functionality, RFC 1034 and RFC 1035, were both written over a quarter of a century ago—1987—and authored by Dr. Paul Mockapetris while at the Information Sciences Institute of the University of Southern California. Although many subsequent RFCs have modified certain DNS behaviors, the core functionality remains intact. This is indeed a remarkable achievement.

Domains and Delegation

The Domain Name System uses a tree (or hierarchical) name structure. At the top of the tree is the root node followed by the *Top-Level Domains* (TLDs), then the *Second-Level Domains* (SLD) and any number of lower levels, each separated with a dot.

■ **Note** The root of the tree is represented most of the time as a silent dot (.), but there are times when it is VERY important.

TLDs are split into two types:

1. *Generic Top-Level Domains (gTLD)*: For example, `.com`, `.edu`, `.net`, `.org`, `.mil`, etc.
2. *Country Code Top-Level Domains (ccTLD)*: For example, `.us`, `.ca`, `.tv`, `.uk`, etc.

Country Code TLDs use a standard two-letter sequence defined by ISO 3166.[2] Figure 1-1 illustrates this diagrammatically.

[2] `www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html`

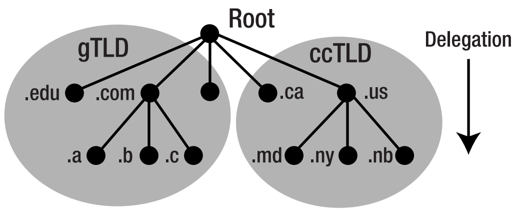


Figure 1-1. *Domain structure and delegation*

What is commonly called a *domain name*, for instance `example.com`, is actually a combination of an SLD name and a TLD name and is written from left to right with the lowest level in the hierarchy on the left and the highest level on the right:

```
sld.tld
```

The term *Second-Level Domain* is technically precise in that it defines nodes at the second level within the domain name hierarchy, but is long-winded. To be even more long-winded, there are also Third-Level Domains, which are especially relevant with ccTLDS, and so on. By convention—or perhaps laziness—the term *domain*, or *domain name*, is generally used to describe a delegated entity, for instance, `example.com`, which consists of the SLD `example` and the TLD `com`. Unless precision is required, the term *domain name* will be used throughout the remainder of this book.

Domain Authority

The concepts of *authority* and *delegation* lie at the core of the Domain Name System hierarchy and exactly mirror its hierarchical organization. Each node within the domain name hierarchy is assigned to an *authority*—an organization or person responsible for the management and operation of that node. Such an organization or person is said to administer the node *authoritatively*. The *authority* for a particular node can in turn *delegate* authority for lower levels of that node within the domain name hierarchy. The rules and limitations of the authority are covered by agreements that flow through the various nodes in the hierarchy.

The authority for the root domain lies with the Internet Corporation for Assigned Numbers and Names (ICANN—`www.icann.org/`). Since 1998, ICANN, a nonprofit organization, has assumed this responsibility from the United States Department of Commerce. When ICANN was established, part of its mandate was to open up that part of the domain name hierarchy for which it is responsible to commercial competition. To facilitate this competition, it created the concept of *accredited registrars*, organizations to which ICANN delegated limited responsibilities for the sale and administration of parts of the domain name hierarchy.

The gTLDs are *authoritatively* administered by ICANN and *delegated* to a series of accredited registrars. The ccTLDs are delegated by ICANN to the individual countries for administration purposes. Figure 1-1 also shows how any authority may in turn delegate to lower levels in the hierarchy; in other words, it may delegate anything for which it is *authoritative*. Each layer in the hierarchy may *delegate* the *authoritative* control to the next or lower level.

In the case of ccTLDs, countries define their own rules for delegation. Countries like the United States (ccTLD `.us`) and Canada (ccTLD `.ca`) and others have decided that they will administer both at the national level and delegate to each state (US) or province (Canada) using a two-character state/province code (for example, `.ny` = New York, `.qc` = Quebec, `.md` = Maryland, etc.). Thus `example.us` is the domain name of `example` that was delegated from the US national ccTLD administration, and `example.md.us` is the domain name of `example` that was delegated from the state of Maryland in the US.

Other countries like the United Kingdom and Brazil among many have opted for functional segmentation in their delegation models. Thus `example.co.uk` is the domain name of `example` registered as a company from the UK registration authority and `example.com.br` is the domain name of `example` registered as a company from the Brazilian registration authority.

Delegation within any domain may be almost limitless and is *decided by the delegated authority*. For example, many states in the US and provinces in Canada delegate cities within state/province domains: the domain name `example.nb.us` would be the town of Example in the State of Nebraska in the United States, and indeed we could have `mycompany.example.nb.us`, which would be the domain name of `mycompany` in the town of Example in the state of Nebraska in the United States.

Reading a domain name from right to left will track its delegation. This unit of delegation is referred to as a *zone* in the DNS specifications.

So What Is `www.example.com`?

From our reading previously, we can see that `www.example.com` is built up from `www` and `example.com`. The domain name `example.com` part was delegated from a gTLD registrar, which in turn was delegated from ICANN.

The owner of the domain chose the `www` part since they are now the delegated authority for the `example.com` domain name. They own everything to the left of the delegated domain name, in this case `example.com`.

The leftmost part, the `www` in this case, is called a *host name*. Keep in mind that only by convention do web sites use the host name `www` (for World Wide Web), but a web site can be named `fred.example.com`—few may think of typing this into their web browser, but that does not invalidate the name!

Every computer that is connected to the Internet or an internal network and is accessed using a name server has a host name. Here are some more examples:

`www.example.com`	The company web service
`ftp.example.com`	The company file transfer protocol server
`pc17.example.com`	A normal PC
`accounting.example.com`	The main accounting system

A host name must be unique within the delegated domain name, but can be anything the owner of `example.com` wants.

Finally, consider this name:

```
www.us.example.com
```

From our previous reading, we figure the domain name is `example.com`; the `www` probably indicates a web site, which leaves the `us` part.

The `us` part was allocated by the owner of `example.com` (who is authoritative) and is called a subdomain. In this case, the *delegated authority* for `example.com` has decided that their organization is best served by a country-based subdomain structure. They could *delegate* the responsibility internally to the US subsidiary for administration of this subdomain, which could in turn create a plant-based structure; for example, `www.cleveland.us.example.com` could indicate the web site of the Cleveland plant in the US organization of `example.com`.

To summarize: the *owner* can delegate, in any way they want, anything to the left of the domain name they own (or were *delegated*). The delegated owner is also responsible for administering this delegation.

■**Note** `www.example.com` and `www.us.example.com` are commonly—but erroneously—referred to as Fully Qualified Domain Names (FQDN). Technically an FQDN unambiguously defines a domain name to the root and therefore must terminate with the normally silent dot; for instance, `www.example.com.` (with the dot) is a valid FQDN, but `www.example.com` (without the dot) is not.

DNS Implementation and Structure

The Internet's DNS implementation exactly maps the domain name delegation structure described previously. There are name servers (servers that run DNS software) at each level in the delegated hierarchy, and the responsibility for running the name server lies with the authoritative control at that level. Figure 1-2 shows this diagrammatically.

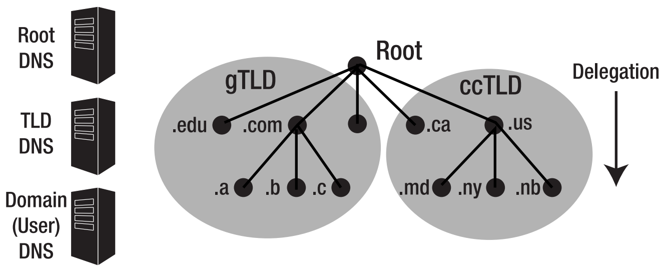


Figure 1-2. *DNS mapped to domain delegation*

The root name servers (hereafter called the *root-servers*) are the most critical resource on the Internet. When any name server worldwide is *queried* for information about a domain name for which it does not currently have information, it first asks (*queries*) one of the root DNS servers. There are currently 13 root-servers worldwide, described in further detail later in this chapter. The root-servers are known to every name server in the world using a special *zone file*, which is distributed with all DNS software.

The TLD name servers (gTLD and ccTLD) are operated by a variety of organizations under ICANN agreements and described more completely later in this chapter.

The owner of a domain name has been delegated the authority for administering the domain name and therefore has the responsibility for the operation of the user (or domain name) name servers—there must be a minimum of two. The name server operational responsibility may be delegated to an ISP, a web hosting company, or increasingly a domain name registrar. Many companies and domain name owners, however, elect to run their own name servers and even delegate the authority and responsibility for subdomain name servers to separate parts of their organization.

When any name server cannot answer, or *resolve*, a request for a name, for instance, `fred.example.com`, the query is passed to a root-server (discussed in the next section), which returns a *referral* to the appropriate TLD name server, which in turn provides a referral to the appropriate domain (user) name server. Figure 1-3 illustrates this process.

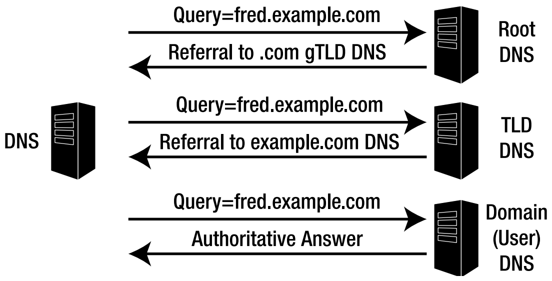


Figure 1-3. *The operational DNS hierarchy*

Root DNS Operations

The root-servers (root DNS) are the responsibility of ICANN but are operated under an agreement known as the Cooperative Research and Development Agreement (CRADA) that was signed between ICANN and the US Department of Commerce (`www.icann.org/committees/dns-root/crada.htm`). This agreement covers the methods and processes by which updates to the root name systems are carried out. ICANN also created the Root Server System Advisory Committee (RSSAC) to provide advice and guidance as to the operation and development of this critical resource. The IETF was requested by the RSSAC to develop the engineering standards for operation of the root-servers. This request resulted in the publication of RFC 2870.

There are currently 13 root-servers. They occupy a reserved domain name, `root-servers.net`. Each root-server typically comprises more than one physical server but shares a common IP address. Root-servers are named from `a.root-servers.net` through `m.root-servers.net` as shown in Table 1-1.

Table 1-1. *Root-Servers*

Server	Operator	Locations	IP Address
A	VeriSign Global Registry Services	Dulles, VA	198.41.0.4
B	Information Sciences Institute	Marina del Rey, CA	IPv4: 192.228.79.201, IPv6: 2001:478:65::53
C	Cogent Communications	Chicago; Herndon, VA; Los Angeles; New York City	IPv4: 192.33.4.12
D	University of Maryland	College Park, MD	IPv4: 128.8.10.90
E	NASA Ames Research Center	Mountain View, CA	IPv4: 192.203.230.10
F	Internet Systems Consortium, Inc. (ISC)	Auckland, Beijing, Brisbane, Dubai, Hong Kong, Jakarta, Johannesburg, Lisbon, Los Angeles, Madrid, Monterrey, Moscow, Munich, New York City, Osaka, Ottawa, Palo Alto, Paris, Prague, Rome, San Francisco, San Jose, Sao Paulo, Seoul, Singapore, Taipei, Tel Aviv, Toronto	IPv4: 192.5.5.241, IPv6: 2001:500::1035
G	US DOD Network Information Center	Vienna, VA	IPv4: 192.112.36.4
H	US Army Research Lab	Aberdeen, MD	IPv4: 128.63.2.53, IPv6: 2001:500:1::803f:235
I	Autonomica/NORDUnet	Amsterdam, Ankara, Bangkok, Brussels, Bucharest, Chicago, Geneva, Frankfurt, Helsinki, Hong Kong, Kuala Lumpur, London, Milan, Oslo, Stockholm, Tokyo, Washington DC	IPv4: 192.36.148.17
J	VeriSign Global Registry Services	Amsterdam; Atlanta; Dulles, VA (2 locations); London; Los Angeles; Miami; Mountain View, CA; Seattle; Seoul; Singapore; Sterling, VA; Stockholm; Tokyo	IPv4: 192.58.128.30
K	Réseaux IP Européens Network Coordination Centre (RIPE)	Amsterdam, Athens, Doha, Frankfurt, London, Milan	IPv4: 193.0.14.129, IPv6: 2001:7fd::1
L	Internet Corporation for Assigned Names and Numbers (ICANN)	Los Angeles	IPv4: 198.32.64.12
M	WIDE Project	Paris, Seoul, Tokyo	IPv4: 12.27.33, IPv6: 2001:dc3::35

■Note The number 13 is not a perverse desire by anyone to operate a number of servers seen by some cultures as unlucky, but rather a technically determined limit enabling common root-server queries to be answered within a single 512-byte UDP transaction and hence reduce root-server loads. The 13 root-server limit will likely remain permanent for the foreseeable future.

The job of the root-servers is to provide a referral to the authoritative name servers for the required TLDs (gTLDs or ccTLDs). For example, if a user requests information about `fred.example.com`, then the root-servers will supply a list of the authoritative name servers for the `.com` TLD. In 2004, ICANN took over responsibility for the maintenance of the root-servers TLD master file—the file that lists the authoritative servers for each TLD. Distribution of this file to each of the operational root-servers is carried out using secure transactions. To further increase security, the server providing the root updates is only accessible from the operational root-servers. It is not a publicly visible server. Figure 1-4 illustrates this process.

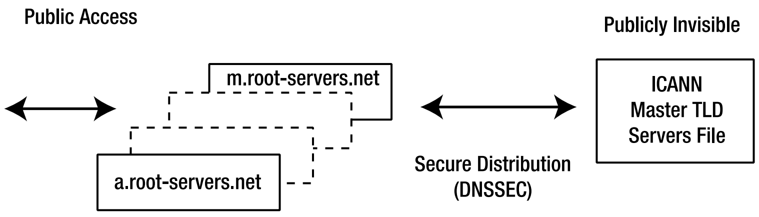


Figure 1-4. *Root-servers update process*

Top-Level Domains

As was mentioned earlier in this chapter, Top-Level Domains are split into Generic Top-Level Domains and Country Code Top-Level Domains. Each group is administered slightly differently, but all are controlled by ICANN. ICANN controls the gTLDs by a purely contractual process. In the case of ccTLDs, since multiple countries are involved, the process is essentially consultative rather than purely contractual.

Generic Top-Level Domains

Generic Top-Level Domains, or gTLDs, are controlled by ICANN using a contractual process. When competition was introduced into the registration of domain names, ICANN established two separate entities:

1. *Registry Operators*: Registry Operators contract with ICANN to operate the authoritative gTLD DNS servers (see Figure 1-2 earlier). There is a single Registry Operator for each of the gTLDs, for example, the US Department of Defense, Network Information Center, is the Registry Operator for the `.mil` gTLD, but each Registry Operator will operate multiple name servers. DNS queries to the root-servers are supplied with a *referral* to the authoritative gTLD servers for the specific gTLD; for example, if the query is for `example.net`, then the root-servers will supply the list of `.net` authoritative DNS servers. Registry Operators obtain the list of SLDs from one or more Registrars. The public has no contact with the Registry Operator. However, a number of Registry Operators are also Registrars; for example, VeriSign, Inc., is the Registry Operator for the `.com` gTLD but is also a well-known Registrar.

2. *Registrars*: Registrars are accredited by ICANN through a contractual process to interact with the public to register one or more gTLDs. When you purchase or renew a domain name, you deal with a Registrar. The Registrar maintains all the required details, including owner name, administrative contact, billing contact, technical contact, the authoritative name servers for the domain name, etc. The Registrar is responsible for providing the Registry Operator for the gTLD with an extract of the data, which consists of the Second-Level Domain Name and the name and IP addresses of the authoritative DNS servers for the Domain. This information is exclusively used to answer DNS queries.

The separation of functionality between the Registry Operator and the Registrar allows the relevant organizations involved to specialize and—importantly—ensures that specialists handle operation of the TLD name servers. Figure 1-5 illustrates this process.

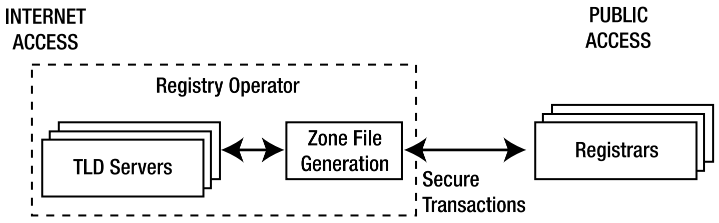


Figure 1-5. *Registry Operator–Registrar relationship*

ICANN inherited the gTLDs listed in Table 1-2 on its establishment in 1998.

Table 1-2. *gTLDs Available Prior to November 2000*

gTLD	Use	Registry Operator	Registrars
`.arpa`	Address and Routing Parameter Area (ARPA) reserved for use in Internet infrastructure	IANA (`www.iana.org/arpa-dom`)	Not available for registration
.com	Historically for abbreviation of company	VeriSign, Inc.	ICANN-Accredited Registrars
`.edu`	Special TLD reserved for use by certain US educational institutions	EDUCAUSE	EDUCAUSE (`www.educause.edu`)
`.gov`	Reserved exclusively for use by the US government	US General Services Administration	US General Services Administration (GSA)
`.int`	Reserved exclusively for use by organizations established by international treaty	IANA (`www.iana.org/int-dom`)	IANA
`.mil`	Reserved exclusively for use by the US military	US DOD Network Information Center	US DOD Network Information Center
`.net`	Historically for use by network operators	VeriSign, Inc. until June 2005	ICANN-accredited registrars
`.org`	Historically a nonprofit organization	Public Interest Registry (`www.pir.org`) DNS operated by Afilias Limited	ICANN-accredited registrars

In November 2000, ICANN authorized the following new gTLDs you see in Table 1-3.

Table 1-3. *gTLDs Authorized by ICANN in November 2000*

gTLD	Use	Registry Operator
`.aero`	Reserved for use by the airline industry	Société Internationale de Télécommunications Aéronautiques (SITA—`www.sita.aero`)
`.biz`	Generic business name domain	NeuLevel, Inc. (`www.neulevel.biz`)
`.coop`	Reserved for use by cooperatives	Dot Cooperation LLC (`www.cooperative.org`)
`.info`	Generic information resources	Afilias Limited (`www.afilias.info`)
`.museum`	Reserved for use by museums	Museum Domain Management Association (`http://musedoma.museum`)
`.name`	For use by individuals—vanity domain names	Global Name Registry (`www.gnr.name`)
`.pro`	Professional organizations	RegistryPro (`www.nic.pro`)

The ICANN agreements with the Registry Operators covering the post-2000 gTLDs have specified that information registration services and WHOIS services be made more easily available by reserving the use of `nic` and `whois` SLD names for each of the gTLDs. For example, to obtain registration information for the `.coop` gTLD, you need enter only `www.nic.coop` (or just `nic.coop`). To obtain WHOIS services for the `.museum` gTLD, you need enter only `www.whois.museum` (or `whois.museum`).

Note WHOIS is quite literally a service by which anyone can find "who is" the owner, and other pertinent details, of domain names or IP addresses. Registrars and in some cases third parties provide access to the registration databases using the standard WHOIS protocol (RFC 3912).

As may be seen from the list in Table 1-3, some of the gTLDs, such as `.aero`, have limited registration policies; others do not. During 2004, ICANN undertook a review of gTLD policy, one of the effects of which was to create a new gTLD *subset* called Sponsored TLDs (sTLDs) to clarify the form of registration access to be offered by new gTLDs. The domains `.museum`, `.coop`, and `.aero` are all now classified as sTLDs, as are the two new domains authorized in April 2005 and shown in Table 1-4. The domains `.biz`, `.info`, `.name`, and `.pro` have unrestricted registration policies. As of the time of writing, an additional eight possible new gTLDs were under consideration by ICANN.

Table 1-4. *sTLDs Authorized by ICANN in April 2005*

TLD	Use	Registry Operator
`.jobs`	Reserved for use by employment companies and human resources organizations	Employ Media LLC (`www.employmedia.com`)
`.travel`	Reserved for use by the travel industry	Tralliance Corporation (`www.tralliance.info`)

Country Code Top-Level Domains

Country Code Top-Level Domains are controlled by ICANN and consist of a two-character code defined by ISO 3166. ICANN has neatly sidestepped the thorny issue of what is a country by the use of ISO 3166. ISO 3166 is controlled by a branch of the United Nations, which is pretty experienced in the matter of defining what is, and what is not, a country!

ccTLDs are delegated by ICANN to a country code manager. *Country code manager* is a historic term reflecting a time when the Internet was a small and intimate place—more often today the country code manager is a branch of government, and the country-code itself has become a valuable economic resource.

The relationship between ICANN and country code managers is complicated by sovereignty and cultural sensitivity, and the process is largely consultative rather than contractual. It is a testament to the good will of all parties that the process works as well as it does. In general, country managers are responsible for administering and operating their delegated country codes and the associated TLD servers with regard to their local circumstances and within the spirit of RFC 1591.

The country delegation models are typically based on a federated model, for example, by state or province—`example.md.us`—or a functional model, for example, `example.co.uk` or `example.com.br`. However, many exceptions do exist reflecting local conditions and needs—the most famous that spring to mind are `.tv` (Tuvala) and `.la` (Laos), whereby those countries have sought to optimize the economic value of the domain name resource.

The Internet Assigned Numbers Authority (IANA) maintains a current list of country code managers at `www.iana.org/cctld/cctld-whois.htm` on behalf of ICANN.

DNS System Components

A Domain Name System includes three components:

1. Data that describes the domain(s) (called *zone files* and introduced in the text that follows)

2. One or more name server (DNS) programs

3. A resolver program or library

A single name server may support zero, one, or many domains. The data for each domain, or zone, describes global properties of the domain and the hosts (or services) provided by that domain. This data is defined in the form of textual *Resource Records (RRs)* organized in zone files. The format of zone files and their Resource Records is standardized in RFC 1035. Zone files are therefore portable across all standard DNS software.

The DNS program typically does three things:

1. It reads one or more zone files, which describe the domains for which it is responsible.

2. Depending on the DNS software functionality, it reads a configuration file, which describes various required behaviors (for example, to cache or not).

3. It responds to questions (queries) from local or remote clients (other name servers or resolvers).

The *resolver* program or library is installed on each host and provides a means of translating a user's request for, say, `www.example.com` into one or more queries to DNS servers using mostly the UDP protocol. A resolver is a complex program, but the standards allow for a much simpler implementation called a *stub resolver*. Almost all resolvers installed on Windows and *nix systems (for example, Linux, UNIX, and BSD) are stub resolvers. A web browser, for example, uses a stub resolver library to translate the name (or URL) entered into the address line of the browser, such as `www.example.com`, to an IP address, which it can use to access the required resource, in this case a web site, via the Internet.

Zones and Zone Files

The term *zone* and its relationship to the domain name can be very confusing. A zone file translates the domain name into operational entities, such as hosts, mail servers, services, and other characteristics, for use by DNS software. Subdomains delegated by the domain name owner are also described using zone files. The original DNS specifications called these *subzones*—a term that has mercifully disappeared over time. A zone file therefore describes that part of the domain name which is being handled by the DNS software—a zone designates an operational entity managed by a DNS or name server.

Zone files contain Resource Records, or RRs, that describe a domain or a subdomain. A zone file will typically consist of the following types of Resource Records:

1. Data that describes the zone authority, known as the *Start of Authority* (*SOA) Resource Record*. This Resource Record is mandatory in all zone files.

2. All hosts within the zone—typically defined using *Address* (*A) Resource Records*.

3. Data that describes global information for the zone—typically *MX Resource Records* describing the domain's mail servers and *NS Resource Records* describing the name servers that are authoritative for the domain.
4. In the case of subdomain delegation, the name servers responsible for this subdomain—using NS Resource Records.
5. In the case of subdomain delegation, a record (called a *glue* record and described in Chapter 8) that allows the name server to reach the subdomain name server(s)—typically one or more A Resource Records.

The following shows a simple example of a zone file showing most of the items mentioned in the preceding list. It is not important at this stage to understand the detail of each line, which is described in the next chapter.

```
; IPv4 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com.
; Start of Authority record defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h        ; refresh
                        15m        ; retry = update retry
                        3w         ; expiry
                        2h         ; min = minimum
                        )
; name servers Resource Records for the domain
             IN      NS      ns1.example.com.
; the second name servers is
; external to this zone (domain).
             IN      NS      ns2.example.net.
; mail server Resource Records for the zone (domain)
       3w    IN      MX  10  mail.example.com.
; the second  mail servers is
; external to the zone (domain)
             IN      MX  20  mail.anotherdomain.com.
; domain hosts includes NS and MX records defined above
; plus any others required
ns1          IN      A       192.168.254.2
mail         IN      A       192.168.254.4
joe          IN      A       192.168.254.6
www          IN      A       192.168.254.7
```

The individual Resource Records are described in Chapter 2, many more sample zone files are presented in Chapter 7, and a complete Resource Record reference is provided in Chapter 13.

Master and Slave DNS Servers

Early in this chapter, you saw that more than one name server is required to increase reliability and performance. It is not uncommon nowadays to see sites with four, five, or more name servers, each of which may be in a physically different location, and each of which must have access to the zone file. In order to reduce the management overheads involved in synchronizing zone files, the DNS specifications allow for a single DNS server to own a *master* copy of the zone file and to allow *zone transfers* (described in Chapter 3) to the other (*slave*) name servers. The terms *zone master*, or *master DNS*, and *zone slaves*, or *slave DNS*, are commonly applied to the respective name servers. The terms *master* and *slave* simply define which name server has the master copy of the zone file; they do not imply any priority of access. The master-slave relationship is illustrated in Figure 1-6.

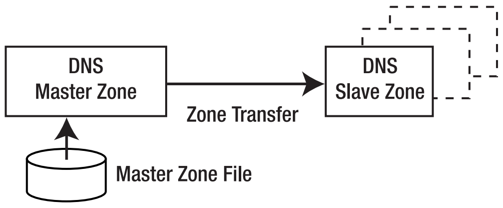


Figure 1-6. *Zone master and slave relationship*

■ **Note** In a perfect world, all terminology is unambiguous. The original DNS specifications used the terms *Primary* and/or m*aster* and *Secondary* (called *slave* previously) to describe the zone transfer process. The terms *Primary* and *Secondary* are still widely used to describe the *order* of DNS in many places such as registration of domain names and when defining network properties on PCs or hosts. In an attempt to reduce confusion, BIND introduced the terms *master* and *slave* in the context of zone transfers as shown earlier. This book will use these terms throughout. When reading other documents and purely in the context of zone transfers, Primary = master and Secondary = slave.

DNS Software

There is a dizzying choice of DNS software tailored to suit a range of user requirements. Berkeley Internet Name Domain—always referred to as BIND—is an Open Source implementation currently developed by the Internet Systems Consortium, Inc. (`www.isc.org`) and is probably the most widely known and deployed of the DNS implementations, and indeed most of this book documents BIND features. BIND, however, is by no means the only DNS solution available or for that matter the only Open Source DNS solution.

BIND has historically been viewed as the high-quality reference implementation of the IETF RFCs that specify DNS functionality. As a consequence, BIND has generally traded performance for generic functionality. The most recent BIND releases, BIND version 9, are placing a renewed emphasis on performance.

Historically, all the root-servers used BIND software. In order to encourage diversity, some of the root-servers now run the NSD DNS (`www.nlnetlabs.nl/nsd`) software, which provides a DNS implementation optimized for high performance. It has traded some generic functionality for raw performance, which may be up to twice that offered by an equivalent BIND configuration.

Microsoft Windows Servers are particularly well provided with DNS solutions. The Microsoft Server packages come bundled with a native DNS server. The current versions of BIND provide a binary package that will run on Windows NT 4.0 and Windows 2000 Server.

One of the major criticisms leveled over the years against many of the DNS software implementations is the lack of ability to dynamically add or remove zones without having to stop and start the DNS server. This criticism reflects both the increasingly dynamic nature of the Internet—more changes, more frequently—and the increased volume of traffic involved. Many users are reluctant to stop answering queries for even the seconds needed to stop and restart DNS software. While Dynamic DNS, supported by BIND and described in Chapter 3, allows editing of individual Resource Records within zones, it cannot add or remove entire zones. A number of the newer DNS implementations are providing database back-ends to contain the zone file data. This database can be dynamically updated and thus obviate the need to restart the DNS server. The corollary of this upside is that errors are also immediately propagated. With DNS data typically cached for 24 hours or longer, it can take a long time to remove even a simple error.

Which DNS solution best works for any user will reflect the functional and organizational requirements and—as always—will require clear understanding of the trade-offs and limitations that may be involved.

It is important to remember that the format of *zone* files used by DNS software is standardized by RFC 1035. Migrating from one implementation of DNS software to another can thus be considerably eased. Where a feature is unique to BIND (not standardized), it will be clearly indicated in the text.

Summary

This chapter introduced a lot of terminology and concepts that will be used throughout the rest of the book. The text described the need for name servers, which translate the descriptive name of a resource to its physical network address, and identified them as being essential for the operation of a dynamic and flexible network of any size.

The Internet's Domain Name System was introduced as being a specific implementation of the name server concept. You learned about the Internet's DNS domain name hierarchy, in particular the separation of the Top-Level Domains into Generic TLDs, for which ICANN is fully authoritative, and Country Code TLDs, which are administered by the individual sovereign countries. You now also know the component parts of a domain name, for instance, `www.example.com` consists of a host name (`www`), an SLD (`example`), and a TLD (`.com`). You also encountered the key concepts of an authority, the entity or person, responsible for a particular node in the domain name hierarchy, and delegation, the process by which the authority at a higher level in the domain name hierarchy may transfer authority to lower levels. The chapter finally introduced DNS software, the server and resolver programs that execute the DNS function, including BIND, the most widely used and implemented DNS server software.

Chapter 2 describes zone files and the most common Resource Records used in these files.

CHAPTER 2

Zone Files and Resource Records

A *zone file* describes or translates a *domain name* into the characteristics, hosts, and services provided by the *domain* in a way that can be used by DNS software. Badly configured zone files can make a domain unreachable, send e-mail to the wrong location, or even redirect customers to a competitor's web site. No question, these are serious consequences, but it gets worse. Answers to *queries* from a badly configured DNS may be *cached* (or *stored*) by other DNS systems for hours, days, or even weeks. It can take a long time for the effects of an error to be rectified—your customers or employees can be left without service or access for prolonged periods. Correctly configured zone files are essential to the running of every service offered by an organization with Internet presence.

This chapter describes the format and layout of zone files and the most common *Resource Records* (RRs) and *directives* that are used in the *forward mapping* of a zone. Forward mapping defines the zone characteristics and the IP addresses used by any hosts (or services) within the zone; for example, it could contain an RR that maps the host `www.example.com` to an IPv4 address such as 192.168.2.3. *Reverse-mapping* zones, which define the IP address-to-host relationship and the unique RRs used in their definition, are described in Chapter 3. A reverse-mapping zone file could, for instance, contain an RR that defines the IPv4 address 192.168.2.3 to have the name `www.example.com`. Because this topic is so central to DNS, zone files and their contents are discussed at length in several other chapters, most notably in Chapter 7, where we'll consider various zone file samples, and Chapter 13, which offers a complete reference on all the RRs and directives.

Zone File Format

Zone files are text files, standardized by RFC 1035, that may be read or edited using any standard editor and can contain three types of entries:

- *Comments*: All comments start with `;` (semicolon) and continue to the end of the line. Comments can additionally be added to any other record type and are assumed to terminate the line.
- *Directives*: All directives start with `$` and are used to control processing of the zone files.
- *Resource Records*: Resource Records are used to define the characteristics, properties, or entities contained within the domain. RRs are contained on a single line with the exception that entries enclosed in parentheses can spread across multiple lines.

The following is a sample zone file fragment that illustrates the preceding points and entry types:

```
; this is a full line comment
$TTL 12h    ; directive - comment terminates the line
$ORIGIN example.com.
; Start of Authority (SOA) record defining the zone (domain)
; illustrates an RR record spread over more than one line
; using the enclosing parentheses
@  IN  SOA  ns1.example.com. hostmaster.example.com. (
              2003080800 ; se = serial number
              3h         ; ref = refresh
              15m        ; ret = update retry
              3w         ; ex = expiry
              2h20m      ; min = minimum
              )
; single line RR
    IN  NS  ns1.example.com.
...
```

The preceding Start of Authority RR could have been written on a single line as shown here:

```
@  IN  SOA  ns1.example.com. hostmaster.example.com. 2003080800 3h 15m 3w 3h
```

■**Note** Standard RFC1035 zone files define time periods in seconds, which results in very large numbers. In the preceding fragment, the values `3h`, `15m`, `3w`, and `2h20m` use a BIND-specific short form for time-in-seconds values. The case-insensitive short forms allowed are `m` = minutes, `h` = hours, `d` = days, and `w` = weeks. The standards-compliant time-in-seconds values used previously would be `10800`, `900`, `1814400`, and `8400`, respectively. This book uses the BIND short format throughout simply because it is more easily understood. A number of alternative DNS implementations have adopted the BIND format as a de facto standard. If you want to stick to the standard and use seconds, keep a calculator handy.

Zone File Contents

One of the many confusing aspects of zone file definition is that it offers many shortcuts and ways to avoid excessive two-finger typing. In general, there is more than one way to do almost everything in a zone file. In the interests of clarity, this chapter uses a single zone file format to avoid confusion. Where appropriate, shortcuts and alternative formats will be illustrated.

In general, a zone file will typically contain the following Resource Records and directives, each of which is described in more detail later in the chapter:

- *The `$TTL` directive*: Defines the default *Time to Live* (TTL) value for the zone or domain—the time a Resource Record may be *cached* (or *saved*) by another DNS server. This directive is mandatory.

- *The `$ORIGIN` directive*: The domain name for the zone being defined. This directive is optional.
- *A Start of Authority (SOA) RR*: The SOA Resource Record, which must appear as the first RR in a zone file, describes the global characteristics of the zone or domain. There can be only one SOA RR in a zone file. This RR is mandatory.
- *The Name Server (NS) RR*: Defines name servers that are authoritative for the zone or domain. There must be two or more NS Resource Records in a zone file. NS RRs may reference servers in this domain or in a *foreign* or *external* domain. These RRs are mandatory.
- *The Mail Exchanger (MX) RR*: Defines the mail servers for the zone. There may be zero or more MX RRs in a zone file. If the domain does not provide e-mail services, there is no need for any MX RRs. An MX RR may reference a mail server in this domain or in a foreign or external domain. This RR is optional.
- *The Address (A) RR*: Used to define the IPv4 address of all the hosts (or services) that exist in this zone and which are required to be publicly visible. IPv6 entries are defined using AAAA (called *Quad A*) RRs. There may zero or more A or AAAA RRs in a zone file. This RR is optional.
- *The CNAME RR*: Defines an *Alias RR*, which allows one host (or service) be defined as the alias name for another host. There may be zero or more CNAME RRs in a zone file. This RR is optional.

Other Resource Record types and directives exist, some of which will be introduced in later chapters. You'll find a full list of Resource Record types and zone file directives defined in Chapter 13. The preceding RRs and directives allow the definition of a fully functional zone file.

An Example Zone File

The example zone file that appears later in this section illustrates the general format of a zone file and shows how Resource Records are used to describe the characteristics of the zone. Each directive and RR is described in detail and in the context of this example zone file. In this example, the zone `example.com` has the following characteristics:

- The zone has two name servers, one hosted in this domain (`ns1.example.com`), the other externally (`ns2.example.net`).
- The zone has two mail servers, one hosted in the domain (`mail.example.com`) and a second (backup) mail server hosted externally (`mail.example.net`).
- The zone has an internal web service with a name of `www.example.com`.
- The zone has an FTP server with a name of `ftp.example.com` (but provided by `ftp.example.net`).
- The zone has a single publicly visible host called `joe.example.com`.

The preceding scenario both illustrates some specific features of zone files and defines a zone that will provide some important services even in the event of failures or outages. Figure 2-1 shows the preceding configuration in operation.

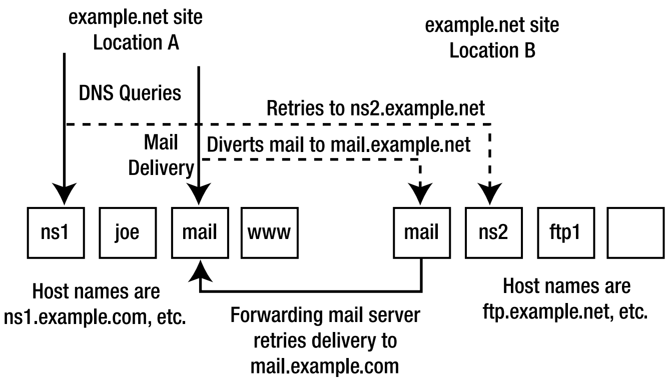


Figure 2-1. *Example configuration*

The configuration provides some simple resilience and will continue to accept mail even if the site at Location A is off-line for some period of time. It achieves resilience using the following strategies:

- There are two name servers located in separate physical locations. In the event `ns1.example.com` is unreachable, `ns2.example.net` will continue to provide DNS service for `example.com`. Failure to provide for geographical separation of name servers led to Microsoft's web sites being off-line for over 23 hours in one famous incident in 2001.[1]

- In the event that mail cannot be delivered to `mail.example.com`, the zone records (the MX RRs described in the section "The MX Resource Record" later in this chapter) will cause redirection to `mail.example.net`. The server `mail.example.net` would be configured as a forwarding mail server for the domain `example.com`. The mail server `mail.example.net` will retry at periodic intervals to deliver the mail to `mail.example.com`. No mail will be lost even during extended outages.

Many smaller sites think this kind of resilient configuration is only for large and complex organizations and therefore locate both the alternate name and mail servers on the same site. There is nothing wrong with this kind of configuration, and indeed it is very common—especially in smaller organizations and a surprising number of large ones as well. However, it is also easier

[1] `www.wired.com/news/technology/0,1282,41412,00.html`

than you think to organize peering by simply swapping backups with another friendly or non-competitive site (that is, *you back up for me and I'll back up for you*). Both sites gain the same resilience, and no money need change hands because the additional traffic should be negligible at both locations as long as the sites are reasonably similar in traffic volumes.

Clearly the web site at `www.example.com` would be non-operational during an outage of location A, but you may already see that by using the name server at `ns2.example.net` this could be replicated by simply defining an alternate IP address for the host `www.example.com` in the zone file used by this name server. Chapter 8 describes some additional ways to provide resilience using the DNS features.

The zone file that describes this configuration is shown here:

```
; IPv4 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com. ; base domain-name
; Start of Authority record defining the key characteristics
; of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; se = serial number
                        12h         ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        2h         ; min = minimum
                        )
; name servers Resource Records for the domain
              IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
              IN      NS      ns2.example.net.
; mail server Resource Records for the zone (domain)
; value 10 denotes it is the most preferred
     3w       IN      MX  10  mail.example.com.
; the second mail server has lower preference (20) and is
; external to the zone (domain)
              IN      MX  20  mail.example.net.
; domain hosts includes NS and MX records defined previously
; plus any others required
ns1           IN      A       192.168.254.2
mail          IN      A       192.168.254.4
joe           IN      A       192.168.254.6
www           IN      A       192.168.254.7
; aliases ftp (ftp server) to an external location
ftp           IN      CNAME   ftp.example.net.
```

The following sections explain each directive and RR type used in the example zone file.

The `$TTL` Directive

Every Resource Record may take an optional Time to Live value specified in seconds. The `$TTL` directive is standardized in RFC 2038 and defines the default TTL value applied to any RR that does not have an explicit TTL defined. TTL in the DNS context means the time in seconds that a record may be cached (stored) by another name server or in some cases a *resolver*. Caching is explained in Chapter 4.

The formal syntax for this directive is as follows:

```
$TTL time-in-seconds
```

From the example zone file:

```
$TTL 2d
```

The preceding `$TTL` directive uses the *BIND-specific* short form `d` to indicate days. The RFC 2038 format equivalent is as follows:

```
$TTL 172800
```

The time-in-seconds value may take the value 0, which indicates never cache the record, to a maximum of 2147483647, which is over 68 years! The current *best practice* recommendation (RFC 1912) proposes a value greater than one day, and on RRs that rarely change, you should consider multiweek values.

The TTL determines two DNS operational characteristics:

- *Access load*: The lower the TTL, the more frequently DNS queries will occur, and the higher the operational load on the zone's name server.
- *Change propagation*: The TTL value represents the maximum time that any change will take to propagate from the zone name server to all users.

It is simple to change the zone-wide TTL by altering a single `$TTL` zone file directive. Many users will set this to a very high value, say, two weeks or more, in normal operational use, and thus minimize name server access. When planned changes and upgrades occur that affect the zone records, for example, IP address changes or new service installation, the `$TTL` will be reduced in advance to a lower value, say 12 hours (`12h` or `43200`). When service has stabilized, the TTL will be restored to the previous high value. The value `2d` used in the example file represents a reasonable balance for stable zones.

The `$TTL` directive must appear before any RR to which it will be applied, and BIND 9 will now refuse to load a zone that does not have a valid `$TTL` directive.

■**Note** In older versions of BIND (prior to BIND 9), the default `TTL` value for the zone was defined in the SOA RR (described in the section "The SOA Resource Record" later in this chapter). RFC 2308 defines both implementation of the `$TTL` directive and the change to the SOA RR.

The `$ORIGIN` Directive

The `$ORIGIN` directive was standardized in RFC 1035 and defines the domain name that will be *appended* to any incomplete name (sometimes called an *unqualified* name) defined in an RR. This process, whereby a value is appended to names that do not end with a dot, is a major source of confusion, anger, and puzzlement when running DNS systems because the process happens invisibly.

■**The `$ORIGIN` Substitution Rule** If a name appears in a Resource Record and does *not* end with a dot, then the value of the last, or only, `$ORIGIN` directive will be appended to the name. If the name does end with a dot, then it is a *Fully Qualified Domain Name* (FQDN) and nothing will be appended to the name. This rule will be illustrated in the following section. The terminating dot in a FQDN is interpreted as the root of the domain tree or hierarchy. Recall from Chapter 1 that although this dot is normally *silent* (omitted), it is occasionally VERY important. This rule requires careful attention as to whether the dot is present.

The formal syntax for `$ORIGIN` is as follows:

```
$ORIGIN domain-name
```

Returning to the example zone file:

```
$ORIGIN example.com.
```

The name of the domain defined by this zone file—`example.com.`—is defined in the `$ORIGIN` directive. The `domain-name` is always a FQDN—it always ends with a dot. `$ORIGIN` directives can appear anywhere in a zone file and will be used from the point they are defined onwards:

```
$ORIGIN example.com.
; RRs from here will append example.com.
....
...
$ORIGIN us.example.com.
; RRs from here will append us.example.com.
...
```

The `$ORIGIN` directive is not mandatory. BIND will assume that the `$ORIGIN` value is defined by the name of the zone defined in its configuration file (the `named.conf` file described in Chapter 7). This book always uses an `$ORIGIN` directive for three reasons:

- A zone file is self-descriptive and self-contained, it requires no reference to any further information.
- The substitution rule (defined previously) is much less confusing—the value to be substituted is immediately apparent (that is, the last `$ORIGIN` directive).
- Not all software may implement the same assumptions about the `$ORIGIN` directive. Zone files are more portable when the directive is included.

It is always tempting to take shortcuts, but as with all things there may be consequences.

The SOA Resource Record

The SOA Resource Record defines the key characteristics and attributes for the zone or domain and is standardized in RFC 1035. As befits the most important RR in the zone file, it is among the most complex and takes a significant number of parameters. The formal syntax of the SOA RR is as follows:

```
name  ttl class rr name-server e-mail sn refresh retry expiry min
```

Here is the SOA RR from the example zone file:

```
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        3h         ; refresh time
                        15m        ; retry = update retry
                        3w         ; expiry
                        3h         ; min = minimum
                        )
```

The SOA RR has two layout rules:

- Typically uses the standard multiline format, in which case the open parenthesis, (, must appear on the first line; the closing parenthesis,), can appear on the same or any subsequent line.
- The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 2-1 maps the values from the example file to the formal syntax.

Table 2-1. *SOA RR Syntax*

Syntax	Example Usage	Description
`name`	`@`	The `@` symbol substitutes the current value of `$ORIGIN` (in the example file this is `example.com.`).
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` (172800 seconds) from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet. Other values exist but are rarely used. They are defined in Chapter 13 purely for the sake of completeness.
`name-server`	`ns1.example.com.`	Defines the Primary Master name server for the zone and has a special meaning only when used with Dynamic DNS configurations, which are covered in Chapter 3. The name server referenced here also needs to be defined using an NS RR. In DNS jargon this is called the `MNAME` field.
`e-mail`	`hostmaster. example.com.`	Defines an administrative e-mail address for the zone. It is recommended in RFC 2142 that the e-mail address *hostmaster* is used for this purpose, but any stable and valid e-mail address can be used. While this field uses unusual dot separators (the `@` symbol has special significance in a zone as described earlier) to define the e-mail address, in the case of the example file, mail will be sent to `hostmaster@example.com`. In DNS jargon this is known as the `RNAME` field.

Syntax	Example Usage	Description
sn	2003080800	Defines the serial number currently associated with the zone. The serial number *must* be updated every time any change is made to the domain. sn can take any number in the range 0 to 4294967295. By convention, but this is only a convention, a date format is used with the form yyyymmddss, where yyyy is the four-digit year number, mm is the month, dd is the day, and ss is the sequence number in case the zone file is updated more than once per day! The value from the example zone file indicates that the last update was on August 8, 2003. This value is used during zone transfer operations (described in Chapter 3) to determine whether the zone file has been changed. Recovery from an out-of-sequence sn value is not trivial as you'll see in Chapter 8. Extreme care should be taken when updating this number. The use of the date convention is designed to minimize errors as well as provide a simple way to track the date of the last change to the zone.
refresh	12h	When the refresh value is reached, the slave name server (described in Chapter 1) for this zone will try and read the SOA RR from the zone master. If the sn value in the SOA RR is higher than that currently stored by the slave, a zone transfer operation is initiated to update or refresh the slave's copy of the zone records. Depending on the how zone transfers are implemented, the value of this parameter may determine how quickly changes are propagated from the *master* to the *slave*. Zone transfers are described in Chapter 3. Typical values are from 3 to 24 hours.
retry	15m	Defines the retry interval in seconds if the *slave* fails to make contact with the zone master during a refresh cycle. Typical values are from 10 to 60 minutes.
expiry	3w	Defines the time in seconds after which the zone records are assumed to be no longer authoritative. BIND interprets this to mean that the records can no longer be considered valid and consequentially stops responding to queries for the zone. Thus, when the refresh time limit is reached, the slave will try and contact the zone master and in the case of a failure will attempt reconnection every retry period. If contact is made, both the refresh and expiry counts are reset. If the slave has failed to make contact when expiry is reached, the slave will stop responding to any queries. The zone is essentially dead at this point. To allow for major outages, expiry is typically set to a very high value—1 to 3 weeks.
min	3h	min was redefined in RFC 2308 to be the period of time that negative responses can be cached by the slave. Thus if a request is made for fred.example.com and it cannot be resolved—it does not exist—then the slave will return NO DOMAIN error (NXDOMAIN). The slave will continue to return this value until min expires, at which point it will retry the failing operation. BIND allows a min value in the range 0 to 10800 (three hours).

To illustrate the use of the $ORIGIN statement and its substitution rule, this zone file fragment shows how it is possible to rewrite the SOA statement:

```
; fragment from example - does not use substitution
$TTL 2d    ; default TTL for zone
$ORIGIN example.com.
; Start of Authority record defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; se = serial number
                        12h         ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        2h         ; min = minimum
                        )
```

The SOA RR could be rewritten to use the `$ORIGIN` substitution rule as shown here:

```
; fragment rewritten to use $ORIGIN substitution
$TTL 2d    ; default TTL for zone
$ORIGIN example.com.
; Start of Authority record defining the key characteristics of the zone (domain)
@         IN      SOA   ns1 hostmaster (
                        2003080800 ; se = serial number
                        12h         ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        2h         ; min = minimum
                        )
```

In the preceding fragment, because `ns1` and `hostmaster` are not FQDNs (they do not end with a dot), the value of the `$ORIGIN` is appended to each name, creating `ns1.example.com.` and `hostmaster.example.com.`, respectively, as in the initial example file. This format is rarely seen, however, as it can be quite confusing, although it is technically and functionally correct.

■**Note** The `name` field used in all the RRs (termed in DNS jargon a *label*) was originally defined to allow any letter, digit, or a dash (—); names or labels must start and end with a letter or a number. The specifications were liberalized by RFC 2181 to allow underscores (_), but there are reputedly still implementations that do not allow them in host names, so it is safest to avoid underscores if possible.

The NS Resource Record

The NS Resource Record is standardized in RFC 1035 and defines the authoritative name servers (there must be at least two) for the domain or zone. The NS RR syntax is as follows:

```
name     ttl    class   rr    name
```

Let's return to the example file:

```
; name servers Resource Records for the domain
            IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
            IN      NS      ns2.example.net.
```

The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 2-2 maps the formal syntax to the first NS record used in the example zone file, which is internal to the zone.

Table 2-2. *NS RR Syntax*

Syntax	Example Usage	Description
`name`		This field is blank (may be either a space or a tab character) and implicitly substitutes the current value of the name field (in this case, the name field of the SOA RR). You could also write this record as `example.com.    IN  NS ns1.example.com.`, which may be less confusing. This is an example of how the same result may be achieved in different ways.
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`name`	`ns1.example.com.`	Defines a name server that is authoritative for the domain. In this example, an FQDN format has been used, but it could have been written as just `ns1` (without the dot) and `$ORIGIN` substitution would take place. This NS record points to a name server within the domain and therefore *must* have a corresponding A RR for IPv4 (or AAAA RR if IPv6) defined.

The second NS RR from the example file is as follows:

```
            IN      NS      ns2.example.net.
```

This is the *classic* method of defining a second name server for the domain. In the event that one name server is not available, the alternate server, ideally at a geographically different location, will be used, thus ensuring access to services such as mail even if the main site is not available due to backbone, power, or other system outages.

The second NS RR is defined to be in a foreign or external zone and therefore does not require an A RR if IPv4 (or AAAA RR if IPv6). In addition, it *must* be defined using an FQDN—it must terminate with a dot. To illustrate the possible errors that may be caused inadvertently by `$ORIGIN` substitution, assume that the terminating dot on this RR was omitted in error, that is, it was written as `ns2.example.net` (without a terminating dot). DNS software would apply substitution and create a name of `ns2.example.net.example.com.`—not the desired result!

■ **Note** The external name server (`ns2.example.net`) must contain a zone file, and be either a master or a slave, for the zone `example.com`. Failure to do so will result in what is called *lame delegation*. Lame delegation occurs when an NS RR points to a name server that does not answer authoritatively for the zone or domain.

The MX Resource Record

The MX RR is standardized in RFC 1035 and defines the mail servers (or mail exchangers in the quaint DNS jargon) for the domain or zone. The formal syntax is as follows:

```
name     ttl    class   rr  preference  name
```

In the example file, the following MX RRs are defined:

```
; mail server Resource Records for the zone (domain)
     3w       IN      MX  10  mail.example.com.
; the second  mail server is
; external to the zone (domain)
              IN      MX  20  mail.example.net.
```

The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 2-3 maps the formal syntax to the first MX record used in the example file, which is internal to the domain.

Table 2-3. *MX RR Syntax*

Syntax	Example Usage	Description
`name`		This field is blank and implicitly substitutes the value of the right-hand name field from the previous RR (in the example file, this is `example.com.`).
`ttl`	`3w`	This illustrates the use of an explicit `ttl` value in a Resource Record that overrides the zone default (defined in the `$TTL` directive). The value shown (3 weeks) is significantly higher than the example zone default, which is 2 days. Because the domain MX RR is unlikely to change—its corresponding A Resource Record may change more frequently—why not minimize the DNS load on what is a normally very actively used RR type? The `ttl` can, however, take any value required including omission, in which case the zone default will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`preference`	`10`	The *preference* field indicates the relative preference or priority of the mail server it defines and can take any value between 0 and 65535. The lower the number, the more preferred is the server. Traditionally, the most preferred mail server has the preference value 10. There is absolutely no reason for this other than it allows another MX record with a more preferred value (a lower number) to be added without changing any other record!

Syntax	Example Usage	Description
name	mail.example.com.	Defines a mail server with the defined `preference` value for the domain. In this example, an FQDN format has been used, but you could write this as just `mail` (without the dot), and `$ORIGIN` substitution will take place. This MX record points to a mail server within the domain and therefore must have the corresponding A RR for IPv4 (or AAAA for IPv6) defined.

The second MX RR from the example file is as follows:

```
            IN      MX  20  mail.example.net.
```

This is the *classic* method of defining a backup mail server, which has a lower preference value—20 in the example case. In the event that the first mail server is not available, the backup mail server, ideally at a geographically different location, would be used. This backup mail server would normally be defined as a simple forwarding mail server for the domain—constantly attempting to pass the mail to the most preferred (or Primary) mail server (`mail.example.com`) when service is happily restored.

The second MX RR is defined to be in a foreign or external domain and therefore does not require an A RR if IPv4 (or an AAAA RR if IPv6) and *must* always be an FQDN—it must end with a dot.

The A Resource Record

The A RR is standardized in RFC 1035 and defines the IPv4 address of a particular host in the domain or zone. The equivalent RR for IPv6 is the AAAA RR described in Chapter 5. The formal syntax of the Address RR is as follows:

```
name     ttl    class   rr  ipv4
```

In the example file, the following A RRs are defined:

```
ns1         IN      A       192.168.254.2
mail        IN      A       192.168.254.4
joe         IN      A       192.168.254.6
www         IN      A       192.168.254.7
```

The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 2-4 maps the formal syntax to the first A RR used in the example zone file.

Table 2-4. *A RR Syntax*

Syntax	Example Usage	Description
`name`	`ns1`	The name is unqualified, causing `$ORIGIN` substitution. You could write this as `ns1.example.com.` (using the FQDN format), which may be more understandable.
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`ipv4`	`192.168.254.2`	Defines that the host `ns1` has the physical IPv4 address192.168.254.2. Records defined by NS or MX RRs that have names contained within this domain *must* have corresponding A RRs as shown in the example zone file for `ns1` and `mail`. Any other hosts the user wishes to make publicly visible are also defined using A RRs; in the example file, this includes the web service (`www`) and the host named `joe` for some reason best known to the owner of the domain.

It is permissible to define the same IP address with multiple names as shown in the following fragment, in which the name server and the web server are colocated on the same machine:

```
ns1           IN      A       192.168.254.2
mail          IN      A       192.168.254.4
joe           IN      A       192.168.254.6
; this A RR has the same IPv4 address as ns1 above
www           IN      A       192.168.254.2
```

The same result can be achieved using a CNAME record (see the code fragment that follows). Multiple IP addresses can also be defined for the same host as in this fragment, where three IPv4 addresses are provided for the host `www.example.com`:

```
www           IN      A       192.168.254.2
              IN      A       192.168.254.7
              IN      A       192.168.254.8
```

DNS software will supply the defined IP address in a round-robin or random order (defined by configuration directives) to successive queries. This feature may be used to provide load balancing and is further described in Chapter 8. The preceding fragment also illustrates the use of a *null* or *blank* name to inherit the previous name, that is, all the entries with a blank name relate to `www` (and assuming an `$ORIGIN` directive of `example.com` will define `www.example.com`).

CNAME Resource Record

The CNAME RR is standardized in RFC 1035 and defines an *alias* for an existing host defined by an A RR. The formal syntax is as follows:

```
name     ttl    class   rr  canonical-name
```

In the example file, the following CNAME RR is defined:

```
ftp            IN      CNAME   ftp.example.net.
```

The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 2-5 maps the formal syntax to the CNAME RR used in the example zone file.

Table 2-5. *CNAME RR Syntax*

Syntax	Example Usage	Description
`name`	`ftp`	The name is unqualified, causing the `$ORIGIN` directive value to be substituted. You could write this as `ftp.example.com.` (using the FQDN format), which may be more understandable.
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`canonical-name`	`ftp.example.net.`	Defines that the name `ftp.example.com` is *aliased* to the host `ftp.example.net.` in a foreign or external domain. In DNS jargon, `ftp.example.net.` is referred to as the *canonical name*, which simply means the expected or real name.

CNAME RRs are often used when assigning *service* names to existing hosts, for example, if a host is actually called `bill` but runs an FTP and a web service, then CNAME RRs are frequently used to define these services as in the following fragment:

```
ftp            IN      CNAME       bill
www            IN      CNAME       bill
bill           IN      A           192.168.254.21
```

CNAME RRs have some limitations. It is permissible but considered very bad practice to chain CNAME records.

```
ns1            IN      A       192.168.254.2
mail           IN      A       192.168.254.3
joe            IN      CNAME   www.example.com.
www            IN      CNAME   mail.example.com.
```

CNAME records should not be used with either NS or MX records; thus in the example file, if the mail server and web server were colocated on the same host, the following would be technically *invalid* but would typically work—an approach that is widely used!

```
               IN      MX        mail.example.com.
mail           IN      CNAME   www.example.com.
www            IN      A       192.168.254.7
```

The following fragment is valid and achieves the same result:

```
               IN      MX        mail.example.com.
mail           IN      A       192.168.254.7
www            IN      CNAME   mail.example.com.
```

The rule defining the preceding (RFC 1034 section 3.6.2) is cautious in regard to use of excessive indirection and says that if a name appears on the right-hand side of an RR (as `mail.example.com` does in the preceding MX RR in the fragment), it should not appear in the left-hand name of a CNAME RR. Many working configurations use this construct routinely. There is always a risk that one day the specification may be tightened and the configuration may not work.

You need to be aware of two other consequences when using CNAME RR. First, CNAME causes the name server to do more work—both the CNAME and the CNAME'd RR must be looked up by the name server. In high-volume name servers, this additional workload may be a consideration. Second, the CNAME RR and the target (CNAME'd) RR record are returned in the answer. When dealing with large answers, this may cause the response to exceed the 512-byte limit of a DNS UDP transaction, thus reducing performance.

When CNAME Records Must Be Used

As noted earlier, CNAME RRs are frequently and commonly used to map services such as FTP, web, gopher, and others onto a single host. Multiple A RRs may also be used to achieve the same result. In general, only two cases require CNAME records to be used—there is no alternative available. The first is when the real or *canonical* host lies in a foreign or external domain, as illustrated in the example file where `ftp.example.com` is aliased to `ftp.example.net`. The second is when the user wishes to address a web site using either `www.example.com` or just `example.com`. In this case, the functionality would be implemented using the following fragment:

```
; define an IP that resolves to example.com
              IN      A       192.168.254.7
; alias www.example.com to example.com
www           IN      CNAME      example.com.
```

The preceding definition will require a configuration change to your web server, which is fully covered in Chapter 8.

Additional Resource Records

In this chapter, you have seen the main RRs used in constructing zone files. Many more RR types exist and are documented, with examples, in Chapter 13. For the sake of completeness, some of the more commonly used additional or specialized RRs are briefly described in the following sections.

PTR Resource Records

Pointer (PTR) RRs are used only for *reverse-mapping* zones and are the corollary of the Address RRs. PTR RRs map an IPv4 address to a name—an A RR maps a name to an IPv4 address. Reverse mapping and PTR records are described in Chapter 3. PTR RRs are also used when *reverse mapping* an IPv6 zone, as you'll see in Chapter 5.

TXT Resource Records

Text (TXT) RRs were historically used to define generic text to be associated with a name. The text may be anything the user wishes. The Sender Policy Framework (SPF) antispam initiative uses the TXT RR to carry its information. You'll find the SPF record format defined in Chapter 8 and the generic TXT RR in Chapter 13.

AAAA Resource Records

The AAAA RR is used to define *forward mapping* of IPv6 hosts, which is covered in Chapter 5.

NSEC, RRSIG, DS, DNSKEY, and KEY Resource Records

These RRs are used in Secure DNS (DNSSEC) configurations as described in Chapters 10 and 11.

SRV Resource Records

Service (SRV) RRs are relatively new and used to map services onto hosts. Chapter 13 describes the SRV RR, and Chapter 8 contains a discussion of the use of SRV records when used in load balancing and resilience.

Standard Configuration File Scenarios

Chapter 7 defines further example configurations, including the required zone files for common DNS types such as master, slave, caching, forwarding, and authoritative-only name servers. Chapter 8 contains a number of common configurations that illustrate various aspects of zone files, and Chapter 13 includes a full reference section on zone files and Resource Records.

Summary

This chapter described the format and content of zone files. You learned about the `$TTL` directive, used to set the default TTL for the zone. You also encountered the `$ORIGIN` directive, used to set the base name for the zone, and the `$ORIGIN` substitution rule, the cause of much DNS aggravation. Using the example zone file as a guide, the text explained the various Resource Record types used to construct basic zone files such as the Start of Authority, Name Server, Mail Exchanger, and Address Resource Records.

Chapter 3 explains DNS operations: the types of DNS queries that may be used; reverse mapping, the process by which an IP address may be mapped to a host name; zone transfers, the method by which zone files are updated from the master to the slave name servers; and finally, a brief overview of the security issues involved in running a DNS service.

CHAPTER 3

DNS Operations

This chapter describes the operation of a DNS system. Namely, you'll learn about the following topics:

- *DNS queries*: How does your browser find `www.example.com`? How does your mail software know where to send your outgoing e-mail? Such operations use DNS queries.
- *Reverse mapping*: How does your mail software determine your identity? How do you find out who is hacking your system? These types of operations use a technique known as reverse mapping.
- *Zone maintenance*: How does the address of your new FTP service get propagated across the Internet? How are your customers notified of any change to your e-mail provider? These operations use zone maintenance.
- *DNS security*: How do you prevent your web site being hijacked? How do you ensure that your e-mail is delivered to you and not someone else? In this chapter, you'll learn key DNS security concepts.

This chapter references configuration directives defined in BIND's `named.conf` file, which controls its operational behavior. Chapter 12 describes these directives. A number of zone files containing DNS Resource Records (RRs) are used to illustrate certain points, as discussed in Chapter 2. Chapter 13 contains a complete reference on each record type. The PTR RRs used in reverse mapping of a zone are described in this chapter.

But first let's get some simple DNS protocol details out of the way.

The DNS Protocol

DNS operations, for example, queries and zone maintenance operations, by default use port 53. For performance reasons, queries use the UDP protocol with a block-size limit of 512 bytes. TCP can be optionally negotiated on a transaction-by-transaction basis for query operations, but due to the performance overhead incurred with TCP, this is essentially a theoretical capability. However, if the response to a query exceeds 512 bytes, TCP is negotiated and used. Exceeding the 512-byte response size limit is typically avoided at all costs, and indeed the limit of 13 root-servers is the maximum that can be returned in a single 512-byte UDP transaction. Zone maintenance operations for reliability reasons use TCP, again by default on port 53.

■**Note** The preceding information is for normal DNS operations. When using DNSSEC (see Chapter 11), response data volumes can increase significantly, and a feature known as *EDNS0* is used to negotiate a UDP block size greater than 512 bytes. BIND can be optionally configured to use a port other than 53 for operations.

DNS Queries

The major task carried out by a name server is to respond to *queries* (questions) from a local or remote resolver or another name server acting on behalf of a resolver. The *resolver* (or more normally the *stub resolver*) is the software library installed on each PC used to translate a user or application request to a query to the local DNS. For instance, a typical query would be "What is the IP address of `www.example.com`?" The resolver will use a locally configured DNS server to perform the queries. Figure 3-1 illustrates this process.

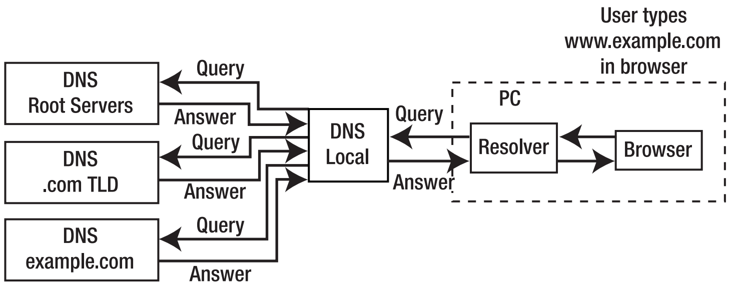


Figure 3-1. *DNS queries*

■**Note** The list of name server systems used by the resolver (or stub resolver) is obtained from the Network Properties in Windows systems and from `/etc/resolv.conf` in Linux, BSD, or UNIX systems.

A name server may have zone files that define it to be authoritative for some (if any) domains and slaves for others, and may be configured to provide caching, forwarding, or other behaviors for other domains or users. Zone files were introduced in Chapter 2.

There are three types of queries defined for DNS systems:

1. *Recursive queries*: A recursive query is one in which the receiving name server will do all the work necessary to return the complete answer to the question. Answering a query recursively may involve the name server in multiple transactions to a number of other name server systems. Name servers are not required to support recursive queries.

2. *Iterative (or nonrecursive) queries*: In an iterative query, if the name server already has the answer, it will return it. If the name server does not have the answer, it will return any information that may be useful, but it will not make additional requests to other name servers systems. All name servers must support iterative queries.

3. *Inverse queries*: The user wants to know the domain name given a Resource Record. Name servers were not required to support inverse queries, and the feature was rarely, if ever, implemented. It finally succumbed to the inevitable when RFC 3425 declared it to be obsolete.

■**Note** The process called *reverse mapping*, which returns a host name given an IP address, does not use inverse queries but instead uses recursive and iterative (nonrecursive) queries using the special domain name `IN-ADDR.ARPA`. Historically, reverse IPv4 mapping was not mandatory. Many systems—especially mail servers—now use reverse mapping for simple security and authentication checks, so proper implementation and maintenance of reverse mapping is now practically essential.

Next, I'll introduce each type of query in further detail.

Recursive Queries

A *recursive query* is one that the name server fully answers (or gives an error). Name servers are not required to support recursive queries, and the resolver (or another name server acting recursively on behalf of another resolver) negotiates the use of recursive service using bits in the query headers. There are three possible responses to a recursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful for example the response to a query for an A RR will follow any CNAME chain. The response will indicate whether the data is authoritative or cached (nonauthoritative).

2. An error indicating the domain or host does not exist (`NXDOMAIN`). This response may also contain CNAME records that pointed to the nonexisting host.

3. A temporary error indication—for instance, it can't access other name servers due to network error, etc.

In a recursive query a name server will, on behalf of the client (resolver or stub resolver), chase the trail of name servers across the universe to get the real answer to the question. The journey of a simple recursive query such as "What is the IP address of `www.example.com`?" to a name server that supports recursive queries but is not *authoritative* for `example.com` (it is not the master or slave for `example.com` zone) will look something like this:

1. A user types the URL `http://www.example.com` into a browser.

2. The browser sends a request for the IP address of `www.example.com` to its resolver.

3. The resolver queries the locally configured name server for the IP address of `www.example.com`.

4. The local name server looks up `www.example.com` in local tables (its *cache*)—but it's not found.
5. The local name server sends a query to a root-server for the IP (the A RR) of `www.example.com`.
6. The root-server only supports iterative (nonrecursive) queries (see the upcoming section "Iterative (Nonrecursive) Queries") and answers with a list of name servers that are authoritative for the gTLD `.com` (this is called a *referral*).
7. The local name server selects one of the authoritative gTLD servers and sends a query for the IP of `www.example.com`.
8. The gTLD name server only supports *iterative* queries and answers with the authoritative name servers for the Second-Level Domain (SLD) `example.com` (a referral).
9. The local name server selects one of the authoritative DNS servers for `example.com` and sends a query for the IP (the A RR) of `www.example.com`.
10. The zone file for `example.com` defines `www.example.com` as a CNAME record (an alias) for `joe.example.com`. The authoritative name server answers with the `www.example.com` CNAME RR and, in this case, the A RR for `joe.example.com`, which we will assume is 192.168.254.2.
11. The local name server sends the response `joe.example.com=192.168.254.2` (together with the CNAME RR `www=joe`) to the original client resolver.
12. The resolver sends `www.example.com=192.168.254.2` to the user's browser.
13. The browser sends a request to `192.168.254.2` for the web page.

Figure 3-2 illustrates a recursive query in action.

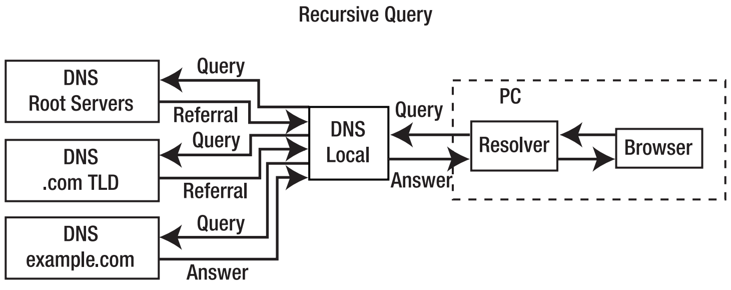


Figure 3-2. *Recursive query*

Which Name Server Is Used

In the case where multiple name servers are available, as is the case with the root-servers or gTLD servers in the preceding explanation, which one should our local DNS use? Most name servers use some algorithm to spread the load and therefore ensure the fastest possible result. In the case of BIND, it maintains a metric called the *round-trip time (RTT)* in which it tracks the response time to queries from each name server. When a list of name servers is initially supplied in a referral, each name server has an RTT of zero (there is no RTT). In this case, BIND will access each name server once in a round-robin at the end of which an RTT metric is available for each name server. Thereafter BIND will select the name server with the lowest RTT and continue to use it until its RTT exceeds the RTT of one of the other name servers, at which time that name server becomes the preferred choice.

Iterative (Nonrecursive) Queries

An *iterative* (or *nonrecursive*) query is one where the name server may provide a partial answer to the query (or give an error). Name servers must support nonrecursive queries.

There are four possible responses to a nonrecursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful. The response will indicate whether the data is authoritative or cached (non-authoritative).

2. An error indicating the domain or host does not exist (`NXDOMAIN`). This response may also contain CNAME records that pointed to the nonexisting host.

3. An temporary error indication—for example, can't access other DNSs due to network error, etc.

4. A referral—a list of two or more name servers (and IP addresses) that are closer to the requested domain name. These may or may not be the authoritative name servers for the final domain in the query. A referral is the normal response method used by root-servers and TLD servers since both name server types only support iterative queries.

The journey of a simple query such as "What is the IP address of `www.example.com`?" to a name server supporting iterative (nonrecursive) queries but that is not authoritative for `example.com` would look something like this:

1. A user types the URL `http://www.example.com` into a browser.

2. The browser sends a request for the IP address of `www.example.com` to its resolver.

3. The resolver on a host sends the query "What is the IP address of `www.example.com`?" to its locally configured name server.

4. The local name server looks up `www.example.com` in local tables (its cache)—but it's not found. The local name server responds with a referral containing the list of root-servers.

5. The resolver sends a query to a root-server for the IP (the A RR) of `www.example.com`.

6. The root-server answers with a list of name servers that are authoritative for the gTLD `.com` (a referral).

7. The resolver selects one of the authoritative gTLD servers returned in the referral and sends a query for the IP of `www.example.com` directly to that name server—not the locally configured DNS.

8. The gTLD name server answers the resolver with the authoritative name servers for the SLD `example.com`.

9. The resolver selects one of the authoritative SLD name servers returned in the referral and sends a query for the IP of `www.example.com` directly to that name server (not the locally configured name server).

10. The zone file for `example.com` defines `www.example.com` as a CNAME record (an alias) to `joe.example.com`. The authoritative name server answers with the `www` CNAME RR and, in this case, the A RR for `joe.example.com`, which we will assume is 192.168.254.2.

11. The resolver sends `www.example.com=192.168.254.2` to the browser.

12. The browser sends a request to `192.168.254.2` for the web page.

Figure 3-3 illustrates an iterative query.

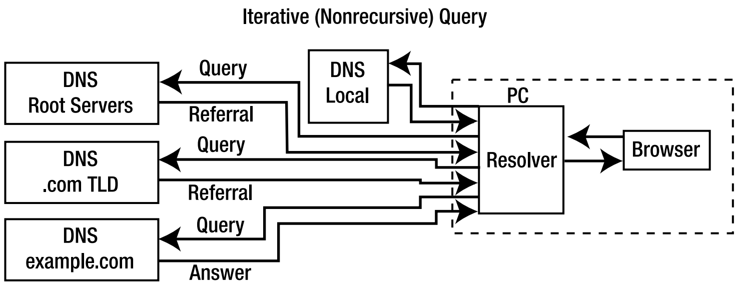


Figure 3-3. *Iterative query*

The preceding sequence is very artificial. A resolver technically must be capable of following referrals. The resolver that is installed on most common systems—this includes Windows, Linux, BSD, and Unix systems—is in fact a stub resolver. A stub resolver, which is defined in the standard, is a minimal resolver that cannot follow referrals. In general, locally configured name servers used by PCs or workstations *must* support recursive queries to avoid returning referrals to the stub resolver.

■**Note** Windows 2000 and XP have what is called a *caching resolver*. This is a stub resolver—it cannot follow referrals—with a simple cache to increase performance and reduce network traffic.

Inverse Queries

An *inverse query* maps a Resource Record to a domain. An example inverse query would be "What is the domain name for this MX record?" Inverse query support was always defined to be an optional service within the DNS specifications, and it was permitted for name servers to return a response of "Not implemented" (`NOTIMP`), which they almost invariably did! Consequently, inverse queries were not widely used and were quietly put to rest when they were made obsolete by RFC 3425.

At first blush it may seem obvious that inverse queries are used to find a host name given an IP address. This not the case. The IP to host query process is called *reverse mapping* or *reverse lookup* and uses normal recursive and iterative (nonrecursive) queries with the special domain `IN-ADDR.ARPA`. Reverse mapping is introduced in the next section.

DNS Reverse Mapping

Given a domain name, a normal DNS query tries to determine its IP address. At times, however, you'll find it useful to be able to determine the name of the host given a particular IP address. While sometimes this is required for diagnostic purposes, more frequently these days it is used for security reasons to trace a hacker or spammer; indeed, many modern mailing systems use reverse mapping to provide simple authentication by using DNS lookup policies, for instance, IP-to-name and name-to-IP, to confirm that the specified IP address does represent the indicated host.

In order to perform reverse mapping using normal recursive and iterative queries, the DNS designers defined a special (reserved) domain name called `IN-ADDR.ARPA`. The next section describes how this special domain is constructed.

`IN-ADDR.ARPA` Reverse-Mapping Domain

Reverse mapping can look very complicated. It is, however, an elegant and simple concept and uses a simple variation of the domain name hierarchy introduced in Chapter 1.

The normal domain name structure is hierarchical starting from the root. A domain name is written left to right, but the hierarchical structure is right to left.

```
domain name = www.example.com
```

The highest node in the hierarchy (or tree) is `.com`, the Top-Level Domain (TLD); the next (lower) is `.example`, the Second-Level Domain; and finally the lowest is `www`, which is the host name and, recall from Chapter 2, is always defined in a zone file. To enable an IPv4 address to be used in a normal query operation, it must converted into a domain name as described next.

An IPv4 address is written as follows:

```
192.168.254.17
```

This IPv4 address defines a host address of 17 in the Class C address range 192.168.254.x (see the sidebar "IPv4 Addresses and CIDR"). In this case, the most important part (the highest node) is on the left (192), not the right. This is a tad awkward and would make it impossible to construct a sensible tree structure that could be searched in a single lifetime.

The solution is elegantly simple: to create the domain name, reverse the order of the address and build the hierarchy under the special domain name `IN-ADDR.ARPA` (the SLD is `IN-ADDR`, the TLD is `ARPA`).

■**Note** `IN-ADDR.ARPA` can also be written as `in-addr.arpa`, which is perfectly acceptable since domains are case insensitive; but the case *should* be preserved, so this book will continue to use `IN-ADDR.ARPA`.

Finally, the last part of the IPv4 address (17) is the host address and hosts, as you may recall from Chapter 2, are always defined inside a zone file. The result of the preceding manipulation is as follows:

```
IPv4 address =192.168.254.17
Class C base = 192.168.254 ; omits the host address = 17
Reversed Class C base = 254.168.192
Added to IN-ADDR.ARPA domain = 254.168.192.IN-ADDR.ARPA
```

The organization of the `IN-ADDR.ARPA` domain is shown in Figure 3-4.

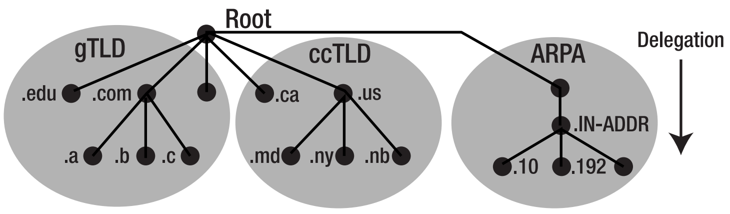


Figure 3-4. `IN-ADDR.ARPA` *reverse mapping*

Finally, a zone file is constructed to describe all the hosts in the reverse-mapped zone using the special PTR Resource Record, which is described in the next section. The resulting zone file will look something like this:

```
; simple reverse mapping zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN 254.168.192.IN-ADDR.ARPA.
; Start of Authority record defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh
                        15m        ; retry
                        3w         ; expiry
                        2h         ; min = minimum
                        )
```

```
; name servers Resource Records for the domain
          IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps an IPv4 address to a host name
2         IN      PTR     ns1.example.com.
.....
4         IN      PTR     mail.example.com.
.....
16        IN      PTR     joe.example.com.
17        IN      PTR     www.example.com.
....
```

Note The PTR RRs in the preceding zone file use Fully Qualified Domain Names (FQDN) ending with a dot because of the `$ORIGIN` substitution rule, which was described in Chapter 2.

IPV4 ADDRESSES AND CIDR

An IPv4 address is a 32-bit value that allows 4,294,967,296 unique addresses. It's difficult to remember numbers of this size, so the conventional way of writing an IP address is in *dotted decimal* format, for example, 192.168.23.17.

A dotted decimal IP address is constructed as follows:

1. A 32-bit address contains 4 × 8-bit bytes (or octets).
2. Each 8-bit byte (octet) may represent 256 (0–255) values. The internal (machine) representation of the byte (octet) is known as *hexadecimal* and may contain the hexadecimal values 00 to FF.
3. Dotted decimal simply converts the 8-bit value for each byte (octet) to its decimal value (which is always in the range 0 to 255) and separates each value with a dot to make it more readable.

Each IPv4 address has two components: a *network address* and a *host address*. The boundary between, or the number of bits in, the network address part and the host address part is determined by the *address class* and the *netmask* or the s*ubnetmask*.

Before the advent of *Classless Inter-Domain Routing* (CIDR), the world was a simple place—we had four classes of IPv4 addresses: A, B, C, and D (there is also a class E, but for all practical purposes it is not used). The IP address Class is defined by the setting of the top (leftmost) 4 bits of the IP address (or bits 0–3 using the IETF's notation). The IP class provides the separation between the host and the network part of the IP address as shown in the following table:

Continued

Class	Example	Bits 28–31	Network Bits	Host Bits	Netmask
A	126.0.0.0	0xxx	8	24	255.0.0.0
B	172.16.0.0	10xx	16	16	255.255.0.0
C	192.22.22.0	110x	24	8	255.255.255.0
D	224.0.0.0	1110			

The following notes explain and expand on some of the points in the preceding table:

1. x = Don't care.
2. Class D addresses are used for multicasting protocols exclusively, for example, OSPF, IGMP, etc.
3. Classes A, B, C, and D are routed IPs (the IPv4 address ranges 10.x.x.x, 172.16.xx to 172.31.xx, and 192.168.x.x are reserved for private use and should only be routed inside a user's private network. They should not be routed across the Internet).
4. The term *netmask* refers to the standard mask for the address class. You will see later that different masks may be used with any IP class.
5. The terms *netmask* and *subnetmask* are subtly different, but they will be used here as if they were synonymous.

Classless Inter-Domain Routing

Classless Inter-Domain Routing essentially removes the idea of class from IPv4 addresses and allows administrations to allocate and route any valid subnet from any convenient base IP class; the idea being that if you want a group of 32 IP addresses, whether you take them from an IP Class C address or from an IP Class B address is *not* important. You simply want 32 IP addresses. The following table shows two 32-address subnets, one from a nominal Class B range, the other from a nominal Class C range—spot the difference!

Class	Network	Netmask
B	172.28.227.192	255.255.255.224
C	192.168.15.64	255.255.255.224

In short, the key factors in a CIDR world become the network (base) IP address and the netmask.

IP Prefix, or Slash Notation

It is common practice to combine IP addresses and their netmask into a single notation called the *IP prefix*, or more commonly *slash notation*. In the preceding example, the IP address 172.28.227.192 with a subnet mask of 255.255.255.224 would be written in the slash or IP prefix notation as 172.28.227.192/27. The IP address to the left of the slash (/) is the network (base) IP address, and the number (1 to 32) to the right of the slash is the number of contiguous bits in the netmask. The following table illustrates this notation:

Slash Form	Network IP	Netmask	No. of IPs
192.168.32.0/19	192.168.32.0	255.255.224.0	8192
172.28.127.64/27	172.28.127.64	255.255.255.224	32
172.28.127.0/24	172.28.127.0	255.255.255.0	256

In the preceding examples, you will see that multiple Class C addresses have been extracted from a Class C IP address and subclass C addresses have been subnetted from a Class B address just to illustrate the flexibility of CIDR.

The PTR Resource Record

The PTR RR is standardized in RFC 1035 and maps an IPv4 address to a particular host in the domain or zone as opposed to an A RR, which maps a name to an IPv4 address. The formal syntax is as follows:

```
name     ttl    class   rr  name
```

In the example file, the following PTR RRs are defined:

```
2         IN      PTR     ns1.example.com.
.....
4         IN      PTR     mail.example.com.
.....
16        IN      PTR     joe.example.com.
17        IN      PTR     www.example.com.
```

As you may recall from Chapter 2, the separators between fields can be either spaces or tabs. Table 3-1 maps the formal syntax to the first PTR RR used in the example zone file.

Table 3-1. *PTR RR Syntax*

Syntax	Example Usage	Description
`name`	`2`	While this looks like a number, it is in fact treated as a name. The name is unqualified, causing the `$ORIGIN` directive value to be substituted. You could have written this as `2.254.168.192.IN-ADDR.ARPA.` (using the FQDN format).
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`name`	`ns1.example.com.`	Defines that a query for 192.168.254.2 will return `ns1.example.com`. This name *must* be written in the FQDN notation (it must end with a dot). If the dot were omitted in error, then `$ORIGIN` substitution would create `ns1.example.com.254.168.192.IN-ADDR.ARPA.`.

Only one IPv4 address may be mapped to a host name using PTR RR. Where multiple A RRs or CNAME RRs can be used to define the same IPv4 address, only one name can appear in the `IN-ADDR.ARPA` zone file. In the zone fragment that follows, either `ns1` or `www` could appear in the `IN-ADDR.ARPA` zone:

```
ns1           IN      A       192.168.254.2
; this A RR has the same IPv4 address as ns1 above
www           IN      A       192.168.254.2
```

Same definition using a CNAME RR:

```
ns1           IN      A       192.168.254.2
www           IN      CNAME   ns1.example.com.
```

Reverse-map lookups are used extensively by e-mail software. If two names are defined for a host, using either A or CNAME RRs, that provides e-mail (SMTP) services, then the mail server name should always be defined in the `IN-ADDR.ARPA` zone file. Failure to do this will result in mail being rejected by any mail server that implements reverse lookup as part of an authentication (antispam) process. The zone fragment that follows shows the same host being defined using two A RRs:

```
mail          IN      A       192.168.254.4
; this A RR has the same IPv4 address as mail above
www           IN      A       192.168.254.4
```

Same definition using a CNAME RR:

```
mail          IN      A       192.168.254.4
www           IN      CNAME   mail.example.com.
```

The `IN-ADDR.ARPA` zone should define the mail host to enable reverse-lookup checks by, say, e-mail software.

```
; the IN-ADDR.ARPA zone file defines mail not www
4             IN      PTR     mail.example.com.
```

The reverse-map may reference a host defined in the forward-map zone file using either an A or a CNAME RR as shown here:

```
www           IN      A       192.168.254.4
ftp           IN      CNAME   mail.example.com.
```

The reverse map defines the host forward mapped with a CNAME:

```
4             IN      PTR     ftp.example.com.
```

IPv6 also uses the PTR RR for reverse mapping in the domain `IP6.ARPA` and is described in Chapter 5.

Reverse-Map Queries

Reverse-map queries use normal *recursive* or *iterative* queries, as described previously, under the special domain `IN-ADDR.ARPA`. The `.ARPA` (now renamed the Address and Routing Parameter

Area) domain is structured hierarchically with ICANN/IANA (`www.icann.org`) at the root as normal and is administered jointly by ICANN/IANA and the IETF/IAB (RFC 3172). Unlike *forward domains*, which use the gTLD or ccTLD servers as the next level of delegation, IPv4 addresses are delegated through the Regional Internet Registries (RIRs), which are shown in Table 3-2.

Table 3-2. *Regional Internet Registries*

RIR Name	Coverage	Web
APNIC	Asia Pacific	`www.apnic.net`
ARIN	North America, Southern Africa, parts of the Caribbean	`www.arin.net`
LACNIC	South America, parts of the Caribbean	`www.lacnic.net`
RIPE	Europe, Middle East, Northern Africa, parts of Asia	`www.ripe.net`
AFRINIC	Africa (This RIR is planned to be fully accredited by ICANN in late 2005/2006 and at that time will assume responsibilities for African registrations that are presently handled by ARIN and RIPE.)	`www.afrinic.net`

IPv4 addresses are allocated in *netblocks* by the RIRs to either a Local Internet Registry (LIR), typically an ISP, or to a National Internet Registry (NIR), which in turn will allocate to an LIR. Each Internet Registry level is delegated the responsibility for reverse mapping the addresses it has been assigned. The LIR *may* delegate the responsibility for reverse mapping to the end user if static IPv4 addresses are involved. However, the organization of reverse mapping is based on each dot-separated value in an IP address as shown in Figure 3-4. If the last part of the IPv4 address assigned to an end user is a subnet (less than 256 addresses), then a problem arises because any entity, a domain name or an address block, in the domain hierarchy can be delegated *once and only once*. In the case of a subnet, the same netblock would require to be delegated to each subnet user. To illustrate this point, assume the netblock 192.168.254.0 is to be allocated to four users, each of whom will have 64 addresses (a subnet of 64 addresses). These will be allocated as shown here in slash or IP prefix notation (see the sidebar "IPv4 addresses and CIDR"):

```
First User - 192.168.254.0/26 (same as netmask of 255.255.255.192)
Second User - 192.168.254.64/26 (same as netmask of 255.255.255.192)
Third User - 192.168.254.128/26 (same as netmask of 255.255.255.192)
Fourth User - 192.168.254.192/26 (same as netmask of 255.255.255.192)
```

When the netblock for this group is reverse mapped, the host part is omitted as defined previously, giving 192.168.254, and then reversed to the IN-ADDR.ARPA domain, giving the following:

```
254.168.192.IN-ADDDR.ARPA
```

Each of our four users would require delegation of this domain in order to provide the reverse mapping of their own assigned address range. This contravenes the single delegation principle defined previously.

In order to overcome this limitation, the construction of reverse maps for the delegation of subnets uses a very specialized *reverse-map name* construct that essentially creates an additional namespace and is described in Chapter 8. Reverse mapping of subnets is very uncommon, since not all organizations are aware of the special techniques involved.

A reverse-map inquiry using a recursive query is shown in Figure 3-5 for the IPv4 address 192.168.250.15, which is assumed to have been reverse-map delegated all the way to the end user.

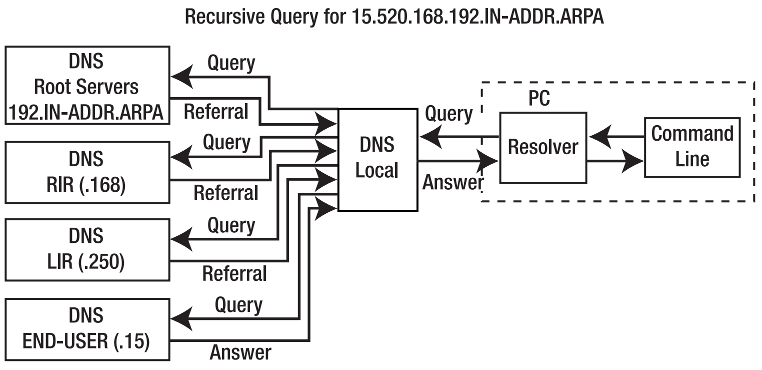


Figure 3-5. *Reverse-mapping query*

The examples appearing in this reverse-map section use, in the interest of promoting good netizen-ship, a private IPv4 address (from the set defined in RFC 1918), which in Figure 8-5 is 192.168.250.15. This is shown as interrogating the root-servers for the purpose of illustration only. These IPv4 addresses are private and are meaningless as far as the public network is concerned, yet recent studies suggest that up to 7% of all queries received at some root-servers comprise reverse-map queries for private IPv4 addresses, which are caused as a result of badly configured name servers. If the local configuration contains private IP addresses in any zone files, then a reverse-mapped zone file for the private IP range must be included in the name server configuration to prevent these meaningless queries being forwarded to the public root-servers. In Chapter 7, the section "Reverse-Map Zone Files" shows an example of such a configuration.

Zone Maintenance

In order to simplify the operation of multiple name servers, it is useful if a single source can update multiple servers. This process—zone maintenance—can involve transfer of zone files from one DNS server to another—between a master and slave DNS for the zone—using features of the DNS protocol.

The time between transferring zone file changes is a major determinant of the speed with which changes to the zone information are propagated throughout the Internet. The initial design of DNS allowed for changes to be propagated using full zone transfer (`AXFR`) operations, but the world of the Internet was simpler and more sedate in those days (1987). The desire to speed up the process of zone update propagation, while minimizing use of resources, has resulted in a number of changes to this aspect of DNS design and implementation from simple—but effective—tinkering such as *incremental zone transfer (IXFR)* and *NOTIFY* messages to the more complex concept of *dynamic update (DDNS)*.

■ **Warning** While zone transfers are generally essential for the efficient operation of DNS systems, they are also a major source of threat. A slave DNS can become *poisoned* if it accepts zone updates from a malicious source. Care should be taken during DNS configuration to ensure that, as a minimum, the slave DNS will only accept transfers from known and trusted sources. The example configurations provided in later chapters implement these minimum precautions.

Full Zone Transfer (AXFR)

The original DNS specifications (RFC 1034 and RFC 1035) envisaged that slave (or Secondary) name servers for the zone would poll the master name server for the zone. The time between polling is determined by the *refresh* value of the domain's SOA Resource Record, which was described in Chapter 2. In a typical zone file, this value will be 12 hours or more.

The DNS polling process is accomplished by the slave name server sending a query to the zone master requesting the SOA RR. If the SOA RR's serial number is greater than the current one maintained by the slave name server, a full zone transfer (AXFR) is requested by the slave DNS. This is the reason it is vital to be disciplined about updating the SOA serial number every time anything changes in any of the zone records. The following example demonstrates updating the serial number using the recommended date number format of yyyymmddss, where yyyy is a four-digit year number, mm is a two-digit month number, dd is a two-digit day number, and ss is a sequence number so that the zone can be updated more than once per day. Assume an SOA RR as shown here:

```
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080803 ; sn = serial number
                        3h         ; refresh time
                        15m        ; retry = update retry
                        3w         ; expiry
                        3h         ; min = minimum
                        )
```

Using the date format, this shows that this zone file was last updated four times (ss = 03) on August 8, 2003. If we assume that today's date is September 7, 2003, then the serial number should be set to the value shown here:

```
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003090700 ; sn = serial number
                        3h         ; refresh time
                        15m        ; retry = update retry
                        3w         ; expiry
                        3h         ; min = minimum
                        )
```

The sequence number has also been reset to `00` to ensure we have plenty of space for fixing errors! If the month and date of the preceding example were to be swapped in error, then the serial number would be

```
2003070900 ; sn = serial number
```

This number is not greater than the previous number, so the slave would not request a zone transfer and the updates would not be propagated. The fix in this case is simple, since the error is back in time. The following example shows the serial number being incorrectly placed forward in time:

```
2005090700 ; sn = serial number
```

To restore this serial number to the correct date is much more complex, and you would only want to do it once in your life. The procedure is documented in Chapter 8. Remember that the date format is a widely used and recommended *convention*; BIND does not validate the number for correct ranges, that is, the following is accepted quite happily by BIND:

```
2003144500 ; sn = serial number
```

which is the 45th day of the 14th month of 2003! In this case, a zone transfer will take place because the number is greater than our initial value.

Zone transfer (`AXFR`) operations use TCP on port 53.

■ **Warning** Not updating the serial number field of the SOA RR when any change is made to the zone file is one of the most common causes of head scratching, screaming, and other more seriously aberrant behavior when dealing with DNS systems. Always update the SOA RR serial number when you make any changes to a zone file.

Incremental Zone Transfer (`IXFR`)

Transferring very large zone files can take a long time and waste bandwidth and other resources. It is especially wasteful if only a single record has been changed! RFC 1995 introduced incremental zone transfers (`IXFR`), which as the name suggests allows the slave name server and master name server to transfer only those records that have changed.

The process works as for `AXFR`. The slave name server sends a query for the domain's SOA RR to the zone master every refresh interval. If the serial number of the SOA RR is greater than the one currently stored by the slave, the name server requests a zone transfer and indicates whether or not it is capable of accepting an incremental zone transfer (`IXFR`). If both master and slave name servers support the feature, an incremental zone transfer (`IXFR`) takes place; otherwise a full zone transfer (`AXFR`) takes place. Incremental zone transfers (`IXFR`) use TCP on port 53.

The default mode for BIND when acting as a slave name server is to request IXFR unless it has been configured not to by use of the `request-ixfr` statement in the `server` or `options` clause of the `named.conf` file (see Chapter 12 for details).

The default mode for BIND when acting as a master name server is to use IXFR only when the zone is dynamic. The use of `IXFR` is controlled through the `provide-ixfr` statement in the `server` or `options` clause of the `named.conf` file (see Chapter 12 for details).

Incremental zone transfers (`IXFR`) affect only the volume of data that is transferred; they have no impact on the time it takes to propagate zone file changes.

Notify (`NOTIFY`)

RFC 1912 recommends an interval of 2 to 12 hours or higher on the refresh interval for the SOA RR. This means that changes to the zone master may not be visible to the zone slave for up to 12 hours or whatever this value is set to. In the fast moving world of the Internet, this may be unacceptable.

RFC 1996 introduced a scheme whereby the zone master will send a `NOTIFY` message to the zone slave name server whenever the zone is loaded or updated. This message indicates that a change *may* have occurred in the domain records. The slave on receipt of the `NOTIFY` message will request the SOA RR from the zone master, and if the serial number is greater than the one currently stored, will attempt a zone transfer using either a full zone transfer (`AXFR`) or an incremental transfer (`IXFR`).

BIND's default behavior is to send `NOTIFY` messages to slave name servers that are defined in the NS Resource Records for the zone. `NOTIFY` behavior in BIND is controlled by `notify`, `also-notify`, and `notify-source` statements in the `zone` or `options` clauses of the `named.conf` file (see Chapter 12 for details).

`NOTIFY` can considerably reduce the time to propagate zone changes to servers.

Dynamic Update

The classic method of updating zone RRs is to manually edit the zone file and then stop and start the name server to read the zone files and propagate the changes. When the volume of changes reaches a certain level, this can become operationally unacceptable—especially considering that in organizations that handle large numbers of zone files, such as service providers, BIND can take a long time to restart as it initializes very large numbers of zone files.

Many larger users of DNS seek a method to rapidly change the zone records while the name server continues to respond to user queries. There are two architectural approaches to solving this problem:

1. Allow runtime updating of the zone RRs from an external source or application.

2. Directly feed the zone RRs from a database, which can be dynamically updated.

RFC 2136 takes the first approach and defines a process, called *Dynamic DNS (DDNS)*, whereby zone records can be updated from one or more external sources. The key limitation in this specification is that a new domain or zone cannot be added or deleted dynamically. All records within an existing zone can be added, changed, or deleted—with the exception that the SOA RR cannot be added or deleted since this would essentially add or remove the zone.

As part of RFC 2136, the term *Primary Master* was introduced to describe the name server defined in the SOA Resource Record for the zone. When dynamically updating zone RRs, it is essential to update only one server even though there may be multiple master servers for the zone. In order to solve this problem, a *boss* server must be selected. The boss server, the primary master, has no special characteristics other than it is defined as the name server in the

SOA RR and may appear in an `allow-update` statement of BIND's `named.conf` configuration file to control the dynamic update process (see Chapter 12 for details).

Dynamic DNS (DDNS) is normally described in conjunction with Secure DNS features—specifically TSIG (RFC 2845) and TKEY (RFC 2930). DDNS, however, does not require or rely on TSIG/TKEY features.

The reason the two features are tightly coupled is that by enabling Dynamic DNS, zone files may be opened up to the possibility of corruption or poisoning by malicious sources. Simple IP address protection can be configured into BIND (using BIND's `allow-update` statement described in Chapter 12), but this provides limited protection. System architecture can further remove risk by positioning both the target name server and all the hosts that are allowed to update it behind secure perimeters. The real power, however, of DDNS is that remote and distributed users are able to semi-autonomously update and control their domain configurations. Under these circumstances, serious users of Dynamic DNS will always use TSIG/TKEY procedures, described in Chapter 10, to authenticate incoming requests.

BIND's default DDNS behavior is to *deny from all hosts*. Control of dynamic update is provided by the BIND `named.conf` statements `allow-update` (usable with and without TSIG/TKEY) and `update-policy` (only usable with TSIG/TKEY) in the `zone` or `options` clauses. The statements and clauses mentioned are described in Chapter 12.

There are a number of Open Source tools that will initiate DDNS updates; these include `nsupdate`, which is one of the utilities distributed with BIND and whose use is described in Chapter 9.

Alternative Dynamic DNS Approaches

As noted earlier, the major limitation in DDNS (RFC 2136) is that new domains cannot be created dynamically. Alternative approaches to this problem do exist.

BIND-DLZ (`bind-dlz.sourceforge.net`) takes a much more radical approach and uses a big patch to BIND 9 that replaces all zone files with a single zone file which simply describes a database. BIND-DLZ supports the major Open Source databases including MySQL, PostgreSQL, BDB, and OpenLDAP. All incoming DNS queries are first directed to the database access routines so that new, modified, or deleted zone data is immediately reflected in the name server's responses. As with all things in life, there is a trade-off. Depending on the selected database, performance can drop precipitously; however, the latest Berkeley DB (BDB) drivers are showing excellent results—with benchmarks approaching raw BIND performance.

PowerDNS (`www.powerdns.com`) is an authoritative-only name server that takes a similar approach with its own (non-BIND) code base by referring all queries to the database back-end and thereby allowing new domains to be added dynamically.

■ **Caution** The use of real-time changes to DNS records without the proper safeguards can result in trivial errors being immediately propagated throughout the Internet with catastrophic consequences. Since DNS caches will typically hold such records for 12 or more hours (determined by either the `$TTL` for the zone file or the TTL value for the specific Resource Record), such errors can take a long time to correct.

Security Overview

DNS operation, the simple act of running a DNS, opens up potential security threats. This is true of any publicly accessible service, for example, a web site or FTP site. Somehow it is easier to forget that DNS is a publicly accessible service.

This security overview steps back from the detail of DNS security configuration minutiae—Chapters 10 and 11 cover DNS security configuration—to try and provide a clear and dispassionate overview of the topic. There is nothing more annoying, on finding oneself in deep trouble halfway up a mountain, that one really didn't need to climb this particular mountain.

The critical point in defining security policies and procedures is to understand what needs to be secured—or rather what threat levels need to be secured against and what threats are acceptable. The answers to these two points will be very different if the DNS is running as a root-server versus running a modest in-house DNS serving a couple of low-volume web sites.

The term *DNSSEC* is thrown around as a blanket term to cover DNS security. This is not quite correct. There are at least three forms of DNS security, two of which are (relatively) painless and a full-blooded DNSSEC implementation that is (relatively) painful.

Security Threats

In order to be able to assess both the potential threats and the possible countermeasures, it is first and foremost necessary to understand the normal data flow in a DNS system. Figure 3-6 shows this flow.

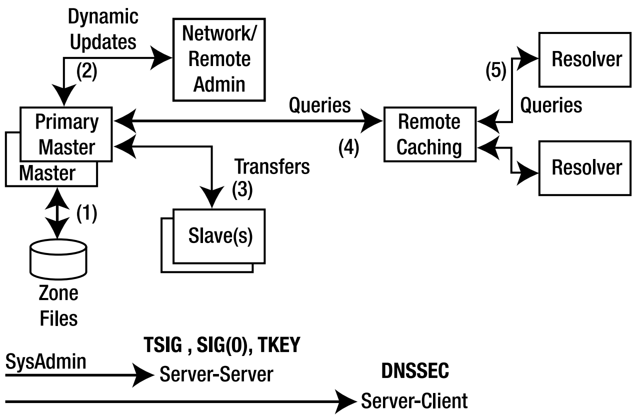


Figure 3-6. *DNS data flow*

Every part of this data flow—each numbered line in Figure 3-6—is a *potential* source of threat. Table 3-3 defines the potential outcomes of compromise at each point and the possible solutions.

Table 3-3. *DNS Security Threats*

Number	Area	Threat	Classification	Solutions
1	Zone files	File corruption (malicious or accidental)	Local	System Administration
2	Dynamic updates	Unauthorized Updates, IP address spoofing (impersonating update source)	Server-Server	Network architecture, TSIG, SIG(0), or disable
3	Zone transfers	IP address spoofing (impersonating update source)	Server-Server	Network architecture, TSIG, TKEY, or disable
4	Remote queries	Cache poisoning by IP spoofing, data interception, or a subverted master or slave	Server-Client	DNSSEC
5	Resolver queries	Data interception, poisoned cache, subverted master or slave, local IP spoofing	Remote Client–Client	DNSSEC, SSL/TLS

The first phase of any security review is to audit what threats are applicable and how seriously they are rated in the particular organizational circumstances. As an example, if dynamic updates are not supported—BIND's default mode—there is no dynamic update threat!

It can be easier to disable a process than to secure it. While alternate processes may be required, these may be far simpler to secure. As an example, for organizational reasons it may be easier to manually update zone records on each name server than to secure or limit zone transfers. In this case, simply disabling all zone transfers is the securest solution. This is sometimes referred to as *security by obscurity*.

Finally, a note of caution: *the further you go from the zone master, the more complicated the solution and implementation*. Unless there is a very good reason for not doing so, it is recommended that you always start from the zone master and work outward. It would be a tad frustrating to have completed a successful implementation of a complex DNSSEC solution only to discover that the installation's zone files were world readable and writable.

Security Classification

The security classification is simply a means to allow selection of the appropriate remedies and strategies for avoiding the implied risk. All the methods described next are discussed in detail in Chapters 10 and 11. The numbering used in the following list relates to Figure 3-6.

- *Local threats (1)*: Local threats are usually the simplest to prevent, typically requiring good system administration polices. Zone files and any DNS configuration files—`named.conf` contains lots of interesting data—should be secure, that is, have limited read and write access and be securely backed up. Stealth (or Split) name servers can be used to minimize public access (described in Chapter 7), and BIND can be run as a *chroot jail* (described in Chapter 10).

- *Server-Server (2)*: If an organization runs slave name servers, it will do zone transfers. It is possible to run multiple master name servers rather than master-slave servers. Alternative methods are required to distribute zone files, but these methods may be easier to secure than zone transfers depending on the organization's requirements and procedures. If zone transfers are required, BIND offers multiple configuration parameters that can be used to minimize the inherent risks in the process (described in Chapter 12), and TSIG and TKEY offer secure methods for transmission of zone files (described in Chapter 10).

- *Server-Server (3)*: The BIND default is to *deny* dynamic zone updates. If an organization requires this feature, then BIND provides a number of configuration parameters to minimize risk (described in Chapter 12). Network architecture design—all systems involved within a trusted perimeter—can further reduce the exposure. TSIG and SIG(0) can be used to secure the transactions (described in Chapter 10).

- *Server-Client (4)*: Remote caches can become poisoned—their contents can become corrupted to point at competitor's web sites—by IP spoofing, data interception, and other hacks. While modest web sites probably have little to fear from this form of attack, if the site is high profile, high volume, open to competitive threat, or a high revenue earner, then the costs and complexity of implementing a full-scale DNSSEC solution may be worthwhile. DNSSEC implementations have immediate applicability to high-volume users (ISPs, service providers), within specialized communities (root-servers and TLD servers) or within controlled groups (intranets, extranets). DNSSEC implementation is described in Chapter 11.

- *Client-Client (5)*: The current versions of the DNSSEC protocol allow for a concept called the *security-aware resolver* whereby the security chain can be propagated to the client resolver. Additionally, BIND allows SSL/TLS to be used to secure the transmission path between the client resolver's host and the local name server.

Summary

This chapter described the various operations and services provided by the DNS protocol. These operations include queries, recursive and interative (nonrecursive); zone transfers; and dynamic update. I described the process known as reverse mapping, in which a normal query is used to obtain the name of a host given its IP address, and illustrated it with some examples. The chapter concluded with a brief overview of the security implications that necessarily arise from running any DNS service.

Chapter 4 describes a number of name server (DNS) types while recognizing that the majority of name servers are required to provide multiple functions.

CHAPTER 4

DNS Types

DNS servers play a wide variety of roles—a single name server may be a *master* for some *zones*, a *slave* for others, and provide *caching* or *forwarding* services for still others. Indeed, much of BIND's power comes from allowing fine-grained control over operational functionality.

The role of the name server is controlled by its configuration file, which in the case of BIND is called `named.conf`. The combination of global parameters in the `named.conf` file (defined in an `options` clause) and the zones being serviced (defined in one or more `zone` clauses) determine the complete functionality of the name server. Depending on the requirements, such configurations can become very complex. In order to provide an approachable starting point to what can become a task of daunting complexity, this chapter breaks down configuration of the name server into a number of basic *types* such as a master server type and an authoritative-only server type. It describes their characteristics and properties, in isolation, to create a series of building blocks from which progressively more complex configurations can be constructed. In some cases, the basic types may themselves be sufficient to create the required name server such as a caching-only server type or a forwarding server type; in other cases, the required name server may consist of, for example, many master server types, many slave server types, and a caching server type. Indeed, in later chapters of this book, we will meet many examples that combine a number of these basic types to create unique solutions.

In order to most effectively introduce the characteristics of the each basic name server type, where appropriate, some BIND configuration file (`named.conf`) fragments are used. The term *clause* is used to describe a group of related *statements* that can appear in the `named.conf` file. This terminology is applied rigorously throughout this book in the interests of consistency and ease of understanding rather than the myriad terms used in other documentation on this subject. The full format and layout of the `named.conf` file is described in Chapter 12, but the following identifies some important clauses and statements used in this file and which appear in the upcoming fragments:

- The `options` clause. The `options` clause groups together statements that control the global behavior of the name server. In some cases, the global statements may be overridden in specific clauses such as the `zone` clause.

- The `zone` clause groups statements that relate to specific zones within the configuration—the zone clause for `example.com` will define all the characteristics or properties of the zone.

- The `type` statement is used within a `zone` clause and defines how the name server will act for the specific zone (for example, it may act as a master or as a slave for the zone).

- The `recursion` statement controls whether recursive queries are supported or not. Caching is an artifact of recursion, and therefore this statement effectively controls the provision of caching services in the name server. This statement may appear either in a global `options` clause or a `view` clause. By default, BIND will support recursive queries and hence provides caching.
- The `file` statement is used to define the physical location of the zone file and appears in a zone clause.

Master (Primary) Name Servers

A *master* DNS configuration, also known as a *zone master* configuration, contains one or more zone files for which this DNS is authoritative and which it reads from a local file system. The term *master* is related to the location of the zone file rather than any other operational characteristics. A master may be requested to transfer zone files—using zone transfer operations (described in Chapter 3)—to one or more *slave* servers whenever the zone file changes.

■ **Note** The term *master* was introduced in BIND 8.x releases and replaced the slightly confusing term *Primary*.

Zone master status for a zone is defined in BIND by including `type master;` in the zone clause of the `named.conf` file as shown by the following fragment:

```
// example.com fragment from named.conf
// defines this server as a zone master for example.com
zone "example.com" in{
      type master;
      file "master.example.com";
};
```

In the preceding fragment, `zone "example.com"` defines the zone to which the following statements apply, `type master` defines this DNS to be the zone master for `example.com`, and `file "master.example.com"` defines the name of the zone file containing the Resource Records (RRs) for `example.com`.

Figure 4-1 illustrates a zone master DNS.

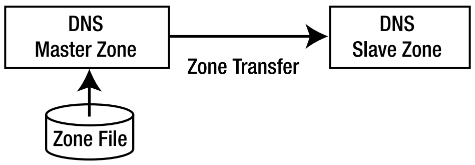


Figure 4-1. *Master and slave servers*

■**Note** The terms *Primary* and *Secondary* name servers are widely used in two contexts. In the context of zone transfer, Primary is used to describe what this book calls the zone master and Secondary describes the zone slave. Unfortunately, the terms *Primary* and *Secondary* are also frequently used when describing the order of name servers, for example, when registering a domain name, and in many PCs when defining the order of DNS used in the network properties on Windows systems especially. BIND 8 introduced the terms *master* and *slave* to try and reduce the confusion caused by the conflicting use of Primary and Secondary. This book uses the terms *master* and *slave* throughout when describing name servers used in zone transfer operations; purely in this context the terms Primary means master and Secondary means slave. Just to further confuse things, the term *Primary Master* has crept into the jargon. This term has a special meaning only in the context of Dynamic DNS (DDNS) updates and is defined to be the name server that appears in the SOA Resource Record (discussed in Chapter 2).

A zone master obtains the zone data from a local zone file as opposed to a zone slave, which obtains its zone data via a zone transfer operation from the zone master. This seemingly trivial point means that it is possible to have any number of zone masters for any zone if that makes operational sense. Zone file changes need to be synchronized between zone masters by a manual or automated process. This may be easier to manage than securing the zone transfer operations inherent in a master-slave configuration.

A master name server can indicate (using `NOTIFY` messages) zone changes to slave servers. This ensures zone changes are rapidly propagated to the slaves rather than simply waiting for the slave to poll for changes at each SOA RR *refresh* interval. The BIND default is to automatically `NOTIFY` all the name servers defined in NS records for the zone.

`NOTIFY` messages may be disabled by use of the configuration statement `notify no` in BIND's `named.conf` file in the `zone` clause for the domain.

When a DNS server that is a master for one or more zones receives a query for a zone for which it is not a master or a slave, then it will act as configured. In BIND, this behavior is defined in the `named.conf` file:

1. If caching behavior is permitted and recursive queries are allowed (described in Chapter 3), the server will completely answer the request or return an error.

2. If caching behavior is permitted and iterative (nonrecursive) queries only are allowed, the server can respond with the complete answer if it is already in the cache because of another request, a *referral*, or return an error.

3. If caching behavior is not permitted (an *authoritative-only* DNS server), the server will return a referral or an error.

■ **Tip** Example configuration files for a master name server are provided in Chapter 7.

Slave (Secondary) Name Servers

The critical nature of DNS—no Internet services can work without it—requires that there be at least two name servers to support each domain or zone; larger or more active domains may rely on many more. For instance, examination of the NS Resource Records using the `dig` tool (see Chapter 9) shows a typical range from 4 to 9 name servers for a number of high-profile zones. It is possible to run multiple master name servers, but any changed zone files must be copied to all masters. Apart from the obvious problem of synchronization when multiple masters are used, each master must be reloaded to use the new zone files, thus taking the name server out of service for a short period of time. With larger sites being hit hundreds of times per second, even modest out-of-service times can *lose* thousands of DNS transactions—effectively making the site unreachable or slowing down access. To resolve this problem, the DNS specifications provide a feature—zone transfer—whereby one name server, the slave (or Secondary), can be updated from a zone master (or Primary) while continuing to provide responses to queries for the zone.

A slave name server obtains its zone information from a zone master, but it will respond as authoritative for those zones for which it is defined to be a slave and for which it has valid zone records, that is, the zone records have not expired. The act of transferring the zone may be viewed as having delegated the authority for the zone to the slave for the time period defined in the *expiry* value of the SOA record (described in Chapter 2) and thus enables the slave to respond authoritatively to queries.

■ **Note** There is no visible difference to other name servers between the response from a zone master and the response from a zone slave.

Slave status is defined in BIND by including `type slave` in the zone clause of the `named.conf` file as shown by the following fragment:

```
// example.com fragment from named.conf
// defines this server as a zone slave
zone "example.com" in{
     type slave;
     file "slave.example.com";
     masters {192.168.23.17;};
};
```

In the preceding fragment, `zone "example.com"` defines the zone for which the following statements apply, and `type slave;` indicates that this name server will act as a slave for `example.com`. The statement `file "slave.example.com";` is optional and allows the zone data to be saved to the specified file. If the name server is reloaded, it can read the zone data from this file rather than forcing a new zone transfer from the master, as would be the case if no file statement were present. The `file` statement can save considerable time and resources. The statement `masters {192.168.23.17};` defines the IP address of the name server that holds the master zone file for this zone. One or more IP addresses may be present. There can be *more than one master DNS for any zone.*

A slave server attempts to update the zone records when the *refresh* parameter of the SOA RR (described in Chapter 2) is reached. If a slave has still not reached the master DNS when the *expiry* time of the SOA RR for the zone has been reached, it will stop responding to queries for the zone. The slave will not use time-expired data.

Slave (Secondary) DNS Behavior

As previously mentioned, slave servers will respond as authoritative to queries for the domain as long as they hold valid zone records. This feature provides the user with a lot of flexibility when registering name servers for a given domain. When registering such name servers, the only requirement is that the servers listed will respond as authoritative to queries for the domain or zone. It is not necessary to define the zone master as one of these name servers; two or more slave servers will satisfy the requirement. This flexibility allows the zone master to be hidden from public access if required. To illustrate why such a strategy may be useful, consider the following scenario: if a slave zone file becomes corrupted through a malicious attack, it can be quickly restored from the master by a zone transfer. If the master zone file were to become similarly corrupted, the zone files may have to be restored from backup media, which could take some time. One way to prevent such a problem is simply to avoid it by not making the master publicly visible. It is visible to the slave only using the `masters` parameter of BIND's `named.conf` but would not appear in any NS RR for the zone. Every name server, master, or slave that the user wishes to make *visible* must be defined using an NS RR in the zone.

Figure 4-2 illustrates a typical master and single slave configuration.

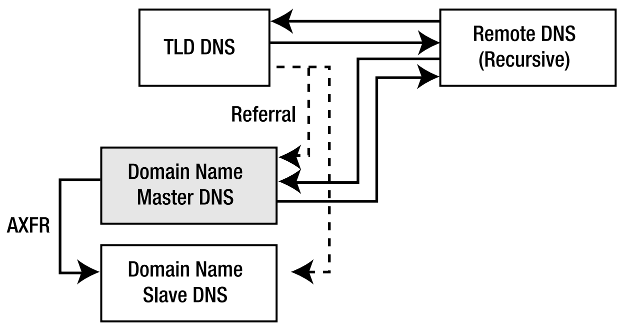


Figure 4-2. *Typical master and slave configuration*

Figure 4-3 illustrates a slave server when used with a hidden master.

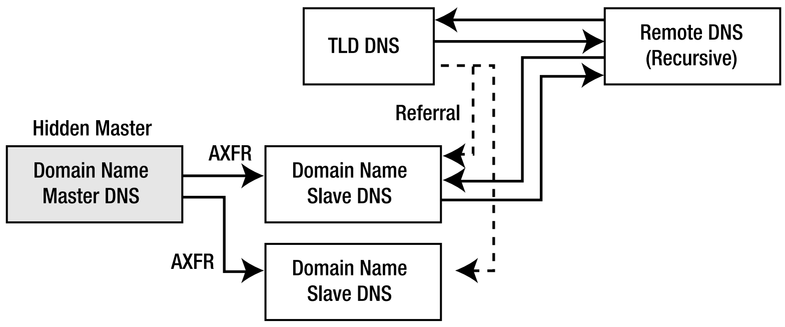


Figure 4-3. *Hidden master-slave configuration*

Slave vs. Cache

A zone slave obtains *all the zone data* for which it is acting as a slave via zone transfer operations, and this process should not be confused with a cache. The slave server uses the *refresh* value from the SOA Resource Record to time-out its zone data and will then retransfer all the zone data. On the other hand, a cache contains individual Resource Records obtained in response to a specific query originating from a *resolver* or another name server acting on behalf of a resolver and discards each RR when its TTL is reached. In addition, a slave server always responds authoritatively to requests for information about its zone. A cache will only respond authoritatively with zone data the first time it obtains the data (directly from the master or slave) and thereafter, when reading from its cache, the data is not marked as authoritative.

Change Propagation Using NOTIFY

The slave will periodically *poll* the zone master for changes at a time interval defined by the refresh parameter of the zone's SOA RR. In this scenario, the refresh parameter, which may be up to 12 hours, controls the time taken to propagate zone changes. If `NOTIFY` behavior is enabled in the zone master—BIND's default—then every time the zone is loaded or reloaded a `NOTIFY` message is sent to all the slave servers defined in the NS RRs of the zone file. On receipt of a `NOTIFY` message, the slave will request a copy of the zone's SOA RR. If the serial number of the current zone data is lower than the serial number of the newly requested SOA RR, then the slave initiates a zone transfer to completely update its zone data. There can be zero, one, or more slave name servers for any given zone.

The `NOTIFY` message—and its subsequent zone transfer operation—presents a potential security threat. To minimize this threat, BIND's default behavior is to only accept `NOTIFY` messages from the zone master (name servers listed in the `masters` statement). Other acceptable `NOTIFY` sources can be defined using the `allow-notify` statement in the `named.conf` file.

■ **Tip** Example configuration files for a slave DNS are provided in Chapter 7.

Caching Name Servers

A caching server obtains specific information in the form of one or more Resource Records about a domain from a zone's authoritative name server (master or slave) in order to answer a host query, and subsequently saves, or caches, the data locally. On a subsequent request for the *same* data, the caching server will respond with its locally stored data, from the cache. This process will continue until the Time to Live (TTL) value of the RR expires, at which time the RR will be discarded from the cache. The next request for this RR will result in the name server again contacting an authoritative name server for the zone. Caches considerably increase DNS performance for local PCs or hosts and can also significantly reduce network loads by obtaining a single copy of frequently accessed data and making it available many times with no additional overhead. Consider the example file in which the mail server was defined using the following RR:

```
 3w       IN      MX  10  mail.example.com.
```

The effect of caching in this case is that every request for the mail server for `example.com` for the next *three weeks* will be satisfied from the cache and will require no further—possibly slow—network access. If a caching name server is reloaded or restarted, then caches are always erased and the process begins again. It is worth emphasizing at this point that the only way RR data is removed from a cache is by either its TTL expiring or the name server being reloaded. This means that changes to the preceding MX record will take up to three weeks to propagate throughout the Internet and thus only stable RRs, such as mail servers, would typically have such very long TTL values.

If the caching server obtains its data *directly* from an authoritative DNS, then it too will respond as authoritative. Otherwise, if the data is supplied from its cache, the response is *nonauthoritative*.

By default, BIND will cache Resource Records. This behavior is defined using the `recursion` parameter—the default is `recursion yes;`—in BIND's `named.conf` file. This may seem a little strange at first, but caching is essentially an artifact of *recursive query* behavior.

■Note There are many configuration examples that show caching behavior being controlled using a `type hint` statement in the zone declaration section of BIND's `named.conf` configuration file. These configurations confuse two *distinct* but *related* functions. If a server provides caching services, then it *must* support recursive queries, and recursive queries need access to the root-servers. Root-server access is provided using the `type hint` statement in a special root-server zone. The root-server zone definition is described in Chapter 7.

A caching server will have a `named.conf` file that includes the following fragment:

```
// options clause fragment of named.conf
// recursion yes is the default and may be omitted
options {
       recursion yes;
};
// zone clause
....
// the DOT indicates the root domain = all domains
zone "." IN {
   type hint;
   file "root.servers";
};
```

The `options` clause indicates the following statements apply to all zones in the configuration unless explicitly overridden with another statement; `recursion yes;` turns on caching behavior, which is the BIND default and could be omitted. The `zone ".";` clause defines the normally silent root domain and is used to access any zone that is not defined in the remainder of the configuration; `type hint;` simply indicates the zone references the root domain and is only ever used in conjunction with a `zone ".";` clause. The statement `file "root.servers";` locates the zone file that contains the Address (A) RRs of the root-servers.

■Tip The `root.servers` zone file, which may be called `named.ca` or `named.root`, is normally supplied with BIND distributions. Chapter 7 illustrates an example `root.servers` zone file.

Caching Implications

To cache or not to cache is a crucial question in the world of DNS, since it incurs substantial performance overheads and runs the risk of cache poisoning, or corruption through malicious attacks. This downside must be offset against the significant performance gains that are obtained when using a caching DNS. The most common uses of DNS caching configurations are as follows:

- As a name server acting as master or slave for one or more zones (domains) and as a caching server for all other queries. A general-purpose name server.
- As a caching-only local server—typically used to support standard PC-based resolvers (stub resolvers), which as you may recall from Chapter 1 require recursive query support that is only provided by a caching name server. This is sometimes called a *proxy DNS server*, but this book associates the term *proxy* with a forwarding name server (described in the next section).

However, if a general purpose name server is being hit thousands of times per second in support of a high-volume site then performance becomes a major factor, and in this case caching would typically be disabled. Furthermore, there are many DNS administrators who, due to the cache-related dangers described previously, will never allow caching behavior on a name server that has any master or slave zones. BIND provides only limited controls to disable caching behavior, principally by including the statement `recursion no;` in the `named.conf` file, but many caching overheads remain. There are now a number of Open Source and commercial DNS servers that stress high performance and that do not provide caching services—they are said to be authoritative-only servers.

Tip Example configuration files for a caching DNS are provided in Chapter 7.

Forwarding (Proxy) Name Servers

A forwarding (a.k.a. proxy, client, or remote) DNS server is one that forwards all queries to another DNS and *caches* the results. On its face, this looks a pretty pointless exercise. However, a forwarding DNS server can pay off in a couple ways when access to an external network is slow, expensive, or heavily congested:

1. The name server to which queries are forwarded will provide recursive query support resulting in a single query-answer DNS transaction. If the local name server were a caching-only server and did not forward queries, multiple transactions would occur, thus increasing network load and time delays.
2. The local or on-site forwarding DNS server will cache results and thereby provide both faster responses for frequently accessed information and eliminate unnecessary external traffic.

Forwarding name servers can also be used tactically to ease the burden of local administration. Each PC may be defined to use a local forwarding name server, which in turn is defined to pass all queries to an external server. If the external DNS server changes, for example, when the user changes ISP, a single configuration change to the local name server's `named.conf` file is required rather than having to change all the local PC configurations. The same result can be accomplished using DHCP, but that is not always convenient.

Forwarding may also be used as part of a Stealth (or Split) server configuration, which is described in the next section, for perimeter defense.

Figure 4-4 illustrates the use of forwarding DNS.

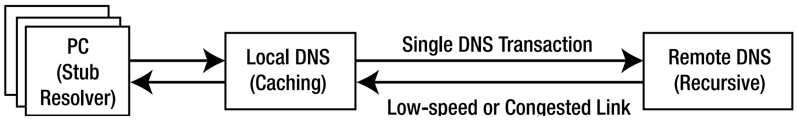


Figure 4-4. *Forwarding DNS server*

BIND allows configuration of forwarding using the `forward` and `forwarders` parameters either at a *global* level (in an `options` clause) or on a per-zone basis (in a `zone` clause) of the `named.conf` file. Both configurations are shown in the following examples.

The following `named.conf` fragment causes global forwarding of all queries received by the name server.

```
// options clause fragment of named.conf
// forwarders can have multiple choices
options {
               forwarders {10.0.0.1; 10.0.0.2;};
               forward only;
};
// zone clauses
....
```

By defining the `forwarders` statement in the `options` clause, it applies to the whole configuration unless overridden in a subsequent `zone` clause. The `forwarders` and `forward` statements are always used in conjunction with each other. The `forwarders {10.0.0.1; 10.0.0.2;};` statement contains two IP addresses that are used in rotation; one, two, or more IP addresses may be used. The `forward only;` statement forces all queries to be forwarded. The next fragment provides forwarding for the specific zone only:

```
// zone clause fragment of named.conf
zone "example.com" IN {
               type forward;
               forwarders {10.0.0.1; 10.0.0.2;};
               forward only;
};
```

Where dial-up links are used with forwarding name servers, BIND's general-purpose nature and strict standards adherence may not make it an optimal solution. A number of alternative DNS solutions specifically target support for such links. BIND defines two parameters, `dialup` and `heartbeat-interval` (see Chapter 12), whose objective is to minimize network connection time.

Tip Example configuration files for a forwarding DNS are provided in Chapter 7.

Stealth (DMZ or Split) Name Server

A Stealth server is defined as a name server that does not appear in any publicly visible NS RRs for the domain. Stealth servers are used in configurations that are sometimes called *demilitarized zone* (DMZ) or *Split servers*, and can be defined as having the following characteristics:

- The organization needs to expose DNS servers to provide access to its public services such as web sites, mail, FTP sites, and so forth.
- The organization does not want the world to see any of its internal hosts either by interrogation (query or zone transfer) or in the event the DNS service or external servers are compromised.

A Stealth or Split server architecture is illustrated in Figure 4-5.

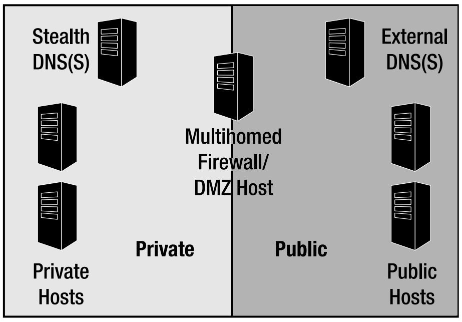


Figure 4-5. *Stealth or Split server architecture*

The external or public servers are configured to provide authoritative-only responses and no caching services—recursive queries are not accepted. In this case, caching is both wasteful in terms of performance and a possible source of pollution or corruption, both of which can lead to system compromise. The zone file used by these public servers is a public subset of the zone file data and will contain only those systems or services that the user needs to make visible, for example, an SOA RR (required), NS RRs for the public (not Stealth) name servers, MX RRs for mail servers, and A RRs for, say, `www.example.com` and `ftp.example.com` for the public web and FTP services.

Zone transfers can be allowed between the public name servers as required, but they should not transfer, or accept transfers from, the Stealth server. This clear separation between the private and public side of the network is necessary, because if the public name server is compromised, then simple inspection of the `named.conf` file or zone files must not yield information that describes any part of the hidden network. BIND's `named.conf` directives such `master`, `allow-notify`, `allow-transfer`, and others, if present, will provide information that allows an attacker to penetrate the veil of privacy.

Stealth Servers and the View Clause

BIND provides a `view` clause that may be used to provide similar functionality using a single server, but this does not address the problem of the name server host system being compromised, and by simple inspection of the `named.conf` file, additional data about the organization can be discovered. Careful consideration of the likelihood of file system compromises on publicly visible servers and the design of the `view` statements must be undertaken before using `view` in a Stealth DNS configuration. BIND's `view` statement can, however, be used to augment the functionality of the public and private parts of a Stealth configuration, and this is described further in Chapter 7.

Stealth Server Configuration

A simple public master zone file containing only those hosts and services that are required to support public or external access for the organization is shown here. This zone file does not contain any hosts or services used in the internal network.

```
; public zone master file
; provides minimal public visibility of external services
example.com. IN      SOA   ns1.example.com. hostmaster.example.com. (
                              2003080800 ; se = serial number
                              12h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h      ; min = minimum
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
              IN      MX  10  mail.example.com.
              IN      MX  20  mail.example.net.
ns1           IN      A       192.168.254.1
ns1           IN      A       192.168.254.2
mail          IN      A       192.168.254.3
www           IN      A       192.168.254.4
ftp           IN      A       192.168.254.5
```

The internal name server—the Stealth server—zone file will make visible internal and external hosts, provide recursive queries and all manner of other services. For instance, this name server would use a private zone master file that could look like this:

```
; private zone master file used by Stealth server(s)
; provides public and private services and hosts
example.com. IN      SOA   ns3.example.com. hostmaster.example.com. (
                              2003080800 ; se = serial number
                              12h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h         ; min = minimum
                              )
              IN      NS      ns3.example.com.
              IN      NS      ns4.example.com.
              IN      NS  10  mail.example.com.
              IN      MX  20  mail.anotherdomain.com.
; public hosts
mail          IN      A       192.168.254.3
www           IN      A       192.168.254.4
ftp           IN      A       192.168.254.5
; private hosts
joe           IN      A       192.168.254.6
bill          IN      A       192.168.254.7
fred          IN      A       192.168.254.8
ns3           IN      A       192.168.254.9
ns4           IN      A       192.168.254.10
....
accounting    IN      A       192.168.254.28
payroll       IN      A       192.168.254.29
```

Clearly, at some point the internal users must cross the perimeter to access external services, including DNS services. There are two possible solutions to this problem:

1. The classic firewall solution in which the internal systems, including the DNS server, are permitted, on a transaction-by-transaction basis, to send and receive data externally.

2. BIND 9's `view` clause may be used to provide support for caching and recursive query services for the internal network on the public DNS server. The `view` clause can be used to provide these services while continuing to deny them to external users and without exposing the structure of the internal network. The relevant configuration files and a further explanation of this style of operation is provided in Chapter 7. This solution does not eliminate the need for a firewall for non-DNS traffic.

Figure 4-6 illustrates the traffic flows for Firewall and BIND `view`-based solutions.

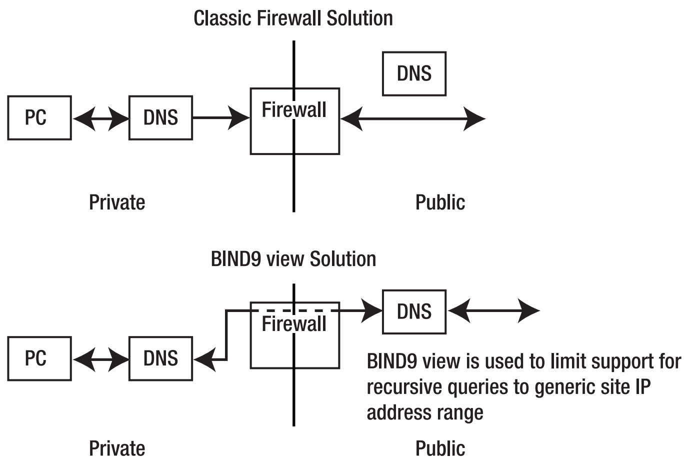

Classic Firewall Solution
PC
DNS
Firewall
DNS
Private
Public
BIND9 view Solution
PC
DNS
Firewall
DNS
BIND9 view is used to limit support for recursive queries to generic site IP address range
Private
Public

Figure 4-6. *Firewall and DNS* `view` *perimeter solutions*

■**Note** There is a third possibility, which is to define the internal network as using exclusively private IP addresses and to use a NAT gateway as the means of securing the internal network and limiting access to the external world. The world of the Internet has many conceptual disagreements. The argument between those who view NAT as a perfect solution that kept the Internet alive when it was threatening to run out of IPv4 addresses and those who see NAT as inherently evil is one of the more contentious. This book will stay gracefully agnostic on the topic of NAT other than to point out that, increasingly, services that are delivered to desktops, such as VoIP, do require network visibility of end-user systems.

Example configuration files for a Stealth DNS configuration are provided in Chapter 7.

Authoritative-only Name Server

The term *authoritative-only name server* is normally used to describe two related properties of a DNS server:

1. The name server will deliver authoritative answers—it is a zone master or slave for one or more domains.
2. The name server does not cache.

Authoritative-only servers are typically used in two distinct configurations:

1. As public or external servers in a Split (a.k.a. DMZ or Stealth) DNS configuration used to provide perimeter security
2. As high-performance name servers, for example, root-servers, TLD servers, or name servers for high-volume sites

Authoritative-only servers typically have high performance requirements. For many years, BIND was the only DNS software used by the root-servers and many of the TLD servers, which also have serious performance requirements, since it provides a high-quality, high-function, and stable platform. However, general-purpose DNS software, such as BIND, while providing an excellent solution, is not optimized for use in high-performance authoritative-only servers. There are now a number of Open Source and commercial alternatives that specialize in high-performance authoritative-only DNS solutions as discussed in Chapter 1. The latest releases of BIND have placed renewed emphasis on its performance characteristics.

It is not possible to directly control caching behavior in BIND, but the `recursion` statement effectively inhibits caching as shown in the following BIND `named.conf` configuration file fragment:

```
// options clause fragment of named.conf
// recursion no = effectively inhibits caching
options {
        recursion no;
};
// zone clauses
....
```

BIND provides three more parameters to control caching: `max-cache-size` (limits the size of the cache on the file system) and `max-cache-ttl` (defines the maximum time RRs may live in the cache and overrides all RR TTL values), neither of which will have much effect on performance in the particular case just discussed, and `allow-recursion`, which provides a list of hosts that are permitted to use recursion—all others are not.

■ **Note** Example configuration files for an authoritative-only DNS are provided in Chapter 7.

Summary

This chapter described a number of commonly used DNS configurations and characteristics. Name servers rarely perform a single function. They are almost by their nature schizophrenic. Indeed, the strength of general-purpose DNS software, especially BIND, is that it can be used to precisely configure multifaceted solutions. You also learned about the behavior of zone masters, zone slaves, caching servers, forwarding servers, and authoritative-only servers. You saw the configuration of Stealth (or Split) servers used in perimeter defense employing both classic configurations and BIND's new (as of BIND 9) `view` clause.

In Chapter 5, we look at the world of IPv6 and its implications for DNS.

CHAPTER 5

DNS and IPv6

While IPv6 provides many improvements in network management, one of the major driving forces behind its design was to greatly increase address space. An IPv4 address uses 32 bits, whereas an IPv6 address uses 128 bits. IPv6 is theoretically capable of providing many millions of IP addresses for every human on the planet!

The original IETF specifications for IPv6 date from 1995 but the Classless Inter-Domain Routing and Network Address Translation (see the sidebar "IPv4 Addresses and CIDR" in Chapter 3) initiatives of the mid-90s effectively postponed the urgent need for additional address space. Until very recently IPv6 usage was largely confined to experimental networks such as the IETF's 6bone (`www.6bone.net`) and large scale deployment was limited to academic institutions.

Yet the year 2002 signaled the start of several significant developments that have given new impetus to IPv6 and have significantly increased its deployment:

- *Hardware availability*: The 3rd Generation Partnership Project (`www.3gpp.org`), consisting of mobile wireless equipment suppliers and operators, has mandated that IPv6 be used to communicate from all 3G (next generation) handsets. The first IPv6 mobile public call was publicly demonstrated in late 2004.

- *DNS support*: IPv6 addresses are already published by 5 of the 13 root-servers and a recent presentation to RIPE (the European-based Regional Internet Registry and operator of `k.root-server.net`) indicates that 8 of the 13 root-servers are IPv6 ready.

- *Address allocation*: IPv6 address block allocations may be obtained via all the currently operational Regional Internet Registries (RIRs), which comprise ARIN (`www.arin.net` covering North America and Southern Africa), RIPE (`www.ripe.net` covering Europe, North Africa, and the Middle East), APNIC (`www.apnic.net` covering Asia Pacific), and LACNIC (`www.lacnic.net` covering South America). The new AFRINIC registry (`www.afrinic.net` covering Africa) is not planned to be operational until late 2005 or early 2006.

- *Software availability*: IPv6 stacks and dual (IPv6/IPv4) stacks are now available for Windows (Server 2003 and XP), Linux, UNIX, and the BSDs (FreeBSD, NetBSD, and OpenBSD).

- *Mainstream technology*: The IETF has recognized the production status of IPv6 and has documented the winding-up of the 6bone experimental and test bed network with final transfer of its special IPv6 addresses range to IANA by June 2006.

Probably the most significant push for IPv6, however, is coming from the changing nature of Internet-based applications. Classic Internet applications such as those providing web access, e-mail, and FTP use the traditional client-server model and can handle mapping private addresses to a limited range of IPv4 public IP addresses using Network Address Translation strategies with some help from Application-Level Gateways (ALGs). However, the new generation of Internet applications such as Instant Messaging (IM) and Voice over IP (VoIP), among others, use a peer-to-peer model and increasingly require *always-on* capabilities (permanent connection to the Internet) and need end-user address transparency—any given user's equipment address must be publicly visible and *fixed*, or *static*, over a reasonable period of time. The current IPv4 address scheme is incapable of providing all peer-to-peer users with end-user address transparency—there simply are not enough addresses. Figure 5-1 illustrates the difference between the client-server model with NAT and peer-to-peer applications.

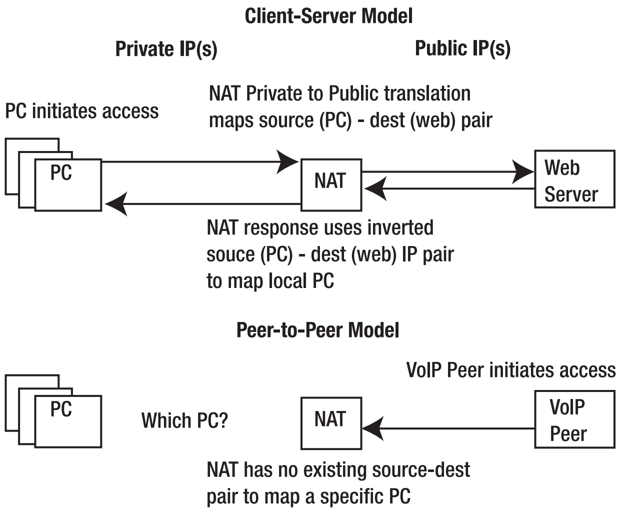

Client-Server Model
Private IP(s)
Public IP(s)
PC initiates access
NAT Private to Public translation maps source (PC) - dest (web) pair
PC
NAT
Web Server
NAT response uses inverted souce (PC) - dest (web) IP pair to map local PC
Peer-to-Peer Model
VoIP Peer initiates access
PC
Which PC?
NAT
VoIP Peer
NAT has no existing source-dest pair to map a specific PC

Figure 5-1. *Address transparency*

The huge investment in IPv4 together with the size of the current installed base means IPv4 will not disappear overnight. IPv6 and IPv4 will have to coexist for some considerable period of time, and serious attention has been paid to transition and interworking schemes in the various IPv6 RFCs. There are significant implications for the DNS systems in both IPv6 and mixed IPv6/IPv4 environments.

Now that you have a better general understanding for why IPv6 will soon become a particularly important part of the network environment, let's take a moment to introduce the IPv6 protocol before delving into the implications it will have on DNS implementations.

IPv6

IPv6 is a big and complex protocol providing many new features for the efficient operation and management of dynamic networks, including the following:

- Significantly expanded address space using 128 bits (IPv4 uses a 32-bit address).
- Scoped addresses—IPv6 addresses can be limited to the local LAN, a private network, or be globally unique.
- Security is defined as part of the protocol.
- Autoconfiguration—stateless or stateful (DHCP enhancements).
- Mobile IPv6 (MIPv6) is significantly more powerful than its IPv4 counterpart.

This section is not designed to fully describe the IPv6 protocol but rather to familiarize you with the addressing features of IPv6 as they affect the DNS system.

Each IPv6 network interface, for instance, a LAN card on a PC or a mobile phone, may have more than one IPv6 address—that is, IPv6 is naturally *multihomed*. An IPv6 address has a *scope*: it can be restricted to a single LAN, a private network, or be globally unique. Table 5-1 defines the types of IPv6 addresses that are supported and contrasts them with the closest IPv4 functional equivalent.

Table 5-1. *Comparison of IPv6 and IPv4 Functionality*

IPv6 Name	Scope/Description	IPv4 Equivalent	Notes
Link-Local	Local LAN only. Automatically assigned based on MAC. Cannot be routed outside local LAN.	No real equivalent. Assigned IPv4 over ARP'd MAC.	Automatically configured by most stacks from the LAN Media Access Control (MAC) address of the network interface. Scoped address concept new to IPv6.
Site-Local	Optional. Local site only. Cannot be routed over Internet. Assigned by user.	Private network address with multihomed interface is closest equivalent.	Work is ongoing by the IETF to clarify the use of the Site-Local address and support has currently been withdrawn.
Global Unicast	Globally unique. Fully routable. Assigned by IANA/delegated *Aggregators*.	Global IP address.	IPv6 and IPv4 similar but IPv6 can have *other* scoped addresses.
Multicast	One-to-many. Hierarchy of multicasting.	Similar to IPv4 Class D.	Significantly more powerful than IPv4 version. No broadcast in IPv6, replaced by Multicast.
Anycast	One-to-nearest. Uses Global Unicast addresses. Routers only. Discovery uses.	Unique protocols in IPv4, for example, IGMP.	Some Anycast addresses reserved for special functions.
Loopback	Local interface scope.	Same as IPv4 127.0.0.1.	Same function.

IPv6 Address Notation

An IPv6 address consists of 128 bits, whereas an IPv4 address consists of 32 bits. An IPv6 address is written as a series of eight hexadecimal strings separated by colons. Each string represents 16 bits. IPv6 examples:

```
# all the following refer to the same address
2001:0DB8:0234:C1AB:0000:00A0:AABC:003F
# leading zeros can always be omitted
2001:DB8:234:C1AB:0:A0:AABC:3F
# not case sensitive - any mixture allowed
2001:db8:234:C1ab:0:A0:aabc:3F
```

Complete zero entries can be omitted entirely but only once in an address. Examples:

```
# full ipv6 address
2001:db8:234:C1AB:0000:00A0:0000:003F
# address with single 0 dropped
2001:db8:234:C1ab:0:A0::003F
# but the following is invalid
2001:db8:0234:C1ab::A0::003F
```

Multiple zero entries can be omitted entirely but only once in an address. Examples:

```
# omitting multiple zeros in address
2001:db8:0:0:0:0:0:3F
# can be written as
2001:db8:3F
# lots of zeros (loopback address)
0:0:0:0:0:0:0:1
# can be written as
::1
# all zeros (unspecified, a.k.a unassigned IP)
0:0:0:0:0:0:0:0
# can be written as
::
# but this address
2001:db8:0:1:0:0:0:3F
# cannot be reduced to this
2001:db8::1::3F  # INVALID
# instead it can only be reduced to
2001:db8::1:0:0:0:3F
# or
2001:db8:0:1::3F
```

IPv6 Address Types

The type of IP address is defined by a variable number of the top bits of its address—the bits are collectively known as the *binary prefix* (BP). Only as many bits as required are used to identify the address type as shown in Table 5-2, which is defined in RFC 3513.

Table 5-2. *IPv6 Address Types*

Use	Binary Prefix	Description/Notes
Unspecified	00...0	IPv6 address = 0:0:0:0:0:0:0:0 (or ::). Used before an address allocated by DHCP (equivalent of IPv4 0.0.0.0).
Loopback	00...1	IPv6 address = 0:0:0:0:0:0:0:1 (or ::1). Local PC Loopback address (equivalent of IPv4 127.0.0.1).
Multicast	1111 1111	IPv6 Multicast replaces both multicast and broadcast in IPv4.
Link-Local Unicast	1111 1110 10	Local LAN scope. Lower bits assigned by user.
Reserved Unicast	1111 1110 11	Was the Site-Local address range. This address range is currently reserved by IANA while the IETF considers the status of the Site-Local features of IPv6.
Global Unicast	All other values	Assigned by IANA and Aggregators. IANA currently assigns addresses to Aggregators from the range starting with the binary prefix 001. The binary prefix 0011 1111 1111 1110 (hex 3FFE) is currently used by the 6bone for experimental purposes, but this use will be discontinued and the address range will be returned to the IANA pool of available addresses in June 2006. The Global Unicast address format is defined in the section "Global Unicast IPv6 Address Allocation."

■**Note** The term *Aggregator* is used to describe various Internet Registries that are responsible for the allocation of IPv6 addresses and for IPv6 reverse-map delegation.

Prefix or Slash Notation

IPv6 addresses use the IP prefix or slash notation in a similar manner to IPv4 to indicate the number of contiguous bits in the netmask. Examples:

```
# a single IP address - 128 bit netmask for loopback
# equivalent of IPv4 255.255.255.255 or /32
::1/128
# Link-Local address mask
FE80::/10
# typical end user site prefix routing mask
2001:db8:222::/48
# typical end user subnet routing mask
2001:db8:222:1::/64
```

Global Unicast IPv6 Address Allocation

The IPv6 Global Unicast address is hierarchical and is divided into what was historically called a *site prefix,* but which has now been renamed a *global routing prefix*, a subnet ID and interface ID address parts. Various agencies or address registrars—called *Aggregators* in IPv6 terminology—assign the global routing prefix as defined in Figure 5-2.

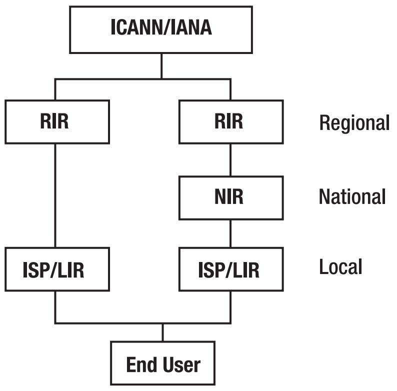


Figure 5-2. *IPv6 hierarchical address allocation*

RFC 3177 defines the current IETF/IAB policy for end-user IPv6 address allocation. End users may be allocated one of three IPv6 address ranges:

1. *Normal end user*: An end user is normally allocated a full 80 bits of address space (see Table 5-4 in the next section for full format). The allocated address space may be assigned in any way required by the end user. This *normal* end-user address range allocation is greater than the whole of the current IPv4 Internet. This allocation is written as /48 in the slash or IP prefix notation.
2. *Single subnet*: Where it is known that only a single subnet (site) will be used, the end user may be allocated only 64 bits of address space. This allocation is written as /64 in the slash or IP prefix notation.
3. *Single device*: Where it is known that only one device will be used, a single IPv6 address may be allocated. This allocation is written as /128 in the slash or IP prefix notation.

Internet Registries may, however, allocate much larger IPv6 address blocks to groups of users such as governments.

IPv6 Global Unicast Address Format

The generic format of an IPv6 Global Unicast address is shown in Table 5-3.

Table 5-3. *Generic IPv6 Global Unicast Address Format*

Name	Size	Description/Notes
Global routing prefix	48 bits	The format of this field may vary depending on the assigned use (see Table 5-4 in the next section).
Subnet ID	16 bits	Used for subnet routing.
Interface ID	64 bits	Equivalent of the host address in IPv4.

End-User IPv6 Address Format

The address block 2001::/16 is assigned by IANA to Aggregators—the Regional Internet Registries which in turn may assign them to National Internet Registries and/or Local Internet Registries (ISPs) for subsequent allocation to end users as shown in Figure 5-2. This address block has the format defined in Table 5-4.

Table 5-4. *IPv6 Global Unicast End-User Address Format*

Name	Size	Description/Notes
Global Routing Prefix of 48 Bits—Assigned by IANA/Aggregator		
Reserved	3 bits	001—Global Unicast address block allocated by IANA (all other values are reserved).
TLA ID	13 bits	0 0000 0000 0001 (address block 2001::/16)—assigned by IANA for use by the Regional Internet Registries.
Sub-TLA	13 bits	Assigned by IANA to the RIRs. The RIRs assign blocks from this range to the Next-Level Aggregator (NLA).
NLA	19 bits	Assigned by RIR to the Next-Level Aggregator, which may be, as you can see in Figure 5-2 earlier, either a National Internet Registry or a Local Internet Registry. The NLA assigns blocks from its allocated address range to end users.
80 Bits—Assigned by the User		
Subnet ID	16 bits	Used for subnet routing.
Interface ID	64 bits	Equivalent of the host address in IPv4.

A request by an end user for an IPv6 address block should be directed to the Local LIR, which is normally, but not always, an ISP. Lists of LIRs may be obtained from each RIR as defined in Table 5-5.

Table 5-5. *Regional Internet Registries*

RIR Name	Coverage	Web
APNIC	Asia Pacific	`www.apnic.net`
ARIN	North America, Southern Africa, parts of the Caribbean	`www.arin.net`
LACNIC	South America parts of the Caribbean	`www.apnic.net`
RIPE	Europe, Middle East, Northern Africa, parts of Asia	`www.ripe.net`
AFRINIC	Africa (This RIR is planned to be fully accredited by ICANN in late 2005/2006 and at that time will assume responsibilities for African registrations that are presently handled by ARIN and RIPE.)	`www.afrinic.net`

■Note Every Aggregator or Internet Registry that assigns IPv6 address blocks also has the responsibility to provide reverse-map delegation. IPv6 is designed to provide complete reverse-map coverage down to every host or end device.

Status of IPv6 DNS Support

The DNS has included support for IPv6 from its earliest definition in 1995. During that time the IPv6 standard has evolved, indeed is still evolving, and the DNS specifications have evolved in parallel. The essence of any such evolution is the need to experiment, to solve new problems, and to provide new functionality. Not all experiments are successful—were this not true, then it is doubtful if any real progress could be made. The following sections describe the current DNS features and functionality that support IPv6 in its current stage of evolution.

The AAAA vs. A6 Resource Record

DNS IPv6 support is the subject of some confusion and requires some historical perspective. As previously stated, DNS has supported IPv6 since 1995 (RFC 1886). This RFC specifies that IPv6 forward mapping will use an AAAA (QUAD A) RR. Reverse mapping (described in Chapter 3) was defined to use an extended version of the PTR RR under the domain name `IP6.INT`.

RFC 2673 (1999) and RFC 2874 (2000) introduced new DNS capabilities to more efficiently support IPv6 services—specifically network renumbering—using new *bit labels* and two new RRs. The A6 RR was designed to be used for forward mapping, and the DNAME RR was designed to enhance support for reverse mapping. These new RRs were defined to deprecate the use of the AAAA RRs in IPv6 and mixed IPv6/IPv4 networks.

However, after considerable debate and amid much controversy, the IETF issued RFC 3363 (2002), which changed RFC 2673 and RFC 2874 to EXPERIMENTAL status—effectively removing the A6 and DNAME RRs from operational use. The current IETF recommendation is contained in RFC 3596 (largely a reissue of RFC 1886) and has DRAFT STANDARD status as summarized here:

- Forward mapping of IPv6 addresses will use the AAAA (Quad A) RR (same as RFC 1886).
- Reverse mapping will use the IP6.ARPA domain (change from RFC 1886).
- Reverse mapping will use the PTR RR (same as RFC 1886).

The AAAA and PTR RRs are used exclusively in all the examples since they constitute the current IETF recommendation. For the sake of completeness, the A6 and DNAME RRs are described in Chapter 13.

Mixed IPv6 and IPv4 Network Support

A DNS system must support both IPv6 and IPv4 networks during what may be a long transitional period. BIND 9 provides two features for supporting this capability:

- Forward mapping of IPv4 addresses (using A RRs as described in Chapter 2) and IPv6 addresses (using AAAA RRs and described in this chapter) may appear in the same *zone file*.
- BIND 9 supports both IPv4 and IPv6 native *protocol* DNS queries. Previous versions, while supporting IPv6 AAAA and PTR RRs, only supported IPv4 protocol queries. Thus, it is possible to query a BIND 9 DNS using IPv4 and obtain an IPv6 AAAA RR and/or an A RR and conversely to query a DNS using IPv6 and obtain an A RR and/or an AAAA RR.

The reverse-mapping files, however, cannot be mixed since IPv4 reverse maps under the domain `IN-ADDR.ARPA` while IPv6 uses `IP6.ARPA`.

IPv6 Resource Records

As previously mentioned, the current IETF recommendation defined in RFC 3596 mandates the use of the AAAA RR for forward mapping of IPv6 address records and PTR RRs for the reverse mapping of IPv6 addresses. To illustrate the use of the two RRs, the standard IPv4 zone file introduced in Chapter 1 is enhanced to provide support for both IPv4 and IPv6. It is assumed that all the defined systems will provide dual stack support; that is, each host is capable of responding to both IPv4 and IPv6 protocol requests. This is one of a number of techniques that may be used during IPv4 to IPv6 transition, and is embraced by all mainstream platforms, including Microsoft Windows, Linux, UNIX, and BSD platforms.

■Note Transition from IPv4 to IPv6 may be handled using a variety of other techniques such as NAT-PT (RFC 2766), which lie outside the scope of this book.

The standard IPv4 version of the example zone file is shown here:

```
; IPv4 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh
                        15m        ; retry = update retry
                        3w         ; expiry
                        2h         ; min = minimum
                        )
; name server RRs for the domain
              IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
              IN      NS      ns2.example.net.
; mail server RRs for the zone (domain)
       3w     IN      MX  10  mail.example.com.
```

```
; the second  mail server is
; external to the zone (domain)
              IN      MX  20  mail.example.net.
; domain hosts includes NS and MX records defined earlier
; plus any others required
ns1           IN      A       192.168.254.2
mail          IN      A       192.168.254.4
joe           IN      A       192.168.254.6
www           IN      A       192.168.254.7
; aliases ftp (ftp server) to an external location
ftp           IN      CNAME   ftp.example.net
```

The IPv6 user configuration used throughout the following sections is as follows:

1. Example, Inc. has been allocated a normal end-user IPv6 address range of 2001:db8::/48. This allocation provides addresses in the range 2001:db8:0:0:0:0:0:0 (or 2001:db8::) to 2001:db8:0:FFFF:FFFF:FFFF:FFFF:FFFF, which may be assigned and used at the discretion of Example, Inc. The global routing prefix is 2001:db8:0::, which is assumed to be allocated by IANA and the various Aggregators (the Regional and Local Internet Registries).

2. Example, Inc. will have two IPv6 subnets: the first contains the hosts `ns1.example.com` and `mail.example.com` and the second contains `joe.example.com` and `www.example.com`.

3. IPv6 addresses in the first subnet will lie in the range 2001:db8:0:1::/64 and in the second 2001:db8:0:2::/64.

4. Each host supports dual stack operation—it has both an IPv4 address and an IPv6 address.

5. IPv6 reverse-map delegation is automatically provided by the Aggregators that allocated the IPv6 address range, and Example, Inc. is required to provide reverse-mapping support for its locally assigned addresses using the `IP6.ARPA` domain.

When this configuration is upgraded to support IPv6 and IPv4, the modified zone file becomes the following:

```
; transitional IPv6/IPv4 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh
                        15m        ; retry = update retry
                        3w         ; expiry
                        2h         ; min = minimum
                        )
; name server RRs for the domain
              IN      NS      ns1.example.com.
```

```
; the second name server is
; external to this zone (domain).
              IN      NS      ns2.example.net.
; mail server RRs for the zone (domain)
       3w     IN      MX  10  mail.example.com.
; the second mail server is
; external to the zone (domain)
              IN      MX  20  mail.example.net.
; domain hosts includes NS and MX records defined above
; plus any others required
; the following hosts are in IPv6 subnet 1
ns1           IN      A        192.168.254.2
ns1           IN      AAAA 2001:db8:0:1::1
mail          IN      A        192.168.254.4
mail          IN     AAAA 2001:db8:0:1::2
; these hosts are defined to be in the IPv6 subnet 2
joe           IN      A        192.168.254.6
joe           IN      AAAA 2001:db8:0:2::1
www           IN      A        192.168.254.7
www           IN      AAAA 2001:db8:0:2::2
; aliases ftp (ftp server) to an external location
ftp           IN      CNAME   ftp.example.net
```

For the purposes of clarity, only the preceding zone file has repeated the name of each host in the AAAA RR. Using blank label substitution, these names could have been omitted as shown in the following fragment for the `www.example.com` RR:

```
www           IN      A        192.168.254.7
              IN      AAAA 2001:db8:0:2::2 ; = www
```

Forward mapping of the IPv6 address is accomplished using the AAAA (Quad A) RR, which is described in the next section.

■ **Note** The address range 2001:db8::/32 is nonroutable and specifically reserved by RFC 3849 for use in documentation.

The AAAA Resource Record

The AAAA (Quad A) RR is the current IETF recommendation for defining forward mapping of IPv6 addresses and is defined in RFC 3596. It is equivalent to the A RR used for IPv4 forward mapping. The formal syntax is as follows:

```
name     ttl    class   rr  ipv6
```

In the enhanced IPv6/IPv4 example file shown previously, the following AAAA RR is defined:

```
ns1            IN      AAAA 2001:db8:0:1::1
```

The separators between fields can be either spaces or tabs. Traditionally, in zone files tabs are used to make a more attractive layout and to clearly indicate which fields are missing.

Table 5-6 maps the formal syntax to the AAAA RR used in the example zone file.

Table 5-6. *AAAA RR Syntax*

Syntax	Example Usage	Description
`name`	`ns1`	The name is unqualified, causing the `$ORIGIN` directive value to be substituted. You could have written this as `ns1.example.com.` (using the FQDN format), which may be more understandable.
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`ipv6`	`2001:db8:0:1::1`	This is a Global Unicast address and takes the format defined by Table 5-4 earlier. The address shown uses the zero elimination feature of IPv6 and could have been written as `2001:db8:0:1:0:0:0:1`. The value `2001:db8:0` is the global routing prefix assigned by IANA and the Aggregators (the Internet Registries). The first `1` is the subnet and the value `::1` is the locally (end-user) assigned interface ID.

Recall from earlier that IPv6 provides scoped addresses. Our hosts will have Link-Local IPv6 addresses as well as the Global Unicast addresses used previously. When software on the host wishes to access a local host, it does not use a name server to look up the address; instead it uses a local Multicast group to find all such local hosts. Only Global Unicast addresses need appear in the zone file.

Reverse IPv6 Mapping

Unlike IPv4, where reverse mapping is frequently not delegated to the end user, IPv6 mandates delegated reverse mapping. The end user is therefore responsible for creation of reverse-mapping zone files using the `IP6.ARPA` domain for the address range they have been assigned. The `IP6.ARPA` domain is similar to the `IN-ADDR.ARPA` domain used for reverse mapping of IPv4 addresses and described in Chapter 3. From the zone files defined previously, you can see that the Global Unicast address range allocated to the end user Example, Inc. is as follows:

```
2001:db8:0::/48
```

Example, Inc. is responsible for reverse mapping the 80-bit addresses in this range (see Table 5-4). IPv6 reverse mapping uses the normal principle of reversing the address and placing the result under the domain `IP6.ARPA`. The key difference from the `IN-ADDR.ARPA` domain (described in Chapter 3) is that a *nibble* is the unit of delegation. A nibble is one of those glorious terms that have survived to enter the jargon. Each byte (or octet) is comprised of 8 bits; a nibble is part of a byte and consists of 4 bits. So a nibble is a small byte! In the context of reverse mapping, each character in the IPv6 address string constitutes a nibble. To illustrate how this works, we must write each character—with no zero elimination—of the Example, Inc. assigned addresses range:

```
2001:0db8:0000::/48
```

Each character is reversed and separated with the normal dot notation to give a reverse-map domain name as shown here:

```
0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA
```

Finally, we construct a zone file to contain the definitions as shown here:

```
; reverse IPV6 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN 0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh = refresh
                        15m        ; retry = update retry
                        3w         ; expiry = expiry
                        2h         ; min = minimum
                        )
; name server RRs for the domain
          IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to a host name
; hosts in subnet ID 1
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0         IN      PTR     ns1.example.com.
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0         IN      PTR     mail.example.com.
; hosts in subnet ID 2
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0         IN      PTR     joe.example.com.
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0         IN      PTR     www.example.com.
```

The individual PTR address labels can become brutally long. The constructed domain name, however, does not have to reflect the address segmentation between the global routing prefix and the end-user part of the address as shown in the preceding example. If we assume that Example, Inc. will only ever have a maximum of 65,535 hosts in each subnet (uses 16 bits of the interface ID), then we can move some more of the end-user address into the zone domain name, which is written once, to reduce the address part in each PTR line, which may be written many hundreds of times. Thus the IPv6 address splits in Table 5-7 achieve the same result.

Table 5-7. *IPv6 Alternative Reverse Mappings*

Zone Name	PTR Part	Note
0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.	1.0.0.0.0.0.0.0.0.0.0.0.➥ 0.0.0.0.0.1.0.0.0.0	Uses a split based on the global routing prefix.
0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.➥ 0.0.8.b.d.0.1.0.0.2.IP6.ARPA.	1.0.0.0	Uses a split based on *user convenience* to reduce the size of each PTR RR. Because the subnet ID appears in the zone name, a second zone file is required in this scenario to describe subnet 2.

The two zone files to implement this alternate structure are shown next. Here's the zone file for subnet ID 1:

```
; reverse IPV6 zone file for example.com subnet ID 1
$TTL 2d    ; default TTL for zone
$ORIGIN .0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh = refresh
                        15m        ; retry = update retry
                        3w         ; expiry = expiry
                        2h         ; min = minimum
                        )
; name server RRs for the domain
          IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to the hostnames in subnet ID 1
1.0.0.0         IN      PTR     ns1.example.com.
2.0.0.0         IN      PTR     mail.example.com.
```

And the zone file for subnet ID 2:

```
; reverse IPV6 zone file for example.com subnet ID 2
$TTL 2d    ; default TTL for zone
$ORIGIN .0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh = refresh
                        15m        ; retry = update retry
                        3w         ; expiry = expiry
                        2h         ; min = minimum
                        )
; name server's RRs for the domain
          IN      NS      ns1.example.com.
```

```
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to the hostnames in subnet ID 2
1.0.0.0.            IN      PTR     joe.example.com.
2.0.0.0             IN      PTR     www.example.com.
```

The PTR RR that is used in IPv6 is described in the next section.

■Note An earlier version of the IPv6 specification used the reverse-map domain `IP6.INT`. This domain has been superseded in RFC 3596 with `IP6.ARPA` to make it consistent with IPv4's `IN-ADDR.ARPA` domain.

The IPv6 PTR Resource Record

The PTR RR is standardized in RFC 1035 and maps an IPv6 address to a particular interface ID (host in IPv4 terminology) in the domain or zone as opposed to an AAAA RR, which maps a name to an IPv6 address. The formal syntax is as follows:

```
name     ttl    class   rr  name
```

In the first reverse-map example zone file, the first PTR RR is defined as follows:

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0         IN      PTR     ns1.example.com.
```

The separators between fields can be either spaces or tabs.

Table 5-8 maps the formal syntax to the first PTR RR used in the example zone file.

Table 5-8. *PTR RR Syntax*

Syntax	Example Usage	Description
`name`	`1.0.0.0.0.0.0.0.0.0.0.0.➥ 0.0.0.0.0.0.0.1.0.0.0.0`	This is the subnet ID and interface ID parts of the IPv6 address written in reverse nibble format. While this looks like a number, it is in fact treated as a name. The name is unqualified causing the `$ORIGIN` directive value to be substituted. You could have written this as `1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.➥ 8.b.d.0.1.0.0.2.IP6.ARPA.` (using the dot-terminated FQDN format).
`ttl`		There is no `ttl` value defined for the RR, so the zone default of `2d` from the `$TTL` directive will be used.
`class`	`IN`	`IN` defines the class to be Internet.
`name`	`ns1.example.com.`	Defines that a query for `2001:db8::1` will return `ns1.example.com`.

Summary

This chapter described the use and implementation of IPv6 as it relates to the DNS. The chapter started by describing the long history of IPv6 starting around 1995 and suggested that a number of factors are currently causing a rapid increase in the spread and deployment of IPv6. A brief tutorial on IPv6 address notation was provided to allow the reader to become familiar with its format and usage.

The status of DNS support was clarified due to some confusion created by the withdrawal of support for bit labels and the A6 and DNAME RR by the IETF in RFC 3363. The current IETF IPv6 DNS recommendation specifies that forward mapping of IPv6 addresses will use the AAAA (Quad A) RR, and reverse mapping will use the PTR RR under the domain IPV6.ARPA.

In Chapter 6, we move from theory to practice by looking at the installation of BIND 9 on Linux, BSD (FreeBSD), and Windows platforms.

PART 2

Get Something Running

CHAPTER 6

Installing BIND

BIND (Berkeley Internet Name Domain) is an Open Source implementation of the Domain Name System protocols originally developed by the University of California, Berkeley. BIND is generally viewed as the reference implementation of the Internet's DNS, the standard against which all other implementations are compared. Due to its stability and high quality, BIND is the most widely deployed DNS software, servicing most of the root and gTLD name servers as well as innumerable ISPs, commercial organizations, and, because of its incomparable cost advantage (freely available under the BSD license), even very small sites and individual PCs. In 1994, the responsibility for BIND development moved from Berkeley to Internet Systems Consortium, Inc. (ISC), a US-based nonprofit company that is also, among other things, the operator of the `k.root-servers.net` (one of the 13 root-servers). ISC funding comes from a wide variety of corporate sponsors for whom the availability of high-quality DNS software is vital to their commercial interests. BIND, reflecting its widespread deployment, is available on a bewildering number of OS platforms.

This chapter describes the installation of BIND 9.3.0—the stable version at the time the test were run—on a variety of widely deployed OS platforms using their packaged formats where available:

- *Fedora Core 2 (FC2)*: A representative Linux platform. Fedora is a community-supported Open Source OS project sponsored by Redhat, Inc. (See `fedora.redhat.com` for more information.) Standard Fedora Core Development RPMs (Redhat Package Managers) were used.
- *FreeBSD*: A representative OS from the BSD/Unix family that comprises FreeBSD, NetBSD, OpenBSD, and DragonflyBSD. (See `www.freebsd.org` for more information.)
- *Windows 2000 Server*: ISC supplies a binary packaged version of BIND for Windows 2000 and NT 4.0.

Note Prior to the book being published, BIND version 9.3.1—a maintenance-only update—was released. Some limited tests were performed with this version of the software. Users are advised to always obtain the latest stable version of software.

Many of BIND's security features require OpenSSL. Both FreeBSD and Fedora Core 2 install this package by default, and no special action is required. The Windows binary version of BIND uses standard Windows services and libraries, and again no special action is required. In the event that a packaged version of software is not available, the chapter describes building BIND from a source *tarball*—the widely used term that describes the file (typically ends with `tar.tz`) containing the source and makefiles necessary to build the software and packaged using the `tar` (archive) and `compress` utilities. In all cases BIND was configured and tested as a simple caching server using the files described in Chapter 7 in the section "Caching-only DNS Server."

It is increasingly common that default installations of BIND are either configured to run in a *sandbox* or *chroot jail* (FreeBSD and Fedora) or offer an optional package to do so (bind-chroot RPM on Fedora). This method of running BIND is described in Chapter 10.

The installation procedures make no attempt to secure the various files before running BIND. This was deliberately done to avoid complication. Various methods of securing a BIND installation are described in Chapter 10, and once thoroughly familiar with the initial installation and configuration, you are urged to read this chapter before running a live or operational server.

We start by looking at installation of BIND 9.3.0 on Fedora Core 2 as a representative example of a Linux installation—it could just as easily have used Debian, Mandriva, Gentoo, SuSe, or one of the many other Linux distributions.

■**Note** BIND typically runs as a *daemon* on Linux, Unix, and BSD systems and as a *service* on Windows OSes. When running as a daemon or service, it is called `named` not BIND. This book uses the term *BIND* to describe the package and `named` to describe the running or operational software.

Fedora Core 2 Installation

This section describes the installation, configuration, and testing of BIND 9.3.0 on a clean Fedora Core 2 system using the standard Fedora Core Development RPMs available from a variety of sources. The Fedora Core software is freely available for download at `fedora.redhat.com` or may be purchased as a multi-CD set for a very modest cost from a number of suppliers. The installed FC2 configuration used the following features:

1. The FC2 graphical configuration option was selected (Anaconda).
2. Automatic partitioning was selected.
3. A server install was selected.
4. The GNOME desktop was selected rather than KDE (both were offered during the install process). Whether a GUI desktop is selected and which one is a matter of user taste and the purpose of the system. The author adheres to the security adage "Only install what you need" and would not normally install a GUI desktop for a pure server-based system. However, BIND can be used on a dedicated DNS server, a multifunction server, and a desktop PC—selecting a GUI interface enabled exposure to the widest variety of possible issues.

5. DNS was not selected from the list of software available during the install procedure since it was a requirement to install BIND 9.3.0. If this option is not selected, it will still lead to a partial installation of BIND for reasons noted later in the section. If DNS software is selected during the install process, it will configure a chrooted caching server but does have the merit that some files that would otherwise be missing are installed. Various notes in the following sections identify the differences in installation resulting from selecting this option or not.
6. The Graphical Internet Utilities option was selected. The browser would be used to find and retrieve the various required packages. In a non-GUI server installation, command-line FTP commands would be used.
7. The development tools were installed only because this system was going to be used later to build BIND from a tarball. Installing development tools is a matter of taste, but a secure server especially would never have development tools installed. Either RPMs would be used exclusively or all development would be done on a separate machine, and the applications moved to the target server.

Installation of FC2 took less than 30 minutes and was very uneventful on a 1GHz PC.

■ **Note** FC2 has the Redhat SE Linux options built-in but disabled by default due to potential kernel incompatibilities. Fedora Core 3 is the first system targeted to fully support the SE Linux extensions. For more information about SE Linux, see `www.nsa.gov/selinux/`.

Upgrading BIND 9

Before describing the specific details of the install, it is worth spending a few moments to review the general policy of Fedora Core to help understand what was done and why. Each Fedora Core release, for instance FC2, is maintained and updated primarily for security reasons. New and updated functionality is typically added to the Fedora Core Development system, which is constantly rolling forward to the next releases. The Fedora Core web pages strongly encourage use of either the `yum` or `apt` tools to keep the base updated. However, to keep the process as generic as possible, neither of these auto-update capabilities was used. Instead, the Fedora Core Development RPMs were obtained and installed. The Fedora Core Development RPMs do not track individual FC releases, and in some cases additional RPMs, not defined in the RPM dependencies, had to be installed. The resulting system has a pure FC2 kernel (2.6.5-1.358), but some of the infrastructure was upgraded to an FC3 or higher standard through the use of these additional RPMs.

Post FC2 Installation

Selecting DNS during the FC2 install process causes a default installation of BIND 9.2.8-13 as shown:

```
# rpm -q bind
bind-9.2.8-13
```

If the DNS software option was *not* selected during the FC2 install process, the previous command will still show the same result. This is because the basic (non-DNS) system requires certain DNS libraries to be installed, which in turn picks up various dependencies leading to a partial BIND install.

If DNS software is selected during FC2 setup, the default BIND install configures a chrooted caching name server whose `/etc/named.conf` file is shown here:

```
// generated by named-bootconf.pl

options {
  directory "/var/named";
  /*
   * If there is a firewall between you and nameservers you want
   * to talk to, you might need to uncomment the query-source
   * directive that follows. Previous versions of BIND always asked
   * questions using port 53, but BIND 8.1 uses an unprivileged
   * port by default.
   */
  // query-source address * port 53;
};

//
// a caching-only nameserver config
//
controls {
  inet 127.0.0.1 allow { localhost; } keys { rndckey; };
};
zone "." IN {
  type hint;
  file "named.ca";
};

zone "localhost" IN {
  type master;
  file "localhost.zone";
  allow-update { none; };
};

zone "0.0.127.in-addr.arpa" IN {
  type master;
  file "named.local";
  allow-update { none; };
};

include "/etc/rndc.key";
```

The default installation also configures the name server to allow rndc (see Chapter 9) usage from the loopback address only. This installation may be started with the following command:

```
/etc/rc.d/init.d/named start
```

The default name server is not configured to load at system startup—a process that is described later in this section.

Version Upgrade

The installation objective was to install BIND 9.3.0. The following process should be used irrespective of whether DNS software was selected or not during the FC2 installation. Using Mozilla (the FC2 default installed browser), the following Fedora Core Development RPMs were obtained from www.rpmfind.net:

- bind-9.3.0-2.i386.rpm: This RPM provides the basic name server application (the named daemon) and is mandatory.
- bind-utils-9.3.0-2.i386.rpm: This RPM provides a number of tools and utilities such as dig, nslookup, and nsupdate (all described in Chapter 9).
- bind-libs-9.3.0-2.i386.rpm: This RPM provides resolver and other libraries to invoke DNS services.
- bind-devel-9.3.0-2.i386.rpm: This RPM is optional and contains source and header modules that may be used when building BIND and other applications.

The preceding RPMs were all downloaded to /tmp.

Prior experience had shown that BIND 9.3.0 requires a minimum version of the GLIBC libraries of 2.3.4 or higher. The currently installed version of GLIBC on FC2 was verified using

```
# rpm -q glibc
glibc-2.3.3-27
```

Since this version is lower than the known requirement, the following Fedora Core Development for i386 RPMs were downloaded from www.rpmfind.net (all demanded by various dependencies):

- glibc-2.3.4-3.i386.rpm: GNU generic libc providing support for the standard C library function calls.
- glibc-common-2.3.4-3.i386.rpm: GLIBC dependency containing library binaries and locale information.
- glibc-devel-2.3.4-3.i386.rpm: GLIBC dependency containing headers and other modules used when building most software.
- glibc-headers-2.3.4-3.i386.rpm: GLIBC dependency from GLIBC also responsible for updating the headers files contained in /usr/include.

- `ncsd-2.3.4-3.i386.rpm`: GLIBC dependency containing support libraries for caching services.
- `selinux-1.20.1-2.i386.rpm`: Dependency from GLIBC. This RPM dependency reflects the use of the Fedora Core Development RPMs.

One of the easiest ways to solve all those terrible interdependencies with RPMs is to create a new directory (in this case `/tmp/glibc`), then download or move all the preceding RPMs into it and issue the following command:

```
# rpm -Uvh /tmp/glibc/*
```

This command upgrades the installed versions of software defined by the RPMs in `/tmp/glibc`. The `U` signifies upgrade an existing installation, and `vh` is used to provide a formatted display indicating progress of the command.

■**Note** If the preceding upgrade is not performed, then BIND may decide to terminate rapidly with a segmentation fault with the following message in syslog (`/var/log/messages`): "/usr/lib/libisc.so.9: symbol __snprint_chk, version GLIBC_2.3.4 not defined in file libc.so.6 with link time reference failed."

Finally, to upgrade BIND, the following command was used:

```
# rpm -Uvh /tmp/bind*
```

■**Note** The various RPMs were downloaded from the `rpmfind.net` service. BIND 9.3.0 RPMs were not available for Fedora Core 2 due to the policy described earlier. Instead, the Fedora RPMs used were all under the generic Fedora Core Development for i386 classification. Various other RPMs existed for Fedora Core 3 and in some, but not all, cases for Fedora Core 2. These RPMs were not used.

Configuring BIND 9

A BIND installation requires the following items. Their presence is determined by whether or not DNS software was selected during the initial installation, as shown here:

- A user and group account called `named`, verified using either `vipw/vigr` or `id named`. This account is present irrespective of whether DNS software was selected during FC2 installation or not.
- A documented `named.conf` file in `/etc`, which is present if DNS software is selected during FC2 installation but not present otherwise. There is also a file called `named.custom`, which is used by the DNS GUI tool (see the upcoming section, "Fedora DNS GUI").

- A named.ca zone file in /var/named, which provides the root-servers zone file. This file is present if DNS software is selected during FC2 installation but not present otherwise.
- A localhost.zone file in /var/named to forward map the localhost domain. This file is present if DNS software is selected during FC2 installation but not present otherwise.
- A named.local zone file in /var/named, which provides a reverse-map zone file for localhost. This file is present if DNS software is selected during FC2 installation but not present otherwise.
- A startup and shutdown script for named in /etc/rc.d/init.d. This script is present irrespective of whether DNS software was selected during FC2 installation or not.
- Startup and shutdown links to the script in the various run-level directories, for instance, /etc/rc.d/rc5.d or /etc/rc.d/rc3.d. These links are not present regardless of whether DNS software was selected during FC2 installation.

Fedora DNS GUI

Fedora Core has adopted a very GUI-oriented interface. The file /etc/named.custom is used by a GUI DNS configuration utility called system-config-named, which can be invoked from the command line or under System Settings ➤ Server Settings ➤ DNS from the main graphical menu. A cursory look at this utility showed that it appeared to require as much knowledge as manual configuration, and its rudimentary help file was not even hyperlinked to the BIND help files. This utility was not used again.

Configuring BIND Files

Depending on how you installed FC2, a number of the required files may be missing as noted earlier. There are three ways that these files may be installed:

1. Manually create them. The master.localhost zone file and localhost.rev formats are described with examples in the section "Required Zone Files," located in Chapter 7.
2. FTP them in from a separate server or location. The root.servers zone file may be obtained from many locations, but the definitive file may be obtained from ftp://ftp.internic.net/domain/named.root.
3. Obtain and run the RPM caching-nameserver-7.3-3.noarch.rpm, which configures BIND as simple caching server and installs all the required files.

The named.conf file for a simple caching server (described in the section "Caching-only DNS Server" located in Chapter 7) was manually constructed in /etc/named.conf. The missing files (named.ca, localhost.zone, and named.local) were FTP'd from a local server and renamed using the conventions adopted by this book:

- The named.ca file was renamed to root.servers.
- The localhost.zone file was renamed to master.localhost.
- The named.local file was renamed localhost.rev.

■**Note** The files are renamed in accordance with the conventions used throughout this book and described in the section "Configuration Conventions" located in Chapter 7. While such renaming may be a matter of taste, it is done purely in the interests of eliminating errors that may be caused through the use of meaningless file names.

A new directory was created and allocated the correct permissions to allow for the standard log file (`/var/log/named/example.log`).

```
# cd /var/log
# mkdir named
# touch named/example.log
# chown named:named named/*
# chmod 0664 named/*
```

BIND was now started using the following command:

```
# /etc/rc.d/init.d/named start
```

To confirm the `named` daemon was running, the following command was issued:

```
# ps ax |grep named
1846   ?    S    0:00 /usr/sbin/named -u named
```

To verify that `named` was operational, the following `dig` command was issued (`dig` is a general-purpose diagnostic utility described in Chapter 9).

```
# dig @192.168.2.2 example.com any
; <<>> DiG 9.3.0 <<>> @192.168.2.2 example.com any
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33647
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;example.com.  IN  ANY

;; ANSWER SECTION:
example.com.  51322  IN  NS  ns2.example.com.
example.com.  51322  IN  NS  ns1.example.com.

;; AUTHORITY SECTION:
example.com.  51322  IN  NS  ns1.example.com.
example.com.  51322  IN  NS  ns2.example.com.
```

```
;; Query time: 9 msec
;; SERVER: 192.168.2.2#53(192.168.2.2)
;; WHEN: Mon Jan 24 13:46:06 2005
;; MSG SIZE  rcvd: 92
```

The `@192.168.2.2` forces the `dig` command to use the name server at the defined address—in this case, the IP address of the newly installed DNS caching server. The preceding is a good result, though you may want to replace `example.com` with your favorite domain name, and confirmed that `named` was working correctly. In order to force local use of the new DNS caching server, `/etc/resolv.conf` must be edited to include the server's IP as the first `nameserver` record in this file. The edited `resolv.conf` file should look something like the following:

```
search example.com
nameserver 192.168.2.2
```

If `named` should be loaded on startup, then the named script must be invoked from the `rc.d` directory for the run levels required. The following shows `named` being started and stopped from both run-level 3 (tty) and run-level 5 (GUI):

```
# ln /etc/rc.d/init.d/named /etc/rc.d/rc5.d/S68named
# ln /etc/rc.d/init.d/named /etc/rc.d/rc5.d/K68named
# ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/S68named
# ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/K68named
```

The 68 in the `S` and `K` values is arbitrary. Finally, the system was rebooted to test that automatic startup was working correctly, syslog checked for errors (`/var/log/messages`), and `named` checked to be running with the following command:

```
# ps ax |grep named
1846   ?    S    0:00 /usr/sbin/named -u named
```

There are a number of additional RPMs that can be used to configure BIND, including the bind-chroot RPM to enable `named` to run in a sandbox or chroot jail. This technique, which seeks to limit the locations where `named` can read and write in the server filesystem, is described in Chapter 10.

FreeBSD Installation

FreeBSD 4.x ships with BIND version 8.x as the default or base installation. FreeBSD 5.3—the first of the stable 5.x series—ships with Bind 9.3.0 and by default runs in a sandbox or chroot jail (use of chroot jails is described in Chapter 10).

FreeBSD differentiates between a base DNS install and a normal (nonbase) DNS install. This differentiation allows two versions of BIND to be installed completely independently of each other—they use separate `named.conf` files and program locations. It is thus possible to test a new DNS release by installing as the nonbase system, changing only the `/etc/rc.conf` file or using the command line to run the tests. Reversion to the previous version is trivial since it has not been removed during the testing process. Once testing is complete, a base system install can be used to replace the base version. The base version of named is installed in /usr/sbin and the tools in /usr/bin, whereas a normal (nonbase) installation is made to /usr/local/sbin and the tools to /usr/local/bin. The base version of named assumes the named.conf file is located in /etc/namedb/named.conf, whereas a nonbase install assumes /usr/local/etc/named.conf. FreeBSD creates the user account bind (as opposed to named for Linux) for use with BIND installations.

In all the cases that follow, the FreeBSD ports collection was used to perform the installs.

BIND 9 Nonbase Install

Assuming you have updated the ports-dns collection to get the latest versions, issue the following commands:

```
# cd /usr/ports/dns/bind9
# make install clean
```

The preceding sequence installs BIND 9 in /usr/local/sbin and the tools in /usr/local/bin and assumes the named.conf file is in /usr/local/etc. To run BIND 9 at startup from this location, edit /etc/rc.conf as follows:

```
# add following line if not present
named_enable="YES"
# the line below must replace the line named_program="/usr/sbin/named' if present
# otherwise add it
named_program="/usr/local/sbin/named"
```

Either copy the named.conf file from /etc/namedb to /usr/local/etc before starting BIND (via the named daemon) or create a new version of the file in this directory. To use the BIND 9 tools installed earlier, the command must be preceded with the BIND 9 tool directory path as shown:

```
# /usr/local/bin/dig @192.168.2.2 example.com any
```

Use of @192.168.2.2 (assuming 192.168.2.2 is the address of this server) will force use of the local DNS irrespective of the state of the /etc/resolv.conf file.

BIND 9 Base Install

This section assumes you either want to run the latest version of BIND as the base system—replacing the existing BIND—or a new install with Bind 9 as the base system. Assuming you have updated the ports-dns collection, issue the following commands:

```
# cd /usr/ports/dns/bind9
# make PORT_REPLACES_BASE_BIND9=yes install clean
```

■**Note** From BIND versions 9.3.1, the preceding should be replaced by `make WITH_PORT_REPLACES_BASE_BIND9=yes install clean`.

The preceding sequence installs BIND 9 in `/usr/sbin` and the tools in `/usr/bin`, and assumes the configuration file is `/etc/namedb/named.conf`. To run BIND 9 at startup, `/etc/rc.conf` may need to be edited as shown in the following fragment:

```
# add the following line if not present
named_enable="YES"
# add the following line if not present
named_program="/usr/sbin/named"
```

No special action is required to run BIND 9 tools: The following command will run the base BIND 9 tool `dig`:

```
# dig @192.168.2.2 example.com
```

Use of `@192.168.2.2` (assuming 192.168.2.2 is the address of this server) will force use of the local server irrespective of the state of the `/etc/resolv.conf` file.

FreeBSD 5.3 Issues

FreeBSD 5.3 installs Bind 9.3.0 as the default (or base) version and automatically configures it to run in a sandbox or chroot jail. Chapter 10 describes the use of chroot jails. The chroot jail configuration assumes all BIND's files are located under `/var/named`—including `named.conf`, log files, and PID files. FreeBSD installs hard links in `/etc/namedb` so you can continue to find the files where you thought they would be. To disable the sandbox or chroot jail if required, add to `/etc/rc.conf` the following line(s):

```
named_chrootdir=""                 # disables jail/sandbox
named_pidfile="/var/run/named/pid" # Must set this in named.conf as well
named_chroot_autoupdate="NO"       # Automatically install/update chrooted
                                   # components of named. See /etc/rc.d/named.
named_symlink_enable="NO"          # Symlink the chrooted pid file
```

The default value of the parameters controlling `named` operation in `/etc/defaults/rc.conf` are as follows:

```
named_enable="YES"                 # Run named, the DNS server (or NO).
named_program="/usr/sbin/named"    # path to named, if you want a different one.
named_flags="-u bind"              # Flags for named
named_pidfile="/var/run/named/pid" # Must set this in named.conf as well
named_chrootdir="/var/named"       # Chroot directory (or "" not to auto-chroot it)
named_chroot_autoupdate="YES"      # Automatically install/update chrooted
                                   # components of named. See /etc/rc.d/named.
named_symlink_enable="YES"         # Symlink the chrooted pid file
```

As always, you should not update the /etc/defaults/rc.conf file but rather edit /etc/rc.conf, which will overrule any entries already defined in /etc/defaults/rc.conf. FreeBSD does not ship with localhost.zone or named.local (localhost.rev in this book's terminology) files; instead there is a script, /etc/namedb/make-localhost, which will help you define these files.

Building BIND from Source

This section describes building BIND from a source tarball. In general, there are only three reasons to build BIND using this method:

1. There is no available package or RPM either for the particular host or OS or the right version number.
2. Unique features are required that are not satisfied by the standard packages or RPMs.
3. You like to control everything yourself and have a high tolerance for pain.

Life is not all simple, however, and if building from tarballs, the advantages of any packaged system (for instance, RPMs) are not available. Any dependencies on other software will have to be manually identified—perhaps even discovered at run time!

The test build was run under the Fedora Core 2 installation described earlier. The objective of this procedure was to replicate the existing Fedora build using software built from the tarball. To build BIND from the source tarball, follow these steps:

1. Download the source tarball from www.isc.org (bind-9.3.0.tar.tz) or one of its mirrors into /usr/src. You can use any suitable location, but /usr/src is a general convention.
2. Unzip the tarball using the following commands:

```
# cd /usr/src
# tar xzf bind-9.3.0.tar.gz
```

3. When this operation is complete, it will have created a new directory named bind-9.3.0. Move to this directory:

```
# cd bind-9.3.0
```

4. The software must now be configured using the following command (the line has been split for formatting reasons only and should be entered as a single line):

```
# ./configure –prefix=/usr –sysconfdir=/etc –localstatedir=/var \
–disable-threads –with-openssl
```

The configure arguments used in this line are

- –prefix: Indicates that named will be installed to /usr/sbin and the tools to /usr/bin

- `–sysconfdir`: Tells `named` to look for `named.conf` in `/etc/named.conf`
- `–localstatedir`: Tells `named` to write the PID file to `/var/run/named.pid`
- `–disable-threads`: The safest (or indeed the only safe) option on most systems
- `–with-openssl`: Indicates that DNSSEC services will be built (requires OpenSSL to be installed—the default for FreeBSD and Fedora)

If you are building for FreeBSD, the arguments `–disable-linux-cap` and `–with-randomdev=/dev/random` should also be used, and by convention `–sysconfdir=/etc/namedb` is used to locate `named.conf`.

Caution The preceding step configures BIND such that after issuing `make install` (see step 6) it will overwrite any currently installed BIND package. If the software built is incorrect or fails, you will have a non-operational DNS system. It may be wiser to use a technique such as described in the FreeBSD section and use `–prefix=/usr/local`. This will have the effect of installing `named` to `/usr/local/sbin` and the tools to `/user/local/bin` and provide two copies of BIND. The newly built system can then be tested (any currently running version will have to be stopped first) and only when you are completely satisfied rebuilt using the preceding parameters. If anything goes wrong during testing of the new software, DNS service can be restored by simply restarting the previous BIND version from `/usr/sbin/named`.

---

5. If anything goes wrong with the `configure` sequence, check the entered line carefully and inspect the file `config.log`, which contains the output of the configure session including error messages. When the `configure` process is finished (less than five minutes on a 1GHz PC), BIND should now be built using the following single command:

```
# make
```

6. The `make` command outputs voluminous data and takes roughly 20 minutes on a 1GHz PC. On successful completion, issue the following command:

```
# make install
```

This will install all the various files generated by the build—well over 200 or so. Running this command with the `./configure` arguments defined earlier will replace any existing installed version of BIND—please read the earlier Caution.

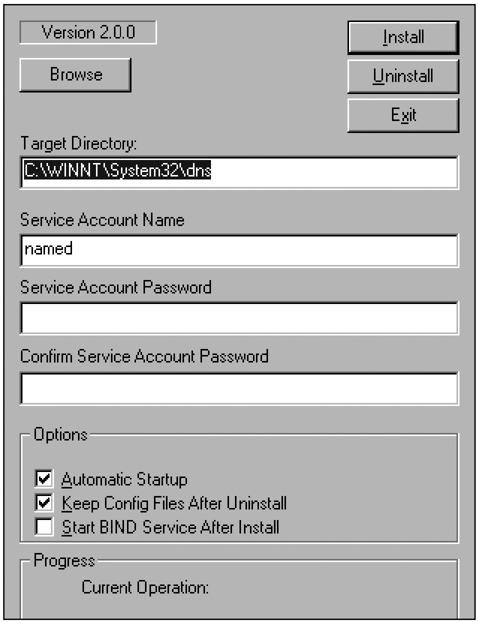
Note In the event that an error occurs during any of the `configure`, `make`, or `make install` procedures, before rerunning, issue the command `make distclean` to remove any previous data before starting the sequence. As noted previously, the `configure` command logs its output to `config.log`. In the case of `make` and `make install`, the commands may be run with data being logged to a file, for instance,. `make >make.log`.

Windows Server 2000 Installation

A packaged binary version of BIND that will install on either Windows 2000 (Server, Advanced Server, or Professional) or Windows NT 4.0 (Server, Advanced Server, or Workstation) is available from the ISC site (`www.isc.org`). The package also includes standard Uninstall functions. This section describes installation of BIND 9.3.0 on a Windows 2000 server using the standard binary package. The installation process was found to be fast and simple.

■ **Note** The Uninstall function supplied with BIND 9.3.0 will work for any BIND 9 versions. Previous Windows BIND versions (BIND 8.x) have their own Uninstall functions. The Windows Server 2003 family is not supported by current BIND releases.

1. Download `Bind 9.3.0.zip` from the ISC site and unzip it into a temporary location, for example, `c:\temp\bind`.
2. There is a small file called `readme1st.txt` with the distribution that provides some information about the installation. In particular it mentions that BIND (or `named.exe`) will run as a *service* on Windows 2000/NT 4.0 and will require a user account called `named` with specific permissions. The install process will create the required account and permissions.
3. In the temporary directory (`c:\temp\bind`), find and double-click `BindInstall.exe`, which will display the screen shown in Figure 6-1.



Target Directory: C:\WINNT\System32\dns
Service Account Name: named
Service Account Password:
Confirm Service Account Password:
Options
☑ Automatic Startup
☑ Keep Config Files After Uninstall
☐ Start BIND Service After Install
Current Operation:

Figure 6-1. *BIND install screen*

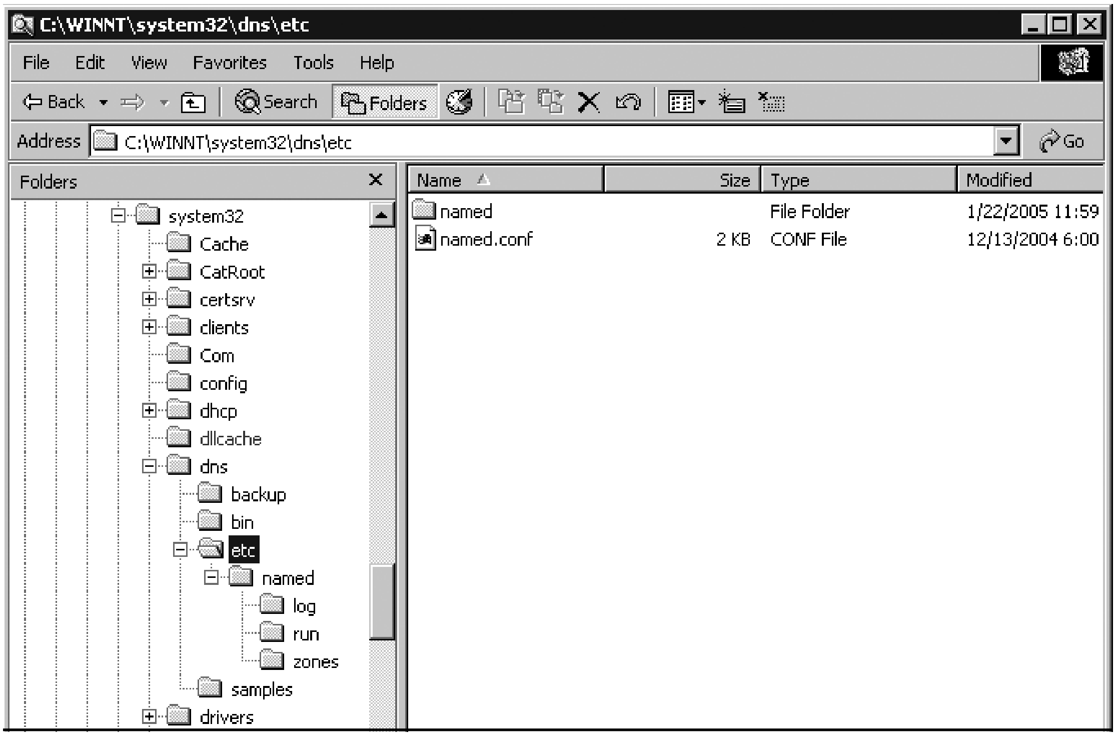
The password entry is optional—it can be left blank or not as you choose. The test installation left the entry blank and used the default Service Account Name as shown in Figure 6-1. The default install directory is `c:\Winnt\system32\dns` (or `%SystemRoot%\system32\dns` in Windows terms). Do *not* check the box labeled Start BIND Service after Install. Click the Install button.

4. Use Windows Explorer to create a directory called `c:\Winnnt\system32\dns\etc\named` and then create three subdirectories called `run`, `zones`, and `log`. Place or create the `master.localhost`, `localhost.rev`, and `root.servers` files in the `zones` subdirectory and the `named.conf` file that follows into the `%SystemRoot%\system32\dns\etc` directory (in the test system, this was `c:\Winnt\system32\dns\etc`). This file is the standard example file used in the Chapter 7 section "Caching-only DNS Server," modified to reflect the Windows path values in the installation. BIND will accept either Windows or Unix line termination conventions.

```
// generated by ME
// CACHING NAME SERVER for WINDOWS 2000 Server
//  Jan 2004
//  a. changed directory statement to windows format
//  b. changed location of log file to named\log\named.log
//   c. changed location of all zone files to named\zones
//   d. added pid-file directive in named\run\named.pid
options {
  directory "C:\Winnt\system32\dns\etc";
  // version added for security, otherwise may be able
       // to exploit known weaknesses
  version "not currently available";
  pid-file "named\run\named.pid";
  recursion yes;
};

// log to named\log\named.log events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// failure messages up to this point are in the event log
  logging{
  channel my_log{
    file "named\log\named.log" versions 3 size 250k;
    severity info;
  };
  category default{
    my_log;
  };
};
zone "." {
  type hint;
  file "named\zones\root.servers";
};
```

```
zone "localhost" in{
  type master;
  file "named\zones\master.localhost";
  allow-update{none;};
};
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "named\zones\localhost.rev";
  allow-update{none;};
};
```

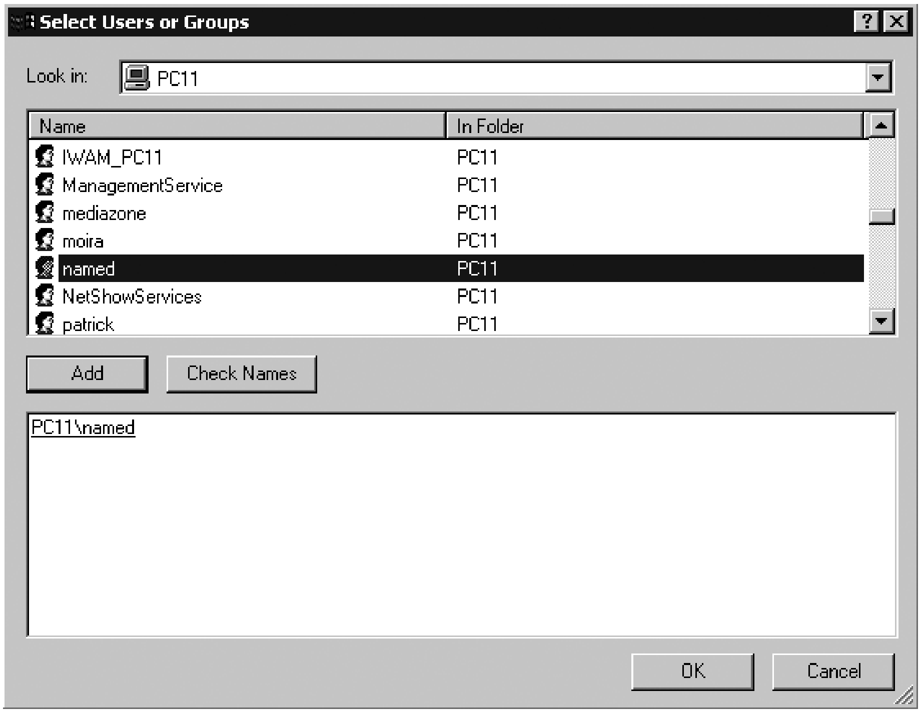
5. The test installation uses the NTFS filesystem and requires permissions to be set to allow BIND to write the log and PID files. Use Windows Explorer to find the BIND install directory (Figure 6-2 shows the default in `c:\Winnt\system32\dns`).



Name	Size	Type	Modified
named		File Folder	1/22/2005 11:59
named.conf	2 KB	CONF File	12/13/2004 6:00

Figure 6-2. *Select BIND install directory*

6. Select the `etc` directory, right-click, and from the pop-up menu click Properties. This will display a tabbed window from which the Security tab should be selected. Click the Permissions button (which will display the window shown later in Figure 6-4 without the named account). To add this account, click the Add button and then find and select the user account called `named` and click Add (see Figure 6-3).



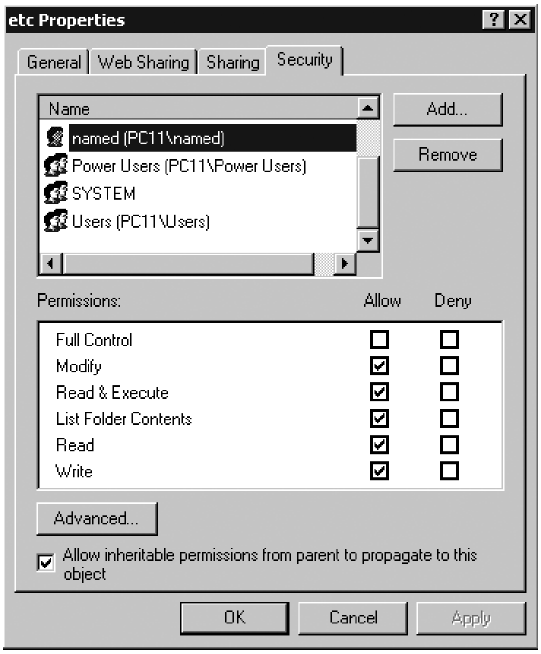
Name	In Folder
IWAM_PC11	PC11
ManagementService	PC11
mediazone	PC11
moira	PC11
named	PC11
NetShowServices	PC11
patrick	PC11

PC11\named

Figure 6-3. *Add the named account*

7. Add all permissions except full control and leave the Allow inheritable permissions . . . check box set (the default) as shown in Figure 6-4. Finally, click OK.

8. BIND installs its software to a nonstandard location (the default is `%SystemRoot%\system32\dns\bin`). To use diagnostic tools such as `dig` and other command-line tools, the full path will be required or the Windows path environment variable can be changed to include the BIND installation directory. You can then forget where the BIND tools are installed! The path variable can be set using the following procedure. Start by selecting Start ➤ Settings ➤ Control Panel ➤ System as shown in Figure 6-5.

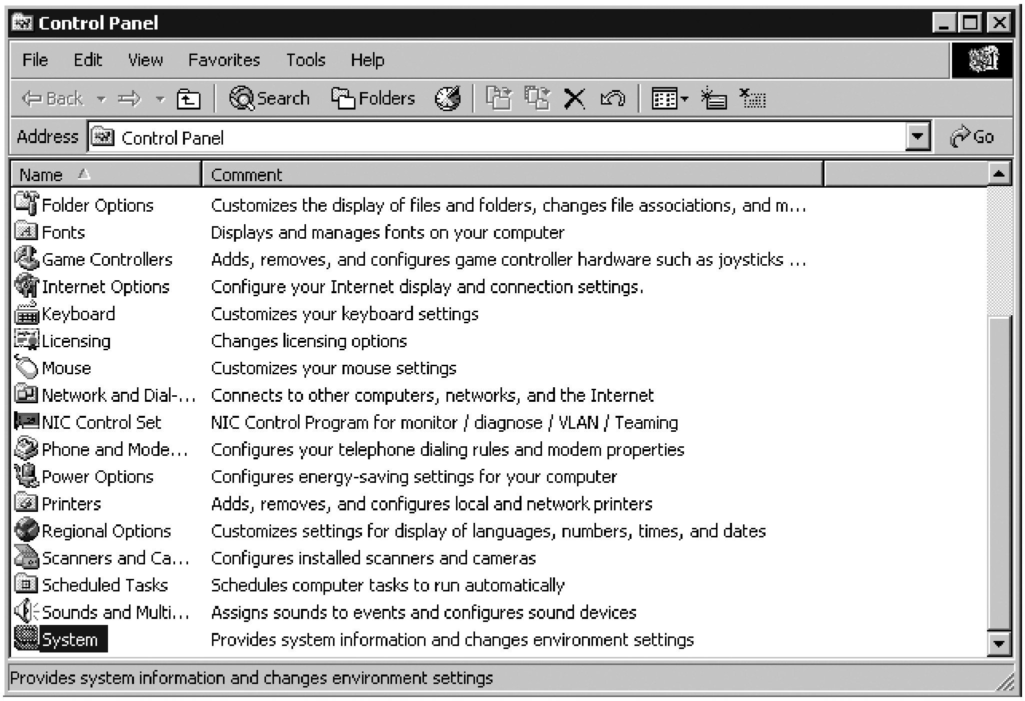


Name
named (PC11\named)
Power Users (PC11\Power Users)
SYSTEM
Users (PC11\Users)

Permissions:	Allow	Deny
Full Control	☐	☐
Modify	☑	☐
Read & Execute	☑	☐
List Folder Contents	☑	☐
Read	☑	☐
Write	☑	☐

☑ Allow inheritable permissions from parent to propagate to this object

Figure 6-4. *Change permissions*



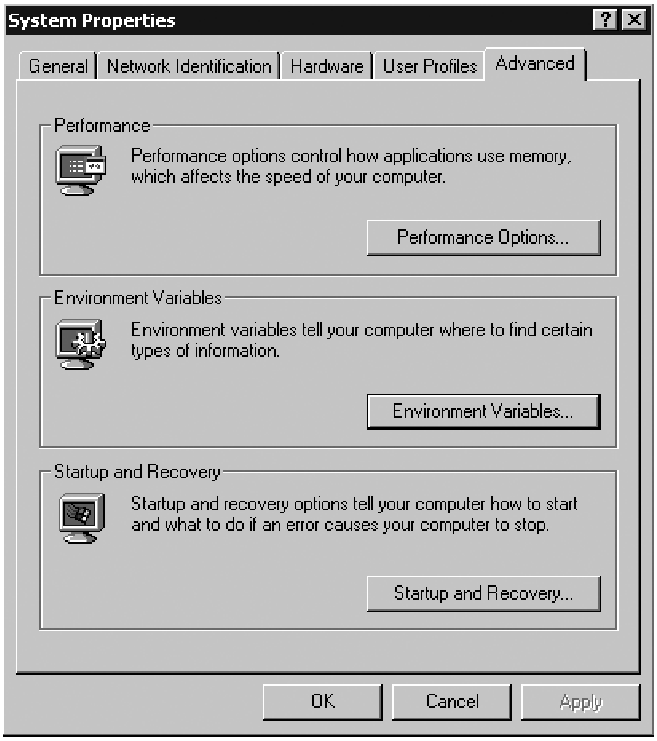
Address Control Panel

Name	Comment
Folder Options	Customizes the display of files and folders, changes file associations, and m...
Fonts	Displays and manages fonts on your computer
Game Controllers	Adds, removes, and configures game controller hardware such as joysticks ...
Internet Options	Configure your Internet display and connection settings.
Keyboard	Customizes your keyboard settings
Licensing	Changes licensing options
Mouse	Customizes your mouse settings
Network and Dial-...	Connects to other computers, networks, and the Internet
NIC Control Set	NIC Control Program for monitor / diagnose / VLAN / Teaming
Phone and Mode...	Configures your telephone dialing rules and modem properties
Power Options	Configures energy-saving settings for your computer
Printers	Adds, removes, and configures local and network printers
Regional Options	Customizes settings for display of languages, numbers, times, and dates
Scanners and Ca...	Configures installed scanners and cameras
Scheduled Tasks	Schedules computer tasks to run automatically
Sounds and Multi...	Assigns sounds to events and configures sound devices
System	Provides system information and changes environment settings

Provides system information and changes environment settings

Figure 6-5. *Select system*

9. Double-click System and then click Environment Variables on the Advanced tab, as shown in Figure 6-6.



Performance options control how applications use memory, which affects the speed of your computer.

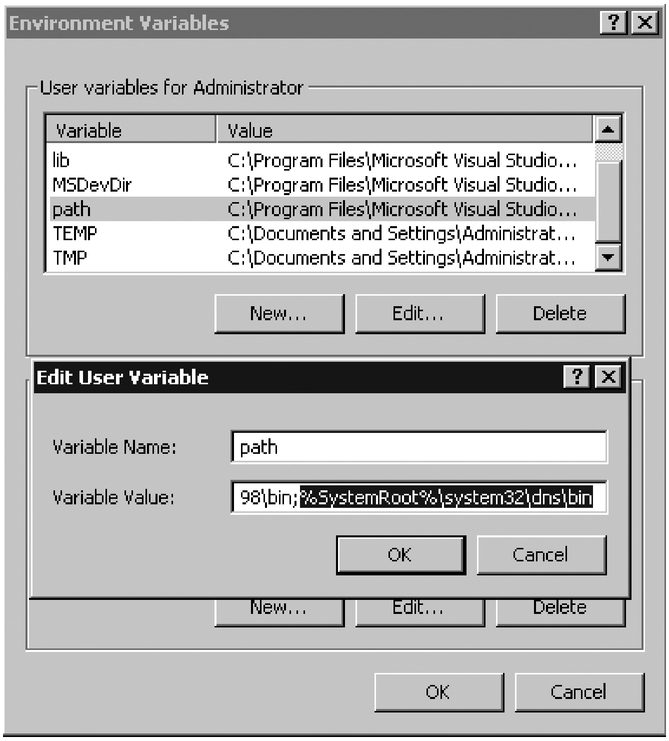
Environment variables tell your computer where to find certain types of information.

Startup and recovery options tell your computer how to start and what to do if an error causes your computer to stop.

Figure 6-6. *Select environment variables*

10. Find and double-click the path line in the top window and add `%SystemRoot%\system32\dns\bin` (or specify wherever your BIND `bin` directory is located) as shown in Figure 6-7. Click OK in both dialog boxes to exit.

■ **Note** The path separator on Windows is a semicolon, not a colon, as in the Unix world. Setting the path has the effect of automatically locating, say, `dig` or `nsupdate`. However, there is a Windows version of `nslookup` that will be found first. Using the BIND version of `nslookup` either requires a full path command, such as `c:\winnt\system32\dns\bin\nslookup` when running it from the command line, or the preceding path directive must be placed first in the list—which has the disadvantage that it will add an extra check for all other program loading operations that use normal Windows locations.



Variable	Value
lib	C:\Program Files\Microsoft Visual Studio...
MSDevDir	C:\Program Files\Microsoft Visual Studio...
path	C:\Program Files\Microsoft Visual Studio...
TEMP	C:\Documents and Settings\Administrat...
TMP	C:\Documents and Settings\Administrat...

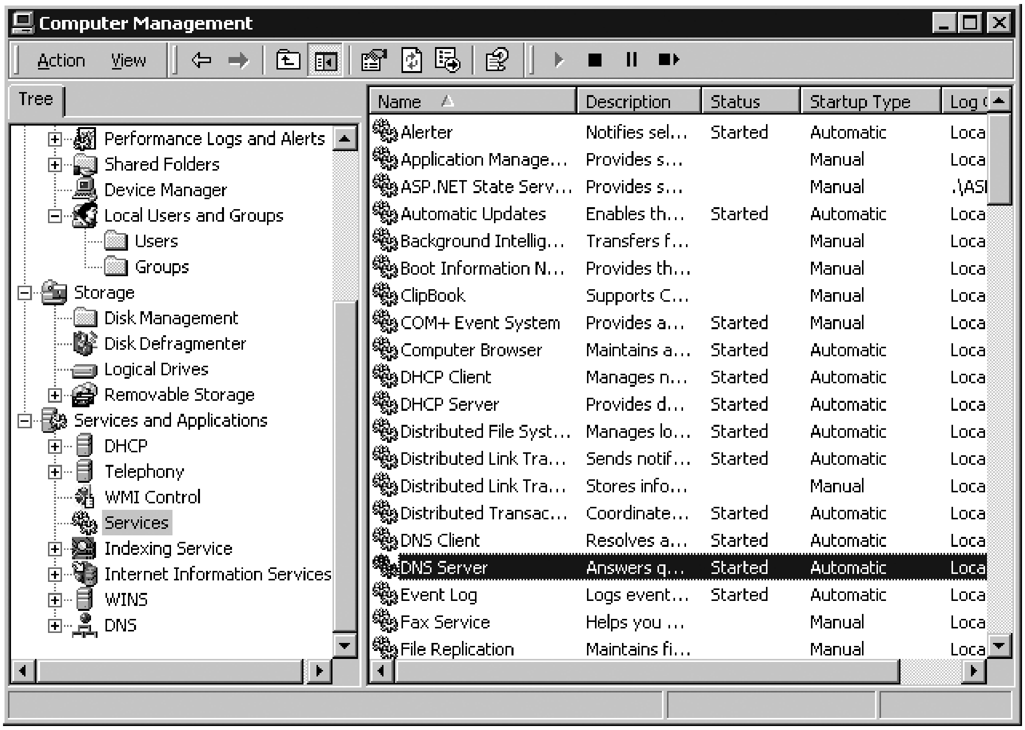
Variable Name: path

Variable Value: 98\bin;%SystemRoot%\system32\dns\bin

Figure 6-7. *Edit path environment variable*

11. The test installation used Windows 2000 Server, which runs the standard Windows DNS service by default. Before the BIND service is started, it is necessary to stop and permanently disable the standard Windows DNS service. To do this, access Computer Management (Start ➤ Programs ➤ Administrative Tools ➤ Computer Management). Expand Services and Applications, and then double-click Services and find DNS Server (see Figure 6-8).

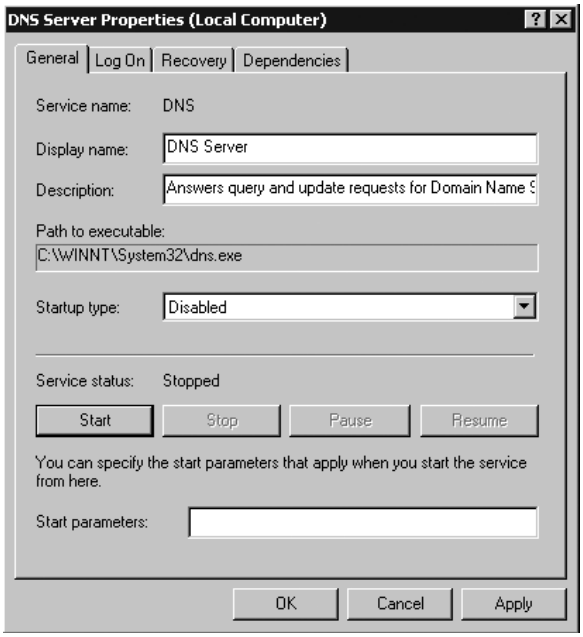
12. Double-click DNS Server, select Disable from the Startup type combo box, and click Stop. When the service has been confirmed as stopped, click OK (see Figure 6-9). Because the standard Windows DNS service is being replaced with the BIND DNS service, this procedure has the effect of permanently disabling the Windows DNS service at startup.



Performance Logs and Alerts
Shared Folders
Device Manager
Local Users and Groups
Users
Groups
Storage
Disk Management
Disk Defragmenter
Logical Drives
Removable Storage
Services and Applications
DHCP
Telephony
WMI Control
Services
Indexing Service
Internet Information Services
WINS
DNS

Name	Description	Status	Startup Type	Log (
Alerter	Notifies sel…	Started	Automatic	Loca
Application Manage…	Provides s…		Manual	Loca
ASP.NET State Serv…	Provides s…		Manual	.\ASI
Automatic Updates	Enables th…	Started	Automatic	Loca
Background Intellig…	Transfers f…		Manual	Loca
Boot Information N…	Provides th…		Manual	Loca
ClipBook	Supports C…		Manual	Loca
COM+ Event System	Provides a…	Started	Manual	Loca
Computer Browser	Maintains a…	Started	Automatic	Loca
DHCP Client	Manages n…	Started	Automatic	Loca
DHCP Server	Provides d…	Started	Automatic	Loca
Distributed File Syst…	Manages lo…	Started	Automatic	Loca
Distributed Link Tra…	Sends notif…	Started	Automatic	Loca
Distributed Link Tra…	Stores info…		Manual	Loca
Distributed Transac…	Coordinate…	Started	Automatic	Loca
DNS Client	Resolves a…	Started	Automatic	Loca
DNS Server	Answers q…	Started	Automatic	Loca
Event Log	Logs event…	Started	Automatic	Loca
Fax Service	Helps you …		Manual	Loca
File Replication	Maintains fi…		Manual	Loca

Figure 6-8. *Select DNS service*



Service name: DNS

Display name: DNS Server

Description: Answers query and update requests for Domain Name S

Path to executable:
C:\WINNT\System32\dns.exe

Startup type: Disabled

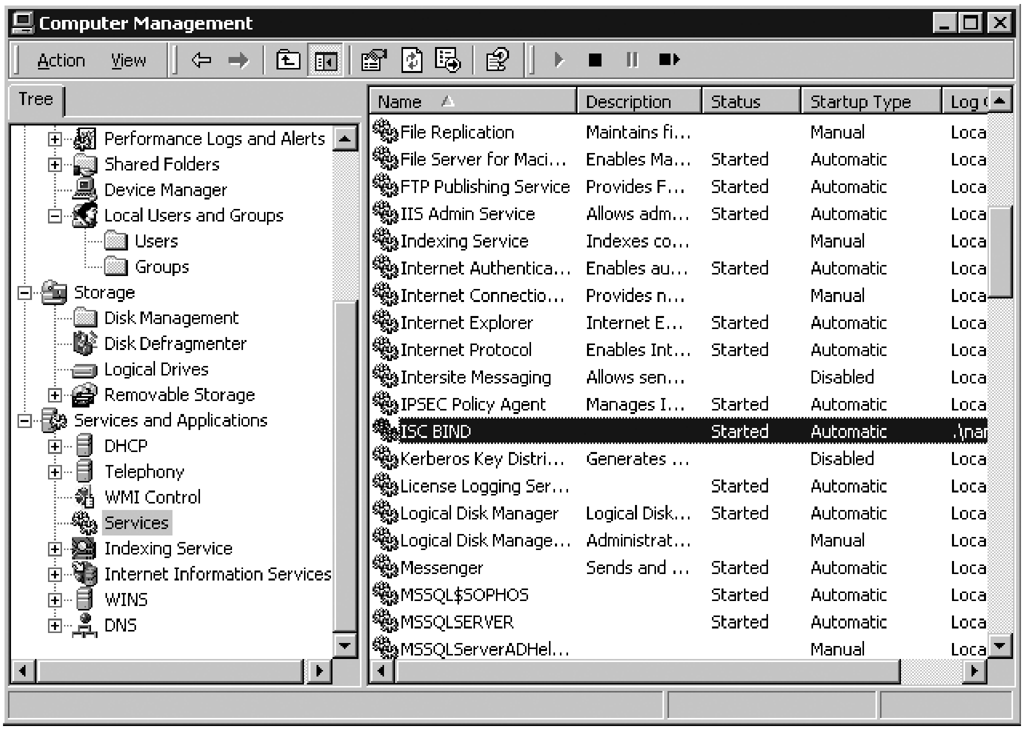
Service status: Stopped

You can specify the start parameters that apply when you start the service from here.

Start parameters:

Figure 6-9. *Stop DNS service*

13. Find and double-click ISC BIND (see Figure 6-10).



Name	Description	Status	Startup Type	Log (
File Replication	Maintains fi...		Manual	Loca
File Server for Maci...	Enables Ma...	Started	Automatic	Loca
FTP Publishing Service	Provides F...	Started	Automatic	Loca
IIS Admin Service	Allows adm...	Started	Automatic	Loca
Indexing Service	Indexes co...		Manual	Loca
Internet Authentica...	Enables au...	Started	Automatic	Loca
Internet Connectio...	Provides n...		Manual	Loca
Internet Explorer	Internet E...	Started	Automatic	Loca
Internet Protocol	Enables Int...	Started	Automatic	Loca
Intersite Messaging	Allows sen...		Disabled	Loca
IPSEC Policy Agent	Manages I...	Started	Automatic	Loca
ISC BIND		Started	Automatic	.\nar
Kerberos Key Distri...	Generates ...		Disabled	Loca
License Logging Ser...		Started	Automatic	Loca
Logical Disk Manager	Logical Disk...	Started	Automatic	Loca
Logical Disk Manage...	Administrat...		Manual	Loca
Messenger	Sends and ...	Started	Automatic	Loca
MSSQL$SOPHOS		Started	Automatic	Loca
MSSQLSERVER		Started	Automatic	Loca
MSSQLServerADHel...			Manual	Loca

Figure 6-10. *Find ISC BIND*

14. Double-click ISC BIND and click Start. The Startup type is set to Automatic by default, which means it will load on startup.

■ **Note** Any errors will be logged under Applications in the Event Log.

15. To test the DNS server, open a DOS box (select Start ➤ Run, enter cmd in the dialog box, and then click OK) and try a `dig` command (described in Chapter 9):

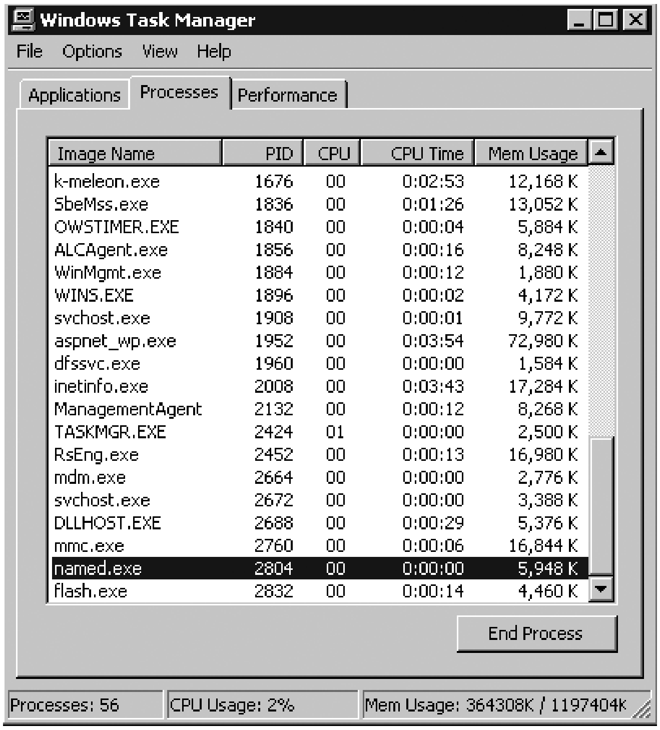
```
# dig @192.168.2.2 example.com any
; <<>> DiG 9.3.0 <<>> @192.168.2.2 example.com any
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33647
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 0
;; QUESTION SECTION:
;example.com.  IN  ANY

;; ANSWER SECTION:
example.com.  51322  IN  NS  ns2.example.com.
example.com.  51322  IN  NS  ns1.example.com.

;; AUTHORITY SECTION:
example.com.  51322  IN  NS  ns1.example.com.
example.com.  51322  IN  NS  ns2.example.com.

;; Query time: 9 msec
;; SERVER: 192.168.2.2#53(192.168.2.2)
;; WHEN: Mon Jan 24 13:46:06 2005
;; MSG SIZE  rcvd: 92
```

The `@192.168.2.2` forces the `dig` command to use the name server at the defined address irrespective of the settings in the TCP/IP properties—in this case, it is the IP of the newly installed DNS caching server. The preceding was a good result, though you may want to replace `example.com` with your favorite domain name and confirm that `named` is working correctly. In order to force local use of the new DNS caching server, either modify the DNS setting in the TCP/IP properties (select Start ➤ Control Panel ➤ Network and Dial-up Connections ➤ Local LAN, right-click and select Properties, and then find and double-click Internet TCP/IP) or create a `%SystemRoot%\system32\dns\etc\resolv.conf` file. The `resolv.conf` file should look something like the following:

```
search example.com
nameserver 192.168.2.2
```

16. To fully test the server, it is necessary to reboot. When the server has rebooted, use Event Viewer to check the Applications log for failure messages, and then use Task Manager to check that the ISC BIND service started up (it loads as `named.exe`) as shown in Figure 6-11.

Installing BIND on Windows 2000 Pro; Windows 2000 Server; Windows 2000 Advanced Server; or Windows NT 4.0 Server, NT 4.0 Advanced Server, or NT 4.0 Workstation is a simple task requiring little user intervention. The entire process takes less than 10 minutes. If you need or want consistency of DNS for maintenance and other purposes across mixed Windows, Unix, Linux, or BSD environments, using BIND is the only solution. As a happy side benefit, you also get `dig`, `nsupdate`, `rndc`, and other tools, meaning that you can diagnose, update, and control BIND installations on other OS platforms from a Windows desktop or server.



```
Image Name          PID   CPU   CPU Time   Mem Usage
k-meleon.exe        1676  00    0:02:53    12,168 K
SbeMss.exe          1836  00    0:01:26    13,052 K
OWSTIMER.EXE        1840  00    0:00:04     5,884 K
ALCAgent.exe        1856  00    0:00:16     8,248 K
WinMgmt.exe         1884  00    0:00:12     1,880 K
WINS.EXE            1896  00    0:00:02     4,172 K
svchost.exe         1908  00    0:00:01     9,772 K
aspnet_wp.exe       1952  00    0:03:54    72,980 K
dfssvc.exe          1960  00    0:00:00     1,584 K
inetinfo.exe        2008  00    0:03:43    17,284 K
ManagementAgent     2132  00    0:00:12     8,268 K
TASKMGR.EXE         2424  01    0:00:00     2,500 K
RsEng.exe           2452  00    0:00:13    16,980 K
mdm.exe             2664  00    0:00:00     2,776 K
svchost.exe         2672  00    0:00:00     3,388 K
DLLHOST.EXE         2688  00    0:00:29     5,376 K
mmc.exe             2760  00    0:00:06    16,844 K
named.exe           2804  00    0:00:00     5,948 K
flash.exe           2832  00    0:00:14     4,460 K

Processes: 56     CPU Usage: 2%     Mem Usage: 364308K / 1197404K
```

Figure 6-11. *Task Manager*

Summary

This chapter covered the installation of BIND 9.3.0 on a variety of widely available OS platforms. In order to keep installations simple, no attempt was made during the installations to secure the various files used by BIND. You are urged to read Chapter 10 before running BIND operationally.

BIND was installed on Fedora Core 2 as a representative of the Linux range of OS platforms. The BIND installation using standard RPMs was simple, but can, depending on how it is installed, result in an incomplete installation—missing zone files—that required some trivial corrective action.

BIND was installed on FreeBSD as representative of the BSDs (FreeBSD, NetBSD, OpenBSD, and DragonflyBSD) and Unix OS platforms. The powerful ports collection was used to perform the installation. No problems were encountered during this installation when using either the 4.x and 5.x releases.

The packaged binary version of BIND for Windows was installed on a Windows 2000 Server and a full set of tests run to ensure it was fully functional. No problems were encountered during this installation.

To cover those situations where a packaged version is not available, BIND was compiled on the Fedora Core 2 platform from a source tarball.

The next chapter looks at the detailed configuration of BIND necessary to run the DNS types defined in Chapter 4.

CHAPTER 7

BIND Type Samples

This chapter presents sample BIND configurations and accompanying descriptions for each of the DNS types described in Chapter 4. But before jumping into these configurations, let's take a moment to quickly review these types:

- *Zone master*: A name server that responds authoritatively for the zone, which reads the zone file from a local file system and which is capable of transferring the zone file to one or more slave servers
- *Zone slave*: A name server that responds authoritatively for the zone but which obtains its zone file by a zone transfer from a zone master
- *Caching server*: A name server that provides recursive query support to clients and that saves the results in a cache
- *Forwarding server*: A name server that passes all queries for which it has no cached results to a caching name server
- *Stealth or Split server*: A name server configuration, typically used in perimeter defense, which separates the services provided to external and internal users
- *Authoritative-only server*: A name server that only provides responses for zones for which it is either a zone master or a zone slave and that does not support recursive queries

Most name server configurations are schizophrenic in nature—they may be *masters* for some zones, *slaves* for others, *forward* others, and provide *caching* services for all comers. Each configuration type described next represents a building block and may be used in a stand-alone configuration or be combined with other types to provide more complex configurations.

Before We Start

In order to make sense of the samples used in this chapter, the following sections cover some background information and formatting issues.

Configuration Layout

A BIND system consists of the following items:

- A `named.conf` file, which describes the server characteristics and the zone files used. The entries in this file are described in Chapter 12. The `named.conf` file is normally located in `/etc` for most Linux distributions, `%SystemRoot%\system32\dns\etc` for Windows, and in either `/etc/namedb` or `/usr/local/etc` for BSD-based systems.
- Depending on the configuration, the name server may use one or more zone files describing the domains being managed. The entries in zone files are described in Chapter 2; Chapter 13 provides a complete reference. By convention the zone files are normally located in `/var/named` for Linux and most Unix-based systems, but this location can be controlled by BIND configuration parameters (using the `directory` statement).
- Depending on the services being provided by the name server, it may require additional zone files describing the *localhost* environment and *root-servers*.

All the configuration files are deliberately kept simple—references are provided to various chapters that describe more advanced parameters as appropriate. Comments are included in the files to describe functionality, but in general they are complete and can be copied directly to a name server configuration with some simple editing to change local name values, IP addresses, and file names.

Configuration Conventions

For reasons of consistency, the configuration scenario used throughout this chapter adheres to these characteristics:

- The domain name is `example.com`.
- The zone has two name servers. One is hosted within `example.com`, the other in an external or foreign domain.
- The IP addresses used assume the private Class C address 192.168.254.0/24—a slightly artificial case (for information on address classes and the / notation, see the sidebar "IPv4 Addresses and CIDR" in Chapter 3).
- The zone consists of the following servers:
  - *Two mail servers*: One is hosted within `example.com`, and a second (backup) mail server hosted in an external or foreign domain.
  - A web server hosted internally and accessed as `www.example.com`.
  - An FTP server hosted externally and accessed as `ftp.example.com`.
  - An additional host called `joe.example.com` for some reason best known to the domain owner.

■**Note** Some readers may think using `example.com` as the default domain name in sample configurations is about as exciting as reviewing a Hello World coding snippet. However, the dilemma is that most of the really bizarre or interesting domain names that would be descriptive or just plain fun to use have already been registered! It would seem a little unfair if the domain name owner were suddenly bombarded with strange diagnostic commands or other artifacts while readers experiment with features. RFC 2606 identifies that IANA (`www.iana.org`) in its infinite wisdom has reserved the domain names `example.com`, `.org`, and `.net` purely for the purposes of experimentation and documentation. In the interests of being a good netizen, this book generally uses `example.com` throughout, but just to spice things up a bit it occasionally uses `example.net`.

Zone File Naming Convention

If your particular situation calls for just one or two zone files, then it may not matter how you title them. However, as the number of zone files increase, this can quickly become a management problem, so establishing a standard file naming convention is key in order to quickly locate a particular file. These days it seems everyone has their own ideas regarding an ideal naming convention, and thus something that is supposed to be useful can become contentious. This book uses the following convention throughout:

- `/var/named/`: This base directory contains all the housekeeping zone files (for example, localhost zone files, reverse-mapping zone files, `root.servers` zone file, etc.) with a subdirectory structure used as follows:
    - `/var/named/master`: This directory contains the master zone files.
    - `/var/named/slave`: This directory contains the slave zone files.
    - `/var/named/view`: This directory contains the view zone files.
- Master zone files are named `master.example.com` (`master.example.net` etc.); if it is a subdomain it will be `master.sub-domain.example.com`.
- Slave zone files are named `slave.example.com` (or `slave.example.fr`, etc.); if it is a subdomain it will be `slave.sub-domain.example.com`, etc.
- The root server zone file is called `root.servers` (typically called `named.ca` or `named.root` in BIND distributions).
- The reverse-mapping file name uses the IP address in its correct or normal order with `.rev` appended to it. For example, if the zone is `23.168.192.IN-ADDR.ARPA`, then the reverse-mapping zone file is called `192.168.23.rev`. There is no reason for the zone file name to be as confusing as the reverse-mapped zone file contents!
- The `localhost` zone file is called `master.localhost` (typically called `localhost.zone` when supplied with BIND distributions). The reverse-mapping file is called `localhost.rev` (typically called `named.local` when supplied with many BIND distributions).

■Note For most Linux and BSD BIND distributions, there is a small overhead after installation to rename the standard distribution files, but the equation *"meaningless file names +2 AM panic = serious chance of error"* is one that should be avoided at all costs. There are plenty of things in the DNS world that need to be remembered—meaningless file names are not one of them.

Keep in mind this is just a convention, and does not affect the behavior of BIND. That said, you are not bound to following these rules; however, it's crucial that you do establish some sort of convention in order to lessen the possibility of administration gaffes.

Required Zone Files

Depending on operational requirements, BIND may need a number of zone files to allow it to provide the required functionality—these are in addition to any zone files that explicitly describe master or slave zones.

root.servers

This file (called `named.ca` or `named.root` in many distributions but renamed `root.servers` in this book) is a standard zone file containing A RRs for the root-servers (`A.ROOT-SERVERS.NET`—`M.ROOT-SERVERS.NET`). When BIND is initially loaded, it uses this zone file to query the root zone to obtain a complete list of the current authoritative root-servers and subsequently uses the obtained list rather than the `root.servers` zone file. When a name server cannot resolve a query from its local zone files or its cache, it uses the name servers obtained via this query to return a referral (if an iterative query) or to find an answer (if a recursive query). The `root.servers` file is defined using a normal `zone` clause with a `type hint` statement as in the following example:

```
// BIND named.conf fragment
zone "." {
  type hint;
  file "root.servers";
};
```

The `zone "."` declaration is short for the root zone (the normally silent dot at the end of an FQDN). A query to this zone will return a list of the root-servers, which is then used by the name server as a starting point for any domain query, for which there is no locally defined zone (slave or master) or a cached answer.

By convention, the hint zone is usually included as the first zone clause in `named.conf`, but there is no good reason for this—it may be placed anywhere suitable. If the configuration is running an internal name service on a closed network, or the name server does not support recursive queries, the `root.servers` file or hint zone is not required. If the zone is not defined, but recursive queries are required, BIND has an internal list that it uses.

The root-servers change very infrequently for obvious reasons; nevertheless, the zone file supplied with any distribution will eventually become outdated. A new zone file can be obtained from a number of locations including ICANN/INTERNIC (`ftp://ftp.internic.net/domain` download file `named.root`). BIND will log any discrepancies from the current `root.servers` zone file and the list it obtains on the initial query of the root zone (see earlier), but will carry on using the retrieved list. The `root.servers` file should be updated perhaps every 12 months or whenever there are log messages noting discrepancies when BIND loads. A `root.servers` fragment is shown here:

```
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  "
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.root
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:    Jan 29, 2004
;       related version of root zone:   2004012900
;
;
; formerly NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
```

The trailing dot in the NS RR line earlier indicates it is an FQDN and signifies this is a name server for the root domain. In total, there are 13 name servers listed in this zone file, namely `a.root-servers.net` to `m.root-servers.net`.

master.localhost

This zone file allows resolution of the name localhost to the loopback address `127.0.0.1` when using the name server. Any query for localhost from any host using the name server will return `127.0.0.1`—namely its fixed localhost address. This file is particularly important because localhost is used by many applications. The localhost zone is defined as shown here:

```
// BIND named.conf fragment
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update {none;}; // optional
};
```

In the standard files supplied with many BIND 9 distributions, the zone-specific statement `allow-update (none;);` is defined, which suppresses any, accidental or malicious, Dynamic DNS (DDNS) behavior that may corrupt the localhost zone file. Dynamic DDNS is disabled by default in BIND 9, and the statement is not strictly required—its inclusion may be regarded as defensive or paranoid at your discretion. An example `master.localhost` file (called `localhost` or `localhost.zone` in many distributions) is shown here:

```
$TTL 86400 ; 24 hours could have been written as 24h or 1d
$ORIGIN localhost.
@  1D  IN    SOA @ hostmaster (
             2004022401 ; serial
             12h ; refresh
             15m ; retry
             1w ; expiry
             3h ; minimum
     )
@  1D  IN  NS @ ; localhost is the name server
   1D  IN  A  127.0.0.1 ; always returns the loop-back address
```

The file embodies the true minimalist (and occasionally incomprehensible) tradition of configuration files. Extensive use is made of `@`, which will force `$ORIGIN` substitution, as explained in Chapter 2. Every record uses a 24-hour (1D) TTL; in RFC 1035 format this time value would be 86400. Even recent BIND distributions use a zone e-mail address of root (the historic practice), current practice (RFC 2142) recommends the use of hostmaster for this purpose, and the file has been correspondingly modified.

The following offers an alternate version of the preceding file that may be more understandable:

```
$TTL 1d ;
$ORIGIN localhost.
localhost.    IN    SOA localhost. hostmaster.localhost. (
             2002022401 ; serial
             3H ; refresh
             15M ; retry
             1w ; expire
             3h ; minimum
     )
localhost.     IN  NS localhost. ; localhost is the name server
localhost.     IN  A  127.0.0.1 ; the loop-back address
```

■**Note** The preceding file uses the *BIND only* abbreviations for time periods in an ugly variety of upper- and lowercase formats to reinforce the point that they are case-insensitive.

IPv6 Localhost

The IPv6 localhost or loopback address is `::1` and is defined using an AAAA RR (a Quad A RR). Recall from Chapter 5 that A and AAAA RRs may be freely mixed in a zone file enabling the standard `master.localhost` zone file to be modified, thus requiring no change to the zone file declaration in the `named.conf` file, as shown here:

```
$TTL 86400 ; 24 hours could have been written as 24h or 1d
$ORIGIN localhost.
@  1D  IN    SOA @ hostmaster (
             2004022401 ; serial
             12h ; refresh
             15m ; retry
             1w ; expiry
             3h ; minimum
     )
@  1D  IN  NS @ ; localhost is the name server
   1D  IN  A  127.0.0.1 ; IPv4 loop-back address
   1D  IN  AAAA  ::1 ; IPv6 loop-back address
```

Reverse-Map Zone Files

Reverse mapping describes the process of translating an IP address to a host name. This process uses the reserved domain IN-ADDR.ARPA and, if it is to be supported, requires a corresponding zone file. Reverse-mapping and its zone file format are described in Chapter 3.

■**Note** Many service providers do not provide delegation of reverse mapping for IPv4 addresses (described in Chapter 8), and as a consequence users can get into the bad habit of not including reverse-map files in their name server configurations. If the name server is behind a firewall/NAT gateway and is using local (RFC 1918) IPv4 addresses, for example, 192.168.0.0/16, it is very important that a reverse-map zone file be included to cover the private IPs being used. Failure to do so will result in queries for these IPs being passed to the public network—consuming both resources and slowing up all local traffic while operations timeout. Recent studies suggest that up to 7% of all traffic hitting certain root-servers comes from badly configured name servers, which generate unnecessary reverse-map queries for local IP addresses.

0.0.127.IN-ADDR.ARPA

This special zone allows reverse mapping of the loopback address 127.0.0.1 to satisfy applications that do reverse or double lookups. It is sometimes called named.local in Linux distributions but is renamed localhost.rev in this book. Any request for the address 127.0.0.1 using this name server will return the name localhost. The 0.0.127.IN-ADDR.ARPA zone is defined as shown here:

```
// BIND named.conf fragment
zone "0.0.127.IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;}; // optional
};
```

In the standard files supplied with many BIND 9 distributions, the zone-specific statement allow-update (none;); is defined, which suppresses Dynamic DNS (DDNS) behavior. This is

BIND 9’s default mode and is not strictly required—its inclusion may be regarded as defensive, paranoid, or prudent at your discretion. An example `localhost.rev` file is shown here:

```
$TTL 86400 ; 24 hours
; could use $ORIGIN 0.0.127.IN-ADDR.ARPA.
@       IN      SOA     localhost. hostmaster.localhost.  (
                        1997022700 ; Serial
                        3h      ; Refresh
                        15      ; Retry
                        1w      ; Expire
                        3h )    ; Minimum
        IN      NS      localhost.
1       IN      PTR     localhost.
```

This file, supplied with most BIND distributions, normally has no `$ORIGIN` directive (the comment line shows the form the `$ORIGIN` directive would take if present) and thus serves to illustrate the additional work required when it is missing. In this case, the `@` name is taken to mean the value in the `zone` clause of `named.conf`, which in the preceding `named.conf` fragment reads as follows:

```
zone "0.0.127.IN-ADDR.ARPA" in{
```

This name will be used by the `$ORIGIN` substitution rule within this file. The absence of an `$ORIGIN` directive means looking in two places (the `named.conf` file and the zone file) to understand exactly what is happening. In the last line of this file, the leading 1 is a name and because it is unqualified (it does not end with a dot), `$ORIGIN` substitution also takes place. This line could have been written as follows:

```
1.0.0.127.in-addr.arpa.    IN   PTR  localhost.
```

IPv6 Localhost Reverse Map

The IPv6 loopback address is written typically as ::1 but its full format is 0:0:0:0:0:0:0:1. Recall from Chapter 5 that reverse mapping for IPv6 uses a reversed nibble format—each 4 bits of the 128-bit address is defined and then placed under the `IP6.ARPA` domain. This leads to the brutally long definition that follows and which comprises one followed by 31 zeros:

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA.
```

The split between the zone or domain name part and the host part defined inside the zone file is arbitrary. The following definitions use a domain name comprising the global routing prefix (or site prefix) of 48 bits and the remainder defined inside the zone file. The zone clause fragment for `named.conf` is shown here:

```
// named.conf fragment
....
zone "0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA" in{
   type master;
   file "localhost-ipv6.rev";
   allow-update {"none";};
};
```

Here is the zone file `localhost-ipv6.rev`:

```
$TTL 86400 ; 24 hours
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA.
@       IN      SOA     localhost. hostmaster.localhost.  (
                        1997022700 ; Serial
                        3h      ; Refresh
                        15      ; Retry
                        1w      ; Expire
                        3h )    ; Minimum
        IN      NS      localhost.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0       IN      PTR     localhost.
```

BIND named.conf File Format and Style

The following notes provide a brief overview of some terminology to enable the reader to make sense of the files presented in this chapter, though some reference to Chapter 12 will be necessary if a detailed description of a particular value is required.

BIND's standard documentation uses a confusing number of terms to describe the various elements in the `named.conf` file. To reduce the confusion that can arise, this book uses only two terms consistently throughout. Individual configuration lines are called *statements*. Each statement is terminated with a semicolon. Statements are defined within clauses. A clause starts on new line, and all its statements are enclosed within braces (curly brackets) and terminate with a semicolon. The following fragment illustrates this organization:

```
// zone starts a new clause
zone "example.com" {
  // all clause statements are contained within braces
  // type, file, and masters are statements and terminate with a semicolon
   type slave;
   file "slave.example.com";
   masters {10.0.0.1;};
  // the zone clause is terminated with a closing brace
};
```

BIND `named.conf` clauses and statements can seem quite complex, and BIND is pretty picky when it comes to syntax—semicolons, braces, and all that wonderful stuff. There are many `named.conf` layout styles possible, the majority of which are simply designed to minimize syntax errors. The following examples show various layout styles, each of which is handled by BIND:

```
// dense single-line layout style
zone "example.com" {type slave; file "sec.example.com"; masters {10.0.0.1;};};
// multiple-line layout style
zone "example.com" {
   type slave;
   file "slave.example.com";
   masters {10.0.0.1;};
};
```

```
// a slightly confusing hybrid layout style
zone "example.com" {
   type slave;
   file "slave.example.com";
   masters {10.0.0.1;}; };
```

Use the layout style that makes the most sense and that will be the least error prone.

Finally, there is the question of quotes or no quotes with names. In the preceding fragment, `zone "example.com"` could have been written as simply `zone example.com`. The rule is if a name contains spaces, it must be enclosed in quotes; if not, the enclosing quotes are optional. This book mostly uses enclosing quotes with names, but especially with reserved names such as `any` and `none` (or `"any"` and `"none"`) will occasionally omit the enclosing quotes.

Standard Zone Files

The next sections describe the detail configuration of BIND's `named.conf` and, where appropriate, the zone files for each of the DNS types. Unless otherwise noted, the standard zone files defined earlier for `root.servers`, `master.localhost`, and `localhost.rev` are unchanged. Also, unless otherwise noted, the `example.com` zone file first introduced in Chapter 2 remains unchanged. However, for convenience it is reproduced here:

```
; simple zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com. ; base domain-name
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; se = serial number
                        12h        ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        2h         ; min = minimum
                        )
; name server RR for the domain
              IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
              IN      NS      ns2.example.net.
; mail server RRs for the zone (domain)
     3w       IN      MX  10  mail.example.com.
; the second  mail servers is  external to the zone (domain)
              IN      MX  20  mail.example.net.
; domain hosts includes NS and MX records defined above
; plus any others required
ns1           IN      A       192.168.254.2
mail          IN      A       192.168.254.4
joe           IN      A       192.168.254.6
www           IN      A       192.168.254.7
; aliases ftp (ftp server) to an external domain
ftp           IN      CNAME   ftp.example.net.
```

Common Configuration Elements

The `named.conf` files used in the example files have a common core containing statements and clauses, which are either required or advisable. This common core is shown here, and each part briefly described:

```
// Master & Caching Name Server for Example, INC.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
// options clause defining the server-wide properties
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
};
// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in syslog e.g. /var/log/messages
//
logging {
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
```

The file always starts with a gentle reminder that, as with all configuration files, disciplined commenting of all changes is one of the simplest and most powerful diagnostic tools available as well as being plain good sense. The `directory` statement in the example shown is the normal path but serves as a constant reminder of the base used for any relative file name (those that don't start with a `/`) such as zone files. The `version` statement inhibits disclosure of the BIND version number. This is done to prevent advertising that the site is running a version of BIND that may have a known exploit—it makes any attacker's life a tad more difficult. The `logging` clause simply streams all messages into a separate file, rotates the log when it reaches 250K in size, and keeps the last three rotated versions. If a logging clause is not present, all logging is done using `syslogd`.

Master DNS Server

Recall from the description in Chapter 4 that a zone master will supply authoritative data for the zone. There may be one or more zone masters and zero or more zone slaves for any given domain or zone. The term *master* simply means that the zone file will be read from the local filestore, and the name server will respond to requests for zone transfer from slaves if permitted by `named.conf` configuration parameters.

Master Name Server Configuration

The BIND configuration samples that follow provide the following functionality:

- The name server is a master for the zone `example.com`. This characteristic is defined by the `zone "example.com"` clause containing a `type master;` statement.
- The name server provides caching services for all other domains. This characteristic is defined by the combination of the `recursion yes;` statement in the `options` clause and the `zone "."`clause `"."` (the root zone).
- The name server provides recursive query services for resolvers or other name servers acting on behalf of resolvers. This characteristic is defined by the `recursion yes;` statement in the `options` clause.

Here is the BIND `named.conf` file:

```
// Master & Caching Name Server for EXAMPLE.COM.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
// options clause defining the server-wide properties
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
  // configuration unique options statements
  // optional - disables zone transfers except for the slave
  // in the example.net domain
  allow-transfer {192.168.1.2;};
  // optional - BIND default behavior is recursion
  recursion yes;
};
```

```
// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in syslog e.g. /var/log/messages
//
logging {
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// root.servers - required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
// zone clause - master for example.com
zone "example.com" in{
  type master;
  file "master/master.example.com";
  allow-update {none;};
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update {none;};
};
// localhost reverse map
zone "0.0.127IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update {none;};
};
// reverse map for local addresses at example.com
// uses 192.168.254.0 for illustration
zone "254.168.192.IN-ADDR.ARPA" in{
  type master;
  file "192.168.254.rev";
  allow-update {none;};
};
```

The `allow-transfer` statement prohibits any zone transfer except to the defined IP address—in this case it is the IP address of `ns2.example.net` (defined in the sample zone file—see earlier). BIND 9's default behavior is to allow zone transfers from any host that requests one. An alternative strategy is to disable all transfers in the `options` clause and selectively enable them in each `zone` clause as shown in this fragment:

```
....
options {
  ....
  allow-transfer {none;};
  ....
}
....
zone "example.com" in {
  ....
  allow-transfer {192.168.1.2;};
  ....
};
```

Additional zone clauses defining either `type master` or `type slave` may be added as required to create larger configurations.

Slave DNS Server

The functionality of the slave name server was described in Chapter 4. The term *slave* simply defines that a name server will obtain the zone records using zone transfer operations but will answer authoritatively for the zone for as long as it has valid zone data (defined by the `expiry` field of the zone's SOA RR). The term *slave* in no sense implies priority of access. As previously described, slave servers will be accessed, in general, just as frequently as any master name server.

Slave Name Server Configuration

The BIND `named.conf` slave sample configuration provides the following functionality:

- The name server is a slave for the zone `example.com`. This characteristic is defined by the `zone "example.com"` clause containing a `type slave;` statement.
- The name server provides caching services for all other domains. This characteristic is defined by the combination of the `recursion yes;` statement in the `options` clause and the `zone "."`clause `"."` (the root zone).
- The name server provides recursive query services for resolvers or other name servers acting on behalf of resolvers. This characteristic is defined by the `recursion yes;` statement in the `options` clause.

The sample configuration file shows that the slave name server is provided in an external or foreign domain called `example.net` (not `example.com`) by the following fragment from the standard zone file:

```
; the second name server is
; external to this zone (domain).
              IN      NS      ns2.example.net.
```

This type of configuration is normally used for physical diversity. If the example.com site is off-line due to communication or other problems, then example.net, assumed to be at a different physical location, will continue to provide service for the example.com zone or domain. Clearly this is not always practical, and the second name server could have been defined as ns2.example.com and located on the same site. There is nothing wrong with such a configuration other than the risk associated with a physical outage. The named.conf sample file that follows, based on the standard sample file, would be located at ns2.example.net.

```
// Slave & caching Name Server for EXAMPLE.NET.
// provides slave name server support for example com
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
  // configuration unique statements
  // disables all zone transfer requests
  allow-transfer {"none";};
  // optional - BIND default behavior is recursion
  recursion yes;
};
//
// log to /var/log/named/examplenet.log all events from info UP
//  in severity
// defaults to use 3 files in rotation
// failure messages up to this point are in (syslog) /var/log/messages
  logging{
  channel examplenet_log{
  file "/var/log/named/examplenet.log" versions 3 size 250k;
  severity info;
 };
  category default{
  examplenet_log;
 };
};
```

```
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
// assumes this server is also master for example.net
zone "example.net" in{
  type master;
  file "master/master.example.net";
  allow-update {none;};
};
// slave for example.com; see following notes
zone "example.com" in{
  type slave;
  file "slave/slave.example.com";
  masters (192.168.254.2;);
  // allows notify messages only from master
  allow-notify {192.168.254.2;};
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
// reverse map for example.net local IPs
// assumed 192.168.1.0 (see notes)
zone "1.168.192.IN-ADDR.ARPA" IN {
  type slave;
  file "slave.192.168.1.rev";
  masters {192.168.1.1;};
};
```

The `example.com` slave zone statement `file "slave/slave.example.com";` is optional and allows the slave to store the zone records obtained on the last zone transfer. If BIND is reloaded, the current stored zone file—assuming it is still within the TTL time defined by the SOA RR `expiry` field—is used rather than immediately requesting a zone transfer and thus wasting both time and network resources. To create the secondary file initially, just create an empty file with the correct file name; BIND will complain the first time it loads the new file but not thereafter.

The zone `example.com` contains a statement `masters {192.168.254.2;};` which has a single IP address referencing `ns1.example.com`; any number of IP addresses could appear in the list. There may be one or more zone masters. The `allow-notify {192.168.254.2;};` statement disables `NOTIFY` messages from any host except the zone master to minimize possible malicious action.

The reverse map for the local IP addresses at `example.net` (`zone "1.168.192.IN-ADDR.ARPA"`) is defined as a slave for administrative convenience—only one copy of this zone file need be maintained. `IN-ADDR.ARPA` zones provide all the normal zone functionality, including master and slave. This zone could have been defined as a master with a local copy of the reverse-map zone file, which is the more normal, but unnecessary, configuration.

The `named.conf` file shows `ns2.example.net` acting as a zone master for its zone or domain (`example.net`). It could equally well have been a slave for the domain or even contained no zone section or clause for `example.net`.

Caching-only DNS Server

The *caching-only name server* is one that provides caching service to its clients—resolvers or other DNSs acting on behalf of resolvers. When the caching-only name server obtains the answer to a query, it saves the resulting Resource Records (RRs) to local file or in-memory storage and will return this saved result to a subsequent query for the same information until the TTL value of the saved RR expires, at which time it will discard the RR. If the caching server is restarted, the current cache will be discarded.

■ **Note** A DNS cache is not the same as a slave's zone data. Zone data consists of all the zone records obtained through zone transfer operations, and importantly this data is timed out using the values in the zone's SOA RR. A cache contains individual RRs obtained as answers to specific queries and timed out according to the TTL value of the specific RR.

Caching-only Name Server Configuration

The BIND `named.conf` configuration sample provides the following functionality:

- The name server is neither a master nor slave for any domain. There are no `zone` clauses for other than essential zones needed for local operations (`master.localhost` and `localhost.rev`) and to support recursive queries (the root zone).
- The name server provides caching services for all domains. This characteristic is defined by the `recursion yes;` statement in the options clause and the `zone "."` clause (the root zone).
- The name server provides recursive query services for resolvers or other DNSs acting on behalf of resolvers. This characteristic is defined by the `recursion yes;` statement in the `options` clause.

Here is the BIND `named.conf`:

```
// Caching Name Server for Example.COM.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something more
//  b. another change
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
  // configuration-specific option clause statements
  // disables all zone transfer requests
  allow-transfer{"none"};
  // optional - BIND default behavior is recursion
  recursion yes;
};
//
// log to /var/log/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
```

```
// localhost reverse map
zone "0.0.127.IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

This is a caching-only name server and contains no zones (other than localhost) with master or slave types. Previous samples for master and slave server types included caching behavior combined with one or more master or slave zones.

The reverse-map zone has been omitted since it is assumed that an external body, for example, an ISP, is the zone master for `example.com` and is therefore also responsible for the reverse map. The reverse-mapping zone could be added if required for local operational reasons.

Forwarding (a.k.a. Proxy, Client, Remote) DNS Server

The functionality of the forwarding name server was described in Chapter 4 and is used primarily to minimize traffic on congested, slow, or expensive external network connections, for example, a dial-up network.

Forwarding Name Server Configuration

The BIND `named.conf` configuration sample provides the following functionality:

- The name server is neither a master nor slave for any domain. There are no zone clauses for other than essential zones needed for local operations (`master.localhost` and `localhost.rev`).
- The name server provides caching services for all domains. This characteristic is an artifact of BIND's normal behavior. When the results of queries forwarded to an external name server are returned, they are automatically cached.
- The name server does not provide recursive query support. This characteristic is defined by the `recursion no;` statement and by the definition of the `forward only;` statement in the `options` clause.
- The name server forwards all queries to a remote DNS—which must provide recursive query support—from all local resolvers (global forwarding). This characteristic is defined by the `forward` and `forwarders` statements in the `options` clause.

Here is the BIND `named.conf`:

```
// Forwarding & Caching Name Server for Example, INC.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something more
//  b. another change
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  // configuration specfic options statements
  forwarders {10.0.0.1; 10.0.0.2;};
  forward only;
  // disables all zone transfer requests
  allow-transfer{"none"};
  // turn off recursion
  recursion no;
};
// log to /var/log/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in (syslog) /var/log/messages
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

The forwarding name server typically contains no zones (other than localhost) with master or slave types.

The reverse-map zone has been omitted since it assumed that an external body, for example an ISP, is the zone master for the domain and is therefore also responsible for the reverse map. It could be added if required for local operational reasons.

The `forward` statement must be used in conjunction with a `forwarders` statement. The statement `forward only` overrides local recursive query behavior. All queries are forwarded to a recursive name server that will return a complete answer in a single transaction, thus minimizing external network traffic, while local clients see an apparently recursive name server.

Since all queries are forwarded to another name server, the `root.servers` zone (`type hint`) is omitted.

Forwarding can also be done on a per-zone basis, in which case the values defined override the global options. The following example shows its use in a `zone` clause:

```
// BIND named.conf fragment
// use of forward in a zone clause
zone "example.net" in{
  type forward;
  forwarders{10.0.0.3;};
  forward only;
};
```

In the preceding fragment, all queries (indicated by `forward only`) for the domain `example.com` will be forwarded to the host 10.0.0.3, whereas the global `forwarders` statement in the main file uses the hosts 10.0.0.1 and 10.0.0.2. If `forward first` had been used, then the sense would be this: forward to host 10.0.0.3 and if no response is obtained only then use the global forwarders 10.0.0.1 and 10.0.0.2.

Stealth (a.k.a. Split or DMZ) DNS Server

The functionality of the Stealth name server configuration—typically used to provide perimeter security—was described in Chapter 4. Figure 7-1 illustrates the conceptual view of a Stealth (a.k.a. Split or DMZ) DNS server configuration.

The key concept in a Stealth DNS system is that there is a clear line of demarcation between the internal Stealth server(s) and the external or public name server(s). The primary difference in configuration is that Stealth servers will provide a comprehensive set of services to internal users to include caching and recursive queries and would be configured as a typical zone master, slave, or a caching-only server (see earlier), while the public server may provide limited services and would typically be configured as an authoritative-only server (see the section "Authoritative-only DNS Server" later in this chapter).

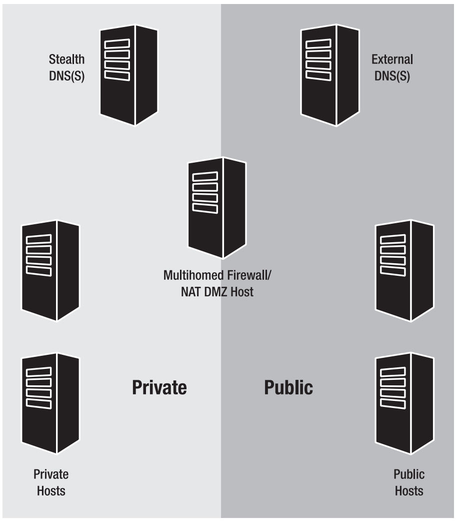


Figure 7-1. *Split/Stealth server configuration*

There are two critical points in a Stealth configuration:

1. The zone file for the Stealth server may contain both public and private hosts, whereas the public server's zone file will contain only public or publicly visible hosts.
2. To preserve the stealth nature, it is vital that the public `named.conf` file does not include statements such as `master`, `allow-notify`, `allow-transfer`, etc. with references to the IP(s) of the Stealth server. If the Stealth server's IP were to appear in the public name server and its file system were to be compromised, the attacker could gain knowledge about the organization—could penetrated the *veil of privacy*—by simply inspecting the `named.conf` file.

Stealth Configuration

The samples that follow depict `named.conf` files for the public and private name servers used in a Stealth or Split configuration.

Stealth (Private) Configuration Files

Here is the BIND `named.conf` file used on the private or Stealth name servers:

```
// Master & Caching Name Server for Example, INC. STEALTH SIDE
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
```

```
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
  // configuration-specfic options statements
  // optional - BIND default behavior is recursion
  recursion yes;
};
//
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in syslog e.g. /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// required zone for recursive queries
// transactions will pass through a classic firewall
zone "." {
  type hint;
  file "root.servers";
};
// zone clause - master for example.com
zone "example.com" in{
  type master;
  file "master/master.example.com.internal";
  allow-update{none;};
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update {none;};
};
```

```
// localhost reverse map
zone "0.0.127IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
// reverse map for local address at example.com
// uses 192.168.254.0 for illustration
zone "254.168.192.IN-ADDR.ARPA" in{
  type master;
  file "192.168.254.rev";
};
```

The zone file `master.example.com.internal` will contain both the public and internal hosts—the standard sample zone file has been modified to add some internal or private hosts.

```
; simple zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN example.com. ; base domain-name
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns3.example.com. hostmaster.example.com. (
                        2003080800 ; se = serial number
                        12h         ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        2h         ; min = minimum
                        )
; name server RRs for the domain
              IN      NS      ns3.example.com.
; mail server RRs for the zone (domain)
     3w       IN      MX  10  mail.example.com.
; the second  mail servers is  external to the zone (domain)
              IN      MX  20  mail.example.net.
; domain hosts includes NS and MX records defined previously
; plus any others required
mail          IN      A       192.168.254.4
joe           IN      A       192.168.254.6
www           IN      A       192.168.254.7
; aliases ftp (ftp server) to an external domain
ftp           IN      CNAME   ftp.example.net.
; private hosts and services
ns3           IN      A       192.168.254.10
accounts   IN      A       192.168.254.11
hr             IN      A       192.168.254.12
....
last           IN      A       192.168.254.233
```

The Stealth side zone file uses a nonpublicly visible name server, `ns3.example.com`, to provide local DNS services. A single name server is used in this configuration file, but two or more could be used depending on the size of the organization and the requirement for resilience. This file does not reference the public name servers `ns1.example.com` and `ns2.example.com`, which are defined in the zone file used by the public server (as discussed in the next section) to minimize unnecessary traffic across the firewall. The mail servers referenced are the same as those used in the public server to avoid having to synchronize mail from multiple servers, and it is assumed all access to the mail servers will be via a firewall.

Public Configuration Files

The BIND `named.conf` file for the public name server is the same as that defined for an authoritative-only name server (discussed in the next section). The zone file used will be the standard sample zone file that contains only public hosts and services.

BIND provides a powerful `view` clause that may be used to provide similar functionality using a single server. The `view` clause allows different users or clients to gain access to different services. When a `view` clause is used, the Stealth and public zone files are hosted on the same server. If this host's file system is compromised for any reason, then simple inspection of these files will reveal information about the organization. Unless the file system can be guaranteed against compromise, the `view` clause cannot provide a Stealth DNS solution in a highly secure environment. The descriptions that follow, however, extend this topic further and present configurations in which the real power of the `view` clause can be used.

Authoritative-only DNS Server

An authoritative-only name server will only provide authoritative answers to queries for zones or domains for which it is either a master or a slave. It will not provide either caching or recursive query support. If security is not the primary requirement, then the `view` clause may be used to provide authoritative-only services to external users and more comprehensive services to internal users as described in the section "Stealth (Private) Configuration Files." An example configuration of this style of operation using a `view` clause is also shown in the "View-based Authoritative-only DNS Server" section.

Authoritative-only Name Server Configuration

The BIND `named.conf` configuration sample provides the following functionality:

- The name server is the zone master for `example.com`. This characteristic is defined by the inclusion of the `zone "example.com"` clause.
- The name server does not provide caching services for any other domains. This is defined by the `recursion no;` statement in the `options` clause and the absence of the `zone "."` clause (root zone).
- The name server does not provide recursive query services for resolvers or other DNSs acting on behalf of resolvers. It supports *only iterative* queries. This characteristic is defined by the `recursion no;` statement in the `options` clause.

- The name server is optimized for maximum performance. Any optional but performance-affecting characteristics should be inhibited. In the following sample, the `allow-transfer {"none";};` statement is shown for this reason as well as reasons of security.

Here is the BIND `named.conf`:

```
// Authoritative only Name Server for Example, INC.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  // configuration specfic options statements
  recursion no;
  // disables all zone transfer requests
  // for performance as well as security reasons
  allow-transfer{"none"};
};
//
// log to /var/log/zytrax-named all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
zone "example.com" in{
  type master;
  file "master/master.example.com";
  allow-update{none;};
};
```

```
// reverse map for local address at example.com
// uses 192.168.254.0 for illustration
zone "254.168.192.IN-ADDR.ARPA" in{
  type master;
  file "192.168.254.rev";
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.IN-ADDR.ARPA" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

The authoritative-only server does not provide services for any domain except those for which it is either a master or a slave; as a consequence the `root.servers` zone file is not present (`zone "."`). The `recursion no;` statement inhibits recursive behavior; the name server will return a list of root-servers (a referral) if it receives a query for a domain or zone for which it is neither master nor slave.

The reverse-mapping zone (`zone "254.168.192.IN-ADDR.ARPA"`) would typically not be present on a performance-oriented server, but the prevalence of reverse lookups by mail systems probably means that it will be present in a practical configuration.

BIND provides three statements to control caching behavior, `max-cache-size` and `max-cache-ttl`, neither of which will have any effect on performance in the preceding case; and `allow-recursion`, which allows a list of hosts that are permitted to use recursion—all others are not (a kind of poor man's `view` clause).

View-based Authoritative-only DNS Server

The functionality of the authoritative-only name server was described in Chapter 4. If high security is not the primary requirement, then the `view` clause may be used to provide authoritative-only services to external users and more comprehensive services, including caching, to internal clients.

View-based Authoritative-only Name Server Configuration

The BIND `named.conf` configuration sample provides the following functionality:

- The name server is the zone master for `example.com`. This characteristic is defined by the inclusion of the `zone "example.com"` clause in both `view` clauses but each referencing a different zone file.

- The name server does not provide caching services for any external users. This is defined by the `recursion no;` statement in the `view "badguys"` clause and the absence of the `zone "."` clause (root zone) within the same `view` clause.
- The name server does not provide recursive query services for any external resolvers or other DNSs acting on behalf of resolvers. It supports only iterative queries. This characteristic is defined by the `recursion no;` statement in the `view "badguys"` clause.
- The name server provides caching services for internal users. This is defined by the `recursion yes;` statement in the `view "goodguys"` clause and the presence of the `zone "."` clause (root zone) within the same `view` clause.
- The name server provides *recursive* query services for internal users. This is defined by the `recursion yes;` statement in the `view "badguys"` clause.

Here is the BIND `named.conf`:

```
// View-based Authoritative Name Server for EXAMPLE, INC.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
//
// global options
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
};
//
// log to /var/log/example.com all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
```

```
// provide recursive queries and caching for internal users
view "goodguys" {
  match-clients { 192.168.254.0/24; }; // the example.com network
  recursion yes;
  // required zone for recursive queries
  zone "." {
   type hint;
   file "root.servers";
  };
  zone "example.com" {
   type master;
   // private zone files including local hosts
   file "view/master.example.com.internal";
   allow-update{none;};
  };
  // required local host domain
  zone "localhost" in{
   type master;
   file "master.localhost";
   allow-update{none;};
  };
  // localhost reverse map
  zone "0.0.127.IN-ADDR.ARPA" in{
   type master;
   file "localhost.rev";
   allow-update{none;};
  };
  // reverse map for local address at example.com
  // uses 192.168.254.0 for illustration
  zone "254.168.192.IN-ADDR.ARPA" in{
  type master;
  file "view/192.168.254.rev.internal";
   allow-update{none;};
};

}; // end view

// external hosts view
view "badguys" {
  match-clients {"any"; }; // all other hosts
  // recursion not supported
  recursion no;
 zone "example.com" {
   type master;
   // only public hosts
   file "view/master.example.com.external";
   allow-update{none;};
  };
  // reverse map for local address at example.com
  // uses 192.168.254.0 for illustration
  zone "254.168.192.IN-ADDR.ARPA" in{
    type master;
    file "view/192.168.254.rev.external";
   allow-update{none;};
  };
}; // end view
```

The principle when using view clauses is that each view contains all the zone clauses required within that view and defines how they will behave within that view. The zone example.com appears in each view clause but references a different zone file (in the file statement). The zone file master.example.com.internal will contain both internal and public hosts, whereas the zone file master.example.com.external will have only the publicly visible hosts. The same principle applies to the reverse-mapping files: 192.168.254.rev.internal will contain reverse mapping for all the internal and public hosts, whereas 192.168.254.rev.external will only reverse map externally visible or public hosts.

The view "goodguys" clause provides recursive support and consequentially requires a root.servers zone file (zone "."). The view "badguys" clause does not require this zone, since it does not support recursive queries, and it is not present. Similarly, there is no need for the master.localhost and localhost.rev zone files in the view "badguys" clause, (all local requests are answered by the view "goodguys" clause) and they are again not present.

The order in which the view statements are defined is very important. In the view "goodguys" clause, the line match-clients { 192.168.0.0/24; } is used to match the 256 IP addresses from 192.168.254.0 to 192.168.254.255 (the IP prefix format, or slash notation, for defining an IP address range is described in Chapter 3 in the sidebar entitled "IPv4 Addresses and CIDR"). Only when this match fails does the process fall through to the view "badguys". In the view "badguys", match-clients {"any"; }; is interpreted to be "any not matched previously." If the order of view clauses were reversed, all IP addresses, including the internal IP addresses (192.168.254.0/24), would match "any" and hence no additional services would be provided to internal clients.

Security and the view Section

Both this chapter and Chapter 4 have suggested that there is a weakness in use of the view clause if the name server's file system is compromised. This is in no sense a reflection on BIND's innate security, quite the contrary. In order to compromise the file system, an attack does not depend upon BIND or BIND's integrity, but rather can focus on any software running in the host with the sole objective of gaining some form of root privilege or even limited (read-only) access to well-known locations. If the zone files master.example.com.internal and 192.168.254.rev.internal could be read, then all the information about the internal organization of the zone could be discovered irrespective of all BIND's attempts to stop it.

However, careful inspection of the named.conf file earlier indicates that it contains relatively innocuous data, which would be of very little use to a hacker and indeed the most revealing data, namely the line match-clients { 192.168.254.0/24; };, may be available via a simple whois enquiry!

This characteristic of the view clause means that it can be used irrespective of the state of the underlying file system where it will not expose private information. The view clause can be used in a Stealth configuration to provide access from the internal network as illustrated in Figure 7-2.

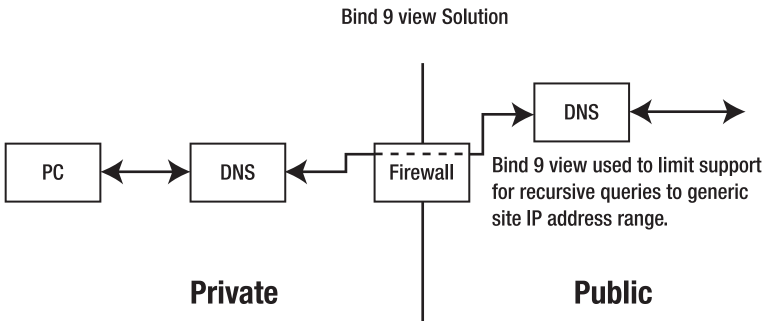

Bind 9 view Solution
PC
DNS
Firewall
DNS
Bind 9 view used to limit support for recursive queries to generic site IP address range.
Private
Public

Figure 7-2. *Use of BIND's* `view` *section in a Stealth configuration*

This `named.conf` sample file on the public side of this configuration provides the following services:

- The name server does not provide caching services for any external users. This characteristic is defined in the `view "badguys"` clause by the `recursion no;` statement in the `options` clause and the lack of a `zone "."` (root zone).
- The name server does not provide recursive query services for any external resolvers or other DNSs acting on behalf of resolvers. It supports only iterative queries. This characteristic is defined in the `view "badguys"` clause by the `recursion no;` statement in the `options` clause.
- The name server provides caching services for internal users. This characteristic is defined in the `view "goodguys"` clause by the `recursion yes;` statement in the `options` clause and the presence of the `zone "."` (root zone).
- The name server provides recursive query services for internal users. This characteristic is defined in the `view "goodguys"` clause by the `recursion yes;` statement in the `options` clause.

The BIND `named.conf` file for this configuration is shown here:

```
// View based Authoritative Name Server for EXAMPLE.COM.
// Recommended that you always maintain a change log in this file as shown here
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something again
//  b. another change
//
// global options
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
};
//
// log to /var/log/example.com all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// provide recursive queries and caching for our internal users
view "goodguys" {
  match-clients { 192.168.254.0/24; }; // example.com's network
  recursion yes;
  // required zone for recursive queries
  zone "." {
   type hint;
   file "root.servers";
  };
}; // end view

// external hosts view
view "badguys" {
  match-clients {"any"; }; // all other hosts
  // recursion not supported
  recursion no;
 zone "example.com" {
   type master;
   // only public hosts
   file "view/master.example.com.external";
  };
  // reverse map for local address at example.com
  // uses 192.168.254.0 for illustration
  zone "254.168.192.IN-ADDR.ARPA" in{
    type master;
    file "view/192.168.254.rev.external";
  };

}; // end view
```

To invoke the service from the Stealth side of the configuration, the `zone "."` (defined as `type hint` in the sample file earlier) should be replaced with the following fragment, which forwards all requests for domains other than `example.com` to `ns1.example.com`—one of the public name servers.

```
// BIND named.conf fragment
// forwards requests for domains other than example.com
// to the public name server ns1.example.com = 192.168.254.2
zone "." in{
  type forward;
  forwarders{192.168.254.2;};
  forward only;
};
```

There are no files involved in this configuration that will divulge additional information that is not already publicly available or that could be found out without access to any of the systems or hosts involved in the configuration. The most revealing information is contained in the line `match-clients { 192.168.254.0/24;`, the IPv4 address range used by the entire configuration, and as previously noted this could probably be obtain with a `whois` enquiry. To further tighten security, communication between the Stealth server and the public name server could use a unique port and thus allow port 53 access to be entirely blocked in the firewall configuration. The point about this configuration is to illustrate the power of the `view` clause and the kind of applications in which it can be used irrespective of the environment in which it is running.

Summary

This chapter introduced a number of configuration samples that reflect widely used DNS types while bearing in mind that name servers are normally multifunctioned. The objective of the chapter is to acquaint you with the configuration of a set of building blocks, DNS types, from which more complex configurations can be constructed. The text described BIND 9's powerful new `view` clause, together with its use in various Stealth configurations. This new clause provides many opportunities to reduce physical configurations in secure perimeter defenses, but careful attention to system design and especially `named.conf` file contents may be required to maximize its potential.

Chapter 8 presents some advanced DNS configurations including delegation of subdomains, load balancing, and resilience, among many others.

CHAPTER 8

Common DNS Tasks

This chapter describes a number of common configurations when working with zone files and in some cases with BIND. These solutions are presented to assist you to quickly implement some commonly used features, to recover from errors, and to illustrate some of the more subtle uses of the DNS. The following topics are covered:

- *How to delegate a subdomain*: This configuration allows the domain name owner to pass the responsibility to a subdomain owner, which may be another party or another part of the organization, who will be entirely responsible for the zone files describing the subdomain.
- *How to delegate a virtual subdomain*: This configuration uses a single zone file to provide subdomain addressing, for instance, `www.us.example.com` or `www.uk.example.com`.
- *How to configure fail-over mail servers*: The configuration allows backup mail servers to be provided to support a domain.
- *How to reverse-map subnets*: This configuration allows the delegation of reverse mapping to subnets of typically less than 256 IPv4 addresses.
- *How to load balance with DNS*: The configurations describe various ways in which load balancing may be implemented using DNS features. The BIND statements that control the order in which addresses are returned are also covered.
- *How to define an SPF record*: The Sender Policy Framework (SPF) is an antispam measure that allows an e-mail server to verify that the SMTP source is valid for the sending e-mail address. SPF records are currently implemented by Microsoft, Google, and AOL to name but three of the many hundreds of thousands of users.
- *How to support* `http://example.com`: The configuration allows both the URL `www.example.com` and `example.com` to directly address a web or other service. The required changes to the Apache server are also covered.
- *How to fix an out-of-sequence SOA serial number*: The process used to fix various SOA serial number errors is covered.
- *How to use DNS wildcards*: The DNS RRs support the use of a wildcard (`*`). The section on wildcards illustrates the use of this error-prone feature.

The examples shown use a number of BIND's `named.conf` statements, which are described in Chapter 12, and standard Resource Records, which are defined in Chapter 13. If you are running name server software other than BIND, the zone files will remain the same, but the configuration statements may differ.

In the next section, the process of delegation of a subdomain, `us.example.com`, is described to illustrate the general principle of delegation within an owner's domain name space. The domain owner can delegate everything to the *right* of the domain name in any way that makes sense—or for that matter that doesn't make sense!

Delegate a Subdomain (Subzone)

This solution configures a zone to *fully delegate* the responsibility for a subdomain to another name server. This is not the only possible method of defining subdomains—another solution involves configuring what this book calls a *virtual* or *pseudo* subdomain, which uses a single zone file to provide subdomain addressing structures. Assume the following addressing structure is required for the subdomain:

- *Zone (domain) name*: `example.com`
- *Domain host name*: `bill.example.com`
- *Subdomain name*: `us.example.com`
- *Subdomain host or service name*: `ftp.us.example.com`

To ease the zone administration load, this solution assumes the responsibility for the subdomain will be fully delegated to the `us.example.com` administrator who will be responsible for the subdomain zone files and their supporting name servers. The zone administrators of the corporate domain for `example.com` want nothing further to do with `us.example.com` other than it has generously agreed to act as the slave DNS for the subdomain name servers. When dealing with subdomains, it is important to remember that as far as the Internet registration authorities and the TLD servers are concerned, subdomains do not exist. All queries for anything that ends with `example.com` will be *referred* to the name servers for the `example.com` zone or domain. In turn, these name servers are responsible for referring the query to the subdomain name servers. For the want of any better terminology, the name servers for `example.com` are called the *domain name servers* and are visible to the gTLD `.com` servers; the name servers for `us.example.com` are called the *subdomain name servers* and are visible only to the *domain name servers* (they are invisible to the gTLD servers).

■ **Note** The term *subzone* was originally defined in RFC 1034 to describe what is today most commonly called a *subdomain*. This book uses the term *subdomain* throughout.

Domain Name Server Configuration

The following is a fragment from BIND's `named.conf` file controlling the `example.com` domain name servers:

```
// named.conf file fragment

zone "example.com" in{
                type master;
                file "master.example.com";
};
// optional - example.com acts as the slave (secondary) for the delegated subdomain
zone "us.example.com" IN {
                type slave;
                file "slave.us.example.com";
                masters {10.10.0.24;};
};
```

The optional definition of a slave (Secondary) name server for our delegated `us.example.com` subdomain is good practice but not essential. The subdomain can use any suitable name server. The zone file `master.example.com` will contain the domain configuration supporting two name servers for both the domain and the subdomain. The following zone file fragment shows this configuration:

```
; zone fragment for 'zone name' example.com
$TTL 2d     ; default TTL is 2 days
$ORIGIN example.com.
@        .  IN      SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               12h         ; refresh =  12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
; main domain name servers
              IN      NS     ns1.example.com.
              IN      NS     ns2.example.com.
; mail domain mail servers
              IN      MX      mail.example.com.
; A records for preceding name servers
ns1           IN      A      192.168.0.3
ns2           IN      A      192.168.0.4
; A record for preceding mail server
mail          IN      A      192.168.0.5
....
; subdomain definitions in the same zone file
; $ORIGIN directive simplifies and clarifies definitions
$ORIGIN us.example.com. ; all subsequent RRs use this ORIGIN
; two name servers for the subdomain
@             IN      NS     ns3.us.example.com.
; the preceding record could have been written without the $ORIGIN as
; us.example.com. IN NS ns3.us.example.com.
; or @         IN    NS ns3
; the second name server points back to preceding ns1
                   IN    NS    ns1.example.com.
; A records for name server ns3 required - the glue record
ns3           IN      A      10.10.0.24 ; glue record
; the preceding record could have been written as
; ns3.us.example.com. A 10.10.0.24 if it's less confusing
```

The preceding fragment makes the assumptions that the domain name server `ns1.example.com` will act as a slave for the `us.example.com` subdomain. If this is not the case, any other name server can be defined the same way; but if this second name server also lies in the `us.example.com` domain, then it will require an A RR. The A RR for `ns3.example.com` for the preceding subdomain is the so-called *glue* record (see the "Glue Records in DNS" sidebar). It is necessary to allow a DNS query for the subdomain to return a referral containing both the name of the name server and its IP address. IP addresses are always defined using an A RR (or an AAAA RR if IPv6).

GLUE RECORDS IN DNS

Strictly speaking, glue records (the IP address of the name server defined using an A or AAAA RR) are only required for every name server lying *within* the domain or zone for which it is a name server. The query response—the referral—*must* provide both the name and the IP address of the name servers that lie within the domain being queried. In practice, the Top-Level Domain (TLD) servers provide the IP address for every Second-Level Domain (SLD) name server, whether in the domain or not, in order to minimize the number of query transactions. When a query to a Generic Top-Level Domain (gTLD) is issued, this name server provides the glue records for all the SLD domain's name servers. These glue records were defined and captured when the domain was registered. In the preceding configuration, the domain name server is acting in this role and must supply the IP addresses of the name servers in response to subdomain queries. To satisfy this requirement, the A RR for the name server (`ns3.us.example.com`) is a glue record and must be present. The reason a glue record *must* exist for servers *within* the domain, but is required only for performance reasons for those in a foreign or external domain, can be illustrated by looking at what would happen if the glue record were not present. If we assume the query to the gTLD server for `example.com` returned the name but not the IP address of `ns1.example.com`, then a further query would be required for the A record of `ns1.example.com`; but since the IP of the SLD name server is not yet known, it must requery the gTLD server, which answers again with the name but not the IP . . . and so on ad infinitum. Name servers for a domain (for instance, `example.com`) that lie in another domain (for instance, `ns1.example.net`) only *need* the name, since a normal query for the A RR of `ns3.example.net` will return the required IP. As noted earlier, to increase performance the IP addresses of all name servers for a domain, whether the name servers lie in the queried domain or not, are always returned by root and TLD name servers.

Subdomain Name Server Configuration

The BIND `named.conf` file controlling the subdomain name servers will contain statements similar to the following fragment:

```
// named.conf file fragment for the subdomain us.example.com

zone "us.example.com" in{
                type master;
                file "master.us.example.com";
};
```

The file `master.us.example.com` will contain the subdomain (`us.example.com`) configuration and use the two name servers that were defined in the preceding domain fragment. Here is a fragment of the subdomain zone file:

```
; zone file for subdomain us.example.com
$TTL 2d ; zone default of 2 days
$ORIGIN us.example.com.
         IN     SOA   ns3.us.example.com. hostmaster.us.example.com. (
         2003080800 ; serial number
         2h         ; refresh =  2 hours
         15m        ; update retry = 15 minutes
         3w12h      ; expiry = 3 weeks + 12 hours
         2h20m      ; minimum = 2 hours + 20 minutes
         )
; subdomain name servers
                  IN      NS     ns3.us.example.com.
                  IN      NS     ns1.example.com. ; see following notes
; subdomain mail server
                  IN      MX 10  mail.us.example.com.
; preceding record could have been written as
;                 IN      MX 10  mail
; A records for preceding name servers
ns3               IN      A      10.10.0.24
ns1.example.com.  IN      A      192.168.0.3 ; 'glue' record
; A record for preceding mail server
mail              IN      A      10.10.0.25
; next record defines our ftp server
ftp               IN      A      10.10.0.28
; the preceding record could have been written as
; ftp.us.example.com. A 10.10.0.24 if it's less confusing
....
; other subdomain records
....
```

The preceding fragment makes the assumption that `ns1.example.com` will act as a slave server for the `us.example.com` subdomain. If this is not the case, other name servers could be defined in a similar manner. The A record for `ns1.example.com` is a so-called glue record and is not strictly necessary because it is already available from a previous query. This point is worth emphasizing further since it illustrates the nature of the DNS hierarchy. To make any query for the subdomain `us.example.com`, the `example.com` domain *must have been queried first.* Since `ns1.example.com` is one of the name servers for `example.com`, its IP address is already known to the DNS that issues the subdomain query. If the second name server for the subdomain lies in an external or foreign domain, there is no need for the glue record. Its inclusion, however, will speed up query response by removing the need for a further query to the root and gTLD servers if the external domain is not present in the cache. An external name server definition would use a standard A RR as shown in the following fragment:

```
; zone fragment for subdomain  us.example.com
;  second name server is external domain
                             IN      NS     ns1.example.net.
; A records for external name server
ns1.example.net.  IN      A      172.17.0.24
```

The FTP service host, and any others required, are only defined in the subdomain zone file and are not visible in the domain name-server zone file.

Virtual Subdomains

This solution defines what this book calls a *virtual* or *pseudo* subdomain in which the domain and the subdomain definitions appear in the same zone file. Subdomains may also be *fully delegated*, and this is the subject of a previous solution. The advantage of this configuration is that unlike a fully delegated subdomain, no additional name servers are required while still creating the subdomain style addressing structure. The disadvantage is that all changes to both the domain and the subdomain will require reloading of the main zone file. The addressing structure required is assumed to be the following:

- *Zone (domain) name*: `example.com`
- *Domain host name*: `bill.example.com`
- *Subdomain name*: `us.example.com`
- *Subdomain host name*: `ftp.us.example.com`

This solution assumes that for operational reasons the owner has decided to maintain all the information for `example.com` and `us.example.com` in a single zone file.

Domain Name Server Configuration

The BIND `named.conf` file will contain statements similar to the following fragment defining the zone `example.com` as normal:

```
// named.conf file fragment

zone "example.com" in{
                type master;
                file "master.example.com";
};
```

The file `master.example.com` will contain the domain and subdomain configuration and support two name servers.

```
; zone fragment for example.com
$TTL 2d ; zone TTL default = 2 days
$ORIGIN example.com.
@              IN      SOA   ns1.example.com. root.example.com. (
               2003080800 ; serial number
               2h         ; refresh =  2 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
; main domain name servers
              IN      NS     ns1.example.com.
              IN      NS     ns2.example.com.
; mail servers for main domain
              IN      MX 10  mail.example.com.
; A records for preceding name servers
ns1           IN      A      192.168.0.3
ns2           IN      A      192.168.0.4
; A record for preceding mail servers
mail          IN      A      192.168.0.5
; other domain-level hosts and services
bill          IN      A      192.168.0.6
....
; subdomain definitions
$ORIGIN us.example.com.
              IN      MX 10  mail
; preceding record could have been written as
; us.example.com.   IN  MX 10 mail.us.example.com.
; A record for subdomain mail server
mail          IN      A      10.10.0.28
; the preceding record could have been written as
; mail.us.example.com. A 10.10.0.28 if it's less confusing
ftp           IN      A      10.10.0.29
; the preceding record could have been written as
; ftp.us.example.com. A 10.10.0.29 if it's less confusing
....
; other subdomain definitions as required
$ORIGIN uk.example.com.
....
```

Additional subdomains could be defined in the same file using the same strategy. For administrative convenience, the standard zone file `$INCLUDE` directive may be used to include the subdomain RRs as demonstrated in the following fragment:

```
; fragment from zone file showing use of $INCLUDE
....
; other domain-level hosts and services
bill          IN      A      192.168.0.5
....
; subdomain definitions
$INCLUDE sub.us.example.com
; other subdomain definitions as required
```

This solution illustrates that subdomain addressing can be easily accomplished in a single zone file at the possible cost of administrative convenience. This structure, as well as being simpler than a *fully delegated* subdomain, does not require any additional name servers.

Configure Mail Servers Fail-Over

This solution is provided for the sake of completeness and uses material already covered in Chapter 2. It configures a DNS server to provide fail-over or alternate mail service when the primary mail service is off-line or not accessible for a period of time. It involves use of the *preference* field of the MX RRs (see Chapter 13) as shown in the following fragment:

```
; zone file fragment
      IN  MX  10 mail.example.com.
      IN  MX  20 mail.example.net.
....
mail  IN  A      192.168.0.4
....
```

If the most preferred mail server, the one with the lowest number (10), which in the preceding fragment is `mail.example.com`, is not available, mail will be sent to the second most preferred server, the one with the next highest number (20), which in the preceding fragment is `mail.example.net`. The secondary mail server (`mail.example.net`), which would ideally be located at a separate geographic location, would typically be configured as a simple relay (or forwarding) mail server with a very long retry time, in which case it will accept the mail and try and relay it to the proper destination (`mail.example.com`) over the next six weeks or whatever you configure the retry time to be.

Delegate Reverse Subnet Maps

This solution describes how to delegate reverse mapping for subnets. Delegation of reverse subnet maps may be used by ISPs or other service providers as a means to enable a user of a static IP range, delegated from the service provider, to be responsible for their own reverse-mapping zone files. In the example shown, a subnet is defined to be less than 256 IPv4 addresses though the solution could be used for any part of an IPv4 address range. Normal reverse mapping is described in Chapter 3 and in our example case is assumed to reverse map down to the third element of an IPv4 address; for instance, assume an IPv4 address of 192.168.199.15, then normal reverse mapping will typically cover the 192.168.199 part, which is then reversed and placed under the domain `IN-ADDR.ARPA`, giving `199.168.192.IN-ADDR.ARPA`. The resulting reverse map will contain the hosts from 192.168.199.0 to 192.168.199.255. We now assume that subnets of 64 addresses are assigned to four separate users (192.168.199.0/26, 192.168.199.64/26, 192.168.199.128/26, 192.168.199.192/26) and to minimize work the assignee wishes to delegate responsibility for reverse mapping to the subnet users (the assignors). The reverse map has been delegated once to the assignee of 192.168.199.0 and cannot therefore be delegated again. Our assignee must use a special technique defined in RFC 2317. The technique involves creating additional space in the reverse-map address hierarchy. Both the assignee and the assignor (end user) are required to implement the technique in their zone files; examples of both are shown in the upcoming text.

Assignee Zone File

The following fragment shows the 192.168.199.64/26 subnet as a fragment of a reverse-map zone file located at the assignee (using the `example.net` domain) of the subnet:

```
; zone file fragment for example.net
$TTL 2d ; zone default TTL = 2 days
$ORIGIN 199.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.example.net. hostmaster.example.net. (
                              2003080800 ; serial number
                              2h         ; refresh
                              15m        ; update retry
                              2w         ; expiry
                              3h         ; minimum
                              )
              IN  NS      ns1.example.net.
              IN  NS      ns2.example.net.
; definition of other IP address 0 - 63
....
; definition of our target 192.168.199.64/26 subnet
; name servers for subnet reverse map
64/26         IN  NS  ns1.example.com.
64/26         IN  NS  ns2.example.com.
; the preceding could have been written as
; 64/26.199.168.192.IN-ARDDR.ARPA.  IN NS ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 127 since they are the subnets multicast
; and broadcast addresses not hosts/nodes
65            IN  CNAME   65.64/26.199.168.192.IN_ADDR.ARPA. ;qualified
66            IN  CNAME   66.64/26 ;unqualified name
67            IN  CNAME   67.64/26
....
125           IN  CNAME   125.64/26
126           IN  CNAME   126.64/26
; end of 192.168.199.64/26 subnet
.....
; other subnet definitions
```

The method works by forcing the CNAME lookup to use the name servers defined for the subnet; that is, the address 65 will find the `CNAME 65.64/26.199.168.192.IN-ADDR.ARPA.`, which is *resolved* to the name servers `ns1.example.com.` and `ns2.example.com.`, both of which in this case are located at the assignor (end user). The `64/26` name, which makes the additional name space look like a IP prefix or slash notation address, is an artificial, but legitimate, way of constructing the additional space to allow delegation. The `/` (slash) relies on a liberal interpretation of the rules for a name or *label* (RFC 2181), but it could be replaced with `-` (dash)—for instance, `64-26`, if that makes you more comfortable. Any number of subnets of variable size can be assigned in this manner; that is, the subnet following the one defined previously could be `128/27` (32 IP addresses) or `128/28` (16 addresses) or `128/25` (128 IP addresses). No changes are required to the BIND configuration to support this reverse map.

Assignor (End-user) Zone File

The zone file for the reverse map (`ns1.example.com` in this example) is a conventional reverse map and looks like this:

```
$TTL 2d ; zone default = 2 days
$ORIGIN 64/26.199.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.example.com. hostmaster.example.com. (
                             2003080800 ; serial number
                             2h         ; refresh
                             15m        ; update retry
                             2w         ; expiry
                             3h         ; minimum
                             )
              IN  NS      ns1.example.com.
              IN  NS      ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 127 since they are the subnets multicast
; and broadcast addresses not hosts/nodes
65            IN  PTR   fred.example.com. ;qualified
66            IN  PTR   joe.example.com.
67            IN  PTR   bill.example.com.
....
125           IN  PTR   web.example.com.
126           IN  PTR   ftp.example.com.
; end of 192.168.23.64/26 subnet
```

Finally, the reverse-map `zone` clause in the `named.conf` file needs to be changed to reflect the revised zone name. The following example shows the reverse-map zone clause fragment:

```
// named.conf fragment at example.com
// revised reverse-map zone name
zone "64/26.199.168.192.IN-ADDR.ARPA" in{
               type master;
               file "192.168.23.rev";
};
```

■**Note** The technique used in the preceding method is credited to Glen A. Herrmannsfeldt, who is obviously a very creative person and one might conjecture had problems persuading his ISP to delegate reverse-mapping responsibility.

DNS Load Balancing

These solutions use the DNS to configure various forms of load balancing. In this context load balancing is defined as the ability to use standard DNS services to share the load between two or more servers providing the same or similar services. The section covers the following topics:

- Balancing mail
- Balancing other services (for instance, web or FTP)
- Balancing services using the SRV RR
- Controlling the order of RRs

This section ends with a brief discussion of the effectiveness of DNS-based load-balancing strategies.

Balancing Mail

Mail is unique in that two possible strategies may be used. The following fragment shows use of multiple MX records with equal-preference values:

```
; zone file fragment
        IN  MX  10  mail.example.com.
        IN  MX  10  mail1.example.com.
        IN  MX  10  mail2.example.com.
....
mail    IN  A       192.168.0.4
mail1   IN  A       192.168.0.5
mail2   IN  A       192.168.0.6
```

The name sever will deliver the MX RRs in the order defined by the `rrset-order` statement (defined later in this section and fully in Chapter 12) and which defaults to round-robin (or cyclic) order. The requesting SMTP server will then apply its algorithm to select one from the equal-preference list that *may* work *against* the BIND `rrset-order` statement. Currently sendmail (8.3.13), Exim (4.44), and Postfix (2.1) all have documented references to indicate they use a random algorithm for records of equal preference; indeed, Postfix allows control over the behavior using the `smtp_randomize_addresses` parameter (default is yes). In this case, the randomizing algorithm may select the very IP that BIND's `rrset-order` algorithm positioned, say, last in the returned order. Documentation for qmail, courier-mta, and Microsoft (Exchange and IIS SMTP) does not describe what these packages do with equal-preference MX values. An alternative approach is to use multiple A records with the same name and different IP addresses as shown in this fragment:

```
; zone file fragment
           IN  MX  10  mail.example.com.
....
mail       IN  A       192.168.0.4
           IN  A       192.168.0.5
           IN  A       192.168.0.6
```

The name server will deliver the A RRs in the order defined by the `rrset-order` statement in BIND's `named.conf` file. In order to satisfy reverse lookup requests used by most mail servers for simple authentication, all the IP addresses listed must be reverse mapped to `mail.example.com` as shown in the following fragment:

```
; reverse-map file fragment
; for 0.168.192.IN-ADDR.ARPA
....
4           PTR      mail.example.com.
5           PTR      mail.example.com.
6           PTR      mail.example.com.
```

The net effect of the two methods is the same. In the case of equal-preference MX records, the control of load lies with the SMTP server's algorithm. In the case of multiple A RRs, control lies with the name server, which in the case of BIND provides the `rrset-order` statement to select the order of A RRs (RRsets) as well as other RRsets. In both the preceding cases, each mail server must be capable of either synchronizing mailbox delivery or all but one of the servers must be mail relays or forwarders.

Balancing Other Services

This section illustrates load balancing with web and FTP services, but the same principle applies to any service. In this case the load-balancing solution uses multiple A RRs as shown in the following fragment:

```
; example.com zone file fragment
....
ftp   IN  A   192.168.0.4
ftp   IN  A   192.168.0.5
ftp   IN  A   192.168.0.6
www   IN  A   192.168.0.7
www   IN  A   192.168.0.8
```

This RR format, which relies on blank name replication, produces exactly the same result:

```
; example.com zone file fragment
....

ftp   IN  A   192.168.0.4
      IN  A   192.168.0.5
      IN  A   192.168.0.6
www   IN  A   192.168.0.7
      IN  A   192.168.0.8
```

The name server will deliver all the IP addresses defined for the given name in answer to a query for the A RRs; the order of IP addresses in the returned list is defined by the `rrset-order` statement in BIND's `named.conf` file. The FTP and web servers must all be exact replicas of each other in this scenario.

Balancing Services

The SRV record provides load balancing by using both a *priority* field and a *weight* field for fine-grained control as well as providing fail-over capability. The SRV RR description in Chapter 13 contains an example illustrating its use in load balancing. The SRV RR is not yet widely supported at this time with two notable exceptions: Lightweight Directory Access Protocol (LDAP), which was partly responsible for development for the SRV record and which is used as a part of the discovery process for LDAP servers and the Session Initiation Protocol (SIP) used in VoIP.

Controlling the RRset Order

BIND versions after 9.2.3 fully implement the `rrset-order` statement, which can be used to control the order in which equal RRs, an RRset, of *any* type are returned. The `rrset-order` statement can take a number of arguments, which are described in Chapter 12, but the following fragment only uses the `order` keyword, which may take the values `fixed`—the order the records were defined in the zone file; `cyclic`—start with the order defined in the zone file and round-robin for each subsequent query; and `random`—randomly order the responses for every query. The `rrset-order` statement can only appear in the global `options` clause for BIND but can take addition arguments that can make it applicable to one or more zones. An example `named.conf` fragment follows that will return any RRset (a set of equal RRs) in round-robin order:

```
// named.conf fragment
options {
// other options
            rrset-order {order cyclic;};
};
```

Assume a zone file has the following MX records:

```
; zone file fragment for example.com
             MX  10   mail1.example.com.
             MX  10   mail2.example.com.
             MX  10   mail3.example.com.
```

The first query to this zone for MX records will return in the order `mail1.example.com.`, `mail2.example.com.`, `mail3.example.com.`; the second query will return `mail3.example.com.`, `mail1.example.com.`, `mail2.example.com.`; and so on in cyclic (or round-robin) order.

Effectiveness of DNS Load Balancing

Clearly the effects of caching can significantly distort the effectiveness of any DNS IP address allocation algorithm. A TTL value of 0 may be used to inhibit caching or the increasingly common very short TTL values (30–60 seconds) could be used to reduce the potentially negative caching effect, but only at the cost of a significant rise in the number of DNS queries. It would be a little unfortunate to achieve excellent load balancing across two or three web servers at the cost of requiring ten more name servers. Intuition, without serious experimentation, would suggest that assuming a normal TTL (12 hours or more) and *any* changing IP allocation algorithm (cyclic or random) loads would be reasonably balanced (measured by request arrivals at an IP) given the following assumptions:

- Traffic is balanced over a number of DNS caches, that is, traffic originates from a number of ISPs or customer locations where DNS caches are maintained. Specifically there are no *pathological* patterns where 90% (or some largish number) of the load originates from one particular cache.

- The volume of traffic is reasonably high since pathological patterns are more likely in small traffic volumes.

DNS load balancing cannot, however, account for service loading; for instance, certain transactions may generate very high CPU or resource loads. For this type of control only a specialized load balancer—which measures transaction response times from each server—will be effective.

Define an SPF Record

This section defines how to configure a Sender Policy Framework (SPF) record for a domain and its mail servers. SPF is being proposed as an IETF experimental standard to enable validation of legitimate sources of e-mail. The information found in this section is based on the current versions of the SPF specification and may change—current information may be obtained from `www.ietf.org/internet-drafts/draft-schlitt-spf-classic-01.txt`.

The design intent of the SPF record is to allow a receiving *Message Transfer Agent (MTA)* to verify that the originating IP (the *source-ip*) of an e-mail from a *sender* is authorized to send mail for the *sender*'s domain. The SPF information is contained in a standard TXT RR (though a new RR type may be allocated if and when SPF reaches standardization by the IETF). The TXT RR is described in Chapter 13. If an SPF (TXT) RR exists and authorizes the source IP address, the mail can be accepted by the MTA. If the SPF (TXT) RR does not authorize the IP address, the mail can be bounced—it did not originate from an authorized source for the *sender*'s domain. If the domain does not have an SPF RR, the situation is no worse than before. Many commercial and Open Source MTAs have already been modified to use the SPF record, including sendmail, qmail, Postfix, courier, Exim, and Microsoft Exchange to name but a few. There is a less widely implemented proposal from Yahoo! called DomainKeys (`http://antispam.yahoo.com/domainkeys`), which is a cryptographic-based solution. Microsoft is advocating a standard called Send ID,[1] which contains SPF as a subset but adds a new Purported Responsible Address (PRA) field to the e-mail to provide additional checking.

This solution both describes the format of the SPF record and presents a number of example configurations. The following terminology is used to simplify the subsequent descriptions:

- *Sender*: The full e-mail address of the originator of the mail item (obtained from the *return path* in the actual SPF checks), for instance, `info@example.com`.

[1] More information on Send ID can be found at `www.microsoft.com/mscorp/twc/privacy/spam/senderid/overview.mspx`.

- *Sender-ip*: The IP address of the SMTP server trying to send this message, for instance, 192.168.0.2.
- *Sender-domain*: The domain name part of the *sender*'s e-mail address, for instance, assume the *sender* is `info@example.com`, then the *sender-domain* is `example.com`.

The SPF record defines one or more tests to verify the *sender*. Each test returns a condition code (defined by the `pre` field shown in the next section). The first test to pass will terminate SPF processing.

TXT RR Format

The standard TXT RR format is defined as follows:

```
name  ttl  class   rr     text
```

The SPF record is entirely contained in the `text` field (a quoted string). SPF defines the contents of the quoted string as shown here:

```
"v=spf1 [pre] type [[pre] type] ... [mod]"
```

SPF records are normally defined for the domain and the mail server(s). The following shows a zone file fragment containing SPF records for the domain and the mail server, which in this case only allows mail for the domain to be sent from the host `mail.example.com`.

```
; zone file fragment for example.com
              IN  MX 10 mail.example.com.
....
mail          IN  A     192.168.0.4
; SPF records
; domain SPF
example.com.  IN  TXT   "v=spf1 mx -all"
; mail host SPF
mail          IN  TXT   "v=spf1 a -all"
```

The following text describes the fields used in an SPF record and references where appropriate the `v=spf1 mx -all` SPF record from the preceding example fragment:

`v=spf1` Field

This field is mandatory and defines the version being used. Currently the only version supported is spf1.

`pre` Field

This optional (defaults to +) field defines the code to return when a match occurs. The possible values are + = pass (default), - = fail, ~ = softfail (indeterminate result), ? = neutral. If a test is conclusive, either add + or omit (defaults to +) as in the first test in the example fragment, which could have been written as `+mx`. If a test might not be conclusive, use ? or ~. - is typically only used with `-all` to indicate the action if there have been no previous matches as in the terminating test from the same fragment.

type Field

This defines the mechanism `type` to use for verification of the *sender*. Multiple `type` tests may be defined in a single SPF record. In the example fragment, there are two type tests, `mx` (or `+mx`) and `all` (`-all`). Each of the `type` values is described in detail in the section "SPF Type Values."

mod Field

Two optional record modifiers are defined. If present, they should follow the last type directive, that is, after the terminating `all`. The current values defined are as follows:

`redirect=domain` Field

This redirects verification to use the SPF record of the defined `domain`. This format may be used to enable additional processing in the event of a failure or may be used on its own in an SPF to provide a single domain-wide definition. This format is the same as the type `include` but may be used without the terminating `all` type.

This SPF allows additional processing using the SPF for `example.net` if the mail from `example.com` tests fail:

```
         IN  TXT :"v=sfp1 mx ?all redirect=example.net"
```

This SPF redirects all processing for `example.com` to a standard SPF record in the domain `example.net`:

```
        IN  TXT "v=spf1 redirect=_spf.example.net"
```

The zone file for `example.net` would include the following record:

```
_spf      IN  TXT  "v=spf1 mx -all"
```

`exp=text-rr` Field

The `exp` type, if present, should come last in an SPF record (after the `all` type if present). It defines the name of a TXT record, `text-rr`, whose text may be optionally returned with any failure message. This fragment shows a trivial example where the *sender* of the mail is informed that they are not authorized to send mail. More complex examples, including the use of macro expansion, can be constructed, referring users to a site that could inform them of the procedure to define SPF records.

```
; domain example.com SPF record
         IN  TXT "v=spf1 mx -all exp=getlost.example.com"
; the getlost TXT record
getlost  IN  TXT "You are not authorized to send mail for the domain"
```

The text field is allowed to contain macro expansions as described in the section "Macro Expansion."

SPF `type` Values

The SPF `type` parameter defines either the mechanism to be used to verify the *sender* or to modify the verification sequence as described in the following sections.

Basic Mechanisms

These types do not define a verification mechanism but affect the verification sequence:

- `include:domain`: Recurse testing using the supplied domain. The SPF record for `domain` replaces the *sender-domain*'s SPF and processing uses the rules defined in the `included` SPF. This is the most common form when clients send mail through an ISP's servers.
- `all`: Terminates a test sequence if no positive results have been found previously.

Sender Mechanisms

These types define a verification mechanism.

Type ip4 Format

This type may take one of the following formats:

```
ip4:ipv4 ip4:ipv4/cidr
```

The `ip4` type uses the *sender-ip* for verification. If the *sender-ip* is the same as `ipv4`, the test passes. This may take the additional argument `ip4:ipv4/cidr`, in which case if the source IPv4 address lies in the range defined by `cidr`(the IP prefix or slash notation), the test passes. This type uses no additional DNS resources and is therefore the recommended solution for IPv4.

This SPF only allows e-mail for the domain to be sent from `192.168.0.2`:

```
    IN  TXT "v=spf1 ip4:192.168.0.2 -all"
```

This SPF allows mail to be sent from any of the 32 addresses that contains the address 192.168.0.38 (CIDR range is from 192.168.0.32–63):

```
    IN  TXT "v=spf1 ip4:192.168.0.38/27 -all"
```

Type ip6 Format

This type may take one of the following formats:

```
ip6:ipv6 ip6:ipv6/cidr
```

The `ip6` type uses the same formats defined for `ip4` previously. This type uses no additional DNS resources and is therefore the recommended solution for IPv6.

The following only allows messages for the domain to be sent from the single address 2001:db8:0:0:0:0:0:10:

```
    IN  TXT "v=spf1 ip6:2001:db8::10 -all"
```

The next example allows mail to be sent from 32 addresses that contain the address 2001:db8:0:0:0:0:0:10 (range is from 2001:db8:0:0:0:0:0:1 to 2001:db8:0:0:0:0:0:1f).

```
    IN  TXT "v=spf1 ip4:2001:db8::10/123 -all"
```

Type a Format

This type may take one of the following formats:

The `a` type uses an A RR for verification. In the basic format with no additional arguments, if the A RR for the *sender-domain* is the same as the *sender-ip*, the test passes. The optional form `a/cidr` will apply the test to the extended range defined by the IP prefix (or slash) notation. The form `a:domain` will cause the test to be applied to the A RR of `domain`, and `a:domain/cidr` will apply the test to the range of IPs defined by the IP prefix (or slash) notation. The `domain` argument may also use macro expansion, defined later in this section. The `a` and `a/cidr` formats require an A RR for the domain as shown in the following fragment:

```
; zone fragment for example.com
$ORIGIN example.com.
...
@             IN   A       192.168.0.2
              IN   TXT "v=spf1 a -all"
....
```

This SPF allows only the host `smtp.example.net` to send mail for the domain `example.com`:

```
              IN   TXT "v=spf1 +a:smtp.example.net -all"
```

The advantage of using the preceding construct is that if the IP address of `smtp.example.com` changes, the preceding SPF record does not change. The cost, however, is one more DNS transaction for every SPF check.

Type `mx` Format

This type may take one of the following formats:

```
mx mx/cidr mx:domain mx:domain/cidr
```

The `mx` type uses the MX RRs and the mail server A RRs for verification. Remember, this type uses the MX RR for the domain, which may not be the same as the SMTP server for the domain. In the basic format with no additional arguments, the MX record for the *sender-domain* and the A RRs for the defined mail host(s) are obtained; if the IP address of the *sender-ip* matches any of the mail host IPs, the test passes. The form `mx:/cidr` applies the address range defined by `cidr` (IP prefix or slash notation) is used for the match. The format `mx:domain` uses the MX and A RRs for `domain` instead of the *sender-domain*, and the format `mx:domain/cidr` extends the IP address check to the `cidr` (IP prefix or slash notation) range of IP addresses. The `domain` argument may also use macro expansion defined later in this section. Use of the `mx` format involves at least two DNS lookups per SPF verification operation.

This SPF allows mail from the domain `example.com` to be sent from any mail server defined in an MX RR for the domain `example.net`:

```
             IN  TXT "v=spf1 mx:example.net -all"
```

This SPF allows mail to be sent from any of the 16 IP addresses containing each of the mail servers defined in MX records for the sending domain:

```
             IN  TXT "v=spf1 mx/28-all"
```

Type ptr Format

This type may take one of the following formats:

```
ptr ptr:domain
```

The `ptr` type uses PTR RRs of the *sender-ip* for verification. In the basic format with no additional arguments, the *sender-ip* is used to query for the host name using the reverse map. The A or AAAA RR for the resulting host is then obtained. If this IP matches the *sender-ip and* the *sender-domain* is the same as the domain name of the host obtained from the PTR RR, then the test passes. The form `ptr:domain` replaces the *sender-domain* with `domain` in the final check for a valid domain name. The `domain` argument may also use macro expansion defined later in this section. The PTR record is the least preferred solution since it places a load on the `IN-ADDR.ARPA` (IPv4) or `IP6.ARPA` (IPv6) reverse-map domains, which generally have less capacity than the gTLD and ccTLD domains.

This SPF would allow any host in the domain `example.com` that is reversed mapped to send mail for the domain:

```
        IN  TXT "v=spf1 ptr -all"
```

Type exists Format

This type may take one of the following formats:

```
exists exists:domain
```

The `exists` type tests for existence of the *sender-domain* using an A RR query. In the basic format with no arguments, an A RR query is issued using the *sender-domain* and if any result is obtained, the test passes. The form `exists:domain` applies the same test but for `domain`. The `domain` argument may also use macro expansion defined later in this section. The `exists` form requires an A RR for the domain as shown in the following zone file fragment:

```
; zone fragment for example.com
$ORIGIN example.com.
...
@             IN   A       192.168.0.2
              IN   TXT "v=spf1 +exists -all"
....
```

Macro Expansion

The SPF record allows macro expansion features using a `%(x)` format where `%` indicates a macro and `x` is a character defining the macro type as defined in Table 8-1.

Table 8-1. *SPF Macro Expansion Arguments*

Macro	Function
`%(c)`	Only allowed in TXT records referenced by the `exp` field. The IP of the receiving MTA.
`%(d)`	The current domain, normally the *sender-domain* `%(o)`, but replaced by the value of any `domain` argument in the `type` field as discussed previously.
`%(h)`	The domain name supplied on `HELO` or `EHLO`, normally the host name of the sending SMTP server.
`%(i)`	The *sender-ip* value. The IP of the SMTP server sending mail for user `info@example.com`.
`%(l)`	Replace with local part of *sender*. For instance, if the *sender* is `infor@example.com`, the local part is `info`.
`%(o)`	The *sender-domain* value. For instance, if the e-mail address is `info@example.com`, the *sender-domain* is `example.com`.
`%(p)`	The validated domain name. The name obtained using the PTR RR of the *sender-ip*. Use of this macro will require an additional query unless a `ptr` type is used.
`%(r)`	Only allowed in TXT records referenced by the `exp` field. The name of the host performing the SPF check. Normally the same as the receiving MTA.
`%(t)`	Only allowed in TXT records referenced by the `exp` field. Defines the current timestamp.
`%(s)`	Replace with *sender* e-mail address, for instance, `info@example.com`.
`%(v)`	Replaced with `in-addr` if *sender-ip* is an IPv4 address and `ip6` if an IPv6 address. Used to construct reverse-map strings.

The preceding macros may take one or more additional arguments as follows:

- `r`: Indicates reverse the order of the field. For instance, `%(or)` would display `example.com` as `com.example`, and `%(ir)` would display 192.168.0.2 as 2.0.168.192. The default splitting point for reversing the order uses `.` (dot) as the separator but any other separator may be used; for instance, `%(sr@)` would split `info@example.com` at the `@` separator and when reversed will display `example.com.info` (when fields are rejoined they will always use a `.` [dot]).

- *Digit*: The presence of a digit (range 1 to 128) controls the number of rightmost names or labels displayed. For instance, `%(d1)` uses the `d` part to extract the current domain (assume its `example.com`) as defined previously, and the qualifying digit (`1`) displays only one rightmost label from the name, in this case `com`; but `%(d5)` would display five right-hand names or labels up to the maximum available, which in this example would display `example.com`.

SPF Record Examples

The following examples are designed to illustrate various uses of the SPF record. The SPF macro expansion features in particular can lead to complex definitions; further examples may be discovered by interrogating such domains as `microsoft.com`, `aol.com`, and `google.com`, all of whom are among the 750,000+ domains that currently publish SPF records. A `dig` command (introduced in Chapter 9) such as shown here will yield an SPF record if published:

```
# dig example.com txt
```

Substitute your favorite domain in the preceding example to verify the existence of an SPF record.

Single Domain Mail Server

This example assumes a single mail server that both sends and receives mail for the domain:

```
; zone file fragment for example.com
$ORIGIN example.com.
          IN  MX 10 mail.example.com.
....
mail      IN  A     192.168.0.4
; SPF records
; domain SPF
@         IN  TXT   "v=spf1 mx -all"
; mail host SPF
mail      IN  TXT   "v=spf1 a -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.com`, and the *sender-domain* is `example.com`. The SPF record only allows the MX host(s) to send for the domain. The mail host SPF is present *in case* the receiving MTA uses a reverse query to obtain the *sender-ip* host name and then does a query for the SPF record of that host. The SPF record states that the A record of `mail.example.com` is permitted to send mail for the domain. If the domain contains multiple MX servers, the domain SPF would stay the same, but each mail host should have an SPF record.

SMTP Server Offsite

This example assumes the domain `example.com` will send mail through an off-site mail server in `example.net`, for instance, an ISP:

```
; zone file fragment for example.com
$ORIGIN example.com.
             IN  MX 10  mail.example.net.
....
; SPF records
; domain SPF
@            IN  TXT    "v=spf1 include:example.net -all"
; WARNING: example.net MUST have a valid SPF definition
```

This format should be used if and only if it is known that `example.net` has a valid SPF record. The `include` recurses (restarts) verification using the SPF records for `example.net`. Mail configuration changes are localized at `example.net`, which may simplify administration. The `include` could have been replaced with redirect as shown here:

```
@            IN  TXT    "v=spf1 redirect=example.com"
```

Virtual Mail Host

This example assumes `example.net` is the host for a large number of virtual mail domains and supplies SMTP services for others. The zone file fragment that follows describes one of the virtual mail domains `example.org`:

```
; zone file fragment for example.org
$ORIGIN example.org.
            IN  MX 10 mail.example.net.
....
; SPF records
; domain SPF
@          IN  TXT   "v=spf1 include:example.net -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.org`, and the *sender-domain* is `example.org`. The SPF record recurses to the domain name `example.net` for verification.

Here is the zone file for `example.net`:

```
; zone file fragment for example.net
$ORIGIN example.net.
             IN  MX 10   mail.example.net.
....
mail         IN  A       192.168.0.37
; SPF records
; domain SPF - any host from
; 192.168.0.32 to 192.168.0.63 can send mail
; and any MX host
@            IN  TXT    "v=spf1 ip4:192.168.0.37/27 mx -all"
; mail SPF
mail         IN  TXT    "v=spf1 a -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.net`, and the *sender-domain* is `example.net` or the `include:example.net` if the mail originated from the `example.org` zone. The SPF record allows any host in the 32 address subnet that contains 192.168.0.37 to send mail for this domain (`example.net`) and any hosted virtual domain, that is, `example.org` in the preceding example. The SPF also allows any host defined in an MX RR as an alternative if the first test fails and allows for a future reconfiguration of the network that may move the host `mail.example.net` IP address outside the defined `ip4` range. The scenario could have used a slightly shorter version:

```
@         IN  TXT    "v=spf1 mx/27 -all"
```

This record has the same effect as `a:192.168.0.37/27` but will cost a further DNS lookup operation, whereas the IP is already available. The scenario relies on the fact that customers will only send mail via the domain `example.net`, that is, they will *not* send mail via another ISP when at home or when traveling. If you are not sure if this is the case, the sequence can be terminated with `?all`, which indicates that the results may not be definite—it allows the mail to pass, perhaps after logging the incident to capture statistics. If the domain contains multiple MX servers, the domain SPF would stay the same, but each mail host would have an SPF record.

No Mail Domain

This example assumes that the domain `example.org` *never* sends mail from any location—ever. Typically this would be done to prevent bogus mail using this domain for everyone else—it is a supreme act of self-sacrifice!

```
; zone file fragment for example.org
; zone does NOT contain MX record(s)
$ORIGIN example.org.
...
; SPF records
; domain SPF
@        IN  TXT   "v=spf1 -all"
```

This SPF test will always fail since the only condition it tests is the `-all`, which, because of the - (minus), results in a fail.

Using Macro Expansion

This example uses macro expansion in the SPF and the polite message sent to users to indicate that the *sender* may be being impersonated. The zone file fragment is as follows:

```
; zone file fragment for example.com
$ORIGIN example.com.
            IN  MX 10 mail.example.com.
....
; SPF records
; domain SPF
@          IN  TXT   "v=spf1 exists:%(d) -all ext=badguy.example.com"
badguy     IN   TXT  "The email from %(s) using SMTP server at %(i) was rejected \
   by %(c) (%(r)) at %(t) \
   because it failed the SPF records check for the domain %(p). \
   Please visit http://abuse.example.com/badguys.html for more information"
```

The `badguy` TXT RR is split across multiple lines (each ending with a `\`) for presentation reasons only and should appear on a single line in the zone file. The `exists:%(d)` tests for the existence of the *sender-domain*, which is the default value for the `exists` test but is used to illustrate use of macros in expressions.

Supporting `http://example.com`

This solution configures a name server to allow URLs of the form `http://www.example.com` and `http://example.com`—both URLs will address (or resolve to) the same web server. Seems it's the cool thing to do these days. To make this feature work also requires a change to the web server. The required change to Apache when using virtual hosts is also provided.

```
; zone fragment for example.com
$TTL 2d ; zone ttl default = 2 days
$ORIGIN example.com.
....
; SOA NS MX and other records
; define an IP that will resolve example.com
@               IN      A      192.168.0.3
; you could also write the preceding line as
; example.com.   IN      A      192.168.0.3
www             IN      CNAME  example.com. ; dot essential
; aliases www.example.com to example.com
; OR define another A record for www using same host
; this is the least number of changes and saves a CNAME
www             IN      A      192.168.0.3
```

The preceding will also work for any other host name as long as different ports are in use; for instance, `ftp://example.com` will work if the FTP server was appropriately configured and on the same host, which in the preceding case is 192.168.0.3.

Apache Configuration

This configuration assumes the use of virtual hosts on an Apache (1.3.x or 2.x) server. Apache's `httpd.conf` configuration file containing the `VirtualHost` section for `example.com` would look something like the following fragment:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ErrorLog logs/www.example.err
    CustomLog logs/www.example.log common
</VirtualHost>
```

A second `VirtualHost` definition is added with `ServerName` modified to reflect the `example.com` change as follows:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName example.com
    ErrorLog logs/example.err
    CustomLog logs/example.log common
</VirtualHost>
```

In the preceding example, a second log and error file is used to avoid possible corruption. An alternate method is to use a single `VirtualHost` definition with the `ServerAlias` directive as shown here and which only requires single log and error files:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ServerAlias example.com
    ErrorLog logs/example.err
    CustomLog logs/example.log common
```

In many cases, when `example.com` is entered, your ever-helpful browser will auto-complete (or guess) that what you really meant was `www.example.com` and add the `www` automatically. So after all that hard work in many browsers, `example.com` would have worked even if you had done nothing!

■ **Caution** If you are using MS FrontPage extensions with a single `VirtualHost` definition, then the `ServerName` must be the name that is used to log in to FP. In the preceding example, the FrontPage login name used would be `www.example.com`.

Out-of-Sequence Serial Numbers

The `serial number` field of the SOA RR (described in Chapter 2) by convention uses a date format defined to be `yyyymmddss` where `yyyy` is the four-digit year number, `mm` is the two-digit month number, `dd` is the two-digit day within month number, and `ss` is a two-digit sequence number within the day. Since this is only a convention, BIND and most other DNS software does not validate the format of this field; it is very easy to introduce errors into this number and get out of sequence. Zone transfer to zone slave will, in the event of zone file changes, occur only if the serial number of the SOA RR is greater that the previous one. So the dreaded day has come and while pondering the meaning of life during a zone file update the serial number is changed, BIND has been restarted, and only with something approaching shock and awe you discover the SOA serial number is incorrect. Apart from ritual suicide, what can be done?

To illustrate the fixes possible, it is assumed that today's date is 28 February 2003 (serial number `2003022800`). If the erroneous serial number entered is less than today, that is, `2003022700`, the fix is trivial: simply correct the serial number and restart or reload BIND or reload the zone with `rndc` (see Chapter 9). If the number is too high, it depends on how high the number is and how frequently the zone file is changed. Assume the changed serial number was set to `2004022900`, which as we all know does not exist, 2003 not being a leap year; however, BIND does not know that and a zone transfer will have taken place, 29 being greater than 28. The simple fix is to increment the date again to `2003030100` and keep using the sequence number until the correct date is reached (tomorrow in this case). This works unless you will require to make more than 99 changes until the erroneous date is reached.

If all the quick solutions are not acceptable, for instance, the serial number is `2008022800`, then it's time to get out the calculator or do some serious mental arithmetic. The SOA serial number is an unsigned 32-bit field with a maximum value of $2^{31,}$ which gives a range of 0 to 4294967295, but the maximum increment to such a number is 2^{31-1} or 2147483647, incrementing the number by the maximum would give the same number. Using the maximum increment, the serial number fix is a two-step process. First, add 2147483647 to the erroneous value, for example, 2008022800 + 2147483647 = 4155506447, restart BIND or reload the zone, and make *absolutely* sure the zone has transferred to all the slave servers. Second, set the SOA serial number for the zone to the correct value and restart BIND or reload the zone again. The zone will transfer to the slave because the serial number has wrapped through zero and is greater that the previous value of 4155506447! RFC 1982 contains all the gruesome details of serial number comparison algorithms if you are curious about such things.

Use of Wildcards in Zone Files

The standard wildcard character * (asterisk) can be used as a name with any RR. Wildcards can have unintended consequences and should only be used with considerable caution.

■ **Note** The Internet Architecture Board (IAB) has even published a paper on the subject of wildcard usage after the infamous use of wildcard A RRs by a gTLD operator to redirect users to a default page when any domain was not found (`www.iab.org/documents/docs/2003-09-20-dns-wildcards.html`).

Wildcards can be very confusing in the DNS specifications because of the normal sense in which wildcards are used in search expressions to find items with imprecisely known information. For example, the command `ls |grep $*.html` will list all files in the given directory ending with `.html`; in this case, the * means any character any number of times. In the case of zone files, the wildcard *creates* records of the RR type they are used with such that any query for a particular RR type and for which an explicit RR does *not* exist will be answered with the wildcard RR data as if it *did* exist—that is, no query for the given RR of that type will fail. This may not be the intended result. While wildcards may be used with any RR type, they are most commonly used with MX records as shown in the following zone file fragment:

```
; zone file for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
@               IN   MX  10  mail.example.com.
*               IN   MX  10  mail.example.com.
```

In the preceding fragment, an MX query for `bill.example.com`, `joe.example.com`, and `everythingelse.example.com` will return the host `mail.example.com`. The following example shows how an existing RR will block the operation of the wildcard:

```
; zone file for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
@               IN   MX  10  mail.example.com.
*               IN   MX  10  mail.example.com.
subdomain   IN  MX  10  mail.example.net.
```

As before, MX queries will return `mail.example.com` except queries for `subdomain.example.com`, which will return `mail.example.net`, and any undefined names below `subdomain` such as `bill.subdomain.example.com` will return `NXDOMAIN` (no name).

A wildcard cannot do anything that cannot be done by one or more (perhaps many more) RRs. There is no *essential* reason to use wildcards other than to reduce the amount of data that may otherwise have to be defined. Whether this reduction in administrative effort is worth the potentially confusing effect of using wildcards in RRs is a matter for local decision.

Summary

This chapter covered a number of common name server configurations and also illustrated some more subtle uses of the DNS system.

The next chapter describes the use of various DNS diagnostic tools and techniques to cover the situations where head-scratching fails to yield the required results.

CHAPTER 9

DNS Diagnostics and Tools

Diagnosing DNS problems can be complex, made so by its interaction with other DNS systems. A DNS problem may originate locally or anywhere in the chain of name servers that provide the response to a query. Finding the location of the problem is, depending on your outlook, either the bane or the challenge of a DNS administrator's life. This chapter is divided into two parts. The first describes a number of tools that may be used to verify, support, or interrogate DNS systems—DNS utilities. The second part looks at diagnosing DNS systems, in some cases using the tools described, in other cases using other methods such as log inspection and invoking debug levels to increase reporting.

DNS Utilities

There are a number of DNS utilities, some of which are specific to BIND distributions and some of which are available on a variety of platforms. The author maintains reference material about this book on his site, `www.netwidget.net/books/apress/dns`; there you will find listed additional DNS utilities—including web-based and Windows utilities—not covered in this book. The following utilities are introduced in this chapter:

- `nslookup`: Utility for interrogating DNS servers. Widely available on multiple platforms, including Windows.
- `dig`: Utility for interrogating DNS servers. Typically only available on BIND-supported platforms.
- `rndc`: Remote maintenance tool for BIND.
- `rndc-confgen`: Utility to generate keys and `rndc.conf` files for use with the `rndc` utility—including a trivial default configuration.
- `nsupdate`: Utility for dynamically updating zone files.
- `named-checkconf`: Utility for checking the syntax of the `named.conf` file.
- `named-checkzone`: Utility for verifying zone files.
- `dnssec-signzone`: Utility for cryptographically signing zones for use with DNSSEC.bis. Chapter 11 makes significant reference to this utility.
- `dnssec-keygen`: Utility for generating keys used in various secure DNS transactions. Chapters 10 and 11 make significant reference to this utility.

The descriptions typically take the form of a quick usage example followed by a detailed description of the various options and parameters available, in some cases followed by further advanced examples. This chapter omits the `host` utility—yet another DNS interrogation utility similar in function to `nslookup` and `dig` but having neither the batch service of `dig` nor the interactive format of `nslookup`. For information on this utility, use `man host` or `host -h` at a command-line prompt.

Every DNS administrator should be thoroughly familiar with either `nslookup` or `dig` for troubleshooting and diagnostic work. If a choice needs to be made, it will depend on circumstance. If you are working on a variety of platforms, `nslookup` is available on Linux, BSD, and Windows platforms and may be the best choice. If DNSSEC (Secure DNS) is being implemented or in the short-term plans there is no choice, `dig` is DNSSEC aware, `nslookup` is not.

The following sections all use the ubiquitous `example.com` domain in conjunction with private IPv4 addresses to illustrate the various commands. This is done purely in the interests of being a good netizen. If you run the `dig` and `nslookup` examples on the public network, some will work, many will not—the actual commands and results were all carried out using a private configuration and are intended to illustrate techniques and formats rather than be taken literally. However, `example.com` does in fact exist on the public network and resolves to an IPv4 address of 192.0.34.166, which is on a reserved IANA netblock. It is not a very interesting domain for experimentation, however, since it has only a limited number of hosts; for instance, there is a `www.example.com` RR, but it does not have MX RRs or an FTP site or anything really exciting. Instead, either build your own name server using one or more of the example configurations in Chapter 7 and use it as the basis of experimentation or get onto the public network and select a domain that you either know well or are curious about, replace the `example.com` in the various examples with your chosen domain name, and try and explore it using the commands as a starting point. You may be astonished at some of the results you get—and don't forget to look for the more exotic RR types (a full list is defined in Chapter 13) such as LOC RRs (geographic location RRs), which are surprisingly frequent, or discover who publishes SPF records (in TXT RRs), or find how many MX or NS RRs some of the more high-profile domains have and whether they are all on the same subnet. An endless world of fun is at your fingertips.

■ **Note** One of the happy side effects of installing BIND on Windows NT 4.0 or Windows 2000 even if it is not used as a name server is that all the diagnostic tools are also installed, including `dig`, `rndc`, and `nsupdate`—see Chapter 6 for how to install BIND on Windows.

The `nslookup` Utility

The `nslookup` utility was officially deprecated in favor of `dig`, but with the latest BIND releases it seems to have received a new lease on life. The major advantage of `nslookup` is its almost universal availability, specifically on Windows systems where `dig` is still pretty exotic. Therefore, if you work in a mixed environment, you are more likely to come across `nslookup` than `dig`. `nslookup` provides both command-line and interactive formats. It can look relatively trivial at first glance, but its configuration parameters (in `.nslookuprc` in the user's home directory or by default from `/etc/resolv.conf` or Windows Network Properties), which may be modified in interactive mode, adds tremendous power to the utility. The default configuration parameters can be displayed using the `-all` option (or `set all` option in interactive mode).

Tip The Windows version of the `nslookup` command is documented in the Microsoft Knowledge Base article number 200525 (`http://support.microsoft.com/default.aspx?scid=kb;en-us;200525`).

`nslookup` Command Format

`nslookup` has four generic command formats:

- *Format 1*: Lookup target using the default name server:

  ```
  nslookup [-opt] target
  ```

- *Format 2*: Lookup target using the specific name server:

  ```
  nslookup [-opt] target dns
  ```

- *Format 3*: Enter interactive mode using the default name server:

  ```
  nslookup [-opt]
  ```

- *Format 4*: Enter interactive mode using the specific name server:

  ```
  nslookup [-opt] - dns
  ```

Format 4 does not work in the native Windows version of `nslookup` but does on any BIND version, including the BIND version installed on Windows. To achieve the same effect on the native Windows version requires use of the `server` command after entering interactive mode using Format 3.

Quick Examples

The following examples are provided for readers who prefer to experiment before reading about the multiple options that can affect the behavior of the `nslookup` command. They are designed to illustrate techniques and should not be taken literally (though the domain `example.com` does exist on the public network) but rather replace `example.com` with your favorite domain name and experiment with various formats. The `nslookup` command is available on Linux, Unix, BSD, and Windows systems.

Format 1: Using the command-line mode with the default name server to perform a simple host lookup:

```
# nslookup www.example.com
Server: ns1.example.com
Address: 192.168.2.53

Name: www.example.com
Address: 192.168.2.80
```

This returns the A record for www.example.com using the default name server—in this case ns1.example.com, which is defined in Windows Network Properties or /etc/resolv.conf in Linux and BSD systems.

Format 1: Using the command-line mode with the default name server to perform a simple reverse map IP lookup:

```
# nslookup 192.168.2.80
Server: ns1.example.com
Address: 192.168.2.53

Name: www.example.com
Address: 192.168.2.80
```

This returns the PTR record for 192.168.2.80 using the IN-ADDR.ARPA domain hierarchy.

Format 2: Using the command-line mode with a specific name server to perform a simple host lookup:

```
# nslookup www.example.com 192.168.255.53
Server: ns1.example.net
Address: 192.168.255.53

Name: www.example.com
Address: 192.168.2.80
```

This returns the A record for www.example.com using the name server at 192.168.255.53. The command format allows either an IP or a name for the specified name server, so the preceding command could have been written as follows:

```
# nslookup www.example.com ns1.example.net
```

Interactive Format

nslookup's interactive format (Formats 3 and 4) provides a single prompt (>) and allows any directive that follows to be entered. To terminate interactive mode, use Ctrl+C (for Windows and for Linux and BSD if no command is currently active), Ctrl+D (Linux and BSD only), or exit (Windows, Linux, and BSD). In Linux or BSD, Ctrl+C will terminate a currently active interactive command or will terminate nslookup if no command is active.

```
# nslookup -all
//  list all records in the domain - needs axfr to be enabled
> ls example.com
// list all text records in domain
> ls -t TXT example.com
// set the base domain to be used for subsequent commands
> set domain=example.org
```

```
//  find host
> mail
// returns mail.example.org
// exit interactive mode
> exit
```

Options

`nslookup` provides a dizzying number of options that vary its processing. Some of these options are only available in interactive mode. The Windows version adds a couple of unique commands. Multiple options can be specified on a single command line.

The set of options defined in Table 9-1 will only work in interactive mode.

Table 9-1. `nslookup` *Interactive Commands*

Option	Parameters	OS	Mode	Processing
`ls`	`[opt] domain`	W	I	Lists all the information for the target domain. This command uses `AXFR` to transfer the zone file. If `AXFR` is disabled on the target domain, the `ls` command will fail with an appropriately obscure error. Takes the standard redirection commands `>` or `>> file name` to output or append to a file for subsequent processing. The options supported are: `-a`: Lists aliases (CNAME) in the domain (synonym for `-t CNAME`). `-d`: The default behavior. Lists all records in the domain (synonym for `-t ANY`). `-h`: Lists all information records in the domain (synonym for `-t HINFO`). `-s`: Lists all well-known service records in the domain (synonym for `-t WKS`). `-t`: Lists the specific record type in the domain, for instance, `-t A`.
`lserver` `server`	`dns`	A	I	Sets the name server for subsequent commands. May be either a name or an IP address. The name is looked up using the original default name server (before any `server` or `lserver` commands are issued). The default `server` is defined in `/etc/resolv.conf` or in the `.nslookuprc` file in the user's home directory for Linux and BSD systems and in Network Properties for Windows systems.
`root`	`root-dns`	A	I	Changes the root server used for certain operations and can be specified as a name, for instance, `k.root-servers.net`, or an IP.

OS Column Key:

W= Windows only

A = Windows, Linux, BSD

U = Linux, BSD

Mode Column Key:

B = Interactive and command-line format

I = Interactive only

C = Command-line only

The options defined in Table 9-2 may be used in interactive mode if preceded with the keyword `set` or on the command line if preceded with `-` (hyphen). When used on the command line, the option only affects the single command (it is not saved). When used with `set` in interactive mode, the option will persist until set by another similar option command. In interactive mode, the command `set all` will list the default settings (the equivalent command-line version would be `nslookup -all`). In a number of cases a short form is also provided.

Table 9-2. `nslookup` *Options*

Option	Parameters	OS	Mode	Processing
`all`		A	B	Displays a list of the default values used by `nslookup`, including the current name server.
`class=`	`class`	A	B	Allows the `class` value to be set for all subsequent commands and may take the case-insensitive values `IN` (Internet—the default), `ANY`, `CH` (CHAOS), and `HS` (HESIOD).
`domain=`	`domain-name`	A	B	Allows a base to be set for all subsequent searches when used in interactive mode. The default domain is defined in `/etc/resolv.conf` for Linux or BSD systems and Network Properties for Windows systems. Setting `domain=` will reset any previously defined `srchlist`.
`[no]debug` `[no]deb`		A	I	Allows control over the debugging information—`debug` (short form `deb`) turns it on, `nodebug` (or `nodeb`) turns it off. The default is `nodebug`.
`[no]d2`		A	I	Enables/disables debugging information—`d2` turns it on, `nod2` turns it off. The default is `nod2`.
`[no]defname` `[no]def`		A	I	Controls whether a domain name (in either `domain` or `srchlist`) is added to a target, which does not end with a dot, that is, it is NOT an FQDN. See also the entry for `search` for full behavior description.
`[no]ignoretc`		A	I	Controls whether packet truncation errors are ignored (`ignoretc`) or whether they cause termination (`noignoretc`—the default).
`[no]msxfer`		W	I	Controls use of MS fast zone transfer. `msxfer` turns it on, `nomsxfer` (the default) turns it off.
`[no]recurse` `[no[rec]`		A	B	Controls recursive behavior. `recurse` (the default) turns it on, and `norecurse` turns it off.
`[no]vc`		A	I	Controls whether to use TCP (`vc`) or UDP (`novc`)—the default is `novc`.
`[no]search` `[no]sea`		A	I	This parameter controls how the `srchlist=` value is used. `search` and `defname` are interrelated based on the matrix shown in Table 9-3 for targets that are not FQDNs.

OS Column Key:	Mode Column Key:
W= Windows only	**B** = Interactive and command-line format
A = Windows, Linux, BSD	**I** = Interactive only
U = Linux, BSD	**C** = Command-line only

Option	Parameters	OS	Mode	Processing
`port=`	`port-no.`	A	B	Changes the default port from the default (`53`) to that specified by `port-no`.
`type=` `querytype=`	`rr-type`	A	B	May take most case-insensitive RR type values, including the meta RRs `ANY`, `IXFR`, and `AXFR`. The default value is `type=A`. `type=ANY` with a domain root name will return any RR with a blank name (label) entry—these include SOA, NS, and MX RRs if directed at an authoritative server for the domain—and thus it provides a quick way to get useful domain info.
`retry=`	`number`	A	B	Controls the number of retries that will be attempted. The default is `4`.
`root=`	`dns`	A	B	Controls the name server used when querying the root-servers. The default is typically `f.root-server.net.` (on Linux and BSD) and `a.root-servers.net` on Windows.
`srchlist=`	`dom1/dom2`	A	I	Allows setting of a search list (up to six names are allowed separated by a forward slash).

OS Column Key:

W= Windows only

A = Windows, Linux, BSD

U = Linux, BSD

Mode Column Key:

B = Interactive and command-line format

I = Interactive only

C = Command-line only

Table 9-3 shows the relationship between the `search` and `defname` options when used with `srchlist`.

Table 9-3. *Effect of the* `search` *and* `defname` *Options*

`search`	`defname`	Result
`search`	`defname`	Adds domain names from `srchlist` or until answer is found
`nosearch`	`defname`	Adds domain name from domain
`nosearch`	`nodefname`	Must be an FQDN
`search`	`nodefname`	Must be an FQDN

In all cases, the first good result will terminate the command—`srchlist` cannot be used to look up multiple targets. In general, the `srchlist` is most useful with subdomains but can be used with different domains.

Examples: Command Line

Use this command to get mail records for a domain:

```
nslookup -type=MX example.com
```

The following lists all the options being used and gets the host address for `mail.example.com`:

```
nslookup -all mail.example.com
```

The next command gets the SOA record using a specific DNS:

```
nslookup -type=SOA example.com 192.168.23.53
```

This one gets all records without labels (the zone apex or root) for the domain (gets SOA, NS, and MX and others if defined) if pointed at an authoritative server for the domain; otherwise, it returns only NS RRs:

```
# nslookup -type=any example.com ns1.example.com
```

Finally, the following gets all domain records if zone transfer (`AXFR`) is not inhibited for the domain (will return `BAD ERROR VALUE` if inhibited using an `allow-transfer` statement or similar):

```
# nslookup -type=axfr example.com
```

Example: Interactive Mode

Enter interactive mode and list the default options—this test was run on a Windows system to illustrate the default superset offered by the Windows native version of `nslookup`—the items noted as Windows only will not be present on a BIND `nslookup` utility. Lines beginning with `//` are intended as comments to describe the function of the following line and should not be entered; only those beginning with the prompt (`#` or `>`) should be entered:

```
# nslookup -all
Default Server:  ns1.example.com
Address:  192.168.2.53
Set options:
  nodebug
  defname
  search
  recurse
  nod2
  novc
  noignoretc
  port=53
  type=A
  class=IN
  timeout=2
  retry=1
  root=A.ROOT-SERVERS.NET.
  domain=example.com
  MSxfr [note: Windows only MS fast zone xfer]
  IXFRversion=1 [note: Windows only incremental zone xfer]
  srchlist=example.com
//  list all records in the domain - needs axfr to be enabled
> ls example.com
// list all text records in domain
> ls -t TXT example.com
// set the base domain to be used for subsequent commands
> set domain=example.org
//  find host
> mail
// returns mail.example.org
// but will handle full format also
> mail.example.org
// return mail.example.org as expected
>set type=any
> example.com
// list apex records for the domain example.com
// and override the domain= value set previously
> set type=mx
// type=mx is persistent
> example.com
// this works as expected for the domain root
> www.example.com
// fails because there is no MX RR for www.example.com!
> set type=a
> www.example.com
// works as expected to give IP
>set type=any
// this is much more useful because it will get all RRs with given name
> www.example.com
// will return ALL RRs with this name.
// exit interactive mode
> exit
```

Tip The `nslookup` default is `type=a` (an A RR only will be returned). It is far better, as illustrated in the preceding sequence, to change this to `ANY` (`set type=ANY`), since it will get *all* records with a particular name, which would include an A RR if present. You get a lot more bang for the buck with `type=ANY`!

BIND dig Utility

`dig` is the current DNS diagnostic tool of preference, but as noted earlier, it is not always widely available and rarely on Windows systems. `dig` has both a command line and a batch mode (but no interactive mode like `nslookup`). In general, the command line of `dig` is more powerful than `nslookup`—even allowing multiple queries in a single line—and the batch mode makes running check files a breeze. `dig` offers a daunting array of options, but the following section provides simple examples.

Quick Examples

The following examples are offered for those readers who wish to experiment before reading about the huge number of options that are available with the `dig` command. They are designed to illustrate that a lot can be done with a limited set of options. They should not be taken literally; instead, replace `example.com` and the various IP addresses with your favorite domain name and start exploring. More examples are shown at the end of this section.

The following returns any RRs without a label—it will provide the SOA, NS, and MX RR at the domain apex if pointed at an authoritative server for the domain; otherwise, it returns only the domain NS RRs:

```
dig example.com any
```

This returns only the MX record for the domain:

```
dig example.com mx
```

The next command returns the A record for the `www.example.com` using a specific name server indicated by the `@` argument—either a name or an IP address may be used:

```
dig @ns2.example.com www.example.com
```

Using the following command will always generate much more interesting results because the pseudo RR type `ANY` is slightly misleading and actually means *all* RRs with the given name, so any hidden RRs such as AAAA, TXT, RP, or KEY RR types with the same name will be displayed as well as the A RR:

```
dig @ns2.example.com www.example.com any
```

To return all domain records using zone transfer (if allowed), try this command:

```
dig example.com axfr
```

This command returns the PTR RR for the IP address:

```
dig -x 192.168.23.23
```

dig Syntax

dig `[@dns] domain [[-c ]q-type] [[-t ]q-class] [+q-opt] [-d-opt] [%comment]`

■ **Note** The `dig` command uses a mixture of positional/contextual arguments and identified options (that is, identified with an option value `@`, `-`, or `+`) to keep simple queries—simple! There are times when it is necessary to disambiguate the `q-type` and `q-class` options described later, and in this case both can be specified in an identified option format—the examples illustrate this usage.

dig Options

Table 9-4 defines the options available with the `dig` command. The `dig` command may be controlled using a file (`.digrc`) in the user's home directory to set defaults that will override the `/etc/resolv.conf` file.

Table 9-4. `dig` *Options*

Parameter	Value	Description
`@dns`		Defines the optional name or IP address (IPv4 or IPv6 format) of the name server to be used for the query. The default is defined in `/etc/resolv.conf` for Linux and BSD systems and Network Properties for BIND's `dig` on Windows. If present, it must be preceded by `@.dig @192.168.2.53 www.example.com`.
`domain`		Defines the name to be used in the query. Unlike `nslookup`, if this is an IP address, it must be preceded with the `-x` option (see `d-opt` field entry).

`q-type`	`RR`	Defines the type of record to return and may take any valid, case-insensitive, RR type, including `ANY`, `IXFR`, and `AXFR`. If omitted, the value `A` is assumed. This parameter may appear following the domain name or may be optionally preceded with `-t` in the identified option format. The following two commands will obtain only the TXT RR at the specified host if present:
		`dig www.example.com txt`
		`dig -t txt www.example.com`
		To get a full listing of the domain records, use the `AXFR` option. The `AXFR` feature may be disallowed by the `allow-transfer` statement in `named.conf`, in which case the command will fail with a "Connection refused" message. When using the `IXFR` type, it takes the form `IXFR=sn`, where `sn` indicates display all changes since the serial number `sn` on the SOA RR. The value `ANY` will list all available records at `domain`; thus to get a listing of the SOA, NS, and MX records, as well as any others at the domain apex, use
		`dig @ns1.example.com example.com any`
		If the dig is directed at a nonauthoritative server, it will return only the domain NS RRs, which can then be used to to issue the above query.

`q-class`	`IN`
	`ANY`
	`HESIOD`
	`CHAOS`
		The default is `IN`. May be optionally preceded with `-c` in the identified option format. The value `ANY` is a valid option for both `q-type` and `q-class`, and to ensure the correct value is used (to disambiguate in the jargon), always specify both `q-type` and `q-class` when using this value as shown here (the lines beginning with `//` are comments and should not be entered):
		`// this will get any record for class IN only`
		`dig example.com any`
		`// this will get any record for any class`
		`dig example.com any any`
		Alternatively, you can use an identified option format with `-c` for `q-class` and `-t` for `q- type`. When the identified option format is used, the parameter order is not important as shown here:
		`dig -c any -t any example.com`
		See the entry for `d-opt` for the identified option format.

`q-opt`		The following options are preceded with a plus (+) and control how the resulting DNS query operates. Multiple values may appear in a single command.

Continued

Table 9-4. *Continued*

Parameter	Value	Description
	`bufsize=bytes`	Defines the number of bytes to be advertised in an EDNS0 OPT meta (or pseudo) RR. May be set to any value in range `0` to `65535`. Only used with the `dnssec` option (the default is `4096`).
	`domain=name`	Replaces the default domain name (found in `resolv.conf`).
	`ndots=num`	Defines the minimum number of dots that must appear in a domain name before it is used as a qualified name. Domain names with a lower number of dots will have any default domain name (from `resolv.conf`) added before the query is issued (the default is 1).
	`[no]aaonly`	Controls whether to use authoritative query only (the default is `noaaonly`).
	`[no]aaflag`	Synonym for `[no]aaonly`.
	`[no]additional`	Controls whether to print the `ADDITIONAL SECTION` (the default is `additional`).
	`[no]adflag`	Controls setting the AD flag. Setting this flag has no effect on a query and is provided for completeness only. (the default is `noadflag`)
	`[no]all`	Sets or unsets all flags that control printed values for example `additional` or `comments` (the default is `all`).
	`[no]answer`	Controls whether to the print `ANSWER SECTION` (the default is `answer`).
	`[no]authority`	Controls whether to print the `AUTHORITY SECTION` (the default is `authority`).
	`[no]besteffort`	Controls whether `dig` will attempt to print malformed responses (the default is `nobesteffort`).
	`[no]comments`	Controls whether to print comments (the default is `comments`).
	`[no]cdflag`	Sets the CD (Checking Disabled) bit, which inhibits a security-aware name server from performing data authentication on signed zones (must be used with the `dnssec` option) (the default is `nocdflag`).
	`[no]cl`	Controls whether to print class information (the default is `cl`).
	`[no]cmd`	Controls whether to echo valid `dig` command-line arguments (the default is `cmd`).
	`[no]defname`	Synonym for `[no]search`.
	`[no]dnssec`	Controls whether to set the DNSSEC OK (`DO`) bit in the OPT pseudo header, thus requesting a security-aware name server to provide security information (the default is `nodnssec`).
	`[no]fail`	Controls whether `dig` will stop processing if it receives a `SERVFAIL` message to one of the default name servers listed in `resolv.conf` (the default is `fail`).
	`[no]identify`	Only valid with `short` option and suppresses or prints the name server identity (the default for the `short` option is `noidentify`).
	`[no]ignore`	Controls whether to ignore truncation errors rather than retry using TCP (the default is `noignore`).
	`[no]multiline`	Displays long RRs in standard parentheses format for multiple-line display. (the default is `nomultiline`).

Parameter	Value	Description
	[no]nssearch	If set, dig will attempt to obtain the SOA RRs for each authoritative name server for the domain name being queried (the default is nossearch).
	[no]question	Controls whether to print QUESTION SECTION (the default is question).
	[no]qr	Controls whether to print the outgoing query (the default is noqr).
	[no]recurse	Controls recursive query behavior. Recursion is automatically inhibited when nssearch and trace are invoked (the default is recurse).
	[no]search	Controls whether to use any domain or searchlist parameters in resolv.conf (the default is nosearch).
	[no]short	Controls whether to display only the answers to the query; for instance, in an A query, short will only display the IP address(es) (the default is noshort).
	[no]sigchase	Controls whether signature chains (chains of trust) will be followed or not for signed zones. This option is not enabled by default and requires dig to be built with DDIG_SIGCHASE.
	[no]stats	Controls whether to display dig statistics (the default is stats).
	[no]tcp	Controls whether to use TCP (tcp) or UDP (notcp) for queries (the default is notcp unless AXFR or IXFR is used).
	[no]topdown	Controls whether signature validation is carried out top-down. Not enabled by default and requires dig to be built with DDIG_SIGCHASE (the default is notopdown).
	[no]trace	Using the trace option causes dig to inhibit its default recursion and issue queries for the requested name to the root-servers and follow (and print) all referrals until an authoritative name server for the domain name is reached (the default is notrace).
	[no]ttlid	Controls whether to print TTL (the default is ttlid).
	[no]vc	Synonym for [no]tcp.
	retry=num	Controls the number of query retries to each server (the default is 2).
	time=secs	Controls the query timeout period (the default is 4 secs).
	tries=num	Controls the number of tries to each server (the default is 3).
	trusted-key=key	Defines the base64 material to be used as a trusted key when chasing signatures. Not enabled by default and requires dig to be built with DDIG_SIGCHASE.
d-opt		The following options control how dig operates and are preceded with a minus (-). Multiple options may appear in a single command line.
	-4	Use IPv4. Only valid for dual-stack (IPv4/IPv6) servers.
	-6	Use IPv6. Only valid for dual-stack (IPv4/IPv6) servers.
	-b	Defines the IP address to be used in the outgoing dig (query) message. Only required on a multihomed server.

Continued

Table 9-4. *Continued*

Parameter	Value	Description
	`-c`	Indicates a `q-class` argument follows (this is the identified option format) and can be used as a convenience or to disambiguate from the same `q-type` options.
	`-f filename`	Specifies a file containing batch commands. Any options specified on the command line will be in effect during the batch run, that is, they are global. Lines beginning with `;` or `#` or `\n` in the batch file are ignored and may be used as comment or white-space lines. Each line in the batch file will represent a single command-line query.
	`-i`	If set in conjunction with an IPv6 reverse-map request (using the `-x` option), this will use the `IP6.INT` (deprecated) reverse-map domain name rather than the current `IP6.ARPA` domain name.
	`-k dir:key`	Signs the message with TSIG using the `key` file in `dir`.
	`-h`	Displays a short list of the `dig` options available and exits.
	`-p port`	Changes the port used for queries to `port` (the default is `53`).
	`-t`	Indicates a `q-type` (RR type) argument follows (this is the identified argument format).
	`-v`	Displays the `dig` version number and exits.

	`-x`	Specifies that inverse notation is being used as shown here (the lines beginning with `//` are comments and should not be entered):
		`// this will fail NXDOMAIN (not found)`
		`dig 192.168.2.53`
		`// instead use`
		`dig -x 192.168.2.53`
		`// OR if you are a masochist!`
		`dig 53.2.168.192.in-addr.arpa ptr`

	`-y key`	Allows the user to enter the base64 shared secret to be used in a TSIG transaction. This both a long process and extremely dangerous. Use only if desperate and the `-k` option is not viable.

`dig` Examples

The following examples are designed to illustrate techniques and should not be taken literally—rather use them as a starting point for experimentation and exploration on a domain of your choice.

`dig` Host Query

Here is a simple host lookup that defaults to an A RR:

```
# dig www.example.com
```

The preceding command could have been written as follows (uses a positional argument so the order is important):

```
# dig www.example.com a
```

Or again, the same command using the identified option format—order is not important:

```
# dig -t a www.example.com
```

Contrast the previous output with the short response by using this command:

```
# dig -t a www.example.com +short
```

As noted previously, type `ANY` will always obtain more interesting results by providing all RRs with any given name.

```
# dig www.example.com any
```

This command forces use of the name server at 192.168.2.224 for the query:

```
# dig @192.168.2.224 www.example.com a
```

To force use of the named server at `ns1.example.com` for the query, use the following:

```
# dig @ns1.example.com www.example.com a
```

The next command is a reverse-map query that returns PTR RR:

```
# dig -x 192.168.2.224
```

`dig` Domain Query

Here is a quick domain lookup that returns all RRs without labels, the domain apex or root, and typically gets SOA, NS, MX, and others. If a nonauthoritative server is used, it returns only the NS RRs:

```
# dig @ns1.example.com example.com any
```

The corresponding identified option format—order not important:

```
# dig @ns1.example.com -t any example.com
```

The next command forces use of the name server at 192.168.2.224 for the query:

```
# dig @192.168.2.224 example.com any
```

This forces use of the name server at `ns1.example.net` for the query:

```
dig @ns1.example.net example.com a
```

dig Multiple Queries

`dig` will accept multiple queries per command line—as long as each query is clearly identified (or disambiguated). This multiple domain lookup returns nonlabel RRs (at domain apex) for both domains:

```
# dig example.com any example.net any
```

The following multiple domain lookup returns A RR for the first domain and domain apex RRs for the second domain:

```
# dig example.com example.net any
```

This multiple domain lookup returns apex RRs for the first domain and an A RR for the second:

```
# dig example.com any example.net
```

If a command line starts with one format, it must be consistent—this fails on the second query:

```
# dig example.com -t any example.net any
```

But this format works for both:

```
# dig example.com -t any example.net -t any
```

And this really does work—though how useful it would be is questionable!

```
# dig example.com any example.net any example.org any
```

And this works as well:

```
# dig www.example.com www.example.net fred.example.net
```

dig Output

The following shows the output from a simple `dig` command to the sample `example.com` zone using one of the authoritative name servers for the zone. Chapter 11 shows the output from a dig command issued to a DNSSEC signed zone, and Chapter 15 contains the output from a dig command to the root-servers.

```
# dig @ns1.example.com www.example.com
; <<>> DiG 9.3.0 <<>> @ns1.example.com www.example.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 826
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.com.  IN  A

;; ANSWER SECTION:
www.example.com.  86400  IN  A  10.1.2.1
www.example.com.  86400  IN  A  192.168.254.3
www.example.com.  86400  IN  A  172.16.2.1
www.example.com.  86400  IN  A  192.168.2.5

;; AUTHORITY SECTION:
example.com.  86400  IN  NS  ns1.example.com.
example.com.  86400  IN  NS  ns2.example.com.

;; ADDITIONAL SECTION:
ns1.example.com.  86400  IN  A  192.168.2.6
ns2.example.com.  86400  IN  A  192.168.23.23

;; Query time: 31 msec
;; SERVER: 192.168.2.3#53(ns1.example.com)
;; WHEN: Tue May 31 20:16:25 2005
;; MSG SIZE  rcvd: 165
```

The output from a dig command is a formatted version of the binary, or *wire format*, message response to the query formed from the dig command parameters (unless the +short option is used). The detailed layout of the message is described in Chapter 15. The preceding response reflects a typical positive response to a dig command and includes the following items:

- The >>HEADER<< is an interpreted version of the message header. The flags and values of the status fields are defined in the next section, "dig Response Values."
- The QUESTION SECTION reflects the original query that is being answered, which was in this case a query for the A RR of www.example.com.
- The ANSWER SECTION provides the four A RRs for www.example.com that fully answer the question in this case. If the ANSWER SECTION is present but contains no entries, then the query was not successful, and the status field in the HEADER typically provides the reason unless the response was a *referral*, in which case the status field will be NOERR (see Chapter 15 for a referral dig response).

- The `AUTHORITY SECTION` provides the NS RRs of the servers that are authoritative for the domain `example.com`.
- The `ADDITIONAL SECTION` provides information that may be useful to the server; in this case, it is the A RRs of the name servers.

`dig` Response Values

This section describes the various fields that are present in the `>>HEADER<<` output to a `dig` command.

DNS Flags

The values of the flags in the `dig` command `>>HEADER<<` are an interpretation of various bits set in the message header, which is described in Chapter 15, Table 15-2.

`qr`: Query Response. This flag is set in the preceding `dig` response. This flag simply means that this is a response to a query and will always be set in a `dig` response.

`aa`: Authoritative Answer. This flag is set in the preceding `dig` response. This flag means that either the response came from an authoritative name server for the domain, which is true for the preceding case, or this was the first time the data was read from an authoritative name server into a caching name server. In the latter case, if the `dig` command is immediately reissued, the `aa` bit will not be set because it will have been read from the cache, in which case the `aa` bit is never set.

`rd`: Recursion Desired. This flag is set in the preceding `dig` response. This flag is copied from the query request (the `dig` command) and means that the incoming query (the `dig` message) requested recursive support.

`ra`: Recursion Available. This flag is set in the preceding `dig` response. This flag means that the responding name server (`ns1.example.com`) supports recursive queries.

`ad`: Authenticated Data. This flag is not set in the preceding `dig` response. This flag is only valid with DNSSEC (the `+dnssec` option was set in the `dig` command) and indicates the target name server is security aware (the `dnssec-enable yes;` statement is present in the `named.conf` file), the query response came from a signed zone, and the data was fully authenticated.

`cd`: Checking Disabled. This flag is not set in the preceding `dig` response. This flag is only valid with DNSSEC and indicates the issuing query wishes to bypass any DNSSEC validation sequence to be performed by the name server when accessing a signed zone. This flag will only be set in the response to a `dig` command if the `+cdflag` option is used.

`do`: DNSSEC OK. This flag is not set in the preceding `dig` response. This flag is only valid with DNSSEC and is set in the extended `OPT PSEUDOSECTION` that is always present in DNSSEC transactions (see Chapter 11). It will only be set in a `dig` response if the `+dnssec` option is used and the target name server is security aware (a `dnssec-enable yes;` statement is present in its `named.conf`).

DNS Status

The values of the status field in a dig response are an interpretation of the RCODE field of the message header and are described in Chapter 15, but reproduced here for convenience:

NOERR: No error condition.

FORMERR: Format error—the name server was unable to interpret the query.

SERVFAIL: Server failure—the name server was unable to process this query due to either a problem with the name server or a requested feature that cannot be satisfied, such as a recursive request to an authoritative-only name server.

NXDOMAIN: Name error—meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.

NOTIMP: Not implemented (versions of BIND prior to 9.3 would respond with NOTIMPL)—the name server does not support the requested operation.

REFUSED: The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester or a name server may not wish to perform a particular operation, for example, a zone transfer (AXFR).

YXDomain: Name exists when it should not (RFC 2136).

YXRRSet: RRset exists when it should not (RFC 2136).

NXRRSet: RRset that should exist does not (RFC 2136).

NotAuth: Server not authoritative for zone (RFC 2136).

NotZone: Name not contained in zone (RFC 2136).

BIND named-checkconf Utility

The named-checkconf utility verifies the syntax of the named.conf file that controls BIND's operation. Whenever changes are made to the named.conf file, this utility should be run before restarting BIND. If you do not do this and you do have a problem, your name server will be off-line while you scramble around making changes—under pressure. If the named.conf file has no errors, the utility provides silent confirmation—it outputs nothing. No news in this case is indeed good news.

named-checkconf Syntax

```
named-checkconf [-j] [-t directory] [-v] [-z] [filename]
```

named-checkconf Options

Table 9-5 describes the options available with the named-checkconf command.

Table 9-5. `named-checkconf` *Options*

Argument	Meaning
`-j`	Relevant only with the `-z` option when checking zone files that are dynamically updated and causes the utility to check any journal files (`zonefile.jnl`).
`-t directory`	Chroots to `directory` when running the check to ensure the correct permissions are available for `include` statements when run in a similar chrooted environment; that is, the `-t directory` argument is the same as would be used on the BIND command line when running in a chroot jail or sandbox.
`-v`	Prints the `named-checkconf` version number and exits.
`-z`	Causes `named-checkconf` to load and verify the master zone files specified in `named.conf`. The utility displays the zone file name and the SOA serial number for each zone found.
`filename`	Optional. The name of the configuration file to be checked. If not specified, it defaults to `/etc/named.conf` on Linux or `/etc/namedb/named.conf` on BSD systems and `%SystemRoot\system32\dns\etc\named.conf` on Windows.

BIND named-checkzone Utility

The `named-checkzone` utility verifies the nominated zone file and provides a useful method to ensure correctness of a zone file before loading into a live name server.

named-checkzone Syntax

```
named-checkzone [-c class] [-d] [-D] [-j] [-k mode] [-n mode] [-o filename]
    [-q] [-t directory] [-v] [-w directory] zonename filename
```

named-checkconf Options

Table 9-6 defines the options available to control processing of the `named-checkzone` utility.

Table 9-6. *BIND* `named-checkzone` *Options*

Options	Parameter	Meaning and Use
`-c`	`class`	Zone class. The default is `IN`. May take values `CH` (CHAOS) or `HS` (HESIOD).
`-d`		Turns on debugging.
`-D`		Writes zone file in canonical (alphabetic by host name) order to `stdout` (console). If used with the `-o` option, this option will write output to a file.
`-j`		If using DDNS, this option will read any journal file when checking the zone.
`-k`	`mode`	Performs `check-name` functions (see the `check-name` statement in Chapter 12) to verify that host names are in compliance with RFC 952 and RFC 1123 formats. The value of mode may be `fail`, `warn`, or `ignore`, which indicates the action to be taken if the check fails. Many modern RRs (notably SRV) will fail these checks.

Options	Parameter	Meaning and Use
-n	mode	Causes all NS RRs to be verified for a corresponding A RR (a so-called glue record). The mode value indicates the action if the check fails and may take one of the values `fail`, `warn`, or `ignore`.
-o	filename	Only valid when used with `-D` and defines an optional output file name.
-q		Quiet mode. Displays no error messages, just the termination code.
-t	directory	Chroots to `directory` when running the check to ensure the correct permissions are available for `include` statements when run in a similar chrooted environment; that is, the `-t directory` argument is the same as would be used on the BIND command line when running in a chroot jail or sandbox.
-w	directory	Defines a directory that will be used for relative addressing in `$INCLUDE` directives. The default is `/var/named`.
-v		Displays the version number of `named-checkzone` and exits.
zonename		The domain name of the zone being checked.
filename		The name of the zone file to be checked.

rndc

The `rndc` utility controls the name server and may, depending on the value of the `controls` clause in the `named.conf` file, be run from one or more local or remote locations, including Windows NT 4.0 (Server or Workstation) and Windows 2000 (Server or Pro). BIND defaults to enable `rndc` access from localhost (127.0.0.1) whether required or not. If `rndc` will not be used, it must be explicitly disabled using a blank `controls` clause, that is, `controls {};`.

The `rndc` utility uses TCP to access the name server on port 953 by default and requires a shared secret to provide TSIG-like authentication on each transaction. The various features supported by the utility are defined in a configuration file called `rndc.conf`. However, to make *initial* setup a trivial process, `rndc` will operate without an `rndc.conf` file and with a default shared secret. The required default secret is created by running the following command:

```
rndc-keygen -a
```

This command generates two files in the directory in which `named.conf` resides. The first is `rndc.key` which contains a default `key` clause, used by both `rndc` and BIND, and `rndc.conf.sample`, which may be edited to provide additional control of `rndc` operation. This default configuration is sufficient to support a localhost service but should be enhanced if remote access is required. The `rndc` command-line options are described next followed by the format of the `rndc.conf` file. The commands available when using `rndc` are then documented, and finally a worked example is shown that supports access to multiple name servers from a remote host.

Note The default shared secret name is nominally defined to be `"rndc-key"`; the Fedora Core and some other Linux distributions, however, seem to use `"rndckey"`. The `rndc.key` file should be inspected to verify the `key` clause name used by any specific distribution.

`rndc` Syntax

`rndc` `[-c config-file] [-k key-file] [-p port] [-s server] [-V] [-y key-id] command`

`rndc` Options

Table 9-7 describes the command-line options available with `rndc`.

Table 9-7. `rndc` *Options*

Option	Parameter	Meaning and Use
`-c`	`config-file`	By default, the configuration file is called `rndc.conf`, but may be changed using this option.
`-k`	`key-file`	If this option is not used, the key is assumed to be in the `rndc.conf` file. If an `rndc.conf` file is not present, the default `rndc.key` file created by the `rndc-confgen -a` command is used. The `key-file` defines a shared secret (HMAC-MD5 algorithm), which was generated by the `rndc-confgen` utility.
`-p`	`port`	The default port used by `rndc` is `953`. This option may be used to change the port number and must be supported by an equivalent `inet` statement in the `controls` clause of BIND's `named.conf` file to change the port number.
`-s`	`server`	The server to be accessed, which may be defined as either a name or an IP address (IPv4 or IPv6). If none is specified, the default value from the `rndc.conf` file is used. If neither is present (the default configuration), localhost is assumed.
`-V`		If used, this turns on verbose logging.
`-y`	`key-name`	Uses the `key-name` when connecting to the server and must be defined in a `key` clause in the `rndc.conf` file. If none is specified, a `key` statement will be looked for in the server clause for the server specified in the `-s` option. If this is not present, the `default-key` statement in the `options` clause is checked, and if not present (the default configuration), `"rndc-key"` is used from the `rndc.key` file created by the `rndc-confgen -a` command.
`command`		The `rndc` command to be executed, for instance, `halt`.

`rndc.conf` Clauses and Statements

The `rndc.conf` file controls the connection and authentication of the `rndc` utility to one or more name servers. This file has a similar structure and syntax to `named.conf`, but with a significantly reduced number of clause types and a limited number of statements.

■ **Note** To provide consistent terminology, this book uses the term *clause* to describe an entity that starts with the name of the clause, which is enclosed in braces and terminated by a semicolon and may contain a number of *statements*. The rationale for this policy is contained in Chapter 12.

The rndc.conf file may contain comments that are exactly the same as those used in the named.conf file: multiline C style (/* */), single line C++ style (//), or single-line Unix style (#). The rndc.conf file may take a single options clause, one or more server clauses, and one or more key clauses.

The options Clause

A single options clause may be defined in the rndc.conf. The options clause defines the default server, authentication key, and port that will be used if not overridden on the command line. It may contain three statements. The first is the default-server statement, which defines the name or IP address of the server that will be used unless overridden by the -s option. If no default-server is specified, localhost is assumed. The second is the default-key statement, which defines the default key-name as a quoted string. It is used if the -y option is not supplied on the command line. If no default-key is defined, the default "rndc-key" is used from the rndc.key file created by rndc-confgen -a. The third is the default-port statement, which, in the absence of a -p option, will define the port number to be used for connection to the server. In the absence of either, port 953 is used. The following example shows an options clause:

```
// rndc.conf fragment
options {
   default-server 127.0.0.1;
   default-key "my-key";
   default-port 3346;
};
```

In the preceding case, a corresponding inet statement in the controls clause of named.conf will be required to specify port 3346 and reference "my-key".

The server Clause

One or more server clauses may exist. The server clause defines a specific name server (either a name or an IP address) that rndc may connect to. It may take two statements. The first is the key statement, which defines the specific key to be used to connect to the server. If not present, the default-key statement is used. The second, the port statement, defines the port number to be used with the specific server. If not present, the default-port statement from the options clause is used; otherwise the rndc default of 953 is used. The example that follows illustrates a typical server clause:

```
// rndc.conf fragment
server "ns1.example.net" {
   key "ns1.example.net";
   port 953;
};
```

The key Clause

One or more key clauses may be defined. The key clause defines the name of a shared secret key that may be used by one of the servers being accessed. The key clause may take two statements. The algorithm statement identifies the encryption algorithm and *must* take the case-insensitive value hmac-md5. This is followed by the secret statement, which contains the base64 encoding of the key enclosed in quotes. The example that follows illustrates a key clause:

```
// rndc.conf file fragment
key "ns1.example.net" {
   algorithm hmac-md5;
   secret "c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K";
};
```

There are two ways to generate the keys clause: either using the rndc-confgen utility (see the section that follows)—this creates a complete key clause that may be edited into the rndc.conf file without change—or using the dnssec-keygen command (see the section “dnssec-keygen Utility” later in the chapter), in which case some editing will be required. There must also be a corresponding key clause in the named.conf for the name server being accessed. The key clause in the named.conf and rndc.conf files are exactly the same, and as with named.conf, key clauses must appear before being used.

rndc Configuration Examples

To set up a default configuration—one in which rndc access is only allowed from localhost—requires no modifications to the named.conf file, and the default authentication key file (rndc.key) is set up using the following command:

```
rndc-confgen -a
```

The following example shows the configuration of rndc.conf on the host bill.example.com (192.168.2.15) to allow access to two name servers with names of ns1.example.com (IP 192.168.2.3, which uses a port number of 3396) and ns2.example.com (IP 192.168.2.4, which uses the default port 953). Each server will use a separate key for security. There are many ways to create the rndc.conf; one such method is shown that keeps typing and editing to a minimum.

The key for use with ns1.example.com is generated using the following command (see the section “rndc-confgen” for details):

```
rndc-confgen -k ns1.example.com -p 3396 -s 192.168.2.3 > rndc.conf
```

The command creates a 128-bit HMAC-MD5 shared secret (the rndc-confgen default) with a name of ns1.example.com (the -k option). The -p option is used to create the default-port statement, and the -s option defines the IP address used in the default-server statement. The rndc.conf file as shown here is created:

```
# Start of rndc.conf
key "ns1.example.com" {
   algorithm hmac-md5;
   secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
};
```

```
options {
   default-key "ns1.example.com";
   default-server 192.168.2.3;
   default-port 3396;
};
# End of rndc.conf

# Use with the following in named.conf, adjusting the allow list as needed:
# key "ns1.example.com" {
#    algorithm hmac-md5;
#   secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
# };
#
# controls {
#    inet 192.168.2.3 port 3396
#    allow { 192.168.2.3; } keys { "ns1.example.com"; };
# };
# End of named.conf
```

The `rndc-confgen` utility helpfully outputs comments to indicate the required changes to the `named.conf` file for the name servers being accessed, which, since they are comments, you can leave in the file or, if they offend you, delete them. This file will work for `ns1.example.com`, but to allow the same `rndc.conf` file to be used for `ns2.example.com`, we create another key again using `rndc-confgen` (though we could use `dnssec-keygen` and some trivial editing).

```
rndc-confgen -k ns2.example.com -s 192.168.2.4 >>rndc.conf
```

This command again creates a 128-bit shared secret using the HMAC-MD5 algorithm (the `rndc-confgen` default) with a name of `ns2.example.com`. It also creates an additional `options` clause that will be removed. The resulting output is appended to the `rndc.conf` file created by the first `rndc-confgen` command as shown next (the comment line containing "`// start of second (appended) rndc.conf file`" was added to indicate the split and would not be present). The presence of multiple comment lines may appear confusing, but is an artifact of the super-friendly `rndc-confgen`'s willingness to help the user. The output has been left intact, since this reflects the real output from this sequence of commands:

```
# Start of rndc.conf
key "ns1.example.com" {
   algorithm hmac-md5;
   secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
};

options {
   default-key "ns1.example.com";
   default-server 192.168.2.3;
   default-port 3396;
};
# End of rndc.conf
# Use with the following in named.conf, adjusting the allow list as needed:
# key "ns1.example.com" {
#    algorithm hmac-md5;
#    secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
# };
#
# controls {
#    inet 192.168.2.3 port 3396
#    allow { 192.168.2.3; } keys { "ns1.example.com"; };
# };
# End of named.conf

// start of second (appended) rndc.conf file
# Start of rndc.conf
key "ns2.example.com" {
   algorithm hmac-md5;
   secret "oSbqEQ7KVw3PZlisH+g/XQ==";
};
options {
   default-key "ns2.example.com";
   default-server 192.168.2.4;
   default-port 953;
};
# End of rndc.conf

# Use with the following in named.conf, adjusting the allow list as needed:
# key "ns2.example.com" {
#    algorithm hmac-md5;
#   secret "oSbqEQ7KVw3PZlisH+g/XQ==";
# };
#
# controls {
#    inet 192.168.2.4 port 953
#    allow { 192.168.2.4; } keys { "ns2.example.com"; };
# };
# End of named.conf
```

To create the final `rndc.conf` file, three edits must be performed using your favorite editor:

1. Move the `key` clause for `ns2.example.com` to just below the key clause for `ns1.example.com` (`key` clauses must always appear before they are referenced).

2. Add a `server` clause for `ns2.example.com`.

3. Delete the second `options` clause that was generated by the last `rndc-confgen` file, which is not required.

The various comment fields from both `rndc-confgen` commands will be retained because they provide some useful information. The final `rndc.conf` file will look as shown here (comments beginning with // have been inserted to show the edits described earlier):

```
# Start of rndc.conf
key "ns1.example.com" {
   algorithm hmac-md5;
   secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
};
// moved ns2.example.com key clause
key "ns2.example.com" {
   algorithm hmac-md5;
   secret "oSbqEQ7KVw3PZlisH+g/XQ==";
};

options {
   default-key "ns1.example.com";
   default-server 192.168.2.3;
   default-port 3396;
};
server ns2.example.com { // create server clause
   key ns2.example.com;
   port 953; // required because of default-port in options clause
};
# End of rndc.conf

# Use with the following in named.conf, adjusting the allow list as needed:
# key "ns1.example.com" {
#    algorithm hmac-md5;
#    secret "tRNNxQ24OB7Gwc/XhS+VLQ==";
# };
#
# controls {
#    inet 192.168.2.3 port 3396
#    allow { 192.168.2.3; } keys { "ns1.example.com"; };
# };
# End of named.conf

// start of second (appended) rndc.conf file
# Start of rndc.conf
// deleted second options clause
# End of rndc.conf

# Use with the following in named.conf, adjusting the allow list as needed:
# key "ns2.example.com" {
#    algorithm hmac-md5;
#   secret "oSbqEQ7KVw3PZlisH+g/XQ==";
# };
#
# controls {
#    inet 192.168.2.4 port 953
#    allow { 192.168.2.4; } keys { "ns2.example.com"; };
# };
# End of named.conf
```

The two key clauses and controls clause must be made available in the named.conf file for the respective name servers, ns1.example.com and ns2.example.com, as shown in the comment lines. The key clauses used in both rndc.conf and named.conf are identical, so it is also possible to use the actual key clause, not the commented versions, which will save some editing. To run rndc to connect to ns1.example.com (the default in the rndc.conf file), the following command would be used:

```
rndc stop
```

This form of the command uses the defaults defined in the options clause to connect to ns1.example.com and therefore requires no options. To run rndc with ns2.example.com, the following command would be used:

```
rndc -s ns2.example.com stop
```

This form uses the values defined in the server clause for ns2.example.com.

■**Note** The rndc.conf file contains extremely sensitive shared-secret information and should be read protected from nonessential users. The key clauses that will added to the named.conf files for the servers ns1.example.com and ns2.example.com should use the normal technique of placing them in a separate file and including them in the named.conf file. The included files should then be read protected from nonessential users.

The next section describes the commands that may be used with rndc.

rndc Commands

The rndc utility provides a number of commands to control the operation of the name server as shown in Table 9-8.

Table 9-8. rndc *Commands*

Command	Options	Meaning or Use
dumpdb		Dumps the cache to the default named_dump.db file in the location defined by a directory statement (or the location and file name defined by the dump-file statement of named.conf).
flush	[view]	Without the optional view, flushes all current caches. If view is used, it will only flush the cache for the specified view name.
freeze	zone [class [view]]	Stops all dynamic updates on the zone and updates the zone file with any outstanding entries in the .jnl file. The dynamic update is reenabled with a thaw zone command. A view within class may be optionally selected.

Command	Options	Meaning or Use
halt		Causes the name server to be immediately halted. The name server cannot be restarted using an rndc command.
notrace		Sets debug level to 0.
querylog		Toggles logging of all queries to the relevant log file.
reconfig		Reloads the named.conf file and new zones only.
refresh	zone [class [view]]	Schedules a zone transfer of a nominated slave zone. May optionally define the class and view to be transferred.
reload	[zone [class [view]]]	Reloads named.conf and all zone files but retains all cache entries. If the optional zone parameter is used, it will only reload the nominated zone, and the class and view may be additionally selected.
retransfer	zone [class [view]]	Forces a zone transfer of a nominated slave zone. May optionally define the class and view to be transferred.
stats		Dumps current zone statistics to the default /var/named/named.stats file (or the name defined in the statistics-file statement of named.conf). Only valid if the zone-statistics yes; statement appears in the named.conf file.
status		Displays various information about the name server including the current status of query logging.
stop		Causes a graceful stop of the name server, allowing any dynamic update and zone transfers to complete. The name server cannot be restarted by an rndc command.
thaw	zone	Enables dynamic updates to the specified zone. This is issued after a freeze zone command. unfreeze is a synonym for thaw.
trace	[level]	If issued without the optional level parameter, this will increment the current debug level by one. The level parameter explicitly sets the debug level.

rndc-confgen Utility

The rndc-confgen utility is used to generate HMAC-MD5 shared-secret (symmetric) keys and a shell rndc.conf configuration file for use with rndc. When the -a option is used, it creates a default configuration (rndc.key file) for use with localhost access only. When used without the -a option, rndc-confgen writes all output to stdout that must be captured to a file using a redirection command. The output file contains comments describing the format of the inet statement required in the controls clause of named.conf.

rndc-confgen Syntax

```
rndc-confgen [-a] [-b keysize] [-c key-file] [-h] [-k keyname] [-p port]
    [-r randomdev] [-s address] [-t chrootdir] [-u user] [>outfile]
```

`rndc-confgen` Options

Table 9-9 describes the options available for use with the `rndc-confgen` utility.

Table 9-9. `rndc-confgen` *Options*

Option	Parameter	Meaning or Use
`-a`		This option creates a configuration for use with `rndc` in its default mode of operation (with localhost only). The file `rndc.key` is written into the same directory as `named.conf` (and is read by both `named` and `rndc`) and an `rndc.conf.sample` file, which may be edited for subsequent configuration of `rndc`.
`-b`	`keysize`	Defines the key size for use with the HMAC-MD5 MAC algorithm. May take a value in the range `1` to `512` and defaults to `128` if not defined.
`-c`	`key-file`	When used with the `-a` option, defines an alternative file name (replaces the default `rndc.key`). If this option is used, the `key-file` name must be included in the `named.conf` file, since BIND only looks by default for `rndc.key`.
`-h`		Displays a list of the options and exits.
`-k`	`key-name`	The `key-name` to be used when creating the `key` clause. The default is `"rndc-key"`, though some distributions change this to `"rndckey"`.
`-p`	`port`	The port number to be used for `rndc` connections. The default is 953. This option overrides the `default-port` statement in the `rndc.conf` file.
`-r`	`randomdev` `keyboard`	Defines the source of randomness used to generate keys. The default is to use `/dev/random`, in which the OS captures randomness (entropy) from various system events. If significant key generation is being done, this source may become depleted, and the utility will apparently freeze, waiting for entropy. Typing any characters on the keyboard will allow the system to capture randomness from the typing intervals. Many systems also provide `/dev/urandom`, which is faster but significantly less random, leading to less secure keys. If neither device is present, the value `keyboard` may be used to force use of the keyboard technique described earlier.
`-s`	`address`	The IP address (IPv4 or IPv6) to which connection will be made. The value overrides the `default-server` statement of the `options` clause in the `rndc.conf` file.
`-t`	`chrootdir`	Only valid with the `-a` option and defines the directory in which BIND will be run chrooted; that is, `directory` will be the same value as used with BIND's `-t` command-line option for defining a chroot base directory. A copy of the `rndc.key` file is placed in this directory.
`-u`	`user`	Defines the user (UID) name whose permission will be applied to the `rndc.key` file. If used in conjunction with the `-a` option, only the copy in the `-t` directory will be allocated the defined user (UID) permission.
`>outfile`		If used without the `-a` option, output from `rndc-confgen` is written to `stdout`, and therefore the standard redirection command will capture the data to the `outfile` name.

BIND nsupdate Utility

The nsupdate utility allows dynamic updating of the zone files for which the name server is the Primary master—the name server that appears on the SOA RR for the zone. nsupdate typically accepts commands from the console but may optionally be used to read commands from a batch file. Zones may not be added or deleted using nsupdate, but a zone's SOA RR may be edited. Zones being dynamically updated should not be manually edited. Should manual editing be required, the server should be stopped, the manual edit carried out, the .jnl files for the zone deleted, and the server restarted; alternatively, you could use the rndc freeze zone command, edit the zone, delete any .jnl files for the zone, and then use rndc thaw/unfreeze zone to enable dynamic updates for the zone. nsupdate uses UDP by default, but if the update is greater than 512 bytes, TCP will be used. nsupdate may be secured using either TSIG or SIG(0) transaction security—both methods, including illustrative examples, are described in Chapter 10. The keys used in both TSIG and SIG(0) are generated using the dnssec-keygen utility. Dynamic updates are controlled by the allow-update or update-policy statements in view, options, or zone clauses of the named.conf file (see Chapter 12). Dynamic updates can be performed on both normal and signed (DNSSEC) zones as described in Chapter 11.

nsupdate Syntax

```
nsupdate [-d] [-k key-file | -y keyname:secret |] [-r udpretries] [-t timeout]
     [-u interval] [-v] [filename]
```

nsupdate Options

Table 9-10 defines the options available with the nsupdate utility.

Table 9-10. nsupdate *Options*

Option	Parameter	Meaning or Use
-d		Turns on debug mode.
-k	key-file	Defines the name of the key-file output when the dnssec-keygen program that created the key was run. This option must have the .private suffix appended on the command line, but both the .key and .private files must be available in the same directory.
-r	udpretries	Defines the number of retries for a dynamic update. The default is 3. The value of 0 means no retries are attempted.
-t	timeout	Defines the time in seconds before the update is regarded as having failed. The default is 300. The value 0 will disable timeout checking.
-u	interval	If an update fails, this option may be used to define the time in seconds between retries. The default is 3.
-v		By default, nsupdate will use UDP unless the block size is greater than 512 bytes, in which case TCP will be used. This option forces use of TCP for all updates.

Continued

Table 9-10. *Continued*

Option	Parameter	Meaning or Use
`-y`	`keyname:secret`	Allows a shared secret to be entered on the command line. This is a dangerous option and should only be used if there is no alternative. The `keyname` field is the name as it appears in the receiving server's `key` clause, and `secret` is the base64 material that comprises the secret key!
`filename`		The optional `filename` may be used to supply update commands from a nominated file. The default is to accept commands from `stdin` (the console).

`nsupdate` Command Format

`nsupdate` commands define the environment, the RRs to be deleted or added, and any required conditions (prerequisites) for the updates to take place. The prerequisites are optional and allow checks to be performed before the update is executed. Commands are built locally and only sent using either the `send` command or a blank line. Any number of RRs (and the required prerequisites) may be added or deleted in a single `send` operation, or individual RRs may be added or deleted in a single `send` operation.

The command formats and their meaning are defined in Table 9-11.

Table 9-11. `nsupdate` *Commands*

Command	Parameter	Meaning or Use
`answer`		Displays the results of the last `send` operation.
`class`	`IN|CH|HS`	The zone class. The default is `IN`.
`local`	`address [port]`	If not specified, the `nsupdate` utility sends updates using a random port number on the configured IP address for the host. This option may be used to define a specific IP and optionally a port number.
`key`	`name secret`	Has the same meaning and overrides the value of the `-y` option on the `nsupdate` command line.
`prereq nxdomain`	`name`	The following `update add` or `update delete` commands will only be executed if the defined `name` does not exist in the zone.
`prereq yxdomain`	`name`	The following `update add` or `update delete` commands will only be executed if the defined `name` does exist in the zone.
`prereq nxrrset`	`name [class] type`	The following `update add` or `update delete` commands will only be executed if the defined `name` and RR `type` do not exist in the zone. `class` is optional and, if not present, defaults to `IN`.
`prereq yxrrset`	`name [ttl] type`	The following `update add` or `update delete` commands will only be executed if the defined `name` and RR `type` do exist in the zone. `class` is optional and, if not present, defaults to `IN`.

Command	Parameter	Meaning or Use
prereq yxrrset	name [ttl] type data	The following update add or update delete commands will only be executed if the defined name, RR type, and data do exist in the zone.
quit		Terminates the nsupdate utility.
send		Sends the current command or commands, equivalent to a blank line being entered.
server	server-name [port]	Defines the name server to which the updates will be sent until the next server command is issued. The optional port parameter may be used to override the default port (53). If not specified, nsupdate will send updates to the Primary master name server for the zone.
show		Displays the last send operation.
update add	name [ttl] [class] type data	The RR to be added as it will appear in the zone file, for instance, update add fred 8600 IN A 192.168.2.3.
update delete	name [ttl] [class] type data	The RR that should be deleted, for instance, update delete fred A 192.168.2.3.
zone	zone-name	Defines the name of the zone that will be used for subsequent updates until another zone command is issued. If not supplied, nsupdate will attempt to guess the required zone from the update add and update delete commands.

nsupdate Example

The following sequence is used to add an MX record and its corresponding A RR for the domain example.com and is secured using SIG(0)—see Chapter 10:

```
cd /var/named/dynamic
# nsupdate -k Kexample.com.+001+00706.private
> server ns1.example.com
> zone example.com
> update add example.com. 36000 IN MX 10 mail2.example.com.
> send
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERR id: 0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
> update add mail2 36000 IN A 192.168.2.5
> send
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERR id: 0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
> quit
```

dnssec-keygen Utility

The `dnssec-keygen` utility is a general-purpose cryptographic key generation utility that generates keys for use with TSIG, SIG(0), TKEY, and DNSSEC (DNSSEC.bis) operations (see Chapters 10 and 11) and well as generic KEY or DNSKEY RRs and, with a modest edit, also for IPSECKEY RRs. When the utility is run, it outputs a `key-file` reference. The `key-file` is used subsequently in `dnssec-signzone` and other commands, and references two files created in the current working directory (the directory from which the command was run) with the following names:

```
Khostname.+algorithm+key-tag.private
Khostname.+algorithm+key-tag.key
```

where `K` is a fixed identifier, `hostname` is the host name value in FQDN format (terminated with a dot) from the `dnssec-keygen` command line (see "`dnssec-keygen` Options" later in this chapter), and + is a fixed separator. `algorithm` is a three-digit number identifying the key algorithm specified in the command line and takes the following values:

`001` = RSA-MD5

`002` = Diffie-Hellman

`003` = DSA

`005` = RSA-SHA-1

`157` = HMAC-MD5

The next + is a fixed separator, `key-tag` is a five-digit value (generated with a modified one's complement algorithm) used to identify this key from others that may have the same `hostname`. The `key-tag` is used explicitly, implicitly in other places, in the Delegated Signer (DS) RR of DNSSEC.bis (see Chapter 11).

The `.private` file contains the private key of a public key (asymmetric) algorithm, for example, RSA-SHA-1, or the shared secret in a symmetric algorithm, for example, HMAC-MD5. The `.key` file contains a formatted KEY or DNSKEY RR, depending on the `-k` and `-n` arguments that follow such that the file may be directly included in a zone file using the `$INCLUDE` directive (see Chapter 13).

■ **Caution** The `dnssec-keygen` utility always generates `.private` and `.key` files. When used with a shared-secret (symmetric) algorithm such as HMAC-MD5 for use in TSIG operations to secure DDNS or zone transfers, the `.key` file will contain a KEY RR with the shared secret! This is a potentially dangerous file and *must not* be included in any zone file; instead, it should be deleted immediately unless there is a very good reason to retain it, in which case it must be secured.

The `.private` file of any public key system contains highly sensitive information, and when it has been used for, say, zone signing, should be taken off-line, which may mean physically removing it from the system or moving to another location and securing with appropriate privileges.

When using dynamic update with signed zone files, however, the `.private` file of the Zone Signing Key (ZSK) must be on-line at all times and should be secured with minimal read permissions.

dnssec-keygen Syntax

```
dnssec-keygen  -a algorithm -b keysize -n type [ -c class ] [ -e ]
       [ -f flag ] [ -g generator ] [ -h ] [ -k ] [ -p protocol ]
       [ -r randomdev ] [ -s strength ]  [ -t type ]  [ -v level ] hostname
```

dnssec-keygen Options

Table 9-12 shows the various options available with the `dnssec-keygen` utility.

Table 9-12. `dnssec-keygen` *Options*

Option	Parameter	Meaning and Use
-a	algorithm	Defines the cryptographic algorithm for which the key is being generated and may take one of the following case-insensitive values: `RSAMD5`: Digital signature using MD5 digest and RSA public key (asymmetric) cryptography `RSASHA1`: Digital signature using SHA-1 digest and RSA public key (asymmetric) cryptography `DSA`: Digital signature using NIST Digital Signature Architecture `DH`: Diffie-Hellman public key (asymmetric) cryptography `HMAC-MD5`: Shared-secret Message Authentication Code (MAC) with MD5 digest
-b	keysize	Specifies the number of bits to be used in the key and depends on the algorithm being used: RSA-MD5 range 512 to 2048 (current RSA recommendation is 1024 but this changes over time) RSA-SHA-1 range 512 to 2048 (current RSA recommendation is 1024 but this changes over time) DSA range 512 to 1024 (must be a multiple of 64) DH range 128 to 4096 HMAC-MD5 range 1 to 512
-c	class	Defines the class of the KEY or DNSKEY RR generated. The default is `IN`. May take the case-insensitive value `IN` (Internet), `CH` (CHAOS), or `HS` (HESIOD).
-e		Valid only with RSA-MD5 and RSA-SHA-1—and specifies to use a large exponent when generating the key. Some cryptographic papers have suggested that use of a large exponent is more secure, but significantly increases computational resources required. The default is to use a normal exponent.
-f	flag	Defines a flag to be set in the `flags` field of the resulting DNSKEY or KEY RR. The only value currently supported is `KSK`, which defines for DNSSEC.bis operations that this a Key Signing Key, or KSK (see Chapter 11). The value of the `flags` field (see Chapter 13) will be set to `257` if this option is defined—indicating the SEP bit is set.
-g	generator	Used only for the Diffie-Hellman algorithm. Value may be either `2` or `5` and defines the generator of the prime number used in the algorithm. The default is to use the values defined in RFC 2539 and, if not possible, to use the value `2`.

Continued

Table 9-12. *Continued*

Option	Parameter	Meaning and Use
-h		Outputs a summary of the dnssec-keygen options and exits.
-k		If present, a KEY RR will be created in the .key file; if not present, a DNSKEY RR will be created. If the algorithm being used is either HMAC-MD5 or DH, the -k option is defaulted, thus creating a KEY RR.
-n	rr-type	Determines, together with the -k option, the value of the flags field on the KEY or DNSKEY RR created (see Chapter 13). Table 9-13 shows the possible values of rr-type (case insensitive) with the flags field and RR type generated.
-p	protocol	Defines the value of the proto field used in the KEY and DNSKEY RRs (see Chapter 13). This is currently restricted to the value 3, which is the default. This field had a historic usage in the KEY RR that was limited to the value 3 by RFC 3445.
-r	randomdev keyboard	Defines the source of randomness used to generate keys. The default is to use /dev/random, in which the OS captures randomness (entropy) from various system events. If significant key generation is being done, this source may become depleted, and the utility will apparently freeze, waiting for entropy. Typing any characters on the keyboard will allow the system to capture randomness from the typing intervals. Many systems also provide /dev/urandom, which is faster but significantly less random, leading to less secure keys. If neither device is present, the value keyboard may be used to force use of the keyboard technique described earlier. The default is to use /dev/random if it exists.
-s	strength	Not currently used. Defines the strength of the generated key and may take the value 0 to 15.
-t	type	No known current use. The default is AUTHCONF (authenticate and encrypt). May take the values AUTHCONF, AUTH, CONF, NOAUTH, and NOCONF NOAUTHCONF.
-v	level	Defines the debugging level and may take the values 0 to 3.
hostname		Defines the name of the KEY or DNSKEY RR that will be generated. For ZSKs and KSKs used in DNSSEC.bis operations, this will be the zone apex; for example, if the zone name is subdomain.example.com, then hostname will be subdomain.example.com. In TSIG operations, the name used is defined in the key clauses of the peers and may be hostname or any other suitable value.

Table 9-13 shows the results of using the -n and the -k option values (the X indicates the rr-type is rejected) together with the value of the flags field.

Table 9-13. *Key RR Matrix*

-n	no -k	-k
host (or entity)	X	512 (KEY)
other	0 (DNSKEY)	X
user	X	0 (KEY)
zone	256 (DNSKEY)	256 (KEY)

RFC 3445 defines the valid `flag` field values to be `0`, `256`, and `257`, so the `host flags` field in Table 9-13 will be interpreted as `0` or exactly the same as the `user` value. DNSSEC.bis ZSKs and KSKs use the `zone` value (without `-k`), and normal KEY RRs use `user` (or `host` if you prefer a change).

dnssec-keygen Examples

The following command will generate a shared secret for use with TSIG operations (when `-a hmac-md5` is used, a KEY RR is *always* generated, so the `-k` option is not required):

```
# dnssec-keygen -a hmac-md5 -b 128 -n user example.com
Kexample.com.+157+23417
```

The following command will generate a public/private key pair using the DSA algorithm for use, say, as a KSK in DNSSEC.bis. It creates a DNSKEY RR with a `flags` field of 257:

```
# dnssec-keygen –a dsa -b 1024 -f KSK -n zone example.com
Kexample.com.+003+03733
```

The following command will generate a public/private key KEY RR for use with, say, SIG(0) dynamic update (DDNS) security using the RSA-SHA-1 algorithm:

```
# dnssec-keygen –a rsasha1 -b 1024 -k -n user bill.example.com
Kexample.com.+005+03733
```

dnssec-signzone Utility

The `dnssec-signzone` utility secures a zone file by cryptographically signing it using a public key (asymmetric) algorithm for use in DNSSEC signed zones (see Chapter 11). Zones are signed using one or more Zone Signing Keys and optionally one or more Key Signing Keys. Use of separate ZSKs and KSKs is the currently IETF recommended best practice. The utility performs the following tasks:

1. Sorts the RRs into canonical order (alphabetic based on name).
2. Adds an NSEC RR for each name in the zone file such that it is possible to chain through the list of all valid names. This process provides *proof of nonexistence* of any name.
3. Signs each RRset in the zone file by adding an RRSIG RR (a digital signature), including the NSEC RRs added in Step 2, using the ZSK.
4. Signs the DNSKEY RRset comprising the ZSK and KSK at the zone apex or root with the KSK—if requested.
5. Optionally creates (through the `-g` option) files containing the DS RR and the KSK for use by the parent zone to create a chain of trust.
6. Writes a signed zone file. The default is to append `.signed` to the zone file name.

The RRSIG RRs that sign each RRset (in Step 3) have a start time value (when they become valid) and an end time value (when they expire). By default, time `dnssec-signzone` uses the UTC value of the local run time minus 1 hour (for clock skew) as the start time and the end time is set to the start time plus 30 days. If nothing else is done to the signed zone file, it will become invalid after this period. Both start and end values can be changed by options described in Table 9-14. The input zone file, the zone file to be signed or re-signed, may be an unsigned zone file, or it can be a signed zone file. If it is a signed file, then existing signatures may be renewed, depending on their remaining period of validity. The default behavior is that any signature that has less than one quarter of its time remaining will be renewed to either the default (30 days) or a user-defined value. Thus if the original signature period was 30 days, then only RRSIG RRs with less than 7.5 days remaining will be renewed.

If the ZSK and KSK values to be used in the signing process are not defined explicitly, the `dnssec-signzone` command will use any DNSKEY value in the zone file for which it can find a corresponding private key in the current directory to sign the file. While leading to much shorter command lines (and the `dnssec-signzone` command line can get to be pretty big), it is always better to explicitly define the ZSK and KSK values in the command line to ensure that the results are as expected. This is especially true when key rollovers are being processed (see the section "Secure Zone Maintenance" in Chapter 11) when inactive DNSKEY RRs may be present in the zone file.

All time values used in `dnssec-signzone` operations are relative to Universal Coordinated Time (UTC, historically known as GMT or Greenwich Mean Time), so it is vital that both the name server clock is correctly synchronized to a suitable time source, for instance, using NTP or the `ntpdate` command, *and* that the time zone is correctly configured on the system.

■ **Note** If a signed file, say, `master.example.com.signed`, is input to a `dnssec-signzone` command, the output file will, unless changed by the `-f` option described in Table 9-14, be `master.example.com.signed.signed`—perhaps not the desired result.

---

dnssec-signzone Syntax

```
dnssec-signzone [ -a ] [ -c class ] [ -d directory ] [ -e end-time ]
    [ -f output-file ] [ -g ][ -h ] [ -i interval ] [ -k ksk-key-file ]
    [ -l domain  ] [ -n threads ] [ -o origin ] [ -p ] [ -r randomdev ]
    [ -s start-time ] [ -t ] [ -v level ]  [ -z ]  zonefile [ zsk-key-file.]
```

dnssec-signzone Options

Table 9-14 describes the options available with the `dnssec-signzone` utility.

Table 9-14. dnssec-signzone *Options*

Option	Parameter	Meaning and Use
-a		Verifies the generated signatures. A new zone file is not written.
-c	class	The default is IN, but this may take the standard optional values of CH (CHAOS) or HS (HESIOD).
-d	directory	Looks for key-files in the defined directory as opposed to the current working directory.
-e	end-time	Defines the time the RRSIG RRs will expire; defaults to 30 days, but may be overridden with this option. May take the format YYYYMMDDHHMMSS or +secs (seconds from -s start-time) or now+secs (seconds from current run time).
-f	output-file	Defines the file name of the signed zone file to be created. The default output file name is the zone file name with .signed appended to it; for instance, if the zone file name is master.example.com, the default output file name is master.example.com.signed. When signing (or re-signing) a signed zone, this value should be the same as the input file name to avoid changes to the named.conf file.
-g		If present, generates files containing the DS RR and the DNSKEY RR to be used by the parent zone when creating a chain of trust. The files are named dsset-domain. and keyset-domain. (both files names terminate with a dot) where domain is the value of the -o domain option.
-h		Displays a short description of each option available and terminates.
-i	interval	Defines the time in seconds after which RRSIG RRs will be retained; otherwise they will be renewed. The default is to take one quarter of the time from the RRSIG start to its expiry. Thus if the default difference of 30 days is being used, any record having more than 7.5 days remaining will be retained; else it will be re-signed for another 30 days or the value defined by the -s and -e options. The -i option may be used to explicitly change the time at which records are re-signed, thus -i 3600 will retain (not re-sign) any RRSIG that has more than 1 hour remaining, and -i 2419200 will only retain RRSIG RRs that have more than 28 days remaining. All others will be re-signed.
-k	ksk-key-file	Defines the key to be used as the KSK (ignores the value of the flags field in its DNSKEY RR) where key-file is the name of the key generated by the dnssec-keygen utility. This key will be used to sign the DNSKEY RRset at the zone apex and to generate any required DS RRs for use in a chain of trust (if the -g option is used). The -k option may appear more than once if a zone is *double-signed* (see Chapter 11). The key-file name is used *without* either the .key or .private suffix (see examples that follow). If this option is not present, dnssec-signzone will attempt to guess the key by inspecting the flags field of the DNSKEY RRs in the zone, but it is much safer to control the behavior using this option at the expense of longer command lines.
-l	domain	Generates a DNSSEC Lookaside Validation (DLV) record set in a file named dlvset-domainname. DLV is an experimental RR type that replaces the normal DS RR with a DLV RR (similar in every respect), which is added to a unique zone controlled by use of the dnssec-lookaside statement in the named.conf file. The domain value is appended to the zone name for all KSK keys in the zone file; that is, if the zone name is example.com and, say, -l dlv.example.net is used, then the DLV RR name is example.com.dlv.example.net. (DLV is described in Chapter 11.)

Continued

Table 9-14. *Continued*

Option	Parameter	Meaning and Use
-n	threads	By default, a single thread is started for each CPU detected. This can be overridden using the threads value.
-o	origin	Defines the name of the zone apex. If not specified, the name of the zone file is assumed to be the zone origin.
-p		Uses pseudo random data; while faster, this is significantly less secure and in general should only be used if a suitable, ample supply of entropy is not available to the server on which the dnssec-signzone is being executed.
-r	randomdev keyboard	Defines the source of randomness. The default is to use /dev/random, in which the OS captures randomness (entropy) from various system events. If this source becomes depleted, the utility will apparently freeze, waiting for more entropy. If this happens, typing any characters on the keyboard will allow capturing of randomness from the typing intervals. Many systems also provide /dev/urandom, which is faster but significantly less secure. If neither device is present, the value keyboard may be used to force use of the keyboard technique described earlier. The default is to use /dev/random if it exists.
-s	start-time	Defines the time the RRSIG RRs will become valid. The default is UTC minus 1 hour for clock skew, but it may be set explicitly using this option. May take the format YYYYMMDDHHMMSS or +secs (seconds from current run time).
-t		Prints statistics on completion of the zone signing.
-v	level	May take the value 1 to 3 for various levels of debugging.
-z		Ignores KSK (SEP) flag on keys found in DNSKEY RRs. Setting this option allows a KSK to be used as a ZSK, but by ignoring the SEP flag, it does not perform the KSK signing function even if a -k option is defined. To avoid problems, DNSKEY RRs should always have the correct flags set, which is controlled by the dnssec-keygen utility options.
zonefile		The name of the zone file containing the records to be signed. May be an unsigned or a signed zone file. This file name may be the same as that used on the -f option if required—if signing a signed zone file, it may be convenient to retain the same zone file name on the output file to save changing the named.conf file.
zsk-key-file		Defines the key-file name (generated by the dnssec-keygen utility) to be used as the ZSK. The ZSK may be omitted, in which case every .private file found in the current directory with a corresponding DNSKEY RR in the zone file that has the ZSK bit set (flags value of 256 or 257) will be used to sign the zone file. At the expense of longer command lines, it is safer to explicitly define the key to be used in all cases. Multiple zsk-key-file values may be used to allow signing with multiple keys.

`dnssec-signzone` Examples

The following examples illustrate the use of the dnssec-signzone utility. The first example signs the zone file master.example.com using a separate KSK and ZSK, both of which are in the current working directory using the default 30-day signature period.

```
# dnssec-signzone -k Kexample.com.+003+12456 -o example.com \
 -t master.example.com Kexample.com.+005+03556
master.example.com.signed
Signatures generated:                       20
Signatures retained:                         0
Signatures dropped:                          0
Signatures successfully verified:            0
Signatures unsuccessfully verified:          0
Runtime in seconds:                      0.357
Signatures per second:                  53.079
```

The \ in the preceding example splits the line for presentation purposes only. The command should appear as a single line to the operating system. In the preceding example, the -t option shows the typical statistics output by the utility. The next example shows use of the end time option to provide a 90-day validity period; separate ZSKs and KSKs are used:

```
# dnssec-signzone -k Kexample.com.+003+12456 -o example.com \
-e 7776000 -t master.example.com Kexample.com.+005+03555
```

The \ in the preceding example splits the line for presentation purposes only. The command should appear as a single line to the operating system. The -e option could have been specified as, say, 20050614110523 (using the date format) if that is more convenient; however, assuming the zone signing policy is fixed (that is, it is always 90 days), the time in seconds is calculated only once rather than adding 90 days to the current date on every run! The following example shows signing the zone with two KSKs and two ZSKs (the line is split with \ for presentation reasons only). The example also requests a DS keyset (the -g option) for sending to the parent. The resulting dsset-example.com. file will contain two DS RRs, one for each of the KSKs:

```
# dnssec-signzone -k Kexample.com.+003+12456 -k Kexample.com.+005+33789 \
 -g -o example.com -t master.example.com Kexample.com.+005+03556 \
 Kexample.com.+005+44776
```

The \ in the preceding example splits the line for presentation purposes only. The command should appear as a single line to the operating system. In this example, the DNSKEY RRset at the zone apex will be signed four times, and all other RRsets signed twice for use in double-signing key-rollover strategies, which are described in Chapter 11.

Diagnosing DNS Problems

DNS problems can come in many shapes and sizes—no single method fits all. Instead, this section approaches DNS diagnosis in two ways:

1. *What to do before the problem happens*: This covers both fault prevention and having the necessary tools and information available before a problem occurs.

2. *What to do when a problem occurs*: Some techniques that may help isolate the problem will be presented.

Finally, the section looks at a relatively nasty problem that may happen increasingly in the future—in this case, a secure zone's signature has expired—and shows how it is also simple to interpret information incorrectly.

Before the Problem Happens

A number of sensible precautions can be taken before any problem happens that may allow you to avoid, or at the very least minimize, the headless-chicken act that can occur if you are told that your domain is unreachable.

Log All Changes

Comment features are available in zone files, `named.conf`, and all other files that may be used in DNS configurations. Keep a log in the file of each change made to the file. Using the file rather than, or as well as, a paper record means the information (usually) cannot be lost or mislaid! As a minimum, the change should contain the date, the name or initials of the person who made the change, and what changes, no matter how trivial, were made to the file. Probably the majority of problems in relatively stable systems arise from a simple change—they are always simple—that had an unintended side effect. Close examination of the change logs may be the fastest way to resolve the problem. Dynamic update can present a problem, but there are strategies available to help here also (see "Logging").

Back Up Files

While it might sound trivial, regular backup of all the major configuration and zone files is essential. CVS or a conventional backup program can be used for this process.

Logging

Design and configure your logs to ensure you have enough data to let you diagnose any problem that may *have* occurred without—and there will always be exceptions to this—having to reconfigure your logs, and then try and reproduce the problems. BIND's logging features are extremely powerful, particularly the ability to control the number and size of the files produced if you are short on disk space. As a general rule, keep at least three days of logs and log as much information as practical (`severity info` or lower in the `channel` statement). Stream the logs if that makes operational sense. Many administrators don't like doing this since they want an overall picture of what is happening from a single log rather than having to look at multiple logs and synchronize times. As a minimum, use the `print-category yes;`, `print-severity yes;`, and `print-time yes;` features of the `channel` statement. If dynamic updates are being used, it is seriously worth considering streaming this log using a `category update` statement as shown in the following fragment:

```
logging {
   ....
    channel example-update {
    file "/var/log/named/update.log" versions 3 size 1m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category update{
    example-update;
    };
};
```

Understand what a normal log looks like. Take some time to review a log file for a normal operational period. If the first time you look at a log is when a problem has been reported, you have no real basis for spotting abnormalities.

Tools

Always run `named-checkconf` after any `named.conf` changes. It won't find everything, but it will pick up those trivial errors. The alternative is to let the BIND reload find the single missing semicolon in your 200-item change and take your name server off-line while you are thrashing around in the bowels of `vi` (or whatever your favorite editor is) trying to fix it before anyone really notices.

Take the time to get thoroughly familiar with either `dig` or `nslookup` before you need to use them. Which one you select is a matter of preference and may be a function of what systems you have available or may need to work with. `nslookup` is typically available on Windows and `dig` is not. If you only work on Linux, Unix, or BSD and are using DNSSEC, you have no choice: `dig` is the only utility that supports DNNSEC.

External Sources

Always keep a list of two or three recursive name servers handy at all times that you can use as an alternative source for `dig` commands (for instance, `dig @ns2.example.net www.example.com`) to diagnose your own domain. Such recursive servers are usually provided by major ISPs and other service providers and can help you triangulate where a problem may be coming from or which users it may be affecting.

Similarly, make sure you keep the names of a couple of whois servers (your favorite domain registration web site should also provide such a service) so that you verify the domain has not expired—yes, it does happen, especially where you may not be the responsible authority.

When the Problem Occurs

So the fateful day arrives and you get the call that you dreaded—you are told that your domain is unreachable. So after the panic attack, what should you do? Unfortunately, there is no single solution. While there is usually a tendency to jump into action, always resist it. Too many administrators confuse the words "action" and "progress." Remember the old doctor's adage, "Do no harm." Unless you know what you are doing and why you are doing it, you may make the problem worse than it was! Instead, the following sections list in rough priority what you can do to try and find the cause.

Make No Assumptions

Take nothing for granted and verify immediately before you even talk to anyone that as far as you are concerned everything is operational—name servers, web sites, mail servers, backbone links, routers, firewalls, etc.

Describe the Problem

Establish the precise nature of the problem: Does it happen all the time? When did it start to happen? What software was being used (for instance, a browser)? Can users still access e-mail or use some other domain-based services? What is their local DNS configuration and addresses? If they bypass the DNS, can they still access, say, the web site using an explicit URL? All this is designed to try and get a precise picture of what is happening.

Scope the Problem

Establishing the scope of affected users is a vital step. Is it all users everywhere or just a single group of users? Is there a common DNS involved (which may be poisoned), or does it affect everyone? Is it a single zone or all zones on a name server?

Once you know where to start looking, you can decide whether log inspection and/or DNS inspection are the most appropriate techniques to use next.

Check Your Logs

Assuming that yours is a disciplined operation, check for recent changes in configuration; for instance, check the change logs in the zone and `named.conf` files, or whatever process you use for change control. Now it is time to take a look at the BIND logs.

The first step is to verify your logs for around the best time that you think the problem started and work forward. In the first instance, this should probably just be a rudimentary check for any obvious error messages and failures. BIND logs are reasonably good and output lengthy text messages describing a problem—it may not always describe the actual problem, but it will at least give you an indication of where to start looking. The following log fragment may serve to illustrate:

```
updating zone 'example.com/IN': adding an RR at 'www.example.com' A
updating zone 'example.com/IN': could not get zone keys for secure dynamic update
updating zone 'example.com/IN': RRSIG/NSEC update failed: permission denied
```

The preceding example (the date and time have been removed for brevity) occurred when using secure dynamic update with a DNSSEC signed zone. The actual error is described by the second log entry; the third log entry merely describes the effect of the error. In the preceding case, the zone key being requested was indeed present and in the correct directory. At first blush, the log message was incorrect; however, further examination of the file containing the zone key showed that its permissions were incorrect. The log message was correct in that it described the effect of the error—BIND could not read the key in this case because it was denied permission. Sure, it would have been nice if the message had said “Permission denied,” but we cannot have everything in life.

Start Digging

Either dig or nslookup are the next tools if nothing obvious has turned up in the log. Start by using a dig (or nslookup) at all the authoritative name servers for the domain for both the domain apex and the particular RR that may be having a problem, say www.example.com. The following commands assume the ubiquitous example.com domain with name servers of ns1.example.com (IP 192.168.2.3) and ns2.example.com (IP 192.168.54.3). The following two commands will give you a quick picture:

```
# dig @ns1.example.com example.com any
# dig @ns2.example.com example.com any
```

The first thing to note in the dig output is whether the IP address associated with ns1.example.com and ns2.example.com resolve to the actual IPs for the two servers, which indicates the local name server is working correctly. If the name format is used, dig will use your locally configured name server to perform the lookup of ns1.example.com and ns2.example.com. If your domain has been hijacked or corrupted, these may not resolve correctly, and your first pointer may be already visible. Check that the SOA serial number, the A RRs for the name servers, and any other RRs such as MX are all correct and as expected. Finally, check that the aa (Authoritative Answer) flag in the >>HEADER<< section is present. Repeat using the IP address of the name servers:

```
# dig @192.168.2.3 example.com any
# dig @192.168.54.3 example.com any
```

Confirm that the data is the same for the two sets of outputs. If all is correct so far, verify any failing record such as www.example.com using the following two commands:

```
# dig @192.168.2.3 www.example.com any
# dig @192.168.54.3 www.example.com any
```

While it is possible to either omit ANY from the preceding command (it defaults to a) or use the value a, it is always better to use the value ANY, which will return all RRs with the preceding name and can occasionally yield very interesting results. Confirm that the values are the same in both responses and correct and that again the aa flag is set.

The preceding process has essentially eliminated the authoritative name servers from the problem, and it is time to start looking further afield. Locate one or more name servers that are being used by any affected users—you can get this information from /etc/resolv.conf on Linux, Unix, or BSD systems, or through Network Properties or using the ipconfig command on most Windows systems. Use the following dig commands with the IP address of this particular name server, assumed in this case to be 192.168.254.1:

```
# dig @192.168.254.1 example.com any
# dig @192.168.254.1 example.com soa
# dig @192.168.254.1 www.example.com any
```

Confirm that the data is the same as that provided from the authoritative name server. It should be exactly the same with the exception that the aa flag would not be typically set; if it is set, immediately repeat the dig command and it should disappear. The reason it may be set is that when a caching name server supplies any RR that it obtains directly from an authoritative server, the aa flag is set; if it is supplied from its cache, it will not be set. Thus if set on the first dig, the cached RR may have timed out and been reread from the authoritative servers, whereas the second dig must have obtained it from the cache, and therefore the aa flag will not be set. While authoritative name servers will provide SOA RRs using the ANY query type when accessing a caching server, they will typically only provide the NS RRs, and all others, the SOA in this particular case, will have to be explicitly requested.

Diagnosing the Problem

This brief example illustrates diagnosis of a particular problem—in this case, a DNS security-related problem. While this may seem a little obscure for many users, it does illustrate a number of points of general interest when diagnosing DNS-related problems, the first one of which is make no assumptions. The second is that, with the increasing use of DNSSEC, there exists two separate worlds: a *security-aware* world and a *security-oblivious* world. Both coexist and may not even be aware of each other until it really matters.

The scenario is that a client of ours is having a problem reaching a particular web site, www.example.com, and the problem started happening about two hours ago. We do not own the domain example.com and know nothing about it, but since we know something about name servers, we have been asked to help. The client is using a caching name server with an address of 192.168.2.3. We checked the web site from our location and we can get the web site perfectly. We use dig to check the address as normal:

```
dig www.example.com
; <<>> DiG 9.3.0 <<>> www.example.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1957
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.com. IN A

;; ANSWER SECTION:
www.example.com. 86400 IN A 10.1.2.1
www.example.com. 86400 IN A 172.16.2.1
www.example.com. 86400 IN A 192.168.2.5
www.example.com. 86400 IN A 192.168.254.3

;; AUTHORITY SECTION:
example.com.  86400 IN NS ns1.example.com.
example.com.  86400 IN NS ns2.example.com.
```

```
;; ADDITIONAL SECTION:
ns1.example.com.  86400 IN A 192.168.2.6
ns2.example.com.  86400 IN A 192.168.23.23

;; Query time: 15 msec
;; SERVER: 192.168.254.2#53(ns1.example.net)
;; WHEN: Thu Jun 02 17:20:36 2005
;; MSG SIZE  rcvd: 165
```

Now we just made two mistakes with this command; the first is we did not use the name server of our client (192.168.2.3)—though in this case it would have made no difference to our results—and the second was we made the assumption that the A record was the important one. By simply changing the `dig` command we issue to use `ANY`, not the default (`a`), this is the output we get:

```
dig www.example.com any
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.3.0 <<>> www.example.com any
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1725
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 8, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.com.  IN ANY

;; ANSWER SECTION:
www.example.com.  86400 IN A 10.1.2.1
www.example.com.  86400 IN A 172.16.2.1
www.example.com.  86400 IN A 192.168.2.5
www.example.com.  86400 IN A 192.168.254.3
www.example.com.  86400 IN RRSIG A 5 3 86400
   20050629162118 (Y43c= )
www.example.com.  3600 IN  NSEC example.com. A RRSIG NSEC
www.example.com.  3600 IN  RRSIG NSEC 5 3 3600 20050629161227 (
   20050530151227 3977 example.com.
   SnZJ96ZkmDaB6q4v9PHAMpZuPOKDshlj7loPXL4= )
www.example.com.  3600 IN  RRSIG NSEC 5 3 3600 20050629161227 (
   20050530151227 12513 example.com.
   s9GMC2J+1LVLOiwWST7yDgD8JC2IzzPEsj+dijE= )

;; AUTHORITY SECTION:
example.com.  86400 IN NS ns1.example.com.
example.com.  86400 IN NS ns2.example.com.
```

```
;; ADDITIONAL SECTION:
ns1.example.com.  86400 IN A 192.168.2.6
ns2.example.com.  86400 IN A 192.168.23.23

;; Query time: 31 msec
;; SERVER: 192.168.254.2#53(ns1.example.net)
;; WHEN: Thu Jun 02 17:21:14 2005
;; MSG SIZE  rcvd: 711
```

A considerable amount of material was cut from this output purely for the sake of brevity. The point, however, is already clear, and we have a significant indication as to the probable error. The various additional RRs in this output, RRSIG and NSEC, are related to DNSSEC security. The `example.com` zone is signed—and further diagnostic work will show us that the signatures have expired, rendering the domain invisible to security-aware name servers but still available to security-oblivious name servers and other diagnostic tools. We did not see the additional output on the first `dig` because the basic `dig` command is not security aware. We can get results with no indication of any security context—that is the way the standards are supposed to work. If we do not ask for it, we do not get it. Had we set the `+dnssec` option, we would have received additional data and an indication of the problem. We were able to reach the web site because our local name server is also not security aware (it does not have a `dnssec-enable yes;` statement). This situation will become increasingly common over the next few years. At the risk of sounding trite, it will be increasingly necessary to `dig` below the surface.

Summary

This chapter covered a number of utilities for diagnosing, maintaining, and verifying the DNS system. The chapter covered the `nslookup` utility used to diagnose name servers, which is generally available on Linux, BSD, and Windows platforms. The `nslookup` utility provides both command-line and interactive formats and uses a default configuration file to add significant power to the interactive format. The `dig` utility provides similar functionality to `nslookup` but is typically available only on systems on which BIND is installed. It is the recommended utility for diagnosing BIND name servers and provides support for the latest DNS features such as DNSSEC.bis. The `dig` utility has both a command line and batch mode format, but no interactive format. The `named-checkconf` and `named-checkzone` utilities are used to verify the `named.conf` and zone files, respectively, before being run on a live name server.

The `rndc` utility may be used to control the name server operation locally or remotely. Operation of this utility is enabled by use of the `controls` clause in the `named.conf` file. `rndc` mandates secure authentication (using a shared secret) but provides a default mode where the minimal required files are provided using the `rndc-confgen` utility with the `-a` option. The `rndc` utility uses an `rndc.conf` file to control server access and the keys to be used. An example is shown that allows remote access to more than one name server, each with a separate key. The `nsupdate` utility allows dynamic updates to the Primary master zone file. The `nsupdate` utility transactions may be secured using either TSIG or SIG(0) security and are enabled by the `allow-update` or `update-policy` statements in the `named.conf` file.

The `dnssec-keygen` utility is used to generate cryptographic keys used in TSIG, SIG(0), DNSSEC, or for other purposes. The `dnssec-signzone` utility is used to cryptographically sign a zone file for use in DNSSEC operations.

The next chapter introduces the topic of DNS security, with is divided into the categories of administrative security, zone transfer security, dynamic update (DDNS) security, and, in Chapter 11, DNSSEC (DNSSEC.bis). Each security category uses different techniques and has a different level of complexity.

PART 3

DNS Security

CHAPTER 10

DNS Secure Configurations

At the macro level, the DNS service is essential to the operation of the Internet. At the micro or local level, the DNS service could be essential to the operation of an enterprise or a humble but much-loved family web site. In all cases, the appropriate investment in security must be made to ensure the effectiveness and safety of the DNS system. The DNS is by its nature a public system, and acts like a honey pot for the bad bees of the Internet world. This chapter and Chapter 11 introduce DNS security, with the intent of allowing the reader to select the appropriate techniques for the perceived level of threat.

Unfortunately, the term *DNSSEC* has a bad reputation because of its perceived complexity, and is frequently used to cover the whole topic of DNS security. There are many facets to DNS security, ranging from relatively simple to implement to brutally complex. This chapter divides security into four topics:

- *Administrative security*: This part of the chapter covers the use of file permissions, server configuration, BIND configuration, and sandboxes (or chroot jails). All of these techniques are relatively simple to implement, and can (and should) be applied to stand-alone DNS servers or to servers that run DNS as one of a number of services. Administrative security is a base-line topic. All the fancy cryptographic techniques in the world are useless if the base system is unstable or has world read-and-write privileges on all the interesting files.

- *Zone transfers*: Unless a multimaster configuration system is being used, zone transfers are essential to normal operation. Limiting and controlling both the source and destinations of zone transfer operations using physical security, BIND parameters, or external firewalls is always prudent. Secure authentication of the source and destinations of zone transfer operations may or may not be worth the effort.

- *Dynamic updates*: Dynamic updates expose a master zone file to possible corruption, destruction, or poisoning. Not taking sensible precautions to limit access through either good system design, BIND parameters, firewalls, or authentication probably constitutes a misplaced reliance on the essential goodness of mankind.

- *Zone integrity*: If it is essential that the zone data used by either another DNS or an end host be correct (that is, query responses have not been tampered with and the returned data could only have come from the zone owner), then DNSSEC is required. DNSSEC has been the subject of considerable experimentation and subsequent change over the past three or four years. This book describes what is colloquially called DNSSEC.bis—the second generation of secure DNS—and the object of much work in the IETF, the root-server operators, and the Regional Internet Registries (RIRs). DNSSEC.bis is described in Chapter 11.

Because it is so critical, the DNS system is the subject of many myths, including the "great bug myth." This myth purports that BIND is so full of bugs that we must go to any lengths to protect our systems from its self-destructive ways. Although this may have been true in the bad old days of early version 4 and version 8 releases of BIND, it is no longer the case. The last CERT advisory (`www.cert.org`) published for BIND was in 2003 and that was for BIND version 8; the last one for BIND 9—a nearly complete rewrite of BIND—was in 2002. When you consider that this software is being accessed many millions of times every second worldwide it is a very impressive performance. DNS systems need to be protected from external sources and attacks, but in general not from BIND itself. The emphasis in the following sections is primarily on outward-facing security, not inward-facing security.

Security Overview and Audit

Figure 10-1 was introduced in Chapter 3 and is reproduced here as a reminder of the possible sources of threat that form the basis of any security audit. Every data path is a potential source of threat.

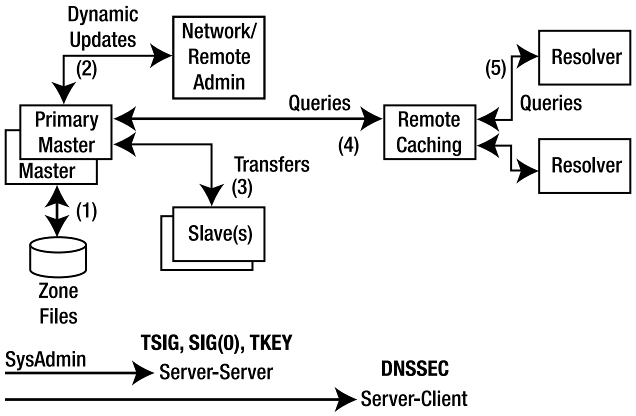


Figure 10-1. *Security overview*

The critical point in defining security policies and procedures is to understand what needs to be secured—or rather what threat levels need to be secured against and what threats are acceptable. The answers to these two points will be different if the DNS is running as a root-server versus running as a modest in-house DNS serving a couple of low-volume web sites. There are no hard and fast rules; defining your policy is a matter of blending paranoia with judgment.

DNS Normal Data Flow

Every data flow—that is, each numbered line in Figure 10-1—is a potential source of threat. Table 10-1 defines the potential outcomes of compromise at each point and the possible solutions.

Table 10-1. *DNS Security Threats*

Number	Area	Threat	Classification	Solutions
1	Zone files	File corruption (malicious or accidental)	Local	System administration
2	Dynamic updates	Unauthorized updates, IP address spoofing (impersonating update source)	Server-to-server	Network architecture, Transaction Signatures (TSIG), SIG(0), or disable
3	Zone transfers	IP address spoofing (impersonating update source)	Server-to-server	Network architecture, TSIG, or disable
4	Remote queries	Cache poisoning by IP spoofing, data interception, or a subverted master or slave	Server–client	DNSSEC
5	Resolver queries	Data interception, poisoned cache, subverted master or slave, local IP spoofing	Remote client–client	DNSSEC

The first phase of any security review is to audit which threats are applicable and how seriously they are rated in the particular organizational circumstances. As an example, if dynamic updates are not supported (BIND's default mode), there will be no dynamic update threat.

It can be easier to disable a process than to secure it. For example, consider zone transfers. If a classic master–slave configuration is being used, then zone transfers will be inevitable, and the configuration's security implications must be evaluated. However, it is possible to replace such a configuration with a multiple-master one in which each name server obtains its zone files locally. Thus, zone transfers may be globally disabled. In this environment, synchronization of master zone files must be done by some out-of-band process, such as secure FTP or secure e-mail. However, these out-of-band processes may be simpler to organize or already exist. Using such alternative procedures is sometimes referred to as *security by obscurity*. It can be a useful tactical fix but is not always a strategic solution.

Finally, a note of caution: there is a single master to secure, in zone transfers there may be one or two slaves to secure, in dynamic updates there may be tens of update sources to secure, and there may be many hundreds or thousands of remote caches to consider in DNSSEC solutions. In general, the further you go from the master, the more systems you have to consider, and consequently the solutions are more complicated. Unless there is a good reason for not doing so, it is always recommended that you start from the zone master and work out. It would be a tad frustrating to have completed a successful implementation of a complex DNSSEC solution, only to discover that anyone could dynamically update your zone files.

Security Classification

The security classification is a means to allow selection of the appropriate remedies and strategies for avoiding the implied risk. Many of the following methods are described in detail in this chapter and Chapter 11. The following numbering relates to Figure 10-1.

- *Local threats (1)*: Local threats are usually the simplest to prevent, and are typically implemented simply by maintaining sound system-administration policies. All zone files and other DNS configuration files should have appropriate read and write access, and should be securely backed up or maintained in a CVS repository. Stealth (or Split) DNS servers can be used to minimize public access, and BIND can be run in a *sandbox* or a *chroot jail* (described in the section "BIND in a Chroot Jail" later in the chapter).

- *Server–server (2)*: If an organization runs slave DNS servers, it needs to execute zone transfers. As noted earlier, it is possible to run multiple-master DNS servers rather than master–slave servers, and thus avoid any associated problems. If zone transfers are required, BIND offers multiple configuration parameters that can be used to minimize the inherent risks in the process. TSIG and Transaction KEY (TKEY) also offer secure methods for authenticating requesting sources and destinations. Both methods are described in detail in the section "Securing Zone Transfers" later in the chapter. The physical transfers can be secured using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

- *Server–server (3)*: The BIND default is to deny Dynamic DNS (DDNS) from all sources. If an organization requires this feature, then BIND provides a number of configuration parameters to minimize the associated risk; these are described in detail later in the chapter. Network architecture design—that is, all systems involved are within a trusted perimeter—can further reduce the exposure. TSIG and SIG(0) can be used to secure the transactions from external sources. Configuration of Stealth (Split) servers was described in Chapters 4 and 7, and TSIG and SIG(0) security are described in the section "Securing Dynamic Updates" later in the chapter.

- *Server–client (4)*: The possibility of remote cache poisoning due to IP spoofing, data interception, and other hacks is likely quite low with modest web sites. However, if the site is high profile, high volume, open to competitive threat, or is a high revenue earner, then the costs and complexity of implementing a full-scale DNSSEC solution may be worthwhile. Significant effort is being invested by software developers, Registry Operators, the RIRs, and root-server operators, among many others, into DNSSEC. We are likely to see significant trickle-down effects within the near term in the public domain, as well as within controlled groups such as intranets and extranets. Indeed, Sweden will be the first country in the world to offer DNSSEC support for the .se domain starting in late 2005.

- *Client–client (5)*: DNSSEC.bis standards define the concept of a *security aware resolver*—a currently mythical entity—that can elect to handle all security validation directly, with the local name server acting as a passive communications gateway.

Administrative Security

Administrative security in the context of this book is concerned with the selection and configuration of the DNS software and the server or servers on which it runs. The items in the following section are listed in approximate order of priority, defined in this case as a combination of return for effort expended and its effect on overall security. Clearly, the order is not rigid, nor is it meant to suggest that if you only keep software up to date, the DNS installation will be secure. However, a fully chrooted installation with a known root exploit can still create serious havoc. Judgment and local circumstance always override any tactical list, such as the one in the following section. The following sections are presented in the form of checklists, with some limited explanations where appropriate.

Up-to-Date Software

Although it may seem trite, keeping software up to date is a vital security component. Many busy administrators who run operational systems dislike upgrading stable software. Upgrading mature, stable software may be an evil, but it is a necessary—and vital—evil for the health and security of an installation. The longer the task is postponed, the worse it gets. The following is offered as a generic upgrade policy:

1. *Known security exploit*: Subscribe to one of the advisory services provided by SANS (`www.sans.org`) or CERT (`www.cert.org`), as well as many others, and take action on BIND and related technology alerts. Depending on the severity of the alert, this can demand an immediate upgrade followed by a quick test before fast system-wide replacement. Better in this case to risk a new problem than a known exploit.

2. *Required new feature*: If a new feature is required, time is not generally of the essence. Upgrading should be done slowly, with serious testing of limited initial deployment before a final upgrade of all operational systems.

3. *Time*: Every 12–18 months, the author upgrades operational systems if neither items 1 nor 2 has required an upgrade, and even if it means some serious work. The reasoning here is that the longer this task is left, the greater the pain of upgrade. Upgrade is slow, with limited initial deployment before system-wide replacement.

The following additional points may be useful or just plain sensible:

- *Maintain an upgrade checklist*: This should include, at a minimum, the order of upgrades, previous problems, specific items to verify (for example log messages, dig test scripts, and results), AXFR block tests, and anything else you deem useful. In a busy environment, your own memory is not a useful planning tool, especially if the frequency of upgrade is low. Add to the list after each upgrade, and keep it readily available on an intranet, in the configuration file as comments, or in some other suitable location. It must be a living document to be useful.

- *Block communication of the BIND software version*: All the sample `named.conf` configuration files (see Chapter 7) use the `version` statement in an `options` clause to hide the current version of BIND being run. If the version number is not blocked, it is simple to use `dig` to discover what version is being used at any particular location. In the event of a known exploit, why boast that you are vulnerable?

Limit Functionality

The best way to limit vulnerabilities is to avoid using exploitable operations, if practical. As an example, by using multiple masters it is possible to run an operational system without doing DNS zone transfers, in which case `allow-transfer` and `notify` statements can be set to `"none"` in the global `options` clause. Take some time to ponder alternate strategies and the relative efforts and returns involved in the operations, rather than just opt to use a BIND feature because it exists. Hackers love that way of thinking—it gives them plenty of targets.

Defensive Configuration

A defensive configuration is one in which all the major, especially security-related, features are explicitly identified as enabled or disabled. Such a configuration ignores any default setting and values. It takes as its starting point the site needs, and defines each requirement, positive or negative, using the appropriate configuration statements or other parameters. Defaults are great for us lazy folks, but they can also be dangerous if they change. As an example, the current version of BIND disables DDNS by default. However, many DNS administrators like to add the statement `allow-update {"none";};` explicitly in a global `options` clause, both as clear indication that the feature is not being used, and as a protection against a future release that may change the default. A defensive configuration file that identifies all the requirements explicitly is also self-defining. That is, by inspecting the file—without needing to find the manual or reference documentation—the functionality is self-evident. At 3 a.m. when pandemonium occurs, such self-defining files may be a useful side-effect.

Deny All, Allow Selectively

Even when operations are permitted, for example in `NOTIFY` or zone transfers, it may be worth globally denying the operation and selectively enabling it, as in the following fragment:

```
options {
....
    allow-transfer {none;}; // no transfer by default
....
};
....
zone "example.com in{
....
    allow-transfer {10.0.1.2;}; // this host only
....
};
```

Although the preceding configuration requires additional typing, it also requires a minor act of thought—always a good thing—before adding the line in the `zone` clause.

Remote Access

BIND releases come with an administration tool called `rndc` (described in Chapter 9) that may be used locally or remotely. On the one hand, `rndc` is a useful tool, while on the other hand, if you can get in, so can someone else. The BIND default is to enable `rndc` from the loopback address only (127.0.0.1). If `rndc` will not be used, it should be explicitly disabled using a null `controls` clause, as shown here:

```
// named.conf fragment
controls {};
....
```

If `rndc` is used, then it is recommended that an explicit `controls` clause be used, even if access is only allowed from `localhost`, as shown here:

```
// named.conf fragment
controls {
   inet 127.0.0.1 allow {localhost;} keys {"rndc-key"};
};
```

In the preceding fragment, the default key name of `rndc-key` is shown (generated by the command `rndc-confgen -a`), and should be replaced with whatever name was allocated to the key being used to control `rndc` access. The `rndc.conf` file and files containing keys such as `rndc.key` must be protected with limited permissions as described in the following section, "Limit Permissions." BIND thoughtfully provides a simple method to create a default key configuration (`rndc-confgen -a`) for use with `rndc`, which for loopback-only (127.0.0.1) use may be adequate to get you started. However, it is neither advisable nor sensible for remote use. Take the few minutes required and learn how to generate your own `rndc` keys. Change them every 30 to 90 days without fail.

■**Note** For some reason best known to the Fedora Core developers, the default `rndc` key name on Fedora is `rndckey`, not the BIND default of `rndc-key`.

Limit Permissions

The theory behind limiting permissions has two distinct parts that must not be confused, and that may have separate implementation issues and strategies:

- *Confidentiality*: This involves limiting access to confidential files used by BIND or the DNS application, to ensure that another application or user cannot read or write to them.
- *Containment*: This prevents BIND from reading or writing to other locations if it is compromised.

As discussed earlier, this book places more emphasis on protecting the name server from external attack such as cache poisoning, accessing confidential information, and other attempts to compromise the data content of the zone files than from damage inflicted by BIND itself being compromised. In this context, the files that BIND uses and their access permissions are of considerable importance. The following list describes seven files or file groups BIND may use, and their protection requirements :

- `named.conf`: This file should be treated as confidential because it contains information about the style of configuration that may assist an attacker, and frequently contains other interesting information, such as IP addresses, that may be used to launch spoofing and other attacks. The `named.conf` file should *never* contain `key` clauses as a matter of policy, including those used for `rndc` access. Instead, `key` clauses should be maintained in separate files within a separate directory and included in the `named.conf` file (using an `include` statement). If `view` clauses are being used, then as a minimum those containing private information for use by a Stealth configuration (see Chapter 4) should also be contained within separate files and included in the `named.conf` file. If the organizational policy allows `zone` clauses or parts of the `named.conf` file to be controlled or edited by end users, or more than one user, then these parts should be saved as separate files in separate directories. In this way, appropriate permissions can be applied and then included in the final `named.conf` file. It is worth noting here that `trusted-keys` clauses contain only public keys, which are not sensitive information, unlike `key` clauses, which contain shared secrets and are extremely sensitive. The requirement for `trusted-keys` clauses is to prevent write corruption (the same as `named.conf`) rather than to prevent unauthorized reading, as is the case for `key` clauses.

- *Included files*: Each file included in the `named.conf` file can have different permissions applied to it. The policy should be to categorize the type of file and the required access, and separate the files into directories whereby directory-level permissions can be applied, rather than fooling around with individual files. Thus, included files containing keys could be saved in a directory called `/var/named/keys`, and private views in a directory called `/var/named/views`. Any private `zone` clauses could be saved in, say, `/var/named/zone-private`. Generally editable `zone` clauses could be saved in the home directory of the user who is allowed to edit it. Each such directory can be assigned appropriate permissions.

- *Zone files*: Zone files typically contain public information, so there seems little point in protecting them (other than from global write permission). However, if a `view` clause-based Stealth system is being used, then the zone files on the private side of the configuration will contain sensitive data and require separate treatment. Again, it is prudent to separate private zone files into a separate directory such as `/var/named/master/private`. Zone files that may be edited by users can be placed in the respective home directories with appropriate user permissions, or you can place them in a `/var/named/master/ddns` directory and allow dynamic updates. Finally, slave zones in `/var/named/slave` require write permission for BIND.

- *PID file*: This is normally written to `/var/run/named.pid` or `/var/run/named/named.pid`. Although it contains sensitive information (the Process Identifier of the `named` daemon), the information can only be used by `root`. If you're faced with a root exploit, then the PID files are among the last items to be concerned about. The PID file requires write access for BIND and read access for scripts (start, stop, restart, and so on) that make use of it.

- *Log files*: This book configures the logs to be written to the /var/log/named directory mostly for convenience rather than security. In general, the log does not contain sensitive information and does not require special handling. However, if a view clause is being used in a Stealth configuration, the log—depending on options—may contain sensitive information relating to private IPs and should be protected.

- rndc *files*: If using rndc, keep in mind that the rndc.conf file (see Chapter 9), and especially any files containing keys, including the default rndc.key file, contain extremely sensitive information and need to be protected.

- *Journal files*: A zone file is normally a read-only file from BIND's perspective. If Dynamic Update (DDNS) is being used, then updates are written to a binary .jnl file for each zone, and only periodically written to the zone file. For public zone files, such information is not sensitive, but for private zone files appropriate permissions are required. Once DDNS is invoked for a zone, special procedures are generally required to edit the zone files manually. Therefore, permissions can be made tight. To reflect these permissions, zone files that will use DDNS could be placed in a directory such as /var/named/master/ddns.

Before building a permission strategy, we need to look at how BIND is run. BIND can run in three possible ways:

- *Run BIND as* root: This is a dangerous thing to do, and normally requires additional work to override the options defined in most standard BIND installations. This method of running BIND is not recommended and will not be discussed further.

- *Run BIND under a unique (nonroot) UID (Linux, Unix, or BSD) or user account (Windows)*: This method uses the -u command line argument of BIND (see Chapter 12), and is the standard method used by most packaged installations on Linux, BSD, and Windows. The User ID (UID) is typically named for Linux/UNIX, or bind if you are running FreeBSD and the user account is named for Windows.

- *Run BIND in a sandbox or chroot jail*: FreeBSD 5.x and Fedora Core 2 default installations use this mode of operation. Most Linux distributions, including Fedora Core 2, provide a bind-chroot RPM that can be applied after BIND has been installed, to add the necessary directories and scripts to apply the appropriate permissions.

Both the last two methods run BIND with a unique UID (normally named or bind if FreeBSD) and are described in detail in the section "Running BIND as Nonroot." Table 10-1 shows the permissions that lock down the various files to their minimum requirements. Before considering the required file permissions, it is necessary to understand the various stages BIND adopts during its initialization sequence. When BIND is loaded it runs as user root because it requires certain privileges—notably the ability to allocate and bind to its normal, but privileged, port number of 53, and if rndc is permitted, also to port 953. During this phase BIND reads all its configuration and zone files and logs any failures to syslogd. On completion of this process, it then issues an suid() call (change user name) to the user name defined in the -u command line argument. Only then does it proceed to write the PID file, log, and any other required files. This structure lends itself perfectly to tailoring precise file permissions, because read-only files (from BIND's perspective) can be set to permissions based on their editing requirements. BIND, because it running as root during its read phase, can read them in all cases. Table 10-2 illustrates the kind of structure and flexibility that may be created. This structure may look complex, but it shows the possibilities, and it has the major merit that once established, it requires little maintenance.

Note The preceding sequence is slightly different when running in a chroot jail, which is described in the section "BIND in a Chroot Jail."

Table 10-2 assumes that BIND runs with a UID of `named`, editing of secure (but not secret) files is done under a nonroot user with a user name of `dnsadmin` and a group of `root` (to allow `su` commands if necessary), and editing of multiuser access files (for example, public zone files and zone clauses) is done under a group called `dnsusers`. The files containing secrets can only be read by BIND and edited by `root`. Files are placed in the directories named under each file type described earlier. The home directory of `dnsadmin` is assumed to be `/var/named`, and for `dnsusers` it is `/home/username` or similar. In Table 10-2, the Mask column shows the directory permission first, separated from the file permissions with a colon (":"). A question mark ("?") indicates that this value may be determined by other system requirements. The setting of limited permissions on Windows systems is described in Chapter 6.

Table 10-2. *Directory and File Permissions*

File/Group	Typical Location	user:group	Mask	Notes
`named.conf`	`/etc`	dnsadmin:root	? : 0660	Read-only BIND file. `dnsadmin` can edit.
Included public `named.conf`	`username` home directory	username: dnsusers	0770 : 0660	Read-only BIND files. Permissions allow `dnsusers` group to edit.
Included key files	`/var/named/keys`	named:named	0400 : 0400	Read-only BIND file. Only `root` can edit or view.
Included private views	`/var/named/views`	dnsadmin:root	0770 : 0660	Read-only BIND file. `dnsadmin` can edit.
Private zone files—no DDNS	`/var/named/masters/private`	dnsadmin:root	0770 : 0660	Read-only BIND file. `dnsadmin` can edit.
Private zone files—with DDNS	`/var/named/masters/ddns`	named:root	0770 :0660	Read/write for BIND. `dnsadmin` can edit.
Slave zone files	`/var/named/slaves`	named:root	0770 : 0660	Read/write for BIND. `dnsadmin` can edit if required.
Public zone files	`username` home directory	username: dnsusers	0770 : 0660	Allows `dnsusers` group to edit. These files cannot be dynamically updated.
`named.pid`	`/var/run/named`	named:named	? : 0664	Allows access by BIND tools/scripts and `root`.
`named.log`	`/var/log/named`	named:root	0770 : 0640	Write access for BIND, `dnsadmin` can read. If not using views, wider permissions can be set depending on local policy.
`rndc.conf`	`/var/named/rndc`	dnsadmin:root	0770 : 0660	Allows access by `dnsadmin` group.
`rndc.key`	`/var/named/rndc/keys`	named:named	0400 : 0400	Only `root` can edit.

In the preceding table, FreeBSD users should replace the group root with wheel and named with bind. If the local policy is to allow only BIND administrators to touch any BIND-related material, then some of the preceding configuration will be unnecessary.

The named.conf file fragment that would reflect such a strategy could look something like the following:

```
// named.conf fragment
include "/var/named/rndc/keys/key.clause"; // single file containing rndc keys
include "/var/named/keys/key.clauses"; // single file containing keys
controls {
   inet 127.0.0.1 allow {localhost;} keys {"rndc-key"};
};
options {
   ....
};
include "/var/named/views/private-view.clause"; // hidden private view
view "public-view" {
   include "/home/firstuser/zone.clause";
   zone "example.com" in {
   type master;
   file "var/named/masters/ddns/example.net";
   // key clause referenced below will be in
   // /var/named/keys/keys.clause above
   allow-update {key "example.net";};
   };
};
```

Running BIND As Nonroot

Most packaged BIND systems, for instance RPMs and FreeBSD ports, install BIND to run with a unique (nonroot) UID—typically named on Linux and Windows and bind on FreeBSD. This section describes how to configure your system if BIND is not installed and configured to run with a unique UID, and to set permissions to lock down the files. Even if your BIND system has been installed to run under a unique UID, you may still want to look at and set appropriate file permissions, especially on the more sensitive files. If BIND is running on your system, its status can be interrogated by issuing the following command:

```
# ps aux |grep named
```

It returns something like the following:

```
named  36120  0.0  0.9  5372 4376  ??  Is    1:02PM   0:00.11 named -u named
```

The preceding output shows that the daemon named is running under the UID named (the first named in the line), which is initiated by the -u argument at the end of the line. If the entry looks like the following, named is running as root indicated by the first root in the line and the absence of a -u argument:

```
root  36120  0.0  0.9  5372 4376  ??  Is    1:02PM   0:00.11 named
```

Action should be taken immediately to change this state, as described in the following section.

Setting the Run Time UID of BIND

To run BIND under its own UID, we need to create a user and group for the named daemon. By convention this is normally `named` (or `bind` under FreeBSD). This book uses `named` throughout, but you can change it to any appropriate value (for example, `dns`) if you wish. First, confirm that you do not already have an existing account by using the following command:

```
# id named
uid=25(named) guid=25(named) groups=25(named)
```

The preceding response indicates the UID already exists. If the user account does not exist, the following response is returned:

```
id: named: no such user
```

Try again with `id bind`, and if there is still no valid user, then create a unique user and group, as follows:

```
# groupadd -r named
```

The preceding command adds the group `named` with the first free system account group (the `-r` argument). The presence of the group can be confirmed with the command `vigr`, which displays and allows editing of the list of groups in the system (use `:q!` to exit `vigr` without making changes).

Now add the system account `named` using the following command:

```
# useradd -c 'Bind daemon' -d /var/named -s /sbin/nologin -g named -r named
```

If the `-c` argument (a comment) contains a space, it must be enclosed in quotes as shown. The `-d /var/named` is the default directory at login, and is required but is not used because this is a system account without a login or password. The `-s /sbin/nologin` argument is the Linux default for a no-shell account, The `-g named` argument defines the initial group to be used by the account and references the `named` group we just created. `useradd` requires that the group `named` exists, so always define groups before users. The `-r` argument defines this to be a system account (typically with a UID < 500 for Linux and < 1000 for FreeBSD) with an account name of `named`.

Setting Permissions for the UID

We now set up and create the permissions for the various essential files. We assume that the user account dnsadmin has already been established as a normal login user account using your favorite tool, and is a member of the root group to allow su commands to be issued if required.

■ **Note** Some of the following permissions differ from those defined in Table 10-1, because they are applied to a directory and are typically intended to allow inspection of file properties. Specific files within the directory may be set to the values defined in Table 10-1.

To create and set permissions for run time write files (log and PID), use the following commands:

```
# cd /var/log
# mkdir named
# touch named/example.log
# chown named:dnsadmin named/*
# chmod 0660 named/*
# cd /var/run
# mkdir named
# touch named/named.pid
# chown named:named/*
# chmod 0664 named/*
```

The following commands all assume that the various directories have been created. If this is not the case, then a preceding mkdir dirname command should be issued, as shown in the preceding command sequence. Set permissions on any keys directory, as shown in the following commands:

```
# cd /var/named
# chown named:named keys/*
# chmod 04000 keys/*
```

Set permissions on any private zone files:

```
# cd /var/named
# chown -R dnsadmin:root master/private/*
# chmod -R 0660 master/private/*
```

Set permissions on any DDNS zone files:

```
# cd /var/named
# chown -R named:root masters/ddns/*
# chmod -R 0660 masters/ddns
```

Set permissions on any private-view `include` files:

```
cd /var/named
chown -R dnsadmin:root views/*
chmod -R 0660 views/*
```

Secure any `rndc` key files:

```
# cd /var/named
# chown -R named:named rndc/*
# chown -R 0660 rndc/*
```

Secure the `named.conf` and `rndc.conf` files:

```
# cd /etc
# chown dnsadmin:root named.conf
# chmod 0660 named.conf
# chown dnsadmin:root rndc.conf
# chmod 0660 rndc.conf
```

Finally, to run BIND, use the following command:

```
# /usr/sbin/named -u named
```

Now verify that BIND is loaded and running using the following command:

```
# ps aux |grep named
```

If it isn't loaded and running, inspect syslog using the following command:

```
vi + /var/log/messages
```

Alternatively, you can use a command such as `tail /var/log/messages` to display the last ten lines of the file if there is not much syslog traffic. Then, verify that BIND has loaded the various zones by inspecting the BIND log file:

```
# cat /var/log/named/named.log
11-Apr-2005 13:02:42.801 zone 0.0.127.in-addr.arpa/IN: loaded serial 1997022700
11-Apr-2005 13:02:42.806 zone example.com/IN: loaded serial 2005032902
11-Apr-2005 13:02:42.813 zone localhost/IN: loaded serial 2002022401
11-Apr-2005 13:02:42.817 running
11-Apr-2005 13:02:42.818 zone example.com/IN: sending notifies (serial 2005032902)
```

To ensure that BIND starts at boot time, we need to create a script that we have chosen to call `named` in the startup directory (for Linux, normally `/etc/rc.d/init.d`, or `/etc/rc.d` for FreeBSD). Such a script would look like the following code, which is a simplified version of the current scripts being used on Fedora Core 2. It provides start, stop, and restart services only:

```
#!/bin/sh
#
# named           This shell script takes care of starting and stopping
#                 named under its own (non-root) UID.
#

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 0

[ -f /usr/sbin/named ] || exit 0

# See how we were called.
case "$1" in
  start)
        # Start daemons.
        echo -n "Starting named: "
        daemon /usr/sbin/named -u named
        echo
        ;;
  stop)
        # Stop daemons.
        echo -n "Shutting down named: "
        killproc named
        echo
        ;;
  restart)
        $0 stop
        $0 start
        exit $?
        ;;
  *)
        echo "Usage: named {start|stop|restart}"
        exit 1
esac

exit 0
```

The preceding script must then be linked to the normal run level(s) used, such as run level 3 (non-X11) and 5 (X11). The default run level is normally defined in /etc/inittab by a line that looks something like this:

```
id:3:initdefault:
```

For the preceding example, we would link the script to the appropriate rc.d run level initialization sequence, which for run level 3 would be as follows:

```
# ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/S68named
# ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/K68named
```

To test this process, a command such as the following should be executed:

```
# /etc/rc.d/init.d/named restart
```

The equivalent startup process for FreeBSD users requires adding the following lines to the /etc/rc.conf file:

```
named_enable="YES"  # Run named, the DNS server (or NO).
named_program="/usr/sbin/named"  # assumes a base installation.
named_flags="-u bind"   # Flags for named
```

To be absolutely certain that everything is working while it is still fresh in our mind, the server ideally should be rebooted, and named confirmed to be running successfully with a command such as this:

```
# ps aux|grep named
named  36120  0.0  0.9  5372 4376  ??  Is    1:02PM   0:00.11 named -u named
```

Although the preceding process may appear to involve a number of steps, it offers the flexibility of being able to control precisely and flexibly the editing permissions of the various files and file groups used in the operation of a BIND-based DNS system. Running BIND in a chroot jail (or sandbox) offers an alternate strategy and is described in the following section.

BIND in a Chroot Jail

The terms *chroot jail* or *chroot cage* (now frequently referred to as a *sandbox*) are named from the system call `chroot("/base/directory");`, which takes a base directory argument and does not let the application read or write outside the base directory. All referenced files and paths within the chrooted application are appended to the base directory. Thus, if the chroot base directory is `/var/named/chroot` and the application accesses a file called `/etc/named.conf`, then the full path is translated to be `/var/named/chroot/etc/named.conf`. When running BIND, the `-t /base/directory` command line argument indicates that BIND should run chrooted and defines the base directory to be used. In a chroot environment, both the `-t` and `-u` (BIND UID) arguments must be present to provide a secure environment.

Most distributions provide a packaged method of running BIND in a chroot jail. The following sections define using such a package for both Linux Fedora Core 2 and FreeBSD 5.x. Finally, if such a package is not provided, or master or slave zones are present, manual configuration of a chroot jail is described.

Fedora Core 2 `bind-chroot` Package

DNS may be run in a chroot jail on Fedora Core 2 in one of two ways:

- Selecting the DNS software option during the install process causes a chrooted caching name server installation by default.
- Installing the bind-chroot RPM (specifically `bind-chroot-9.3.0-2.i386.rpm`, which is the same release installation described in Chapter 6).

In both the preceding cases, the process is the same because the install process also runs the bind-chroot RPM. The chroot RPM does the following:

- It creates the chroot base directory as `/var/named/chroot`.
- The following directories are added under `/var/named/chroot`: `etc`, `var/named`, `var/run/named`, and `/dev` (containing only `null` and `random`).
- Relevant files are copied from the corresponding directories. For instance, `/etc/named.conf` is copied to `/var/named/chroot/etc/named.conf`, and ownership of the chroot directories is set to `root:named` with permissions of 0640.
- The startup script (in `/etc/rc.d/init.d/named`) is modified to add the argument `-t /var/named/chroot` to invoke the chroot feature.

The Fedora default configuration files a log using `syslogd`. If a log file is required, then an appropriate directory must be created. For instance, assume you're creating a log file using the following fragment:

```
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
```

In this case, a directory /var/named/chroot/var/log/named is required with write permission for the named UID.

FreeBSD 5.x

The installation of DNS on FreeBSD 5.x creates a chroot installation by default with a chroot base of /var/named. The installation performs the following tasks:

- Creates the directory /var/named/etc/namedb and links it to /etc/namedb (by default, FreeBSD organizes all its files, including zone files, under this base directory). Thus, going to the normal location for these files (etc/namedb) follows the link to the chroot location.
- Additionally, the following directories are created under /var/named: var/dump, var/stats, var/run/named, and var/log (with a default file name of named.security.log). Ownership is bind:bind for the directories, and world read permissions are set on all files.
- The file /etc/defaults/rc.conf contains the defaults, as shown in the following fragment:

```
#
# named.  It may be possible to run named in a sandbox, man security for
# details.
#
named_enable="YES"  # Run named, the DNS server (or NO).
named_program="/usr/sbin/named"  # path to named, if you want a different one.
named_flags="-u bind"   # Flags for named
named_pidfile="/var/run/named/pid" # Must set this in named.conf as well
named_chrootdir="/var/named" # Chroot directory (or "" not to auto-chroot it)
named_chroot_autoupdate="YES" # Automatically install/update chrooted
  # components of named. See /etc/rc.d/named.
named_symlink_enable="YES" # Symlink the chrooted pid file
```

  As always, if changes are required to this file, they should be made to /etc/rc.conf, which overrides the equivalent value in /etc/defaults/rc.conf.

- The script (/etc/rc.d/named) processes the parameters in rc.conf to create or update the configuration during startup. This startup script creates a default rndc configuration by running the command rndc-confgen -a (see Chapter 9), which allows rndc access from localhost only (assuming the default controls clause).

Manual Configuration of Chroot Jail

This section identifies the manual setup of a chroot jail or sandbox. You might want to do so, perhaps because you enjoy doing this kind of thing, perhaps because there may not be an available RPM to install the chroot option, and perhaps because things may go wrong. The configuration has been tested on Linux and FreeBSD (both are documented separately). It assumes a chroot base directory of /chroot/named. The configuration could have used the more normal location of /var/named/chroot or /var/named/ for FreeBSD, but using /chroot/named means we can create a clean chroot environment and avoid any partial results from default installations.

It is further assumed that the user `named` and group `named` accounts have been set up (FreeBSD users would normally use `bind:bind`). If these accounts are not present, the process is described in the section "Setting the Run Time UID of BIND" in this chapter. The standard caching name server `named.conf` file (from the section "Caching-only DNS Server," located in Chapter 7) is used as the target configuration and is reproduced here:

```
 // Caching Name Server for Example.com.
// We recommend that you always maintain a change log in this file as shown below
// CHANGELOG:
// 1. 9 july 2005 INITIALS or NAME
//  a. did something
// a. 23 july 2005 INITIALS or NAME
//  a. did something more
//  b. another change
//
options {
  // all relative paths use this directory as a base
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is published
  version "not currently available";
  // configuration specific option clause statements
  // disables all zone transfer requests
  allow-transfer{"none"};
  // optional - BIND default behavior is recursion
  recursion yes;
};
//
// log to /var/log/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 250k;
   severity info;
 };
  category default{
  example_log;
 };
};
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
```

```
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

Finally, it is assumed that a default `rndc` configuration was established using the command `rndc-confgen -a` so that a default `/etc/rndc.key` file is present.

Linux (Fedora Core 2) Chroot

This configuration builds a chroot environment in a unique location to show the entire process involved. The following series of commands creates the required directories and moves the basic files required. Lines beginning with // are comments and should not be entered:

```
# cd /
# mkdir chroot
# mkdir chroot/named
# mkdir chroot/named/var
# mkdir chroot/named/var/named
# mkdir chroot/named/var/run
# mkdir chroot/named/var/run/named
// create empty default pid file
# touch chroot/named/var/run/named/named.pid
# mkdir chroot/named/var/log
# mkdir chroot/named/var/log/named
// create empty log file
# touch chroot/named/var/log/named/example.log
# mkdir chroot/named/dev
// create chroot/named/dev/null and /dev/random
# mknod chroot/named/dev/null c 1 3
# mknod chroot/named/dev/random c 1 8
// copy required files
# cp /etc/named.conf chroot/named/etc/named.conf
# cp /etc/localtime chroot/named/etc/localtime
# cp /var/named/localhost.rev chroot/named/var/named/localhost.rev
# cp /var/named/master.localhost chroot/named/var/named/master.localhost
# cp /var/named/root.servers chroot/named/var/named/root.servers
// rndc default key file (if not disabled)
# cp /etc/rndc.key chroot/named/etc/rndc.key
// set permissions and ownerships
```

```
# chown -R named:named chroot/named/*
# chmod -R 0660 chroot/named/*
# chmod 0666 chroot/named/dev/null
# chmod 0644 chroot/named/dev/random
# chmod 0664 chroot/named/var/run/named/named.pid
```

If the name server has additional zone files (for instance, if it is a zone slave or master), then additional directories and file copies are required for the relevant files. If a default rndc configuration has been created (using rndc-confgen -a), then the key file needs to be copied as shown. If rndc has been disabled with an empty controls clause (control {};), then this file is not required. If a custom rndc configuration has been built, then /etc/rndc.conf needs to be copied together with any specific .key file. Although Linux device types tend to remain stable, it may be worth verifying that the major and minor device numbers are as shown in the mknod commands by issuing the following command:

```
# ls -l /dev/null
crw-rw-rw- 1 root root 1,3 Feb 23 2004 /dev/null
# ls -l /dev/random
crw-r—r— 1 root root 1,8 Feb 23 2004 /dev/random
```

Finally, named may be started using the following command:

```
# named -u named -t /chroot/named
```

Assuming named must be started at system boot time, the startup script (/etc/rc.d/init.d/named) needs to be edited to add the -t /chroot/named argument.

The preceding configuration is a simplified version using a minimum of commands to show the process involved. If more complex configurations are required, then the procedures and techniques described in the section "Limit Permissions" may be applied.

FreeBSD Chroot

FreeBSD users have two options. The first method assumes that all files will use the standard (default) FreeBSD locations, and simply involves adding the following three lines to /etc/rc.conf:

```
named_chrootdir="/chroot/named" # Chroot directory (or "" not to auto-chroot it)
named_chroot_autoupdate="YES" # Automatically install/update chrooted
named_symlink_enable="YES" # Symlink the chrooted pid file
```

The preceding code overrides any values in /etc/defaults/rc.conf and automatically configures the required values, including directory creation according to the FreeBSD standard (all files are stored under the etc/namedb directory) at the next system boot.

The second method should be used if non-FreeBSD default locations are being used for any files. This method uses the same command sequence as defined for Linux in the preceding section, with the exception that the values on the mknod commands should be verified using the ls -l commands as shown for Linux. The current values for FreeBSD 4.x are as follows:

```
# mknod chroot/named/dev/null c 2 2
# mknod chroot/named/dev/random c 2 3
```

The values for FreeBSD 5.x are as follows:

```
# mknod chroot/named/dev/null c 2 3
# mknod chroot/named/dev/random c 249 0
```

Finally, the following lines need to be added to `/etc/rc.conf` if not already present:

```
named_enable="YES"  # Run named, the DNS server (or NO).
named_program="/usr/sbin/named"  # path to named, if you want a different one.
named_flags="-u bind -t /chroot/named"   # Flags for named
named_chrootdir="" # Chroot directory (or "" not to auto-chroot it)
```

The second method bypasses the default chroot initialization process, and allows much tighter control over configuration—at the expense of the user doing all the work.

Dedicated Server

The ultimate in permission limitation or the ultimate sandbox is a dedicated server either running as part of a Stealth server configuration (see Chapter 4) or as a stand-alone server. Such a server relies on minimalism to reduce the possibility of subversion, and would typically look something like the following:

- No GUI interfaces, to reduce software complexity
- No compilers or other development tools
- Firewall (packet filter) to inhibit access to all ports other than port 53
- No remote access to system—Secure Shell (SSH) or BIND (`rndc`)
- No Network File System (NFS) or Samba connections
- Removal of all unnecessary utilities; for example, Telnet, FTP, and so on
- BIND or NSD software running in a sandbox and typically configured as an authoritative-only server

Stream the Log

If security is a significant concern, then monitoring for security violations using intrusion-detection software such as Snort (`www.snort.org`) is important, but such tools lie outside the scope of this book. However, BIND's logging features can assist in this process by streaming security messages into a separate log file to minimize the work content of scanning logs manually, and hence the likelihood of missing key events. The following `named.conf` fragment streams the security events into a separate log:

```
// named.conf fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel security_log { // streamed security log
    file "/var/log/named/security.log" versions 3 size 500k;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category security{
    security_log;
    };
};
....
```

The `severity` setting (see Chapter 12) can be experimented with to find the most acceptable value to balance volume and information. BIND's `server` clause with a `bogus yes;` statement or the `blackhole` statement can be used to inhibit service completely to a persistent security offender.

Software Diversity

Significant effort has been spent by many of the root-server operators to minimize exploitable risks by running BIND on multiple host operating systems (for example, Linux, Solaris, FreeBSD, and so on), to reduce exposure to a single weakness. The theory is that if an exploit is discovered in one OS, it is unlikely to be present in all OSes at the same time. Therefore, only the vulnerable systems can be retired immediately while service continues. The NSD package (`www.nlnetlabs.nl/nsd`), which is an Open Source authoritative-only name server, has been running in the RIPE operated root-server (`k.root-servers.net`) since 2003, and fully supports DNSSEC.bis features. If the thought of a single BIND exploit taking all your systems off the air at the same time keeps you awake at night, then the possibly significant additional effort of maintaining a second version of DNS software may be worthwhile.

A Cryptographic Overview

The next sections and Chapter 11 include techniques that make extensive use of modern cryptographic processes. This section is designed to give the reader a brief overview of the terminology used, as well as the functionality and limitations associated with each technique. The mathematical processes used in the cryptography are treated as automagical ("stuff happens") and are not described at all. For a cryptanalyst, such a statement is pure heresy. However, understanding how the math works in the actual algorithmic processes is not necessary to understand the security concepts. Additional resources are provided at `www.netwidget.net/books/apress/dns` for those readers who revel in the gruesome mathematical details. However, before ignoring the mathematics entirely, it is important to understand a couple of points. Cryptographic techniques are not provably secure. Instead, they are exposed to attacks by dedicated researchers and specialists. Only after having weathered such attacks are the techniques made available for operational use. Research is ongoing to keep ahead of the bad guys, and occasionally results in new weaknesses being discovered. Finally, all cryptographic techniques are based on a concept known as *computationally infeasible*. This means either it would cost too much to assemble the computing power necessary to find the key, or that it would take too long. This concept is relative, not absolute, and changes over time.

Cryptology can be used for three purposes:

- *Confidentiality*: Only the parties to the communication can understand the messages sent between the parties.
- *Authentication*: The data could only have come from a known source.
- *Data integrity*: The data that is received by one party is the data that was sent by the other party.

In the context of DNS standards, only authentication and data integrity are of interest. Where confidentiality is required, it is assumed to be provided by a communications process such as SSL or its successor TLS, and is not defined within the DNS standards. BIND does support SSL.

Most of us have been cryptographers at some stage in our lives. The secret codes and methods we invented to send notes to our school friends also reflect, perhaps crudely, the earliest cryptographic processes whereby the "secret" was contained in the process. For example, we could shift the letters two positions in the alphabet and encode the message. The disadvantage with this method was that once the process was discovered, the algorithm was useless; it had to be discarded and a new one invented. By contrast, modern cryptography assumes that the algorithm used—the method of encryption—is known to everyone, including the bad guys, and indeed can only be proven to be secure by repeated attack. The secret part of the process lies with a unique key or keys. If a key is compromised, it is simply discarded and a new one created. An attacker must start again with no greater knowledge than before, even though the basic algorithm or process stays the same. There are two classes of key-based cryptographic algorithms in modern usage: symmetric and asymmetric.

Symmetric Cryptography

Symmetric encryption algorithms, also called *single-key*, *shared-secret*, or even, confusingly, *private-key* systems, use a single key to encrypt and decrypt the data. This single key—the shared secret—must be securely exchanged between the parties that will use it prior to the actual secure communication. The limitations of shared-secret systems are twofold. First, the key must be distributed securely using a process called key management, which itself is not trivial. Second, the method of securing the key once distributed lies with all the parties to the communication: "I trust myself but do I trust all the other parties to keep the key secret?" Examples of common symmetric key algorithms are DES, AES, IDEA, and RC4, and typical key sizes are 64, 128, or 192 bits. Figure 10-2 shows the operational use of a shared secret for classic confidential communications.

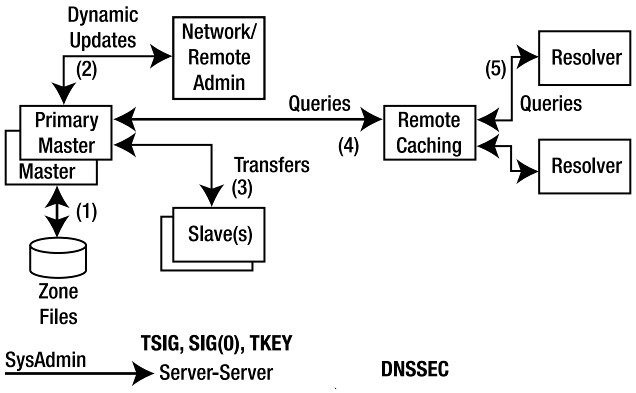


Figure 10-2. *Symmetric, or shared-secret, cryptography*

■**Note** The term *shared secret*, which describes a single key used, or shared, by both ends of the communication should not be confused with *secret sharing*, which describes a process whereby the shared, or single, secret key is broken up into parts and shared between multiple persons to make it more secure.

Shared-secret algorithms are used in the DNS in TSIG operations. The problem of distributing the keys (key management) is not defined in the DNS standards, and can be anything that works for the user; for instance, telephone, fax, secure e-mail, or carrier pigeon. The shared-secret key(s) used by DNS software must be constantly available (known as *on-line* in the jargon) to allow their use when validating transactions. However, the keys require minimum visibility; thus, it is impossible to store them in the zone file. Instead, such keys are stored in one or more `key` clauses within BIND's `named.conf` file. Due to their extremely sensitive content (a shared secret), they are normally stored as separate files with limited read permissions and included (using the `include` statement) into the `named.conf` file.

Asymmetric Cryptography

Asymmetric encryption algorithms use a pair of keys and are generally referred to as *public-key cryptographic systems* or sometimes as *nonsecret encryption* (a slight oxymoron). In these systems, data (called *plain-text* in the jargon) that is encrypted with one key can only be decrypted with the paired key. Given one key, it is computationally infeasible to derive the paired key. The system works by making one key, called the public key, widely available, while maintaining the other key, surprisingly called the private key, a secret. This process has an interesting side effect. If a message is encrypted with a private key and can be decrypted with its paired public key, then only the owner of the private key could have done it. This property is used in digital signatures and is described further in the section "Digital Signatures." The most widely used public-key encryption systems are RSA (after the inventors Rivest, Shamir, and Adelman) and elliptic curves. Typical key sizes for public-key systems are 512 bits, 1,024 bits, or higher. The public keys of a private/public key pair can be safely stored in a public service such as DNS, while the private key must be maintained securely in a private location. Figure 10-3 illustrates the use of public-key cryptography for classic confidential communications.

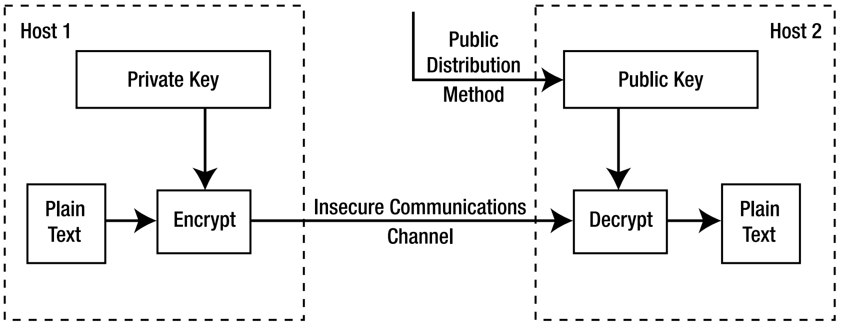


Figure 10-3. *Asymmetric or public-key cryptography*

Public-key systems have one significant limitation, in that they rely on knowing, or trusting, that the public key that will be used in communications with a person or organization really is the public key of the person or organization and has not been spoofed by a malicious third party. The method by which this is usually accomplished is sometimes called a Public Key Infrastructure (PKI), in which a trusted third party securely manages public keys. If the third party is requested to provide the public key of *X*, they are trusted to provide the correct key. The third party is trusted to have satisfied themselves by some process—attestation, notarization, and so on—that *X* is the one and only, or globally unique, *X*.

Message Digests

As stated previously, DNS systems require authentication and data integrity, not confidentiality. To provide data integrity, the message could be simply encrypted. Thus, only the possessor of the single key (in symmetric systems) or the public key (in asymmetric systems) could decrypt it. However, encryption systems use complex mathematical functions, and are therefore big users of CPU resources. To encrypt all messages would incur unacceptably high overheads. Fortunately, other techniques can be used to reduce this load. The most common is a lightweight procedure called a *one-way hash*, or more commonly a *message digest*. The hash or digest creates a unique and relatively small fixed-size block of data (irrespective of the original message length) that cannot be reversed. The messages being sent typically include both the plain text (unencrypted) and a digest of the message. The hash algorithm is applied to the received plain text and if the result matches the message digest, this means the received data was not altered. The message digest is in some senses similar in concept to a checksum but has significantly different mathematical properties. The most common forms of message digest are MD5 and SHA-1 (part of the SHA family). Figure 10-4 shows the message digest in action.

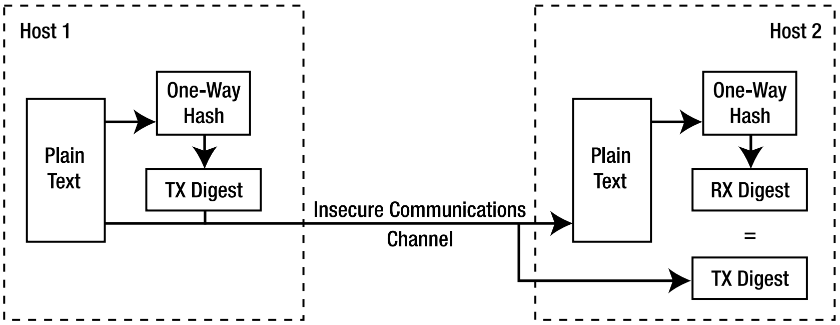


Figure 10-4. *Message digests*

Message Authentication Codes

Two possible solutions exist for authenticating the sender as well as ensuring integrity. In the case of symmetric, shared-secret systems, a Message Authentication Code (MAC) is created that combines the message digest with a shared key. The key part authenticates the sender, and the hash part ensures data integrity. The most common forms of MACs are HMAC-MD5 and HMAC-SHA-1. MACs are used for TSIG secure operations in DNS. Figure 10-5 shows how the MAC is used.

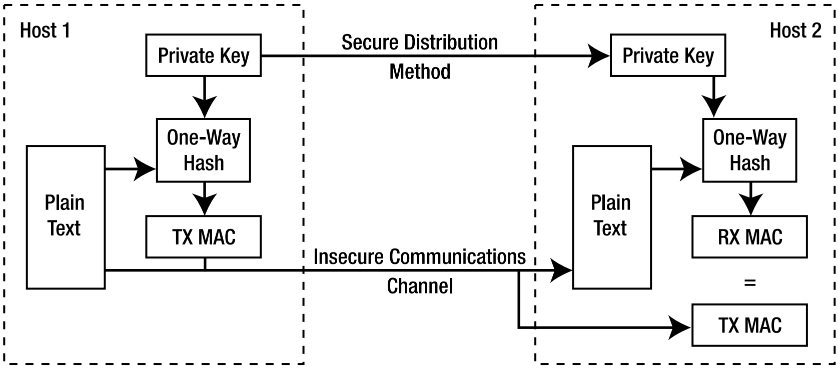


Figure 10-5. *Message Authentication Code*

Digital Signatures

In the asymmetric or public-key world, the process of authentication and data integrity uses what is called a *digital signature*. The message being sent is again hashed to create a message digest using, say, MD5 or SHA-1 to ensure data integrity. The resulting message digest is then encrypted using the private key of the sender. Both the plain-text message and the encrypted digest are sent to the other party. The receiver decrypts the message digest using the public key of the sender, applies the hash algorithm to the plain-text data, and if the results match, then both the authenticity of the sender and the integrity of the data are assured. Typical key sizes for digital signature systems are 512 bits, 1,024 bits, or higher. The most common digital signature algorithms are RSA-MD5, RSA-SHA-1, and Digital Signature Architecture (DSA; a US Government standard). Digital signatures are used in the DNS for SIG(0) secure transactions and for all DNSSEC transactions described in Chapter 11. Figure 10-6 shows how the digital signature is used.

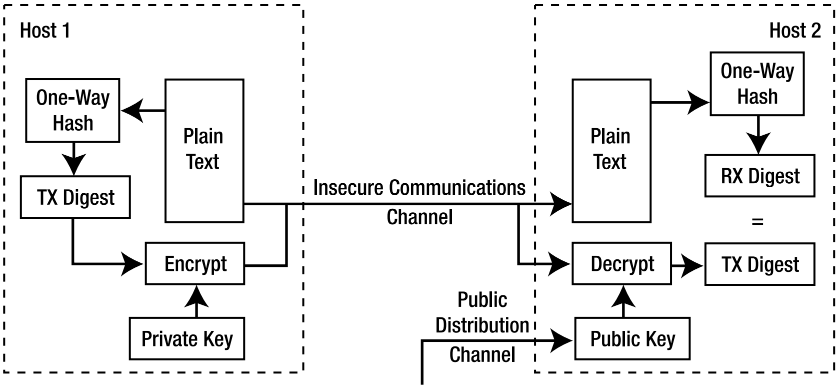


Figure 10-6. *Digital signatures*

■ **Note** The MD5 hash algorithm, and by implication any algorithm that uses it, such as RSAMD5, has been moved to a "not recommended" status in most IETF documents, due to some theoretical weaknesses published in early 2005. These weaknesses do not invalidate the use of the algorithm.

DNS Cryptographic Use

The DNS standards that cover security—generically and confusingly referred to as DNSSEC—use cryptographic security in two distinct ways. Transaction security, such as that used in zone transfer and dynamic updates, uses a point-to-point security model in which both parties to the transaction are assumed to trust each other. The parties exchange information, including security information, that authenticates the source and data integrity and is relevant only for that transaction. TSIG (shared-secret) and SIG(0) (public key) methods are used to perform the validation. Both methods are described, with examples, later in this chapter.

Client-server security, now known as DNSSEC.bis, allows the receiving DNS to validate the source and integrity of data received in response to any query from a suitably configured DNS. For such a system to work, it relies critically on an assurance that the source of the data is what it says it is. This problem, which was described in the preceding section, normally relies on the presence of a PKI, whereby a trusted third party verifies that some information, normally a public key, belongs to *X*, and that *X* is truly the one and only *X*. DNSSEC.bis security does not rely on a PKI, but instead creates a hierarchy or *chain of trust* based on the delegation of DNS names. A trusted party forms the root or the Security Entry Point (SEP) of the chain of trust, in which certain RRs at the point of delegation are cryptographically signed (using a digital signature) by the parent zone. This creates a secure link to the next domain in the chain, which in turn signs the delegation records and so on, until the end point has been reached. The authenticity of each link in the chain, with the exception of the starting point, is verified by the previous, or parent, domain. DNSSEC.bis security is described in Chapter 11.

The nature of secure systems is that they must safeguard against many forms of attack. One attack form is called a replay attack, in which a transaction is captured and replayed at a later time. To avoid such forms of attack, all systems involved in cryptographic security must be time synchronized. The protocols typically allow a "fudge" factor of 300 seconds (5 minutes), but the implementation of Network Time Protocol (NTP) is essential in systems that use cryptographic techniques. Implementation of NTP lies outside the scope of this book, but Open Source implementations are available for most major OSes and their distributions. Further information may be obtained from `www.ntp.org`, including a list of public time servers.

■ **Caution** Time synchronization for all hosts involved in cryptographic exchanges is crucial. BIND failure messages do not always indicate clearly that time is the source of a failure in authentication, when 90% of the time that is indeed the problem. NTP uses an incremental approach to synchronizing clocks, and can take a considerable period to adjust the clocks on any host system. If you are not running NTP software and wish to experiment with the techniques described throughout this chapter and Chapter 11, then each host that will participate should synchronize its clocks to Internet time by issuing an `ntpdate name.of.time.server` command. In this command, `name.of.time.server` should be replaced with some accessible time server; a list of publicly available time servers can be found at `www.ntp.org`. Note that `ntpdate` is a one-time update, and the accuracy of the local clock determines how long its effect will last. Operational systems that participate in DNSSEC must implement NTP.

Securing Zone Transfers

In most DNS configurations, zone transfers are essential. If you are of a security-conscious frame of mind, perhaps zone transfers are viewed as a necessary evil. The default option in BIND is to allow zone transfers to any requesting host. Although on its face this may look like a remarkably friendly act, it is based on the simple premise that a public DNS contains public data. Everything that is transferred can be discovered by exhaustive queries, even if zone transfers are completely banned. If data should not be public, it should not be in the zone file on a public server. Simply securing zone transfers is not a solution to hiding data. Nevertheless, there are cases where it is necessary as part of a security-in-depth configuration to restrict zone transfers—for example, on the private side of a Stealth server configuration (see Chapter 4). The simplest way to secure zone transfers is through the use of IP addresses in BIND's `named.conf` file. The following `named.conf` fragment limits transfers to `named` hosts based on the zone name:

```
// named.conf fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel security_log { // streamed security log
    file "/var/log/named/security.log" versions 3 size 2m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category security{
    security_log;
    };
};
options {
....
    allow-transfer {none;}; // none by default
....
};
....
zone "example.com in{
....
    allow-transfer {10.1.2.5;}; // this zone only
....
};
```

The preceding configuration fragment denies all zone transfer requests and selectively permits the allowable hosts on a per-zone basis. For instance, the single IP address 10.1.2.5 is allowed to perform zone transfers for the zone `example.com`. The log is streamed for security events, because it is assumed that as part of this defensive strategy it is of interest to see where transfer requests are coming from. If necessary, after log inspection a `server` clause with `bogus yes;` or a `blackhole` statement could be used to stop service completely to a persistently inquisitive host.

Given the right circumstances, IP addresses can be spoofed, which can result in *man-in-the-middle* attacks such that a third party may pretend to be the zone master. When requested to transfer a zone, this third party could transfer counterfeit data resulting in, say, a web site being hijacked by providing alternate IP addresses in the Resource Records (RRs). To prevent such a possibility, zone transfers can be secured through the use of cryptographic techniques to ensure both authentication (the master and slave are who they say they are) and data integrity (the data received by the slave was the same as the data sent by the master).

Authentication and Integrity of Zone Transfers

The bad news is that of the three methods for securing zone transfers, for practical purposes, there is only one, as may be seen from the following list:

- *TSIG*: TSIG was defined in RFC 2845 and uses a single shared secret between the master and slave servers as part of a MAC. The key must be distributed to the slave locations by some secure process, such as fax, mail, courier, or secure e-mail, and it must be maintained securely at all the sites. Shared secrets, because they rely on a single key maintained at two or more locations, should be changed frequently (perhaps every 30 to 60 days). If there is more than one slave server, either separate shared secrets may be used for each master-slave pair, or a single shared secret may be used for all slaves. The latter policy is significantly riskier, because any subversion or discovery of the key at a single site invalidates all slave transfers, whereas if separate secrets are used the subverted slave can be temporarily disabled until the key is replaced. There is no change to the operational zone files when using the TSIG method; only the `named.conf` file is modified.

- *SIG(0)*: SIG(0) was defined in RFC 2931 and uses a public-key system to generate a digital signature that both authenticates and ensures the integrity of the data involved in each transaction that includes zone transfers. However, there are no tools available with current BIND releases to support SIG(0) for zone transfers. SIG(0) may be used with DDNS; see the section "Securing Dynamic Updates" later in the chapter.

- *TKEY*: The TKEY provides a method of securely exchanging shared-secret keys so that the poor carrier pigeons can have a rest (or whatever other method you use to securely distribute shared keys). The method is defined to support both the Diffie-Hellman algorithm and the Generic Security Services API (GSSAPI). However, the standard (RFC 2930) mandates that the exchange must be authenticated with either TSIG or SIG(0) methods. Consequently, it appears not to be widely implemented, and is not covered further in this book.

For practical purposes, the only method available to secure zone transfers is TSIG. The detailed configuration required to support this service is covered in the following section.

TSIG Configuration

Transaction Signatures (TSIG) use a Message Authentication Code (MAC) with a shared secret both to authenticate and ensure the data integrity of every transaction involved in zone transfers between the nominated slave and its master. It is vital to keep in mind that shared-secret data is never placed in the DNS zone files. Instead, the shared secret is used by the two servers when exchanging data, such as a zone transfer. Figure 10-7 illustrates how shared secrets are used in securing transactions.

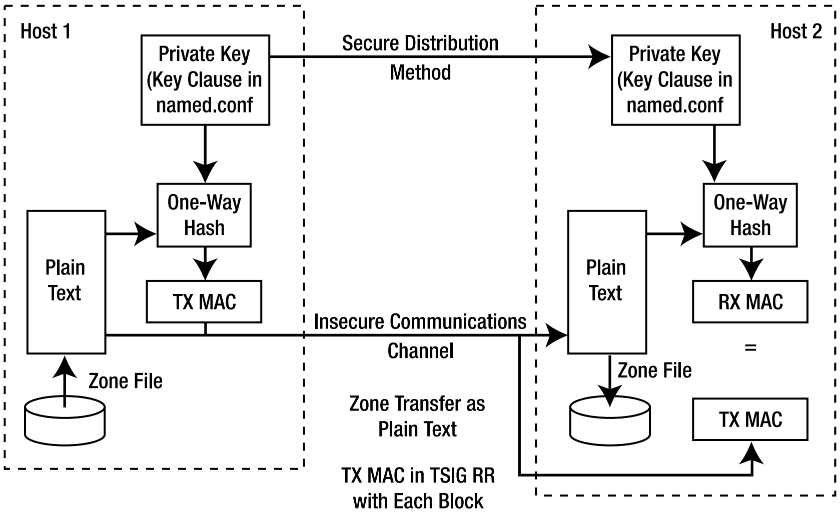

Host 1
Private Key (Key Clause in named.conf
Secure Distribution Method
Host 2
Private Key (Key Clause in named.conf
One-Way Hash
Plain Text
TX MAC
Zone File
Insecure Communications Channel
Zone Transfer as Plain Text
TX MAC in TSIG RR with Each Block
One-Way Hash
Plain Text
RX MAC
=
Zone File
TX MAC

Figure 10-7. *Shared-secret TSIG*

The shared secret is generated using the `dnssec-keygen` utility, which is the general-purpose cryptographic utility provided with BIND and is described in Chapter 9. The TSIG standard (RFC 2845) allows both HMAC-MD5 and HMAC-SHA-1 algorithms to be used as MACs. However, the current release of the `dnssec-keygen` utility only supports the HMAC-MD5 algorithm. The shared-secret key is assumed to be generated in a directory called `/var/named/keys` using a command similar to the following:

```
# cd /var/named/keys
# dnssec-keygen -a hmac-md5 -b 128 -n host example.com
```

This command generates a 128-bit key (the `-b` argument) suitable for use with the HMAC-MD5 MAC algorithm (HMAC-MD5 allows keys from 1 to 512 bits). The `-n host` argument indicates that a host KEY RR is generated with a name of `example.com`. This KEY RR is not used in TSIG transactions for reasons explained later in this section. However, the `dnssec-keygen` program treats the `-n` argument as mandatory, so it must be present. The command writes two files to the current directory, and when complete outputs a short message to identify the created files, as shown here:

```
Kexample.com.+157+31313
```

The preceding file name consists of the fixed value K, followed by the host name reflected from the dnssec-keygen command (in this case example.com). 157 identifies the algorithm (HMAC-MD5). The 31313 is called the key-tag; it is generated using a variant of the "one's complement" checksum algorithm to identify this key set uniquely. Looking in the directory in which the files were written displays two files:

```
Kexample.com.+157+31313.private
Kexample.com.+157+31313.key
```

Viewing the file Kexample.com.+157+31313.private displays something like the following data:

```
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: JuxDyYXIJhAia5WQe9oqUA==
```

The preceding information contains three lines. The line beginning with the text Key: is the base64 (RFC 3548) encoded version of the shared-secret key. The next step is to edit this data into a key clause that will be used in the named.conf file, as shown here:

```
key "example.com" (
     alogorithm hmac-md5;
     secret JuxDyYXIJhAia5WQe9oqUA==;
};
```

The key name example.com, which can be a quoted string and contain spaces, or unquoted if there are no spaces, is normally the name used as the hostname in the dnssec-keygen command, as in the preceding case. Depending on the application, it can be any useful string, as long as the same key clause name is used by both parties in the transaction. In the example case, both parties (master and slave) contain a key clause with the name example.com, as shown in the example fragments that follow. The name of the key clause could have been "transfer-key" if that was more meaningful; again, the same key clause name must be used by both parties. For a TSIG transaction, there is no required relationship between the name used in the -n argument of the dnssec-keygen utility and the name of the key clause. The name defined in the key clause is sent in the TSIG meta (or pseudo) RR with each secure transaction to identify the shared secret being used. If the key clause name is not the same in each party, the transaction will fail with a BADNAME error. The algorithm line of the key clause identifies the algorithm being used (hmac-md5 as defined in the dnssec-keygen command). The data following secret is a copy of the data from the Key line of the Kexample.com.+157+31313.private file, terminated with a semicolon. This key clause should be saved as separate file—we'll call it example.com.key—and placed in a directory we'll call /var/named/keys and included in the named.conf file. This file, containing the shared-secret key clause, must now be made available by some secure process (such as floppy disk, secure e-mail, or other secure service), to the slave server or servers. Because this file contains highly sensitive data it should be immediately secured on the master and slaves such that it can only be read with the UID of BIND. The commands to secure the file look something like this:

```
chown named:named /var/named/keys/example.com.key
chmod 0400 /var/named/keys/example.com.key
```

The preceding commands assume that BIND is being run with the `-u` argument (as described earlier in this chapter) and allow BIND's UID read access to the file. However, the `root` user can both read and write as normal if subsequent modification is required. Alternatively, you can use a `chmod` setting of 0600 and allow all editing to be done under the BIND UID if you have a deep-seated objection to using `root` for anything nonessential.

■**Note** The UID is assumed to be `named`, as shown in the preceding example. `named` is the normal value used with Linux and Windows. However, FreeBSD typically uses a UID of `bind`.

Viewing the file `Kexample.com.+157+31313.key` shows the following text:

```
example.com. IN KEY 512 3 157 JuxDyYXIJhAia5WQe9oqUA==
```

This is a DNS-ready KEY RR containing the shared-secret key! It is generated as an artifact of the `dnssec-keygen` standard processing; unfortunately, there is no way to prevent it. The KEY RR is *never* used with any shared-secret algorithms and *must not under any circumstances* be added to the zone file. Instead, only the `named.conf` file `key` clause contains the shared-secret key that is used independently by both ends during the communication, as illustrated in Figure 10-7. Once the `key` clause is established on both the master and slaves, either secure `Kexample.com.+157+31313.key` and `Kexample.com.+157+31313.private`, or better still, delete these files completely—they will not be used again and represent an additional security headache.

The `named.conf` file at the master will look something like the following fragment:

```
// named.conf example.com master fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   ....
};
// include the key clause for example.com key name
include "keys/example.com.key"; // include the key clause
// server clause references the key clause included above
server 10.1.2.3 {
   keys {"example.com";}; // name used in key clause
};
....
zone "example.com" in{
   type master;
   file "master.example.com";
   // allow transfer only if key (TSIG) present
   allow-transfer {key "example.com";};
};
....
```

To assist in testing and experimentation, the log has been streamed to log DNSSEC events separately, as shown in the preceding fragment. The `severity debug 3;` line generates copious amounts of logging and should be used during testing only. In a production environment, this value can be set to `severity info;` or higher. DNSSEC is not turned on by default in BIND. The `dnssec-enable yes;` statement *must* be placed in the global `options` clause to invoke the feature. The `key` clause contained in the file `keys/example.com.key` must appear before it is referenced in the `server` clause, as shown in the preceding fragment. The `server` clause defines the IPv4 address of the slave server for `example.com`, and the `keys` statement in this clause references the `key` clause containing the secret key to be used. The `allow-transfer` statement in the `zone` clause for `example.com` is an `address-match-list` construction using the `key` option (see the section in Chapter 12, "BIND `address_match_list` Definition") and provides the linkage to validate incoming TSIG messages. The corresponding slave server `named.conf` file looks something like that shown here:

```
// named.conf example.com slave fragment
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   ....
```

```
include "keys/example.com.key"; // include the key clause
server 10.1.2.5 {
   keys {"example.com";}; // name used in key clause
};
....
zone "example.com" in{
   type slave;
   file "slave.example.com";
   masters {10.1.2.5;};
};
```

The `key` clause again is included from the file `keys/example.com.key` (remember both sides are sharing this key) and must appear before it is referenced in the `server` clause, which in this case is the IPv4 address of the zone master for `example.com`. The `masters` statement in the `zone` clause for `example.com` contains an IPv4 address to link it to the `server` clause. This triggers the initiation of the authentication sequence using the defined `keys` statement. Although the `masters` statement can contain a `key` option in this case, because the slave initiates the request for zone transfer it must know where to send it, so it uses an IP address. The corresponding `allow-transfer` statement in the zone master fragment can use the `key` format because it is *responding* to the request.

For those with insatiable curiosity, it may be worthwhile to look at the resulting zone transfer with a suitable sniffer application (see Chapter 15). A meta (or pseudo) RR called TSIG, containing the MAC for each transaction, and with a name of the shared-secret `key` clause is placed in the `ADDITIONAL SECTION` of the query and its response (see Chapter 13 for an explanation of meta RRs). In this case, the response is the zone transfer (AXFR). These TSIG RRs are discarded once the message has been verified; that is, they are not saved as part of the zone transfer data.

■**Note** The KEY RR generated as part of the `dnssec-keygen` process (contained in the `.key` file) is used in public-key systems only. When using shared-secret techniques such as TSIG, the KEY RR is an annoying and dangerous artifact, and must not be placed in the DNS zone file. Unless there are good reasons not to, it should be deleted immediately.

Securing Dynamic Updates

Dynamic DNS (DDNS) was defined in RFCs 3007 and 2136, and describes a process whereby RRs for a zone can be added, deleted, and modified by a third party. However, zones cannot be deleted or added using this process. To ensure consistency of zone data, the dynamic updates are only carried out on the *primary master* server, which is defined as the name server that appears in the SOA RR for the zone (the `MNAME` field). The BIND default is to disallow dynamic updates from all IP addresses. Dynamic updating is a powerful capability, and many sites use it extensively to enable customers to edit their zone data directly, and in some cases to synchronize Dynamic Host Control Protocol (DHCP) with both forward and reverse mapping files automatically. As with all positives, there is an accompanying negative: unscrupulous access by malicious third parties can corrupt or poison the zone file. As previously stated, not securing DDNS unless it occurs behind a secure perimeter and between consenting adults constitutes an over-reliance on the essential goodness of mankind. It is imperative to secure DDNS. The simplest way to secure DDNS is through the use of IP-based restrictions. The following fragment uses BIND's `allow-update` statement to limit access:

```
// named.conf fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel security_log { // streamed security log
    file "/var/log/named/security.log" versions 3 size 2m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category security{
    security_log;
    };
};
options {
....
};
....
zone "example.com in{
....
    allow-update {10.1.2.5;}; // this zone only
....
};
```

The preceding configuration fragment denies all dynamic updates and selectively permits the allowable hosts on a per-zone basis. For example, the single IP address 10.1.2.5 is allowed to perform updates for the zone `example.com`. The log is streamed for security events because it is assumed that as part of this defensive strategy, it is of interest to see where update requests are coming from. If necessary, after inspection of the `security.log` file, a `server` clause with `bogus yes;` or a `blackhole` statement could be used to stop service completely to a persistent host.

Given the right circumstances, IP addresses can be spoofed, which can result in the bad guys doing naughty things to the master zone file. To prevent such a possibility, dynamic updates can be secured through the use of cryptographic techniques to ensure both authentication (the master and slave are who they say they are), and data integrity (the data received by the master being updated was the same as the data sent by the client performing the update).

Both TSIG and SIG(0) methods are supported by the `nsupdate` utility provided with BIND releases and described in Chapter 9. Implementation of both TSIG and SIG(0) methods is described in the following sections.

TSIG DDNS Configuration

TSIGs use a Message Authentication Code (MAC) with a shared secret both to authenticate and ensure the data integrity of every transaction involved in dynamic updates between the primary master and the update source. The method of generating the shared secret is exactly the same as that defined for the earlier section "TSIG Configuration," and is not repeated here. The shared secret is not shared with another name server in this case, but with the source of the dynamic updates; for instance, the `nsupdate` utility. Again, it is vital that the KEY RRs generated as part of the `dnssec-keygen` process *must not* be added to the zone file. When using a shared-secret algorithm such as TSIG, the `key` clause or clauses in the `named.conf` file—which is assumed not to be a public file—store the secret keys.

■**Note** It is possible to use the same shared-secret key to perform both dynamic update and zone transfer authorization, especially if the same host is being used for both operations. However, in general, a separate shared secret should be used for every host pair because this minimizes exposure to compromised keys.

The `named.conf` file fragment to support the dynamic update is shown in the following code, using both the `allow-update` and the `update-policy` statements:

```
// named.conf example.com master fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   ....
};
include "keys/example.com.key"; // include the key clause
server 10.1.2.3 {
   keys {"example.com";}; // name used in key clause
};
....
zone "example.com" in{
   type master;
   file "master.example.com";
   allow-update {key "example.com";};
};
....
zone "example.net" in{
   type master;
   file "master.example.net";
   update-policy { grant example.com subdomain example.net ANY;};
   update-policy { grant * self * A;};
   update-policy { grant fred.example.net name example.net MX;};
};
....
```

To assist in testing, the log has been streamed to log dnssec events separately, as shown in the preceding fragment. The severity debug 3; line generates copious amounts of logging and should be used during testing only. In a production system, this value can be set to severity info; or higher. DNSSEC is not turned on by default in BIND, so the dnssec-enable yes; statement must be placed in the global options clause to invoke the feature. The allow-update statement in the zone clause for example.com uses the key option of the address_match_list to permit any updates to the example.com zone file. The zone clause for example.net uses update-policy statements to provide tight control over what can be done and by whom. The first update-policy statement allows a TSIG transaction with the name example.com to update any record in the zone file example.net. The keyword subdomain means that the following parameter, in this case example.net, is treated as a base name. Any name that includes or terminates with example.net matches; for example, joe.example.net terminates with example.net and therefore it matches, as would the MX RR for the domain. The second update-policy statement allows any TSIG transaction with a name of, say, bill.example.net and for which there is a key clause with the same name (bill.example.net) to update only an A RR with a name of bill.example.net. The additional key clauses are not shown in the example, but this construct requires a key clause and a unique shared secret for every possible A RR that could be updated. The final update-policy statement says that a TSIG transaction with a name of fred.example.net is allowed to update only the MX RR(s) for the domain example.net.

To reinforce the process of key generation for shared-secret applications, the following sequence shows creation of the shared secret for fred.example.com. This shared secret is used in the last update-policy statement in the zone clause for example.net in the preceding fragment. Use the following command to generate the key:

```
dnssec-keygen -a hmac-md5 -b 128 -n host fred.example.net
```

When complete, the command responds with a file identifier such as the following:

```
Kexample.com.+157+32713
```

Create a new key clause with a name of fred.example.net using the data from the Key: line of the file called Kexample.com.+157+32713.private, as shown here:

```
key "fred.example.net" {
     alogorithm hmac-md5;
     secret 7aBDy3XIJhA775WQ4FoqUA==;
};
```

Add this key clause to the existing file example.com.key, which contains the original key clause we created, or create a new file and add a new include statement in named.conf. Finally, if the data is added to the existing file or a new file is created, remember to check that the file permissions only allow read, or read and write access only, for the BIND UID. To illustrate the dynamic update process in action, the example uses the nsupdate utility supplied with all BIND releases. In this case, we use the example.com key, which can update both the example.com and example.net zone files. Before invoking the nsupdate utility, the files Kexample.com.+157+31313.private and Kexample.com.+157+31313.key need to be moved by a secure process into a suitable working directory on the host that will run the nsupdate utility. In this case we assume the directory is /var/named/dynamic.

■**Note** Recall from earlier that when using shared secrets, the file containing the KEY RR (in the preceding case Kexample.com.+157+32713.key), which is generated automatically by the dnssec-keygen utility, must not be added to the zone file. However, this file is required by the nsupdate utility for operational reasons. Once securely transferred to that host or hosts, it should be deleted from the primary master host.

The following sequence adds a new A RR to the zones example.com and example.net:

```
# cd /var/named/dynamic
# nsupdate -k Kexample.com.+157+31313.private
```

```
> server ns1.example.com
> zone example.com
> update add new 36000 IN A 192.168.5.4
> send
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERR id: 0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
> zone example.net
> update add another.example.net.  36000 IN A 192.168.7.15
> send
> quit
```

The preceding example shows adding an A RR to each of the domains `example.com` and `example.net`. The key file used with the `nsupdate` utility has a name of `example.com`, which has permission to update both `example.com` (via the `allow-update` statement in the example `named.conf` fragment) and `example.net` (through the first `update-policy` statement). A `dig` command can be used to verify that the new RRs are available, as shown here:

```
# dig @192.168.5.12 new.example.com A
; <<>> DiG 9.3.0 <<>> @192.168.5.12 new.example.com A
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1082
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;new.example.com.      IN  A

;; ANSWER SECTION:
new.example.com. 36000 IN  A  192.168.5.4

;; AUTHORITY SECTION:
example.com.       86400  IN  NS  ns1.example.com.
example.com.       86400  IN  NS  ns2.example.com.

;; ADDITIONAL SECTION:
ns1.example.com.  86400  IN  A  192.168.5.12
ns2.example.com.  86400  IN  A  192.168.5.11

;; Query time: 15 msec
;; SERVER: 192.168.5.12#53(192.168.5.12)
;; WHEN: Thu Apr 07 21:59:48 2005
;; MSG SIZE  rcvd: 124
```

The preceding output confirms that the update to the `example.com` domain was successful and is immediately available at the primary master. The update process automatically adds

1 to the sequence number field of the SOA RR. Unless disabled by a `named.conf` statement, a `NOTIFY` is sent to the slave servers for the zone, and the update is cascaded to all the slave servers within minutes.

■**Note** It is worth reminding readers that once a dynamic update is invoked, the zone file should not be manually edited because updates are initially written to a journal file (`zone.file.name.jnl`), and the zone file is only periodically updated. If manual editing is required, then either stop BIND and perform the edit, or use the `rndc` command `flush`, followed by `freeze zone.name`; perform the manual edit; and then `thaw zone.name`. In either case, the zone's `.jnl` file should be deleted before either restarting BIND or issuing the `rndc thaw` command to ensure subsequent consistency.

SIG(0) Configuration

The `nsupdate` utility also supports SIG(0) authentication and data integrity checking through the use of digital signatures, which are based on public-key technology. Public-key technology has the advantage that no special action is required to distribute the public keys. They are simply placed as KEY RRs in the zone file, and may be read by anyone, because without the matching private key they are useless. In an update sequence, the zone master uses the public key. The client performing the update uses the private key to generate the signature, which is verified by the receiving zone primary master. If an encrypted response is required, the server uses the public key to sign the response, which in turn is verified using the private key of the updating client. The downside of public-key technology is that it uses significantly more CPU resources than shared-secret technology. If any volume of updates is likely on a busy server, then use of SIG(0) may warrant careful consideration. As long as the key distribution and management problem associated with shared secrets can be handled, TSIG may be a better option. The term SIG(0) can be a little confusing, because there was a SIG RR type that performed a function similar to the current RRSIG RRs used in DNSSEC configurations. However, the SIG(0) RR used to secure transactions is a meta (or pseudo) RR type that is dynamically created by the sending application or server and added to the `ADDITIONAL SECTION` of the transaction (see Chapter 15). The SIG(0) RR is discarded immediately after verification. Specifically, it is not cached or added to the zone file. This form of the SIG RR is uniquely identified by having a type 0 in its `label` field (see the section "SIG RR" in Chapter 13), and hence has the name SIG(0).

The private and public keys for the SIG(0) transaction are created using the `dnssec-keygen` utility (see Chapter 9). Because the client that updates the zone uses the private key, key generation should be done on this host. If this is not possible, the generated files have to be moved to the client machine using a secure process. These commands generate a public/private key pair in the directory `/var/named/keys`:

```
# cd /var/named/keys
# dnssec-keygen -a rsasha1 -b 512 -k -n host update.example.com
```

In the preceding dnssec-keygen command, -a rsasha1 generates a digital signature using the RSA algorithm with the SHA-1 message digest (the dnssec-keygen utility supports the DSA, RSA-SHA-1, and RSAMD5 public-key methods). The -b 512 argument indicates the key will be 512 bits long. An RSA-SHA-1 key may be from 512 to 2,048 bits. The higher the number, the greater cryptographic strength of the key; however, more CPU is used in encryption/decryption. The -k argument indicates that a KEY RR type is required (not a DNSKEY RR). The -n host indicates a host KEY RR will be created with a name of update.example.com. When complete, the command will output a message similar to this:

```
Kupdate.example.com.+001+00706
```

K is a fixed value, update.example.com. is the name from the dnssec-keygen command, 001 indicates the algorithm (RSA-SHA-1), and 00706 is the key-tag that is algorithmically generated and uniquely identifies this key pair. Inspection of the directory /var/named/keys shows two files:

```
Kupdate.example.com.+001+00706.private
Kupdate.example.com.+001+00706.key
```

The file Kupdate.example.com.+001+00706.key contains a single KEY RR and looks something like the following:

```
update.example.com. IN KEY 512 3 1 (AQPL1jlhf7Of9l1P/h
  PFNMxU55IpkMX1O7EzvDk5OrhOeM7xF+YQdQKD
  brvR1rf6J8oTPFM2MM26sK98aj5MAsJX)
```

The preceding data has been edited to enclose the key material in parentheses (allowing it to be split across several lines for presentation reasons only), but it appears as a single line in the file. This is the public key associated with the public/private key pair, and it may be sent via any suitable method for inclusion in the master zone file for example.com, either by cutting and pasting, or by using the $INCLUDE directive. The Kupdate.example.com.+001+00706.key file containing the KEY RR is public data and requires no special handling. The following example shows the use of the $INCLUDE directive in the zone file for example.com. It assumes the .key file is placed in the directory /var/named/keys on the host of the zone master for example.com:

```
; example.com zone file fragment
$TTL 2d ; zone TTL default of 2 days
$ORIGIN example.com.
....
$INCLUDE keys/Kupdate.example.com.+001+00706.key ;DDNS key
....
```

The named.conf file on the primary master server must now be modified to allow the zone to be updated using an update-policy statement (an allow-update statement could also be used) in the zone clause, as shown in the following example:

```
// named.conf fragment
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   ....
};
....
zone "example.com" IN{
   type master;
   file master.example.com;
   update-policy {grant update.example.com subdomain example.com ANY};
};
```

■Note To assist in testing, the log has been streamed to provide additional information about DNSSEC events using `severity debug 3;`. This value should not be used in a production environment unless you like large log files. A setting of `severity info;` or higher should be used. Because DNSSEC is not enabled automatically, the `dnssec-enable yes;` statement must be present in the global `options` clause.

The preceding `update-policy` allows the KEY RR with the `name update.example.com` to update any RR in the domain `example.com`. The following `update-policy` statement only allows `update.example.com` to modify NS records for the domain:

```
update-policy {grant update.example.com subdomain example.com NS};
```

By careful selection of the host name when generating keys, fine-grained controls can be created at the cost of multiple key RRs. The following example illustrates how to use this process to allow individual users to modify only their own host records. The target zone file fragment is shown here:

```
; example.com zone file fragment
$TTL 2d ; zone TTL default of 2 days
$ORIGIN example.com.
....
bill   IN  A   192.168.2.3
       IN  TXT "one fine day"
       IN  RP  bill.example.com.
fred   IN  A   192.168.2.4
       IN  RP  fred.example.com.
       IN  AAAA 2001:db8::15
....
```

To control the process, two public/private key pairs with the preceding host names are generated, as shown here:

```
# dnssec-keygen -a rsasha1 -b 512 -k -n host bill.example.com
# dnssec-keygen -a rsasha1 -b 512 -k -n host fred.example.com
```

It is assumed that these keys are generated on the respective hosts `bill.example.com` and `fred.example.com`. The public KEY RRs are included in the zone file, as shown here:

```
; example.com zone file fragment
$TTL 2d ; zone TTL default of 2 days
$ORIGIN example.com.
....
bill   IN  A   192.168.2.3
       IN  TXT "one fine day"
       IN  RP  bill.example.com. .
$INCLUDE keys/Kbill.example.com.+001+77325.key ; bill KEY RR
fred   IN  A   192.168.2.4
       IN  RP  fred.example.com. .
       IN  AAAA 2001:db8::15
$INCLUDE keys/Kfred.example.com.+001+08634.key ; fred KEY RR
....
```

The following `named.conf` `update-policy` ensures that the appropriate key can update only its own A, TXT, AAAA, and RP records:

```
update-policy {grant * self * A AAAA TXT RP};
```

The first * says that a reference to any KEY RR with the same name (`self`) as the host record (the second *) is allowed (`grant`) to update the A, AAAA, TXT, and RP RRs only with the same host name. Thus, an incoming update with the name `bill.example.com` (references the KEY RR of bill.example.com) is only allowed to update or add any A, AAAA, RP, or TXT RRs with a host name of bill.example.com. Similarly, if the update uses the name fred.example.com, it can only update the defined RR types that have a host name of fred.example.com.

Having digressed to cover the use of update-policy, it is time to return to the original public-key example. The file Kupdate.example.com.+001+00706.private, which is located on the client that updates the zone file, looks something like this:

```
Private-key-format: v1.2
Algorithm: 1 (RSA)
Modulus: y9Y5YX+9H/ZdT/4TxTTMVOeSKZDF9TuxM7w5OdK4dHjO8RfmEH
UCg267Oda3+ifKEzxTNjDNurCvfGo+TALCVw==
PublicExponent: Aw==
PrivateExponent: h+QmQP/TaqQ+NVQNLiMy4OUMG7XZTifLd9L
QOTclovoCO/y4wq3QNg4jNa5kb4Y4UQfx/2HcK84HrM/T66fzew==
Prime1: /sRMFcz/OnBnuueuvvQi4SCKlKSCi1loWgPTHsmKLZO=
Prime2: zNLQux9xD8HxzBmiYl67lHklO5KbeB+TSfVfYaD8p4M=
Exponent1: qdgyuTNUOaBFJOUfKfgXQMBcYxhXB5DwPAKMvzEGyRM=
Exponent2: iIyLJ2pLX9ahMrvBlunSYvtujQxnpWpiMU4/lmtTGlc=
Coefficient: S5di+sst/DCqT5MSNaiNLPNODJWRjxivgkiifB7DPl4=
```

A number of the preceding lines have been split across more than one line for presentation reasons only. This file contains the private key of the public/private key pair and is only used by the nsupdate utility. It should be immediately secured for read-only permission under the UID of the user who will perform the dynamic update. For the purposes of illustration, it is assumed that the user name that will perform the update is updater, with a group name of users. The following commands secure the .private and .key files in /var/named/dynamic:

```
# chown -R updater:users /var/named/dynamic/*
# chmod -R 0400 /var/named/dynamic/*
```

To invoke and test the SIG(0) dynamic update process, the nsupdate utility is invoked and the following sequence is used to add an MX record and its corresponding A RR for the domain example.com:

```
# cd /var/named/dynamic
# nsupdate -k Kexample.com.+001+00706.private
> server ns1.example.com
> zone example.com
> update add example.com. 36000 IN MX 10 mail2.example.com.
> send
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERR id: 0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
> update add mail2 36000 IN A 192.168.2.5
> send
> show
```

```
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERR id: 0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
> quit
```

As with the TSIG example, a `dig` command can be issued to verify that the MX and A RRs are available at the primary master. By pointing the `dig` command at the slave servers, the cascaded update, initiated by a `NOTIFY` message, can also be verified.

It is possible to mix TSIG and SIG(0) dynamic update clients if that makes operational sense. It is also possible to support TSIG for zone transfers and SIG(0) for dynamic update operations, or any such combination.

Summary

This chapter introduced DNS security by categorizing the topic into administrative security, zone transfers, dynamic updates, and zone integrity. The first three topics are covered in this chapter; zone integrity using DNSSEC.bis is described in Chapter 11.

The administrative security discussion covered the selection and configuration of DNS servers and discussed software updating, limiting functionality, limiting permissions (including sandboxes or chroot jails), log streaming, and the use of multiple sources of both OS and DNS software to reduce the risks involved in running DNS systems. The packaged installation of a chroot jail on Linux Fedora Core 2 and FreeBSD was described, as well as the manual installation of a chroot jail in the absence of an available package.

The chapter described the use of cryptographic techniques to secure various transactions. The various techniques were described in outline for readers unfamiliar with general cryptographic processes, including symmetric (shared-secret) systems, asymmetric (public-key) systems, message digests, MACs, and digital signatures.

The use of simple BIND statements to secure zone transfers using IP addresses and the use of TSIG (shared-secret) transactions to secure zone transfers was described and illustrated with example files.

The chapter described, with examples, the use of BIND commands to secure dynamic updates using IP addresses. Both SIG(0), using public-key or asymmetric cryptographic techniques, and TSIG (shared-secret) methods to secure dynamic updates were described and again illustrated with example files and configurations.

The next chapter describes the design intent and implementation of DNSSEC (colloquially referred to as DNSSEC.bis) to ensure the source and integrity of zone data during normal query operations.

CHAPTER 11

DNSSEC

When a name server receives the response to a query for, say, the A record of a web site, for instance, www.example.com, it can only hope that the data is correct. It has no way of *proving* that this is the case, and in fact it could have been duped or spoofed in a variety of ways. For instance, the query response may have been supplied from a poisoned zone file, or the query may have been intercepted and bad data substituted in the response. Another possibility is the query may have been redirected to a bogus server for the domain in question, or the response could be perfectly valid, containing good data from the correct source. In a situation where revenues, reputation, or security (that is, commercial or national) are at stake, such uncertainty may be unacceptable. DNSSEC was designed to eliminate the doubt involved in DNS query operations by providing verifiable certainty to suitably configured name servers and was originally defined in RFC 2535. Significant efforts have been expended over the last several years by many organizations, notably ISC (www.isc.org), Nlnetlabs (www.nlnetlabs.nl), some of the root-server operators (www.root-servers.org), and Regional Internet Registries (www.nro.net), to build and test secure DNS systems such that they can be scaled and deployed in operational environments. A significant number of RFCs have been published on the topic of DNSSEC, many clarifying very specific points of implementation and usage. This Herculean effort has led to what is now colloquially called DNSSEC.bis (defined by RFCs 4033, 4034, and 4035) and constitutes a substantial enhancement to the original specifications. This chapter starts by describing the design of DNSSEC.bis followed by examples that illustrate the various processes involved in securing and maintaining DNSSEC systems.

Note The suffix ".bis" is widely used in the standards world and simply means the second version of a standard. The rest of this chapter will use the term *DNSSEC* not *DNSSEC.bis* to clarify the point that the processes and procedures described here represent the current IETF standard—all prior RFCs relating to DNSSEC were made obsolete by RFCs 4033, 4034, and 4035. Both BIND (release 9.3+—see www.isc.org) and NSD (release 2.3+—see www.nlnetlabs.nl/nsd) support DNSSEC.bis.

The DNSSEC Environment

DNSSEC defines a process whereby a *suitably configured* name server can verify the authenticity and integrity of query results from a *signed* zone. Public key (or asymmetric) cryptography and a special set of Resource Records (RRs), specifically Resource Record Signatures (RRSIGs), DNSKEY, and Next Secure (NSEC) RRs, are used by DNSSEC.bis to enable a *security-aware* receiving name server to do the following:

- Authenticate that the data received could only have originated from the requested zone.
- Verify the integrity of the data. The data that was received at the *querying* name server was the data that was sent from the *queried* named server. The data content is protected, not the communication channel.
- Verify that if a negative response (`NXDOMAIN`) was received to a host query, that the target record does not exist (called *proof of nonexistence* and occasionally *denial of existence*).

The first item to note here is that to support DNSSEC, both the authoritative zone source (master and slave) and the receiving name server must be configured to support DNSSEC. The authoritative name server has to cryptographically *sign* its zone data and becomes in the jargon a *Secure Entry Point (SEP)*, and the receiving name server must be configured to support a security service and is said in the jargon to be *security aware*.

Islands of Security

It is unreasonable to suppose that every name server in the world will overnight be configured to support DNSSEC nor that every zone in the world will be secured. Figure 11-1 shows the possible configurations that could exist and that the DNSSEC standards have to handle.

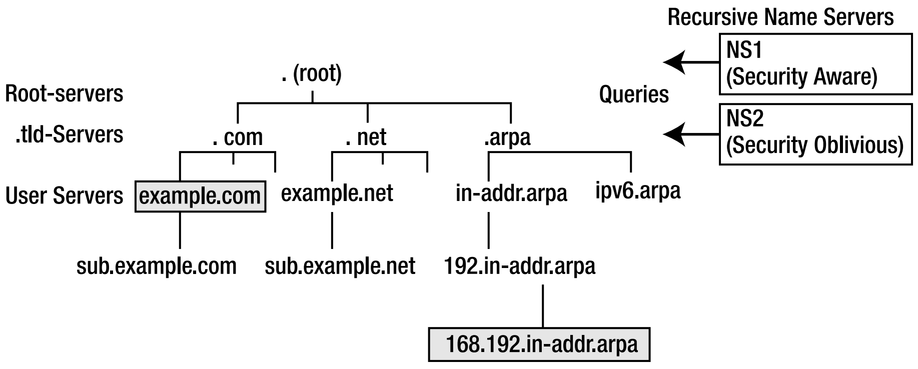


Figure 11-1. *Isolated islands of security*

Figure 11-1 assumes that the colored domains are secure. The security-aware name server (NS1) must continue to provide query results for all domains including the secure domains of `example.com` and `168.192.in-addr.arpa`, and this includes passing through the secure domain of `example.com` to obtain results for the insecure subdomain of `sub.example.com`. Equally the *security-oblivious* name server (NS2 is a name server not configured for DNSSEC security) must continue to obtain transparent results for all the domains, both secure and insecure. NS1 is configured to become security aware by the `dnssec-enable yes;` statement in a global `options` clause, which causes the name server to advertise its security awareness by including an OPT meta (or pseudo) RR in the additional section of any query with the DNSSEC OK (DO) bit set (see Chapter 15 for details). Conversely, any query without both of these characteristics is said to be *security oblivious*. If the authoritative zone source (master or slave) for `example.com` receives a query that indicates the sender is security aware (NS1 in the Figure 11-1 scenario), it responds with additional security information such as RRSIG RRs that enable the requested RRs to be authenticated. If the name server receives a query from a security-oblivious name server (NS2 in the Figure 11-1 scenario), it responds without security information. In the latter case, the query results will be exactly the same as would have been supplied if neither server were security aware (that is, security is invisible).

Public key cryptography relies on a public and private key pair (see Chapter 10 for a description of public key crytography). The zone at `example.com` is cryptographically signed using the zone's private key. The receiving name server must have access to the zone's public key in order to perform the required security verification. This gives rise to the classic asymmetric cryptography problem—how to obtain the public key, in the preceding case for `example.com`, in a manner that ensures it could only have come from `example.com`. There are two possible solutions:

1. Publish the public key using a DNSKEY RR in the zone file. This method is vulnerable to two problems. If we use a secure query to get the key, then the response data requires the public key, which we are requesting but don't yet have, so the security validation will fail—a chicken-and-egg situation. If a nonsecure query is used, then the response could have been spoofed, since it has all the weaknesses of a standard insecure query discussed earlier.

2. Obtain the key using an out-of-band process such as secure e-mail, telephone, or some other acceptable process. This is the method adopted by DNSSEC, and in BIND the public key, called a *trusted anchor* for reasons that will be clear later, is defined using the `trusted-keys` clause of `named.conf`. Figure 11-2 shows this process with NS1 only having the `trusted-keys` clause for `example.com`.

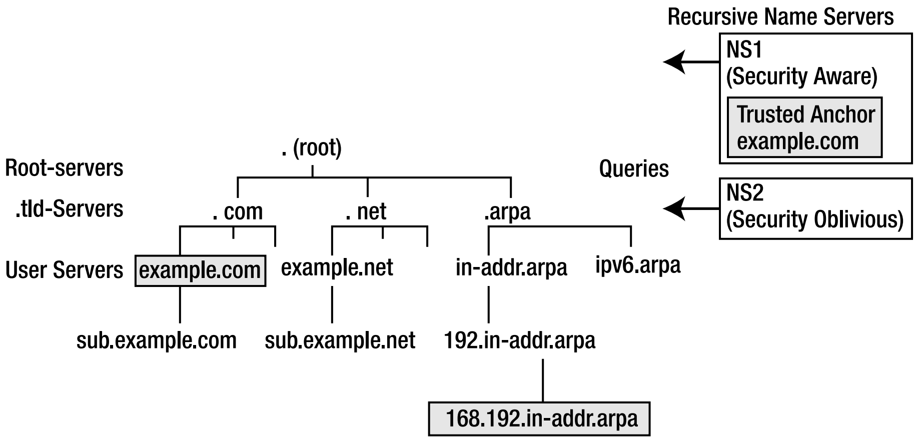


Figure 11-2. *Trusted anchors*

In Figure 11-2, NS2 will continue to operate transparently as before, but NS1 has been configured with a trusted anchor for the domain `example.com` such that all queries for this domain can be securely authenticated—indicated by setting the Authenticated Data (AD) bit in the message header response (see Chapter 15 for details). It does not, however, have a trusted anchor for the domain `168.192.in-addr.arpa`, and in this case responses from this zone will continue to behave as if they were not secure. Theoretically, NS1 is able to determine the following states from the responses from any name server:

- *Secure*: A trusted anchor is present for the zone and has been used to validate the received data successfully. In Figure 11-2, only `example.com` will generate such response states indicated by the Authenticated Data (AD) bit being set.
- *Insecure*: A trusted anchor is present and information allows the name server to prove that at a delegation point there is no secure link to the zone. In Figure 11-2, `sub.example.com` is the only domain that will generate such a response state.
- *Bogus*: A trusted anchor exists, but the data failed to authenticate at the receiving name server using the trusted anchor. An attempt to spoof or corrupt any response from the domain `example.com` will generate this state.
- *Indeterminate*: There is no trusted anchor for the domain. This will be the response state for all domains in Figure 11-2 (including `168.192.in-addr.arpa`) except `example.com` and `sub.example.com`.

Clearly, it is not practical for every name server to have a trusted anchor for every secure domain on the Internet. If this were the only part of DNSSEC, it would simply not scale for Internet-wide deployment. However, before looking at the next set of features, it is worth noting that communities of interest that have finite membership such as extranets, affinity groups, and enterprise networks could implement DNSSEC—even with the relatively limited features described so far—and gain *immediate* access to secured capabilities within the interest groups while continuing to provide transparent service to the wider security-oblivious community. The critical point to make here is that the benefits of DNSSEC can be leveraged immediately given the right circumstances and environment, while users accumulate knowledge and operational experience. As time goes by, the benefits will only increase.

Chains of Trust

Figure 11-3 shows that any single island of security can be joined to another secure (signed) domain through its delegation point—the NS RRs that point from the parent domain or zone to the child domain or zone—and can be authenticated using the final RR in the DNSSEC set called a Delegated Signer (DS) RR.

In Figure 11-3, a *chain of trust* is shown from `example.com` to `sub.example.com`. Three points flow from this process:

1. The child zone, `sub.example.com` in Figure 11-3, must be secure before secure delegation can occur. Securing the zone is an essential prerequisite to creating chains of trust.

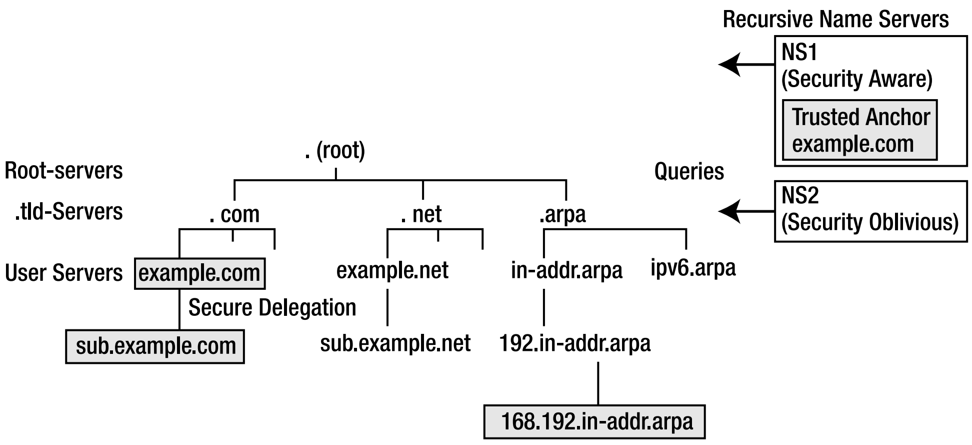


Figure 11-3. *Creating chains of trust*

2. The trusted anchor for `example.com` covers the secure zones that are delegated from it. In the case of Figure 11-3, the trusted anchor for `example.com` covers the child zone `sub.example.com`. The delegation can be securely tracked from `example.com` (the parent that is covered by the trusted anchor) to `sub.example.com` (the child) using a chain of trust. Any number of levels can be covered using this chain of trust concept.

3. Delegation chains can be built both upward as well as downward. Thus if the gTLD domain `.com` were secured, the existing secure domain `example.com` can immediately join the chain, while unsecured domains will continue to operate unchanged (that is, they will not enjoy the benefits of security until action is taken to secure them). The NS1 server would require a new trusted anchor to cover the secured `.com` domain, but this single trusted anchor would cover the *whole* `.com` domain, including `example.com`, as shown in Figure 11-4.

While no gTLDs have yet announced plans to secure their domains, a significant number of tests and trials are underway, as is tool development to mechanize the various processes involved, which are described later in this chapter. Sweden is the first country in the world to announce that in late 2005 it will start signing the `.se` ccTLD domain (`dnssec.nic.se`) and offering a public DNSSEC service that will both validate the various technical and business processes involved and increase the global awareness of the benefits of securing zones. It is only a matter of time before others follow.

■ **Note** In addition to the Swedish announcement mentioned previously, a public DNSSEC trial using what is called a *lookaside* validation process is also underway. This trial, which uses an experimental DLV RR (supported by BIND), is described later in the "DNSSEC Lookaside Validation" section. The trial service theoretically covers all TLD domains and the root domain with a single trusted anchor.

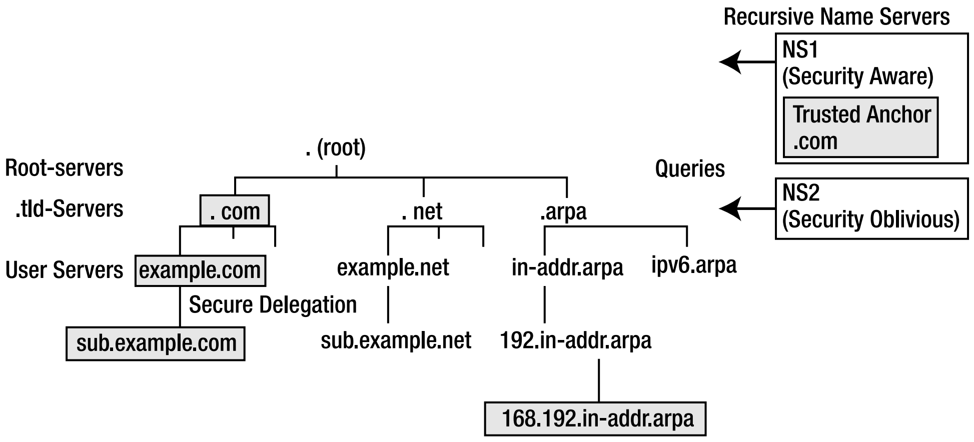


Figure 11-4. *Joining chains of trust*

Having described the DNSSEC process, it is time to start looking at the details of how it all works starting with securing the zone file—the first step in the implementation sequence.

Securing or Signing the Zone

The first step in implementation of DNSSEC is to cryptographically sign the zone files. This is done using the `dnssec-signzone` utility provided with all BIND distributions. However, before we get anywhere near the details of running this utility, it is necessary to step back and understand what is being done.

Zones are digitally signed using the *private* key of a public key (asymmetric) encryption technology. DNSSEC allows for the use of RSA-SHA-1, DSA-SHA-1, and RSA-MD5 digital signatures. The public key corresponding to the private key used to sign the zone is published using a DNSKEY RR and will appear at the apex or root of the zone file; for example, if the zone being signed is `example.com`, then a DNSKEY RR with a name of `example.com` will appear in the zone file.

■Note The private key of the signing algorithm is only required to be available during the signing process—all verification at security-aware name servers is accomplished using the public key only. Thus following signing, it is common practice to take the public key *off-line*, which may involve physically removing the key from the server or moving it to a more secure part of the server. In the case of Dynamic DNS (DDNS), taking the private key off-line may not be possible—see the section "Dynamic DNS and DNSSEC" later in this chapter.

Two *types* of keys are identified for use in zone signing operations. The first type is called a *Zone Signing Key (ZSK)*, and the second type is called a *Key Signing Key (KSK)*. The ZSK is used to sign the RRsets *within* the zone, and this includes signing the ZSK itself, as you shall see later. The public key of this ZSK uses a DNSKEY RR at the apex or root of the zone; that is, if the zone being signed is `example.com`, the ZSK's public key will be defined by a DNSKEY RR, which has a name of `example.com`. The KSK is used to sign the keys at the apex or root of the zone, which includes the ZSK and the KSK and may also be used outside the zone either as the trusted anchor in a security-aware server or as part of the chain of trust by a parent name server. The KSK is also defined in a DNSKEY RR at the root or apex of the zone, that is, if the zone is called `example.com`, then the name of the DNSKEY RR of the KSK will also be `example.com`. The difference between the ZSK and the KSK is therefore one of usage not definition, and it is a matter of *local operational choice* whether a *single* DNSKEY RR is used as both the ZSK and the KSK or whether *separate* DNSKEY RRs are used as the ZSK and KSK. The RFCs allow both methods. This book will use separate keys throughout as the ZSF and the KSK to clearly separate the functionality. The current draft of a best practice RFC on DNSSEC also recommends the use of separate keys (`www.ietf.org/internet-drafts/draft-ietf-dnsop-dnssec-opera➥ tional-practices-05.txt`). The difference in definition is that a ZSK DNSKEY RR has a `flags` field of 256 (see Chapter 13), whereas a KSK is indicated by a `flags` field value of `257`.

■**Note** The `flags` field in a DNSKEY RR is a decimal representation of a bit-significant field; thus the decimal value 256 represents bit 7 of the 16-bit `flags` field (bits numbered from the left starting from 0) and indicates a ZSK. The decimal value `257` represents both bit 7 (ZSK) and bit 15, the Secure Entry Point bit. The SEP bit is used to indicate, *solely for administrative purposes*, that this DNSKEY RR is used as a KSK, and indeed this bit is becoming increasingly known as the KSK bit. This bit is not required and plays no role in the secure validation process or the protocol. While all of the following examples use this feature, specifically trusted anchors and DS RRs point to DNSKEY RRs with the SEP bit set, they could just as easily have pointed to DNSKEY RRs with only the ZSK bit set (`flags` value of 256). The SEP bit is, in modern jargon, pure *sugar*. Its job in life is to make matters more pleasant! All that being said, the current recommended best practice is to use separate keys, which means the SEP bit will be set on the KSK, and indeed the current DLV Pilot trial (see the section "DNSSEC Lookaside Validation") mandates it.

Figure 11-5 shows the usage of the two key types.

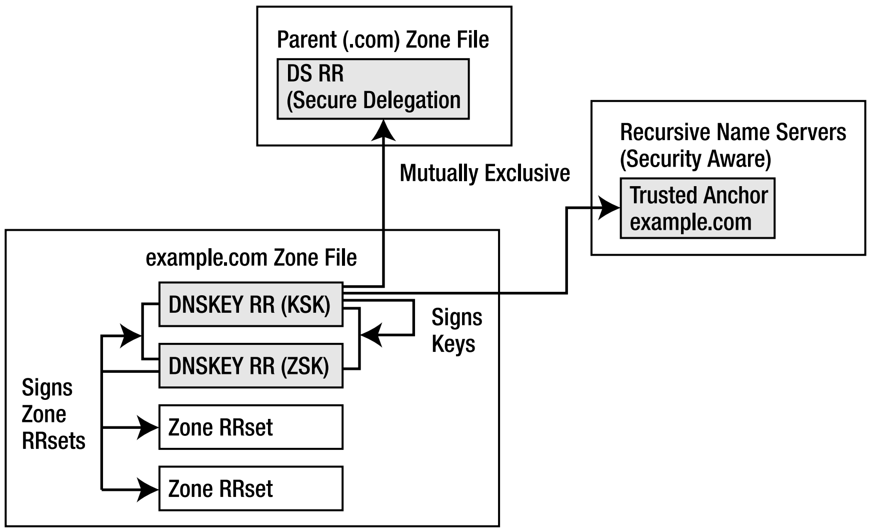


Figure 11-5. *Usage of Zone Signing Key and Key Signing Key*

When a zone is signed, the ZSK and the KSK (remember they can be one and the same key) are generated using the normal `dnssec-keygen` utility (described in Chapter 9) with the name of the zone. The detail process including the parameters used will be illustrated later in the various operational examples. The resulting DNSKEY RRs are either edited into the zone file directly or by using an `$INCLUDE` directive (see Chapter 13) as shown in the following `example.com` zone file:

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@       IN SOA ns1.example.com. hostmaster.example.com. (
                  2005032902 ; serial
                  10800      ; refresh (3 hours)
                  15         ; retry (15 seconds)
                  604800     ; expire (1 week)
                  10800      ; minimum (3 hours)
                  )
            NS ns1.example.com.
            NS ns2.example.com.
            MX 10 mail.example.com.
            MX 10 mail1.example.com.
_ldap._tcp  SRV 5 2 235 www
ns1         A 192.168.2.6
ns2         A 192.168.23.23
www         A 10.1.2.1
            A 172.16.2.1
mail        A 192.168.2.3
mail1       A 192.168.2.4
```

```
example.com. IN DNSKEY 257 3 5 (AQPnvgDqCShrBmFEh5vW7k
                     M4DG/kMwa3EBnPSLAqWRbFOffIWP9ZA2v
                     cZn5ngUjVZ/1IdOViZBOOFCm63bakNgpQ
                     4UNH6e4LH8hnTDMyrlw9smNC xLr4ROqL
                     lcLWDT4ANysDpCZmHUPilvJB1WnVhGKV1
                     I6TO1x+u4uNoe1/ uocNOQ==) ;KSK (SEP)
example.com. IN DNSKEY 256 3 5 (AQPmYqOH3zNwuX4l2+hkh8U
                     G1P14Gv8dfCSi6MbEXON424EX+EIMl4OOO
                     OBkep/ZtIRRJ4rfJONPGs8+HWJDMQOapZn
                     VSYOmSH9V5V32c+j7Gx628y/MyyzwDuT6+
                     zQ3cbobUKrlzL/PLEHegqIDpGkF2VBWXWH
                     LDTCJ5nXB sayYeQ==) ; ZSK
```

The first DNSKEY RR is the KSK indicated by the `flags` value of `257`; the second DNSKEY RR, the ZSK, has a `flags` value of 256 (for details see Chapter 13). The zone is now ready for signing, which is done using the `dnssec-signzone` utility—details of running this utility are fully illustrated later in the example. When a zone is signed, the `dnssec-signzone` utility does a number of automagical things:

1. It sorts the RRs into a canonical order (essentially alphabetic based on host name).
2. It adds an NSEC RR after each RR to chain together the valid host names appearing in the zone file. The last NSEC RR will point back to the zone apex or root.
3. It uses the ZSK to sign each RRset by creating an RRSIG RR. This includes both the DNSKEY RRs and the newly added NSEC RRs from step 2.
4. It uses the KSK to sign (create an RRSIG RR) for the DNSKEY RRset at the zone apex.

The resulting file—which by default has `.signed` appended to the name of the master zone file—after running the `dnssec-signzone` utility will look like that shown here:

```
; File written on Thu Apr 14 12:39:03 2005
; dnssec_signzone version 9.3.0
example.com.  86400 IN SOA ns1.example.com. hostmaster.example.com. (
                      2005032902 ; serial
                      10800      ; refresh (3 hours)
                      15         ; retry (15 seconds)
                      604800     ; expire (1 week)
                      10800      ; minimum (3 hours)
                      )
              86400 RRSIG SOA 5 2 86400 20050514153903 (
                      20050414153903 38420 example.com.
                      P8DKXJwN2dmfl16sqJqk9eVv6HfDs6tgs9B2
                      k/J406v1dyxtl7lUq6oaOVSh9WzqDZTe3dis
                      Ji/DGNVDfXvx3gUnN26sHjkAqZIpTtzYR/ql
                      R+dXKfK14Sqeva0kl5OGqWCmOtuaxlJ9h249
                      w7P3qKtEs4nL1ELrtyEnOLyCX4k= )
              86400 NS ns1.example.com.
              86400 NS ns2.example.com.
```

```
 86400 RRSIG NS 5 2 86400 20050514153903 (
         20050414153903 38420 example.com.
         TK9eFTMHpYqtyLZ+L6qWJmh5PfAsJlUFVI/Y
         Z4P5XBzbEerW85U7SsgrdKCil52qZ8a8OzQI
         5cbsGNrQHfrkvpPdE/D3RiIJzVGrGOmRDvkC
         kvdywljdadVg+8xsCp2XMGfebG2xzKfehO7G
         pFb+TtN2XYfXBVlFa+ZgGbJSkM8= )
 86400 MX 10 mail.example.com.
 86400 MX 10 mail1.example.com.
 86400 RRSIG MX 5 2 86400 20050514153903 (
         20050414153903 38420 example.com.
         MPFBtkjE12FoNbUFO6rgpXA6FCOEnqu6g6jB
         zH3nT4lE9TP89LOErrD13XqKYik27RALoEL2
         y1UFvbk78rZKIeyRPRZh4/6O3qMcMqXq/BCa
         ITsCtlFjcPy4OOFb/76SN9soK8pcC3w3Nkg5
         BSDgbDRImKth+l+PTPiu+iQuUYY= )
 10800 NSEC ldap._tcp.example.com. (NS SOA MX
                             RRSIG NSEC DNSKEY)
 10800 RRSIG NSEC 5 2 10800 20050514153903 (
         20050414153903 38420 example.com.
         NYpw5eq7aW6jO2eybm6Lj/T+4llyvCYuFLQW
         oqtTec38kGHxWtwMdZckSm3V+ColSnjJK8+N
         2YuoCJdooEetrwkUWZv/C/68ES3VVoFHHFqk
         cCMs+7OIG3nMcuGB91yuGcpwBNqkYvm3hW/P
         ZBzj+ikuphPQ7x5O7F2VP9t1rC4= )
 86400 DNSKEY 256 3 5 (
         AQPmYqOH3zNwuX4l2+hkh8UG1P14Gv8dfCSi
         6MbEXON424EX+EIMl40OOOBkep/ZtIRRJ4rf
         JONPGs8+HWJDMQOapZnVSYOmSH9V5V32c+j7
         Gx628y/MyyzwDuT6+zQ3cbobUKrlzL/PLEHe
         gqIDpGkF2VBWXWHLDTCJ5nXBsayYeQ==
         ) ; key id = 38420
86400 DNSKEY 257 3 5 (
         AQPnvgDqCShrBmFEh5vW7kM4DG/kMwa3EBnP
         SLAqWRbFOffIWP9ZA2vcZn5ngUjVZ/1IdOVi
         ZBOOFCm63bakNgpQ4UNH6e4LH8hnTDMyrlw9
         smNCxLr4ROqLlcLWDT4ANysDpCZmHUPilvJB
         1WnVhGKV1I6TO1x+u4uNoe1/uocNOQ==
         ) ; key id = 12513
86400 RRSIG DNSKEY 5 2 86400 20050514153903 (
        20050414153903 12513 example.com.
        lUSl/8AXfEcdocB9syYuONk8AeRXSJy13ixO
        tbAQaH++DjDa+GZOw+eUpSLegMdW7uXdU2Hk
        hWdPoZOTg7+KnjlyJ6uJ+ZozaxYYCpwZrot1
        mP9Jnot6VU58PurwJ8YB2MnQR5rylWYZk84L
        UNoJq8FohGy3/f+Fj1fp4pZ3chM= )
```

```
             86400 RRSIG DNSKEY 5 2 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     awjJL2h6NNhfZ/4HXOiDMJbIYPr+blIaaeK/
                     xEr91vP6myOd2S7dWypZc+qbrm5ew5v6n/OV
                     8UC69u/MZPTBEetRLhi1+D++YIZ7GXmdtUjL
                     A+js30Pgb2cR5cJRDK8yCqi5SlxhNxwO713V
                     kSl/1rlKy+LSl8nQ6XJt8/pkjDM= )
_ldap._tcp.example.com. 86400 IN SRV 5 2 235 www.example.com.
             86400 RRSIG SRV 5 4 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     CnmMTizzSerS4ePFONANviTRFEdJ4OKUwaBu
                     JZiPmX2ZkvQQ2ZWEl6Vvxu7NTyhi6OYRvQWt
                     yMKSOL1LqkT61XfN8v7XWscfZTx8qb6K5qu4
                     n+3xghHRDPBn6yHCOuOvaC4iZeEZyxOWO4jf
                     ce+mtaVkBq5p2dhsH3/t+Msw5qk= )
             10800 NSEC mail.example.com. SRV RRSIG NSEC
             10800 RRSIG NSEC 5 4 10800 20050514153903 (
                     20050414153903 38420 example.com.
                     zC3hXQX82oT9yYEH/OUtPukyTglTat4MxwQ4
                     p4VEWP8qPoEKP3hhtjz4rt1ylTdpLWH6tLNS
                     NFCE/Mdol6kjspfVXRcsL2MLVdRNOKLeqjhl
                     U8Sdut2kjXOnBFp4hAcicALzaN7/PpFGgaOf
                     /KnKcD7aM5a/grOzZBHjswszGt4= )
mail.example.com. 86400 IN A 192.168.2.3
             86400 RRSIG A 5 3 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     W7QBEKLgh6v7AK6T3OKQ4tgSgO4RCrddAN61
                     cD8MYUrq5l1W7I1QgAxdT8zAlADiUjafnnlk
                     QNM4vJnToS3BxprhZ2mFJwiTWV3DIGBqCnPJ
                     c62pueOM+DmsCEBKxbVUZOKf2nW5bim+GIqH
                     CsxfiyOXqDRgzDF6ZZUo/njMqcA= )
             10800 NSEC mail1.example.com. A RRSIG NSEC
             10800 RRSIG NSEC 5 3 10800 20050514153903 (
                     20050414153903 38420 example.com.
                     XiHOXLyR7uC9MvJS3m8AMVtY6QtgSdhy6tId
                     uaylkHjt/EjolKuZdy1F71yP1rICWDdcWgy2
                     eKSKVZy97RfKMlRKBbWruBspmfBfKHSUvl27
                     sOwehJ3n7H4OD5xEO/tzJrHnL1tjHkaqeVpe
                     V16vLTVUUbday9HeRl/388HU1Ok= )
mail1.example.com. 86400 IN A 192.168.2.4
             86400 RRSIG A 5 3 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     lRounAjKet/54jW1oYxoF2LXOxwOyjoVfKX8
                     wNuVKXKv+wW+VoLLjToM8IgIO+AVbIacRbpx
                     PmHgr2CVZo1wRT1guaDCgejk1qI5uYuy9bgD
                     EO2gjaPL2nXYyjTcU3xNOcsWsHLp5PT72Kps
                     bIlGQDAry/xcQSk8mF2scDVj9mc= )
             10800 NSEC ns1.example.com. A RRSIG NSEC
             10800 RRSIG NSEC 5 3 10800 20050514153903 (
                     20050414153903 38420 example.com.
                     VMkPYVtLdiRO6sQFseMss3Xn56WSPRkeYF/q
                     WqRLEMbPv5GrsafzQdExKmj2XFOOJKmbgz/p
                     uhyGKSdzmLcZosjg+hFZnrlMI2kBP5pJ67dN
                     AhZynF+S+A1hymxWQ9lT2+h4zCgW2zEDhy+J
                     PkMi4ra9voDWau3COsRmxcO38Eg= )
ns1.example.com. 86400 IN A 192.168.2.6
             86400 RRSIG A 5 3 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     mQioT8nfRq6dOyFvmR7kO9dU8wohWUOE35ki
                     LTKPrQON1ERi/dhI/YhXtqBP4GDAAbBBOCQU
                     AUJFJP7lnV3oP5FP5YuTvL4eHBoSVcWpdhFG
                     bSV1OejH7CN6e/QACksNmMo7jwQ9woSZ6n5y
                     fpOiPnGUa39awWK+WXegz1UhZfo= )
             10800 NSEC ns2.example.com. A RRSIG NSEC
             10800 RRSIG NSEC 5 3 10800 20050514153903 (
                     20050414153903 38420 example.com.
                     DJmXHGjUZCkbMOUkVSCxFe7eouHr2GHjKGhl
                     7P4etVVkhNMafMBfrsy+J7/Nf4vfbYKCzDEa
                     ARmN1gWBTW/xt8diFk8GKdhsZoiGDkLGOg12
                     rpNhwSOwJK7fdblFSoEZyCrwMQYdEUpdfsGY
                     Xq+7IbdUR9gMFW+ecNcKA9jtpYA= )
ns2.example.com. 86400 IN A 192.168.23.23
             86400 RRSIG A 5 3 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     tvqos7ZVNO4ZWGWDS+uVqj4juNt+N+uNHem3
                     bIOaKAmHKamQzE9ecDfX2HFTO2Pr6OF7v6JQ
                     q9yPoVtGvsYrYrZM7jLTaPdnUhko34KpSThq
                     5SU2OCSUqkIgtYVCMxM18QtnZ4tsy9883OgC
                     9OJTxOkOHdjgYfxLRDuO1AEZfww= )
             10800 NSEC www.example.com. A RRSIG NSEC
             10800 RRSIG NSEC 5 3 10800 20050514153903 (
                     20050414153903 38420 example.com.
                     OxKKdDdRi9ICISwF9Eo4vBH7IkF+Khn8K2yC
                     1TFBpW2CeTAnn67Ngxw3mnNuD8Jh+1k7lFWJ
                     dcvI3+5COycOLnL2+7ncuUg+OMv7kSYOiSaW
                     GlMXKHqzh9rZH8NYraCeqFQu4Zmh99w5w6NH
                     W9JwJOxbQU6hkq8nq8274owj/9M= )
www.example.com. 86400 IN A 10.1.2.1
             86400 IN A 172.16.2.1
             86400 RRSIG A 5 3 86400 20050514153903 (
                     20050414153903 38420 example.com.
                     MQKOnxT6+SO1du5gUcW71CRNDYlHgp4Wddqx
                     py/m92dJwl1XFMOqcNVcuhz9YmCV+zn59vi6
                     Hj5pWpvFRVE5VsrDYtPosKkxyUHSOSeVJkfg
```

```
                7jd33Mz771i/jtQdvkr4Ti3DTcNEBBYZvF59
                sA+ncD56AwG+8NEgxfKJt59d7wE= )
        10800 NSEC example.com. A RRSIG NSEC
        10800 RRSIG NSEC 5 3 10800 20050514153903 (
                20050414153903 38420 example.com.
                JsP+o3gqJgopHosuT1QK+F9z/uXigSv+2Ntg
                Q1GRmBrtUawxdiaX7jCnFOvUKLDJcPDFv2cU
                ceBLVxhpfu9KYQZYghXAR8SvW4XKCOzwMJ1s
                HXtvU8Zx/R+SOj1FnfkndP8VXwPn2Z92ai+Q
                AuOAWELN837tnnFMHIn67sUId7w= )
```

Once you get over the initial feelings of relief that the process is automated, you should note the following points:

- The records have been reordered, specifically the DNSKEY RRs have been moved to the top or apex of the signed file and the A RRs for `ns1`, `ns2`, and `www` have been sorted into their expected (canonical) order.
- NSEC RRs have been added (the first one is after the last MX RR for the zone) such that it is possible to chain using these records through the zone file and thus prove that any particular host name does not exist (remember NSEC RRs are used as proof of nonexistence as described previously). The last NSEC RRs (there are two of them) for the A RRs for `www.example.com` point back to the domain root (`example.com`), indicating there are no additional records in the zone file. The reason there are two NSEC RRs both pointing back to the start of the domain is simply because the `www` A RRs are an RRset (they are equal); had there been only a single A RR, there would have been only one NSEC RR.
- Every RRset has been signed with an RRSIG RR. There are four multiple RRsets (the DNSKEY RRs, the NS RRs, the MX RRs, and the `www.example.com` A RRs); all the other RRsets comprise single RRs—which are still RRsets!
- The DNSKEY RRs have been signed twice (there are two RRSIG RRs). The first signature uses the KSK and is an artifact of the use of a separate ZSK and KSK. The second signature uses the ZSK and is an artifact of the rule that says that all RRsets are signed by the ZSK. If a single DNSKEY RR was used for both the ZSK and KSK functions as allowed by the standards, there would be only one RRSIG RR.

Finally every RRSIG RR has a start time (the time after which it is regarded as being valid) that begins at the Universal Coordinated Time (UTC) minus 1 hour (to allow for clock skew) corresponding to the local run time of the `dnssec-signzone` utility and will expire 30 days after its start time—these are the `dnssec-signzone` utility defaults. The utility run time is always included as a comment on the first line of the file; the expiry and start times are respectively the fourth and fifth parameters after the RRSIG type value (see also Chapter 13). If the zone file is not *re-signed* before the value defined by the expiry time is reached (in this case `20050514153903`, or 14th May 2005 at 3:39 p.m. UTC) a security-aware name server will discard any data from the zone as being bogus (insecure); paradoxically, a security oblivious name server will continue to receive the data successfully. Signing a zone always introduces an element of time that is not present in an unsigned zone file and requires periodic maintenance of the zone file. The next section will look at the implications of re-signing as well as other aspects of secure zone maintenance, including the essential topic of changing keys by what is called in the jargon *key rollover*.

Note It is worth pointing out that NSEC RRs are the subject of some controversy since, as a side effect of their purpose, they have the capability of "walking" or enumerating the zone file. By simply following the NSEC chain for any zone, a user can find all the entries in that zone. Some users find this behavior unacceptable, since it speeds up a process that would otherwise require exhaustive search of the zone. This topic is the subject of ongoing standards work.

Secure Zone Maintenance

Re-signing a zone involves simply rerunning the original `dnssec-signzone` utility using either the original zone file or the currently signed version of it. Secure zone files need to be re-signed for three reasons:

1. *When any change is made to a zone record*: In the insecure world, changes were simply a matter of updating the SOA RR serial number; in the world of DNSSEC, whenever a change is made to the zone file, the SOA serial number needs to be updated and the zone needs to be re-signed. The issue of dynamic update (DDNS) and re-signing is discussed later in the chapter.

2. *When the signatures expire*: As shown in the example signed zone file earlier, each RRSIG RR will expire by default every 30 days. While this can be controlled by parameters to the `dnsssec-signzone` utility, nevertheless periodic zone re-signing will always be required to avoid signature expiry.

3. *When one or more of the ZSK or KSK needs to be changed*: This process, called key rollover, may be required either as part of a regular maintenance process or an emergency—the key is either known to be or suspected of having been compromised.

The first two processes use the existing DNSKEY RRs and have no impact on external name servers. The last process involving key rollover has significant implications for any external name server that has a DS RR (the parent) referencing the KSK, a trusted anchor that references the current KSK (a `trusted-keys` clause in BIND) or cached DNSKEY RRs for either the KSK or ZSK.

Cryptographic keys must be periodically changed, typically every 30 to 90 days, for three reasons:

1. Over a period of time it may be possible for an attacker to accumulate enough plaintext and encrypted material to perform an analysis of the key.

2. A brute-force attack will take some period of time. If the key is changed prior to that interval, the attacker will have to start again.

3. If a key is silently compromised (unknown to the user or operator), it is unlikely the attacker will boast about it but rather elect to continue quietly decrypting the material or subverting the system. Changing the key will at least limit any damage that may result from this and force the attacker to start again.

When a key is changed it may, depending on whether it is a ZSK or a KSK, impact one or more of the following processes, which most likely will be controlled by entities other than the zone administrator who initiates the change:

- *Updating of the DS record at the parent (KSK only)*: If the parent zone at which the DS RR must be changed is not controlled by the same owner as the child zone, then synchronization of the DS RR change with the KSK change is impossible without a level of automation that is not currently available. The time difference may be considerable and involve multiple days.
- *Updating of the trusted anchors at security-aware name servers (KSK only)*: This process will depend on the method being used, but the worst case may involve users manually updating name servers, which could easily take many days. If the update is not performed, a likely event if a manual update is involved, then the zone data will be rejected as *bogus* at any name server that has not updated the trusted anchor, thus rendering the zone unavailable.
- *RRSIG RR and DNSKEY RR caching in name servers (ZSK and KSK)*: Since the RRSIG RR used to sign any RRset and the DNSKEY RRs used to validate them may be acquired at a different time, they will therefore expire from any cache at different times, even if all zone TTLs are the same. It is therefore possible if a zone is re-signed with a new ZSK or KSK for either an old RRSIG RR (an RRSIG RR created with the old ZSK or KSK) and a new DNSKEY RR or the reverse situation to occur. In both cases, queries for the associated RRs data will cause a bogus response.

It may be seen from the preceding that there can be no single point in time at which a zone can change from one key to another key, no matter whether it is a ZSK or a KSK. The standards, however, allow for multiple keys to exist (in multiple DNSKEY RRs) at the zone apex and mandate that all available keys should be tried before the zone data is marked as bogus. This feature allows a signed zone to operate for a period of time with old and new keys until the various processes can be guaranteed to have acquired the new key material, at which point old key(s) can be retired. There are two methods by which a zone may operate with multiple keys—the *prepublish* method and the *double-signing* method.

The Prepublish Method

The *prepublish* method allows one or more new keys to be simply introduced into the zone apex before they are used. Their inclusion in the zone prior to use ensures that the appropriate keys are available in the cache of all security-aware name servers when the key is finally rolled; that is, the zone is re-signed with the new ZSK and/or KSK while leaving the old key(s) at the zone apex, even though they are apparently performing no function. To illustrate this process, it is assumed that all TTLs for a signed zone are for 24 hours (`86400` seconds).

1. At least two days before the zone signatures expires (it could be any time prior to that if required), a new ZSK or KSK would be added to the DNSKEY set at the zone apex. The zone is re-signed using the current, not the new, key(s). The new DNSKEY RR is not used to sign the zone in any way—it is merely present in the DNSKEY RRset.
2. After 24 hours, it can be guaranteed that all caches in security-aware name servers will have the new DNSKEY RRset containing both the current and the new keys. The RRSIG records for any RR types in these caches will have been signed with the current keys.
3. At this point, the zone is re-signed using the new key or keys. The old key is retained in the DNSKEY RRset.

4. From this point and for the next 24 hours, the RRSIG RRs associated with any requested RR, say the A RR for `www.example.com`, may be signed with either the old key, if the RR is already in the cache, or the new key, if it has expired from the cache or was not available in the cache and had to be obtained from the authoritative source. Recall that the standards mandate that all available keys should be tried before rejecting any data as being bogus. In either case, a DNSKEY RR that will successfully authenticate the requested RR data will be present in the cache, or if the DNSKEY RRset has expired or is not present, it can be requested from the authoritative source (master or slave).
5. After 24 hours from re-signing the zone file with the new key(s), the caches can all be guaranteed to contain only RRSIGs signed with the new key. The old key, sometimes called the *stale key*, may be removed at any subsequently convenient time from the DNSKEY RRset at the zone apex and the zone again re-signed with the new key.

The Double-Signing Method

As its name suggests, the *double-signing* method involves the use of more than one key to sign the zone if a ZSK, or the DNSKEY RRset at the zone apex if a KSK. Since RRsets are signed with all keys, double signing ensures that any key contained in the cache at a security-aware name server or referenced in a DS RR or a trusted anchor will authenticate any requested RRset. When all users of the key have migrated to the new key (the DS RRs have been updated at the parent zone and the trusted anchors have been replaced), the old key can be deleted from the zone file and the zone re-signed only with the new key.

Key Rollover Summary

Which method is used is largely a matter of operational decision, but to minimize the volume of records involved, especially in larger zone files, the prepublish method is more suited to changing Zone Signing Keys and the double-signing method to changing Key Signing Keys. The reasoning here is that ZSKs sign each RRset in the zone file, of which there will typically be many. Double signing each RRset will significantly increase the amount of data in the zone file as well as the volume of data sent on each query response. On the other hand, KSKs only sign a single RRset, the DNSKEY RRset at the zone apex, and double signing this RRset will therefore incur a relatively modest overhead.

The process of key rollover involves a number of steps, some of which may involve third parties, and some of which lend themselves to automation. Each step itself is not complex, but the totality of the process, coupled with the fact that a zone could become inaccessible (by being treated as bogus) if any step fails, suggests a number of observations:

- The key-rollover process must be thoroughly planned and subject to continuously evolving refinement. It is clear that Internet Registries and Registry Operators must and are taking the lead in this area since it is fundamentally, but not exclusively, an operational problem.

- The process must be automated wherever practical. While automated tool development is still in its infancy, some high-quality ones have been already been released under Open Source licenses (see `www.netwidget.net/books/apress/dns`).
- The key rollover process must be exercised on a frequent and periodic basis. "Practice makes perfect" to quote the old proverb. While it is possible even now to create keys that could be valid for many years, such attempts to postpone the agony of creating an efficient and streamlined key-rollover process by making it an infrequent event probably exacerbate the problem simply because they ignore the point that, due to a crucial key compromise any key-rollover process (especially a creaky one), may have to be carried out in short order. The prospect of many thousands of administrators giving great imitations of headless chickens while desperately trying to figure out what they did three years ago, coupled with the signature expiry clock inexorably ticking down, does not leave one with a warm and fuzzy feeling.
- KSKs and ZSKs should be separated and rolled at different intervals. The KSK change is clearly the most significant and, by using a larger key size, can be rolled perhaps every 90 days or so versus a ZSK interval of perhaps 30 days. The larger the key size, the more CPU load placed on the server. However, since the KSK is used very infrequently relative to the ZSK, having a larger key size for the KSK should present only a modest additional load on the server.

Secure Delegation

Once a zone is secured, it can then to added to an existing chain of trust or can be used to secure delegation to a subdomain. In both cases, this is accomplished using a Delegated Signer RR. The DS RR is placed in the parent of the zone that will be securely delegated and acts as a pointer to the next key in the chain of trust. The DS RR contains a hash (or digest) of the KSK, defined using a DNSKEY RR, at the apex or the root of the child domain. Thus if the subdomain `sub.example.com` is to be securely delegated (joins the chain of trust), then a DS RR containing a digest of the DNSKEY RR with a name of `sub.example.com` and having a `flags` field value of `257` will be added to the domain `example.com` at the point of delegation (the NS RRs pointing to the subdomain `sub.example.com` in this case). Secure delegation can only occur if the parent and child zones are secure, that is, they both are signed. Figure 11-6 illustrates this process.

The `dnssec-signzone` utility can generate a DS RR during the signing process for the child zone using the `-g` option. Depending on the policies in place, the DS RR and perhaps a copy of the KSK (a DNSKEY RR) for the zone may be sent to the owner of the parent domain for inclusion in the zone file, which must then be re-signed. The child zone is said to join the chain of trust and is authenticated by virtue of the authentication of the parent zone and its secure link (the DS RR) to the child zone. A security-aware name server receiving RRs from a secure domain can track the delegation route for `sub.example.com` back through one or more DS RRs in signed zones to one for which the name server has a trusted anchor.

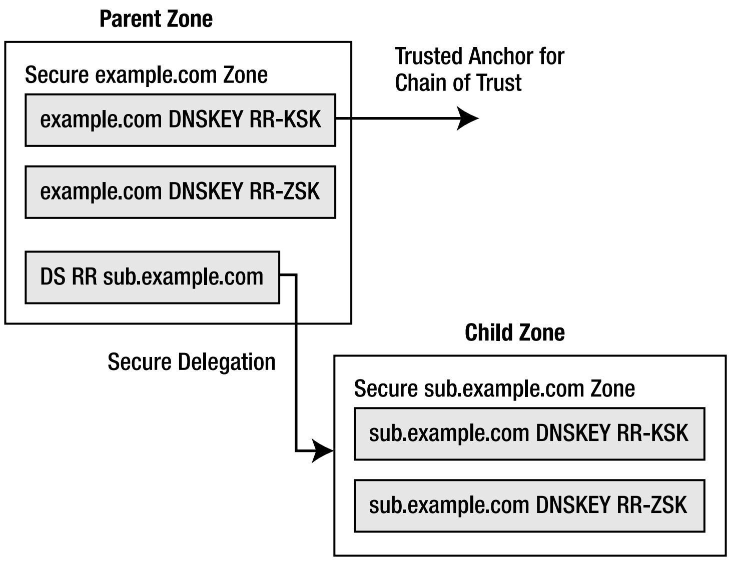

Parent Zone
Secure example.com Zone
example.com DNSKEY RR-KSK
example.com DNSKEY RR-ZSK
DS RR sub.example.com
Trusted Anchor for Chain of Trust
Secure Delegation
Child Zone
Secure sub.example.com Zone
sub.example.com DNSKEY RR-KSK
sub.example.com DNSKEY RR-ZSK

Figure 11-6. *Secure delegation in DNSSEC.bis*

Dynamic DNS and DNSSEC

Dynamic DNS can be used with signed zones. The server will automatically update any required NSEC RRs and will re-sign the RRset. The following points, however, apply when working with dynamic updates and signed zones:

- Either TSIG or SIG(0) security can be used as described in Chapter 10. If SIG(0) (public key) security is used, it requires a KEY RR (not a DNSKEY RR as used in DNSSEC), and in any case a unique key should always be used that must be included or added to the zone file.

- The `.private` file of the ZSK *must* be available (on-line) in either the directory defined by the `directory` statement or uniquely defined using the `key-directory` statement (in a `global`, `view`, or `zone` clause) of `named.conf` during any update. If this file is not available, then any update attempt will fail with the following log message:

```
'example.com/IN': adding an RR at 'www.example.com' A
'example.com/IN': could not get zone keys for secure dynamic update
'example.com/IN': RRSIG/NSEC update failed: permission denied
```

- If separate KSK and ZSKs are defined and if both `.private` files are on-line when the update is executed, the modified RRsets will be signed with *both* keys. Always make sure that the `.private` file of the KSK is taken off-line as soon as the zone is signed.

- When dynamically updated zones are signed, then modified procedures for *manual* edit of the zone file must be followed:

1. Either stop BIND or use `rndc freeze zone`.
2. Bring the KSK `.private` key on-line (the ZSK `.private` file is assumed to be on-line as noted previously)
3. Re-sign the signed zone since it contains the current updates (requires the `-f` option)
4. Delete the `.jnl` file
5. Take the KSK `.private` file off-line, and then either start BIND or `rndc thaw zone`.

In particular, it is vital to delete the `.jnl` file for the zone before restarting or reloading the zone file after the re-signing is complete to ensure that there is no playback of stale values from this file.

If you are currently using secured dynamic updates, adding DNSSEC to the zones is a transparent process. Care must be taken when signing zones that are dynamically updated to observe the additional steps required.

■ **Note** Some of the processes described previously can be automated and indeed a number of such tools already exist, some of which are identified at `www.netwidget.net/books/apress/dns`.

DNSSEC Implementation

In order to illustrate the DNSSEC implementation process, the following procedures will be described with examples:

- Securing the zone `example.com` using a separate ZSK and KSK
- Establishing a trusted anchor for `example.com` in a name server at `ns1.example.net`
- Securing the zone `sub.example.com`
- Adding the DS RR for `sub.example.com` to the zone `example.com` to create secure delegation within the chain of trust
- Rolling the ZSK and KSK for `example.com`

The examples are based on Figures 11-1 through 11-3, presented earlier in the chapter.

Securing the `example.com` Zone

The zone `example.com`, which will be signed during this process, is an *island of security* and has a zone file as shown here:

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@             IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
            IN  NS ns1.example.com.
            IN  NS ns2.example.com.
            IN  MX 10 mail.example.com.
            IN  MX 10 mail1.example.com.
_ldap._tcp  IN  SRV 5 2 235 www
ns1         IN  A  192.168.2.6
ns2         IN  A  192.168.23.23
www         IN  A  10.1.2.1
            IN  A  172.16.2.1
mail        IN  A  192.168.2.3
mail1       IN  A  192.168.2.4
$ORIGIN sub.example.com.
@           IN  NS ns3.sub.example.com.
            IN  NS ns4.sub.example.com.
ns3         IN  A  10.2.3.4 ; glue RR
ns4         IN  A  10.2.3.5 ; glue RR
```

The zone file contains delegation to a subdomain called `sub.example.com`, which is assumed at this point to be insecure (as shown in earlier in this chapter in Figure 11-1).

To secure the `example.com` zone will require a ZSK and a KSK, which again may be a single key or two separate keys. For the purposes of clarity and because the processes are significantly different, the examples use the recommended method of using separate keys for each of the ZSK and KSK functions. Both keys are generated by the `dnssec-keygen` utility. The examples assume that these operations will be carried out in the directory `/var/named/keys`. To generate the ZSK, the `dnssec-keygen` utility (see Chapter 9) is run with the following command:

```
# dnssec-keygen -a rsasha1 -b 1024 -n zone example.com
Kexample.com.+005+03977
```

This generates a key pair for the RSA-SHA-1 digital signature algorithm, with a key length of 1024 bits and a zone record with the name of the zone apex, which in this case is `example.com`. The command response of `Kexample.com.+005+03977` indicates that a file `Kexample.com.+005+03977.key` contains the DNSKEY RR with the public key of the key pair, which will be added to the zone file, and the file `Kexample.com.+005+03977.private` contains the private key used in subsequent signing operations. The value `03977` is the `key-tag` that uniquely identifies this key. Next, the KSK is generated using the following command:

```
 # dnssec-keygen -a rsasha1 -b 1400 -f KSK -n zone example.com
Kexample.com.+005+12513
```

This generates a key pair for the RSA-SHA-1 digital signature algorithm, with a key length of 1400 bits and a zone DNSKEY RR with the name of the zone apex, which in this case is `example.com`. The `-f KSK` argument signifies that this will generate a KSK DNSKEY RR, indicated by the `flags` field having a value of `257` (the SEP bit is set). Full details of the DNSKEY RR are defined in Chapter 13.

Note The key size of the KSK is set to the value 1400, whereas the key size of the ZSK is set to 1024. The KSK is significantly stronger and therefore needs to be changed (rolled over) with a lower frequency than the ZSK. However, it should be noted that the current RSA recommendation for key size is 1024, and RSA considers that keys greater than 768 are still immune from brute-force attacks.

The command response of `Kexample.com.+005+12513` indicates that a file with the name `Kexample.com.+005+12513.key` contains the DNSKEY RR with the public key of the key pair, which will be added to the zone file, and the file `Kexample.com.+005+12513.private` contains the private key used in subsequent signing operations. The value `12513` is the `key-tag` that uniquely identifies this key. The DNSKEY RRs generated by the previous operations are contained in the files `Kexample.com.+005+12513.key` (KSK) and `Kexample.com.+005+03977.key`. These may be edited into the zone file (as shown in "Securing or Signing the Zone" earlier in the chapter) or included as shown here:

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@             IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
            IN  NS ns1.example.com.
            IN  NS ns2.example.com.
            IN  MX 10 mail.example.com.
            IN  MX 10 mail1.example.com.
_ldap._tcp  IN  SRV 5 2 235 www
ns1         IN  A  192.168.2.6
ns2         IN  A  192.168.23.23
www         IN  A  10.1.2.1
            IN  A  172.16.2.1
mail        IN  A  192.168.2.3
mail1       IN  A  192.168.2.4
$ORIGIN sub.example.com.
```

```
@           IN  NS ns3.sub.example.com.
            IN  NS ns4.sub.example.com.
ns3         IN  A  10.2.3.4 ; glue RR
ns4         IN  A  10.2.3.5 ; glue RR
$INCLUDE keys/Kexample.com.+005+12513.key ; KSK
$INCLUDE keys/Kexample.com.+005+03977.key ; ZSK
```

An alternative way to add the DNSKEY RR directly to the preceding file would be to use the following commands:

```
# cat keys/Kexample.com.+005+12513.key >> master.example.com
# cat keys/Kexample.com.+005+03977.key >> master.example.com
```

Since the DNSKEY RR is a public key, there are no security requirements—either method is perfectly acceptable.

The zone is now ready for signing using the `dnssec-signzone` command (see Chapter 9) as shown here:

```
# dnssec-signzone -o example.com -t -k Kexample.com.+005+12513 master.example.com \
Kexample.com.+005+03977
master.example.com.signed
Signatures generated:                        19
Signatures retained:                          0
Signatures dropped:                           0
Signatures successfully verified:             0
Signatures unsuccessfully verified:           0
Runtime in seconds:                       0.357
Signatures per second:                   53.079
```

■ **Tip** When signing a zone with a single ZSK, rather than separate the KSK and the ZSK as shown in the example, just omit the `-k` option.

The \ in the preceding example indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. The `-o example.com` arguments indicate the name of the domain being signed. The `-t` argument displays some statistics, which are shown on the following lines. `-k Kexample.com.+005+12513` indicates that `Kexample.com.+005+12513.private` contains the private key that should be used as the KSK. `master.example.com` is the name of the zone file to be signed, and `Kexample.com.+005+03977` indicates that `Kexample.com.+005+03977.private` contains the private key that will be used as the ZSK. The first line of the resulting output is `master.example.com.signed`, which is the default file name (input zone file name with `.signed` appended) allocated if the `-f` option is not used.

The resulting file looks as shown here:

```
; File written on Mon Apr 18 10:48:29 2005
; dnssec_signzone version 9.3.0
example.com.  86400 IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
              86400 RRSIG SOA 5 2 86400 20050518134829 (
                          20050418134829 3977 example.com.
                          Pcj36/iCWbY+9/sq9Dw7+QaeRbs= )
              86400 NS ns1.example.com.
              86400 NS ns2.example.com.
              86400 RRSIG NS 5 2 86400 20050518134829 (
                          20050418134829 3977 example.com.
                          6sfpgAuKarGSbhN3elYozOaBU6c= )
              86400 MX 10 mail.example.com.
              86400 MX 10 mail1.example.com.
              86400 RRSIG MX 5 2 86400 20050518134829 (
                          20050418134829 3977 example.com.
                          2y4QQlM7+RsO39wLaxA/I+69d38= )
              10800 NSEC ldap._tcp.example.com. (NS SOA MX
                                        RRSIG NSEC DNSKEY)
              10800 RSIG NSEC 5 2 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        k4T48nVQVZuPBW3aQOBhlQmYP6c= )
              86400 DNSKEY 256 3 5 (
                        t/4w8JgeybiVZeHbYXHIljSOkHt8vw==
                        ) ; key id = 3977
              86400 DNSKEY 257 3 5 (
                        1WnVhGKV1I6TO1x+u4uNoe1/uocNOQ==
                        ) ; key id = 12513
              86400 RRSIG DNSKEY 5 2 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        ihcz6BqjNRBFk4vCSGjS2UWdx7M= )
              86400 RRSIG DNSKEY 5 2 86400 20050518134829 (
                        20050418134829 12513 example.com.
                        vv2TqynHfZI8I9GA9zpyd+y/54M= )
_ldap._tcp.example.com. 86400 IN SRV 5 2 235 www.example.com.
              86400 RRSIG SRV 5 4 86400 20050518134829 (
                        4hzYqMuD+YfCe6CYijkvxaK2AI8= )
              10800 NSEC mail.example.com. SRV RRSIG NSEC
              10800 RRSIG NSEC 5 4 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        8qOgADAR86IvfVUT7eXtRbXhyQg= )
```

```
mail.example.com. 86400 IN A 192.168.2.3
              86400 RRSIG A 5 3 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        ntx8VinqRDuVGdLv6j1aTZPk26c= )
              10800 NSEC mail1.example.com. A RRSIG NSEC
              10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        bsjUM4szz6k1kJj1eASDVh+PPdc= )
mail1.example.com. 86400 IN A 192.168.2.4
              86400 RRSIG A 5 3 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        s5jnGdHVOzLEN9OooydL5QOq6Bg= )
              10800 NSEC ns1.example.com. A RRSIG NSEC
              10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        /CaOz+gPDCxpgXp9vVBwoCDZyNs= )
ns1.example.com. 86400 IN A 192.168.2.6
              86400 RRSIG A 5 3 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        WLwYOeMj29hoehng6Q8MOqP/Fps= )
              10800 NSEC ns2.example.com. A RRSIG NSEC
              10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        iUmOZtFd2tlB1kCGdO3TWHA6XLE= )
ns2.example.com. 86400 IN A 192.168.23.23
              86400 RRSIG A 5 3 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        D5g1Bc235ra+kcgdLyOi5oOxyKs= )
              10800 NSEC sub.example.com. A RRSIG NSEC
              10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        KrYgcGOtK2EZkbMBpedYBjVLVwE= )
sub.example.com. 86400 IN NS ns3.sub.example.com.
                 86400 IN NS ns4.sub.example.com.
               10800 NSEC www.example.com. NS RRSIG NSEC
               10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        lwTngtzMsECH+ZsOqzaOd8mxORE= )
ns3.sub.example.com. 86400 IN A 10.2.3.4
ns4.sub.example.com. 86400 IN A 10.2.3.5
www.example.com. 86400 IN A 10.1.2.1
              86400 IN A 172.16.2.1
              86400 RRSIG A 5 3 86400 20050518134829 (
                        20050418134829 3977 example.com.
                        5djR2cKlFB5XUU4uT92hFWGfsKE= )
              10800 NSEC example.com. A RRSIG NSEC
              10800 RRSIG NSEC 5 3 10800 20050518134829 (
                        20050418134829 3977 example.com.
                        8OcJsjO6zzkINiR2nqLUh2GEbvI= )
```

In the interest of brevity and because it adds no value to the human reader, most of the base64 material in the DNSKEY and RRSIG RRs has been removed. The following points should be noted:

- The included .key files have been expanded and moved into the signed zone file.
- The host names have been sorted in host name (canonical) order; specifically, the DNSKEY RRs from the included files, the ns1, ns2, and www RRs have all been reordered.
- NSEC RRs have been added to all RRs to chain through the zone file with the exception of the two glue A RRs for ns3.sub.example.com. and ns4.sub.example.com. at the point of delegation.
- All the RRsets have been signed with RRSIG RRs with the exception of the glue A RRs for ns3.sub.example.com. and ns4.sub.example.com as well as the two NS RRs for sub.example.com. It is worth explaining the reason for this omission. The zone example.com is not authoritative for sub.example.com. The NS RRs (and their corresponding A RRs) for the delegation are deemed to be owned by sub.example.com (the child) but placed at example.com (the parent). The lack of signature is a consequence of the fact that this zone is not authoritative for these, or any other delegation, RRs and therefore cannot sign them.
- The DNKEY RRset at the zone apex (example.com) has been signed twice: once with the ZSK and once with the KSK.
- The RRSIG RRs expire 30 days from the date of the start time (in the preceding case, 20050518134829, which is 1:48:29 p.m. on the 18 May 2005 UTC). The zone will have to be re-signed prior to this date. Re-signing simply involves running the same dnssec-signzone command used to originally sign the zone.

The signed zone file is ready to become operational in ns1.example.com, the primary master for the zone, using a named.conf fragment such as defined here:

```
// named.cong fragment for ns1.example.com
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   allow-transfer {"none"};
   ....
};
....
zone "example.com" in{
   type master;
   file "master.example.com.signed";
   allow-transfer {192.168.23.23;}; // ns2.example.com
   allow-update {"none";};
};
....
```

The log has been streamed for `dnssec` events to assist in any test debugging. A sample log output is shown later in the section "DNSSEC Logging." The `severity debug 3;` statement should not be used for production, since it will generate huge amounts of log data; instead `severity info;` or higher should be used. DNSSEC is not enabled by default, and the `dnssec-enable yes;` statement is required in the global `options` clause. The zone file in the `example.com` zone clause references the signed file created by the zone signing process earlier. No special treatment is required on the slave server (`ns2.example.com`) whose `named.conf` would look as defined here:

```
// named.conf fragment for ns2.example.com
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   allow-transfer {"none"};
   ....
};
....
zone "example.com" in{
   type slave;
   file "slave.example.com.signed";
   masters {192.168.2.6;}; // ns1.example.com
   allow-update {"none";};
};
....
```

Verifying the Signed Zone

To confirm the zone is working successfully, use a `dig` command to verify the results. If a normal `dig` command is issued, it will emulate the behavior of a security-oblivious name server, and therefore no security information will be displayed. If the `+dnssec` option is added, it will respond with the security information as shown in the following example, which has been issued to `ns1.example.com`, an authoritative name server for the `example.com` zone:

```
dig @192.168.2.6 www.example.com +dnssec +multiline
; <<>> DiG 9.3.0 <<>> @192.168.2.6 www.example.com +dnssec
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1307
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 3, ADDITIONAL: 9

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.example.com.  IN  A

;; ANSWER SECTION:
www.example.com.  86400  IN  A  10.1.2.1
www.example.com.  86400  IN  A  172.16.2.1
www.example.com.  86400  IN  RRSIG  A 5 3 86400 (
   20050628180003 20050529180003 46979 example.com.
   jitcoTkXbNIO8kbME/EyTlKyn6QwBQ==)
```

```
;; AUTHORITY SECTION:
example.com.  86400  IN  NS  ns2.example.com.
example.com.  86400  IN  NS  ns1.example.com.
example.com.  86400  IN  RRSIG  NS 5 2 86400 (
   20050628180003 20050529180003 46979 example.com.
   R8Vsb5sjXJwbJOD5rcPZocaf2Rz==)

;; ADDITIONAL SECTION:
ns1.example.com.  86400  IN  A  192.168.2.6
ns2.example.com.  86400  IN  A  192.168.23.23
ns1.example.com.  86400  IN  RRSIG  A 5 3 86400 (
   20050628180003 20050529180003 46979 example.com.
   jHwcZ18dDvGqmoszU5MUOBbJA==)
ns2.example.com.  86400  IN  RRSIG  A 5 3 86400 (
   20050628180003 20050529180003 46979 example.com.
   jzfYhRBXEC5svDCUwJk7U2EPB8==)
example.com.  86400  IN  DNSKEY  256 3 5 (
   AQPYSk9lcDWan3QTOrI2kTjHz)
example.com.  86400  IN  DNSKEY  257 3 5 (
   AQO9gvDKN7WDVeluu3ec)
example.com.  86400  IN  RRSIG  DNSKEY 5 2 86400(
   20050628180003 20050529180003 38070 example.com.
   R73FYKx4sjR88smPpEm==)
example.com.  86400  IN  RRSIG  DNSKEY 5 2 86400 (
   20050628180003 20050529180003 46979 example.com.
   AoGwqxZxQyvViBmMvyf1k8f==)

;; Query time: 15 msec
;; SERVER: 192.168.2.6#53(192.168.2.6)
;; WHEN: Sun May 29 15:26:24 2005
;; MSG SIZE  rcvd: 1838
```

Again, in the interest of brevity, most of the base64 material has been eliminated, since it is of no interest to the human reader. The following points should be noted:

- The `+multiline` option simply adds parentheses to each long RR to create a slightly more readable output format.
- The `OPT PSEUDOSECTION` shows that EDNS0 features are in use and that a UDP block size of 4096 bytes has been negotiated for use in the much bigger responses from DNSSEC transactions. The OPT meta (or pseudo) RR is actually placed in the `ADDITIONAL SECTION`, but `dig` chooses to display and format it separately (see Chapter 15).
- The `ANSWER SECTION` includes the RRSIG to cover the A RRs returned with the query and can thus be used to authenticate the RRset.
- The `AUTHORITY SECTION` also includes the RRSIG RR to cover the NS RRs returned and allow verification of this section as well.

- The `ADDITIONAL SECTION` contains as expected the A RRs for the authoritative name servers and its covering RRSIG RR. In additional, it also contains the DNSKEY RRs for the KSK and ZSK for the zone and the two RRSIGs (one signed with the ZSK and the other using the KSK).
- The `HEADER` flags do not include the `ad` (Authenticated Data) flag (see Chapter 15) because this `dig` was issued to one of the authoritative name servers for the signed zone. The authoritative name server's job is to supply the information, the various RRSIGs and DNSKEY RRs, by which a receiving name server can perform the authentication. If, however, the `dig` had been issued to a security-aware name server that was not authoritative for the zone `example.com`, then that name server *would have performed the authentication* and, assuming it was successful, the `ad` flag would have been set. This process is illustrated later in the chapter.

Observant readers will note that the preceding zone was re-signed at a much later date than that shown in the original zone signing example earlier, but this has no impact on any results.

Establishing a Trusted Anchor

The example assumes that a security-aware name server at `ns1.example.net` wishes to authenticate the data from `example.com`. This name server needs to establish a trusted anchor for the domain `example.com`. The administrator of `ns1.example.net` obtains by some secure process the DNSKEY RR for the KSK of the domain `example.com`. While the DNSKEY RR itself is not sensitive information (it contains a public key), the administrator must be able to authenticate the source of the key, and therefore a secure distribution process such as secure e-mail or secure FTP must be used to obtain the trusted anchor. This DNSKEY RR is available from the signed `example.com` zone file shown earlier and is identified as having a `flags` field value of `257` (which includes the SEP or KSK bit):

```
86400 DNSKEY 257 3 5 (
          1WnVhGKV1I6TO1x+u4uNoe1/uocNOQ==
          ) ; key id = 12513
```

The reader should note that much of the base64 material has been eliminated in the interest of brevity and that a real DNSKEY RR would be considerably larger. The trusted anchor is created by editing this DNSKEY RR into a `trusted-keys` clause for the `named.conf` file at the server `ns1.example.net`, as shown in the following fragment:

```
// named.conf fragment for ns1.example.net
logging{
    channel normal_log {
    file "/var/log/named/normal.log" versions 3 size 2m;
    severity error;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel dnssec_log { // streamed dnssec log
    file "/var/log/named/dnssec.log" versions 3 size 2m;
    severity debug 3;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    normal_log;
    };
    category dnssec{
    dnssec_log;
    };
};
options {
   ....
   directory "/var/named";
   dnssec-enable yes;
   allow-transfer {"none"};
   ....
};
trusted-keys{
   "example.com" 257 3 5 "1WnVhGKV1I6TO1x+u4uNoe1/uocNOQ==";
};
....
```

The `trusted-keys` clause contains the trusted anchor for `example.com` and is an edited version of the DNSKEY RR created by removing the TTL and DNSKEY, and adding the domain name in quotes (a quoted string that can be an FQDN, but will work quite happily without the trailing dot); the `flags`, `proto`, and `algorithm` fields are left intact, and the base64 public key material (`key-data`) is enclosed in quotes and terminated with a semicolon. For the full format and layout of the trusted anchor layout within the `trusted-keys` clause, see Chapter 12. Since this a security-aware server, the `dnssec-enable yes;` statement must be included. The log is again streamed and the `severity debug 3;` used to generate information that may be useful during debugging, but should not be used in production unless the reader likes managing large logs! Instead, `severity info;` or a higher value should be used based on comfort and experience.

Using a Trusted Anchor

The following shows a `dig` command issued to the recursive server `ns1.example.net` that is neither the zone master nor the zone slave for `example.com`, but that has been configured to be security aware (using a `dnssec-enable yes;` statement) and has a trusted anchor for the zone `example.com` (in a `trusted-keys` clause):

```
dig @ns1.example.net www.example.com
; <<>> DiG 9.3.0 <<>> @ns1.example.net www.example.com +dnssec +multiline
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60711
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.example.com.  IN  A

;; ANSWER SECTION:
www.example.com.  86061  IN  A  172.16.2.1
www.example.com.  86061  IN  A  10.1.2.1
www.example.com.  86061  IN  RRSIG  A  5 3 86400 20050628191945 (
   20050529191945 3977 example.com.
   tbdelN28tzTudlYyjDhy4OlUIjXyqQUayzyzAzY= )

;; Query time: 1 msec
;; SERVER: 192.168.254.23#53(ns1.example.net)
;; WHEN: Sun May 29 23:35:15 2005
;; MSG SIZE  rcvd: 247
```

Again, in the interest of brevity, most of the base64 material has been eliminated, since it is of no interest to the human reader. In this case, the response is significantly shorter than that shown when the authoritative server was queried directly (shown in the preceding section, "Verifying Signed Zones"). The reason is simply that the name server 192.168.254.23, because it is security aware, has verified the various signatures on our behalf, and confirmed this action by setting the `ad` (Authenticated Data) flag in the `HEADER`, and therefore only the query results are supplied to the `dig` command. The next section shows the security log at the name server to confirm that it has indeed performed this validation.

DNSSEC Logging

The following shows typical log output using the streamed security logging configured as shown in the `named.conf` fragment example; this is the resulting output from the preceding `dig` command:

```
dnssec: validating www.example.com A: starting
dnssec: validating www.example.com A: attempting positive response validation
dnssec: validating example.com DNSKEY: starting
dnssec: validating example.com DNSKEY: attempting positive response validation
dnssec: validating example.com DNSKEY: verify rdataset: success
dnssec: validating example.com DNSKEY: signed by trusted key; marking as secure
dnssec: validator @0x8257800: dns_validator_destroy
```

```
dnssec: validating www.example.com A: in fetch_callback_validator
dnssec: validating www.example.com A: keyset with trust 7
dnssec: validating www.example.com A: resuming validate
validating www.example.com A: verify rdataset: success
dnssec: validating www.example.com A: marking as secure
dnssec: validator @0x81ab000: dns_validator_destroy
```

For the sake of brevity, the date and time have been removed from this log output, which shows both A RRs being validated and being marked as secure.

Signing the `sub.example.com` Zone

The process for signing a subdomain is essentially similar to that defined for signing a zone with one single difference. The zone `sub.example.com` is the child of the secure zone `example.com`, or, if you prefer, `example.com` is the secure parent of `sub.example.com`. A Delegated Signer RR can be added to the `example.com` zone file to create secure delegation. The zone `sub.example.com` will join the chain of trust whose current secure entry point is `example.com`. For clarity and ease of key rollover, separate KSK and ZSK RR will be used.

Generate the ZSK for `sub.example.com`:

```
# dnssec-keygen -a rsasha1 -b 1024 -n zone sub.example.com
Ksub.example.com.+005+48560
```

Generate the KSK for `sub.example.com`:

```
# dnssec-keygen -a rsasha1 -b 1400 -f KSK -n zone example.com
Ksub.example.com.+005+64536
```

Include the keys in the `sub.example.com` zone file:

```
$TTL 86400 ; 1 day
$ORIGIN sub.example.com.
@            IN SOA ns1.sub.example.com. hostmaster.example.com. (
                    2005032902 ; serial
                    10800      ; refresh (3 hours)
                    15         ; retry (15 seconds)
                    604800     ; expire (1 week)
                    10800      ; minimum (3 hours)
                    )
              IN NS ns3.example.com.
              IN NS ns4.example.com.
              IN MX 10 mail.example.com.
ns3           IN A 10.2.3.4
ns4           IN A 10.2.3.5
fred          IN A 10.1.2.1
$INCLUDE Ksub.example.com.+005+48560.key ; ZSK
$INCLUDE Ksub.example.com.+005+64536.key ; KSK
```

Sign the zone sub.example.com:

```
# dnssec-signzone -o sub.example.com -t -g -k Kexample.com.+005+64536 \
master.sub.example.com  Kexample.com.+005+48560
master.sub.example.com.signed
Signatures generated:                    19
Signatures retained:                      0
Signatures dropped:                       0
Signatures successfully verified:         0
Signatures unsuccessfully verified:       0
Runtime in seconds:                   0.357
Signatures per second:               53.079
```

The \ indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. This command line is the same as that for the zone example.com using the revised keys and zone file names, with the exception that the -g argument is used to generate two special files called dsset-sub.example.com. (containing the DS RR for the parent) and keyset-sub.example.com. (containing a copy of the public key DNSKEY RR of the KSK). One or both of these files may, depending on policy, be sent to the parent DNS administrator by any suitable, but secure, process to enable secure delegation, the creation of a chain of trust, which is described in the next section. While neither file contains secure information (they contain normal RR data), it is vital that the recipient be able to authenticate the sender and hence create the appropriate level of trust; thus a secure process such as secure e-mail or secure FTP should always be used when making these files available. The named.conf file for the master and slave servers for this subdomain are the same as those used for example.com and require no special treatment. Because sub.example.com is authenticated via the zone example.com, no action is required at the name server ns1.example.net—its trusted-keys clause with a trusted anchor for example.com will cover sub.example.com as well through the chain of trust.

Creating the Chain of Trust

When the parent administrator receives the dsset-sub.example.com. and, optionally, the keyset-sub.example.com. files, they are placed in the same directory where the example.com zone is signed. The dsset-sub.example.com. file is included in the original example.com zone as shown here (the location in the zone file is not important, but note that the file name always ends with a dot):

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@            IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
```

```
            IN  NS ns1.example.com.
            IN  NS ns2.example.com.
            IN  MX 10 mail.example.com.
            IN  MX 10 mail1.example.com.
_ldap._tcp  IN  SRV 5 2 235 www
ns1         IN  A  192.168.2.6
ns2         IN  A  192.168.23.23
www         IN  A  10.1.2.1
            IN  A  172.16.2.1
mail        IN  A  192.168.2.3
mail1       IN  A  192.168.2.4
$ORIGIN sub.example.com.
@           IN  NS ns3.sub.example.com.
            IN  NS ns4.sub.example.com.
ns3         IN  A  10.2.3.4 ; glue RR
ns4         IN  A  10.2.3.5 ; glue RR
$INCLUDE keys/Kexample.com.+005+12513.key ; KSK
$INCLUDE keys/Kexample.com.+005+03977.key ; ZSK
$INCLUDE dsset-sub.example.com. ; DS RR
```

Re-sign the zone by executing the `dnssec-signzone` command exactly as before:

```
# dnssec-signzone -o example.com -t -k Kexample.com.+005+12513 \
master.example.com Kexample.com.+005+03977
master.example.com.signed
Signatures generated:                      20
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Runtime in seconds:                     0.357
Signatures per second:                 53.079
```

The \ in the preceding example indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. The only thing that has changed is that the `Signatures generated` line has gone from 19 in the first version to 20 in this version because of the additional DS RR, which has now been signed. The resulting zone file is exactly the same as the first signed zone but with an updated signature expiry, and the additional DS RR has been added and signed as shown in the following fragment:

```
sub.example.com. 86400 IN NS ns3.sub.example.com.
                 86400 IN NS ns4.sub.example.com.
                 86400 DS 64536 5 1 (
                       CE0711D34D21C069A4C91215C50B4F38E3D5
                       65D1 )
```

```
                  86400 RRSIG DS 5 3 86400 20050518171727 (
                        20050418171727 3977 example.com.
                        RRApmGQ3fKmzbAF7ev4G6eRpWOI= )
                  10800 NSEC www.example.com. (NS DS RRSIG
                                              NSEC)
                  10800 RRSIG NSEC 5 3 10800 20050518171727 (
                        20050418171727 3977 example.com.
                        gNp5LyMVZ8wcH5lNgGpKNJSsfcs= )
ns3.sub.example.com. 86400 IN A 10.2.3.4
ns4.sub.example.com. 86400 IN A 10.2.3.5
www.example.com. 86400 IN A 10.1.2.1
                  86400 IN A 172.16.2.1
                  86400 RRSIG A 5 3 86400 20050518171727 (
                        20050418171727 3977 example.com.
                        srHGYT4F2T8IRQTRctl/ZzQa494= )
                  10800 NSEC example.com. A RRSIG NSEC
                  10800 RRSIG NSEC 5 3 10800 20050518171727 (
                        20050418171727 3977 example.com.
                        5dkPy1jAM2izam5W9Eri/7PdaXI= )
```

BIND will need to be reloaded or `rndc` (freeze/thaw) used to pick up the new zone file. Because `sub.example.com` gets its authentication through the delegation point in `example.com`, the trusted anchor configured at `ns1.example.net` also covers `sub.example.com`, and no additional configuration is required.

Key Rollover

As described earlier, the ZSK and the KSK are required to be periodically changed, or *rolled over*, using either a prepublish or double-signing strategy. In general, prepublish is best used for ZSKs and double signing for KSKs. The process of key rollover is messy but not difficult, and lends itself to a level of script or other automation, for example, running from `cron`.

Note When performing any zone re-signing, for key rollover or normal zone signing maintenance procedures, on zones that are dynamically updated, then the additional procedures documented in "Dynamic DNS and DNSSEC" in this chapter should be followed.

Prepublish ZSK Rollover

The objective in the prepublish strategy is to get the current and new DNSKEY RRs into the caches of all security-aware name servers. This is done by first adding a new ZSK to the zone file. This example will assume that the signed zone file for `example.com` created previously with a current ZSK `key-tag` of `03977` and a current KSK `key-tag` of `12513` will have only the ZSK (`key-tag` of `03977`) rolled. By looking at the zone file, the longest TTL is 24 hours (`86400` seconds). At least two days before the zone signatures expire or before the new ZSK is required, a new ZSK is created using the following command:

```
# dnssec-keygen -a rsasha1 -b 1024 -n zone example.com
Kexample.com.+005+39539
```

The new ZSK is included in the zone file:

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@             IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
            IN  NS ns1.example.com.
            IN  NS ns2.example.com.
            IN  MX 10 mail.example.com.
            IN  MX 10 mail1.example.com.
_ldap._tcp  IN  SRV 5 2 235 www
ns1         IN  A  192.168.2.6
ns2         IN  A  192.168.23.23
www         IN  A  10.1.2.1
            IN  A  172.16.2.1
mail        IN  A  192.168.2.3
mail1       IN  A  192.168.2.4
$ORIGIN sub.example.com.
@           IN  NS ns3.sub.example.com.
            IN  NS ns4.sub.example.com.
ns3         IN  A  10.2.3.4 ; glue RR
ns4         IN  A  10.2.3.5 ; glue RR
$INCLUDE keys/Kexample.com.+005+12513.key ; KSK
$INCLUDE keys/Kexample.com.+005+03977.key ; current ZSK
$INCLUDE dsset-sub.example.com. ; DS RR
$INCLUDE keys/Kexample.com.+005+39539.key ; new ZSK
```

The zone is signed using the *current* ZSK and KSK—there are no changes to the command used in the previous section:

```
# dnssec-signzone -o example.com -t -k Kexample.com.+005+12513 \
master.example.com Kexample.com.+005+03977
master.example.com.signed
Signatures generated:                      20
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Runtime in seconds:                     0.357
Signatures per second:                 53.079
```

The \ indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. When the file is signed, BIND is either reloaded or `rndc` (freeze/thaw) commands used to refresh the zone file. The DNSKEY RRset at the zone apex looks like that shown here:

```
             86400 DNSKEY 256 3 5 (
                        AQPCf56dKVA+TAzVQVedURNd/twKcbg0z
                        t/4w8JgeybiVZeHbYXHIljS0kHt8vw==
                        ) ; key id = 3977
             86400 DNSKEY 256 3 5 (
                        AQPCrtJceGC5REQ4khX5VKSvnlWgBxH/1
                        dV0aRDNEebrwNVohBMEVI1j3Nh7UIQ==
                        ) ; key id = 39539
             86400 DNSKEY 257 3 5 (
                        AQPnvgDqCShrBmFEh5vW7kM4DG/kMwa3E
                        1WnVhGKV1I6T01x+u4uNoe1/uocNOQ==
                        ) ; key id = 12513
             86400 RRSIG DNSKEY 5 2 86400 20050518182149 (
                        20050418182149 3977 example.com.
                        OyWzCmieTtR2bES6l+KFXBOosv8= )
             86400 RRSIG DNSKEY 5 2 86400 20050518182149 (
                        20050418182149 12513 example.com.
                        IQOteNohrH5oZ1+2EM22LrPFHrk= )
_ldap._tcp.example.com. 86400 IN SRV 5 2 235 www.example.com.
```

The new DNSKEY RR (the second DNSKEY RR shown previously) is now available in the DNSKEY RRset at the zone apex where it can be used by security-aware name servers to try and verify the signatures. At this point, all such attempts will fail because no RRSIG records use this key. Recall, however, that name servers are mandated to try all available DNSKEY RRs, so the current ZSK will also be used and the RRSIGs will be validated. After 24 hours from the zone being reloaded, all security-aware name servers using the `example.com` zone will be guaranteed to either have the new DNSKEY RRset in the cache or have timed out the old version, which only has the current KSK and current ZSK.

After the 24 hours cache propagation period has passed, the zone is again re-signed using the KSK as before and the new ZSK (`key-tag` of `39539`) using the following command:

```
# dnssec-signzone -o example.com -t -k Kexample.com.+005+12513 \
master.example.com Kexample.com.+005+39539
master.example.com.signed
Signatures generated:                      20
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Runtime in seconds:                     0.357
Signatures per second:                 53.079
```

The \ indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. All the RRSIG RRs have now been signed with the new ZSK. Again, BIND is reloaded or `rndc` used to refresh the zone. After a further 24-hour period, all security-aware name servers that use the `example.com` zone will have the new DNSKEY RRset either cached or have timed out the old DNSKEY RRset. Any time after this the zone file may be modified to delete the previous ZSK (`key-tag` is `03977`), the zone re-signed—using the new ZSK (`key-tag` is `39539`) as in the preceding command—and then reloaded. There is no particular urgency to delete the old key and to minimize re-signing operations; this can be postponed until either the next scheduled zone re-signing or the next scheduled key rollover.

Double-signing KSK Rollover

Recall the KSK only signs the DNSKEY RRset at the zone apex. The double-signing strategy uses two KSKs to sign this RRset. Again, the example file created previously and now signed with the new ZSK will be used as a starting point for the KSK rollover. Create a new KSK using the following command:

```
# dnssec-keygen -a rsasha1 -b 1024 -f KSK 1024 -n zone example.com
Kexample.com.+005+50148
```

This new DNSKEY is included in the `master.example.com` zone file:

```
$TTL 86400 ; 1 day
$ORIGIN example.com.
@             IN SOA ns1.example.com. hostmaster.example.com. (
                        2005032902 ; serial
                        10800      ; refresh (3 hours)
                        15         ; retry (15 seconds)
                        604800     ; expire (1 week)
                        10800      ; minimum (3 hours)
                        )
            IN  NS ns1.example.com.
            IN  NS ns2.example.com.
            IN  MX 10 mail.example.com.
            IN  MX 10 mail1.example.com.
_ldap._tcp  IN  SRV 5 2 235 www
ns1         IN  A  192.168.2.6
ns2         IN  A  192.168.23.23
www         IN  A  10.1.2.1
            IN  A  172.16.2.1
mail        IN  A  192.168.2.3
mail1       IN  A  192.168.2.4
$ORIGIN sub.example.com.
@           IN  NS ns3.sub.example.com.
            IN  NS ns4.sub.example.com.
ns3         IN  A  10.2.3.4 ; glue RR
```

```
ns4         IN  A  10.2.3.5 ; glue RR
$INCLUDE keys/Kexample.com.+005+12513.key ; current KSK
$INCLUDE keys/Kexample.com.+005+50148.key ; new KSK
$INCLUDE dsset-sub.example.com. ; DS RR
$INCLUDE keys/Kexample.com.+005+39539.key ; new ZSK
```

The zone is signed with the following command:

```
# dnssec-signzone -o example.com -t -k Kexample.com.+005+12513 \
-k Kexample.com.+005+50148 master.example.com Kexample.com.+005+39539
master.example.com.signed
Signatures generated:                      21
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Runtime in seconds:                     0.357
Signatures per second:                 53.079
```

The \ indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. The command has two -k arguments, indicating the DNSKEY RRset will be signed three times, once each with the current and new KSK and once with the ZSK. Note also that the `Signatures generated` line has increased to 21 to confirm this. BIND should now be reloaded or `rndc` (freeze/thaw) commands used to refresh the zone. The DNSKEY RRset zone file fragments looks as shown here with three RRSIG RRs covering the DNSKEY RRset:

```
         86400 DNSKEY 256 3 5 (
                  AQPCrtJceGC5REQ4khX5VKSvnlWgBxH/1xpg
                  dV0aRDNEebrwNVohBMEVI1j3Nh7UIQ==
                  ) ; key id = 39539
         86400 DNSKEY 257 3 5 (
                  AQPZjeWTe9q998Oo2XWmaaMYNb9xxMDdwHNH
                  waQ5CQ6tVQwg5udwmnWTJt5ryBI+DQ==
                  ) ; key id = 50148
         86400 DNSKEY 257 3 5 (
                  AQPnvgDqCShrBmFEh5vW7kM4DG/kMwa3EBnP
                  1WnVhGKV1I6T01x+u4uNoe1/uocNOQ==
                  ) ; key id = 12513
         86400 RRSIG DNSKEY 5 2 86400 20050518190823 (
                  20050418190823 12513 example.com.
                  ZI95WgwWViUm7YYe2dwznC5M17Q= )
         86400 RRSIG DNSKEY 5 2 86400 20050518190823 (
                  20050418190823 39539 example.com.
                  wLonbgY9l3DigWoLb6zDZxlnEVQ= )
         86400 RRSIG DNSKEY 5 2 86400 20050518190823 (
                  20050418190823 50148 example.com.
                  OVU+uuC8LYj8561OsmDlNE9pzVY= )
```

Again, in the interest of brevity, most of the base64 key material has been omitted. Recall that the KSK is identified by having a `flags` field of value `257` (see Chapter 13). The DNSKEY RRset is signed three times, once each with the current KSK (`key-tag` is `12513`), the new KSK (`key-tag` is `50148`), and the current ZSK (`key-tag` is `39539`). The file `Kexample.com.+005+50148.key` should be sent to all security-aware name server administrators that have a trusted anchor configured for the zone `example.com`. While this file does not contain sensitive information, it contains a public key; it is important that the recipient be able to authenticate the sender in order to establish the right level of trust, and therefore a secure process such as secure e-mail, HTTPS, or secure FTP should be used.

There are two possible strategies for distribution of a new trusted anchor:

1. Delay 24 hours (or whatever TTL is being used) from the time of re-signing before distributing or notifying users of the availability of the new anchor. This ensures that all security-aware name servers will have cached the new DNSKEY RRset or timed out the old (single KSK) DNSKEY RRset, in which case, they will query and obtain the new DNSKEY RRset. At this point, the existing trusted anchor may be *replaced* in the `trusted-keys` clause.

2. Distribute or notify immediately users of the zone's trusted anchor when the zone is re-signed, in which case the trusted anchor must be *added* to the server's `trusted-keys` clause to allow for cache delays. The example shown uses this procedure for no very good reason.

Only when positive confirmation is received that the new trusted anchor has been *added* or *replaced* in the `trusted-keys` clause, as shown in the following example, can the current KSK be removed from the zone file and the zone file be re-signed using the new KSK (`key-tag` is `50148`) and the current ZSK (`key-tag` is `39539`). The new DNSKEY RR for the KSK is as follows:

```
      86400 DNSKEY 257 3 5 (
                AQPZjeWTe9q998Oo2XWmaaMYNb9xxMDdwHNH
                waQ5CQ6tVQwg5udwmnWTJt5ryBI+DQ==
                ) ; key id = 50148
```

It is shown being added as a trusted anchor to the current `trusted-keys` clause in all affected name servers such as `ns1.example.net` as shown here:

```
trusted-keys{
   "example.com." 257 3 5 "1WnVhGKV1I6TO1x+u4uNoe1/uocNOQ=="; // old KSK
   "example.com." 257 3 5 "AQPZjeWTe9q998Oo2XWmaaMYNb9xxMDdwHNH
                waQ5CQ6tVQwg5udwmnWTJt5ryBI+DQ=="; // new KSK
};
```

Using the delayed notification process described previously, the new trusted anchor could also have replaced the previous one. When the old KSK (`key-tag` is `12513`) is removed from the zone file and the zone re-signed, all users of the *stale* trusted anchor can be informed so they can remove the trusted anchor from the `trusted-keys` clause at some suitable time. There is no pressing need to do this, so it can be scheduled as part of a regular DNS maintenance session or even postponed until the next key rollover.

Any security-aware name server that does not upgrade, by adding or replacing, to the new trusted anchor will suddenly start generating bogus data responses to zone data so it vital that a reliable process is in place to get feedback. The consequences of re-signing too early with only the new KSK are also severe, and again, unless there is a pressing reason such as KSK compromise with active damage occurring, it is better to wait.

DNSSEC Lookaside Validation

The DNSSEC Lookaside Validation (DLV) service is an alternative method by which a chain of trust may be created and verified without the need to sign the parent zone file. The service makes use of a DLV RR, which is not currently defined by an RFC—its status is therefore experimental—but which is fully supported by the current (9.3+) versions of BIND. The DLV RR is functionally identical to the DS RR and may be generated by the `dnssec-signzone` utility by use of the `-l` domain-name option (see Chapter 9). A DLV RR is placed in a special signed zone called a *lookaside zone* instead of the DS RR that would normally be added to the parent zone, thus removing the need to sign the parent zone. The DLV service works by providing an alternative method to verify a chain of trust as described next.

Assume that the lookaside domain is called `dlv.example.net` and the name server is trying to verify the chain of trust for the signed zone `example.com`. In a normal sequence, when a security-aware name server tries to verify the chain of trust for `example.com`, it will first check for a trusted anchor in its `trusted-keys` clause, and if one is not found, it will issue a query to find a DS RR at the parent `.com` zone. If neither is found, the zone will be marked as insecure. DLV adds an additional step by allowing the name server to query a lookaside zone, for which it must have a trusted anchor, for the DLV RR of the zone being verified. When the verifying name server detects that the lookaside feature is enabled (by a `dnssec-lookaside` statement in `named.conf`), it will issue a DLV query with the domain name `example.com.dlv.example.net`, which, if found, and assuming the trusted anchor for `dlv.example.net` is present in a `trusted-keys` clause, the `example.com` zone is verified to be secure. Figure 11-7 illustrates the DLV process.

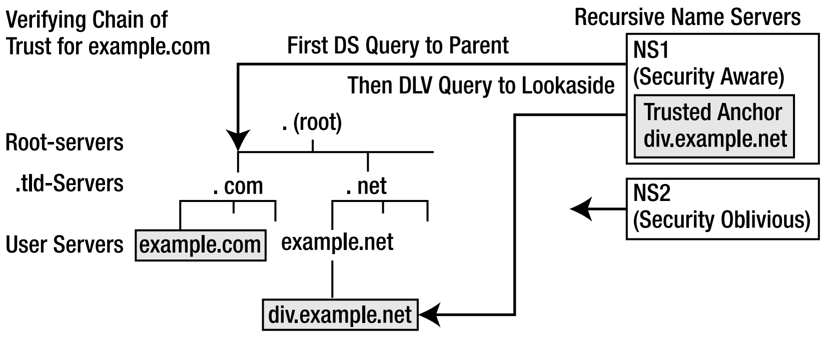


Figure 11-7. *DLV verification procedure*

The initial query will try to find a DS RR for `example.com` at the parent `.com` zone and only if that fails will the DLV query be issued to the lookaside zone. While the lookaside zone dlv.example.net must be signed, the trusted anchor at NS1 means that its parent, example.net, does not have to be signed, as is shown in Figure 11-7.

A public pilot of DLV is currently being run by VeriSign Labs, a division of VeriSign, Inc. (www.verisignlabs.com), which covers *all* of the TLDs with a single trusted anchor, without the need for any of the TLD zones to be signed.

DLV Configuration

This section describes the various steps to be taken when joining the zone example.com to a DLV chain of trust. While the specific example of the VeriSign Pilot is used, the explanations cover the general case wherever appropriate. The lookaside zone for the VeriSign Pilot is dlv.verisignlabs.com.

The DLV system, like all other DNSSEC systems, starts with a signed zone. The example.com zone is signed in the normal way as described earlier using the dnssec-signzone utility with the addition of a -l dlv.verisignlabs.com option to create a DLV RR with the correct name (example.com.dlv.verisignlabs.com). Creation of this DLV RR is the only reason the zone needs to be re-signed. This step is actually not required for the current VeriSign Pilot project, which creates 'the DS' RR automatically when a zone is submitted for addition to the pilot project. The process is described in full, since other DLV services may, however, require a DLV RR to be supplied.

Assuming the same configuration as the last example but using only the new KSK from the rollover (key-tag is 50148), the zone signing would use the following command:

```
# dnssec-signzone -o example.com -t -l dlv.verisignlabs.com \
-k Kexample.com.+005+50148 master.example.com Kexample.com.+005+39539
master.example.com.signed
Signatures generated:                      20
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Runtime in seconds:                     0.357
Signatures per second:                 53.079
```

As noted previously, joining the VeriSign Pilot does not require a DLV RR, and since this is the only reason for re-signing the zone, it may be omitted if that is the only objective. The \ indicates that the line has been split for presentation reasons only, meaning the first and second lines actually appear as a single line to the operating system. The -l dlv.verisignlabs.com argument defines the name of the lookaside zone that will be appended to this zone name (defined by the -o example.com argument) when the DLV RR is created. This option causes dnssec-signzone to create a new file called dlvset-example.com., which contains a formatted DLV RR as shown here:

```
example.com.dlv.verisign.com. IN DLV 50148 5 1 (0CAE34D
   C1BDE4A5D12A777A8DEC3B703E516DC71)
```

This DLV has been edited to use multiple lines using the normal zone file method of enclosing in parentheses for presentation reasons only, and from inspection and comparison with the DS RR from the previous example files, it may be seen to be functionally identical.

Depending on the operational or business requirements of the lookaside zone service operator, the DLV RR may need to be sent by a secure process. While the data itself is not sensitive, secure transmission allows the recipient to authenticate the *sender*, not just the data. In the case of the VeriSign Pilot, the DLV RR is synthesized when the zone is registered on the project's secure web site (`https://www.dlv.verisignlabs.com`). The submitted zone is inspected by VeriSign software by querying for DNSKEY RRs at the zone apex, and it will automatically create and add to its database a DLV RR for any DNSKEY RR with a `flags` field value of `257` (the SEP or KSK bit is set). This type of procedure may or may not become common practice, but it certainly demonstrates a level of automation that would also be practical for DS RRs.

■ **Note** The VeriSign Pilot *requires* that the DLV RR points to a KSK; that is, the DNSKEY RR has a `flags` value of `257` as described earlier (the SEP bit is set)—therefore a separate KSK and ZSK are required per the recommended practice.

To configure the name server to use the DLV service requires two changes to the `named.conf` file. The first involves definition of the lookaside zone name (in a `dnssec-lookaside` statement), and the second requires the inclusion of a trusted anchor for the DLV zone. The trusted anchor for the pilot project (at the time of writing there were two such anchors) may be obtained from `https://www.dlv.verisignlabs.com/trusted.html`. This is a secure web page, and users are recommended by text on the page to verify the security certificate—a simple but effective authentication process—before using the public keys defined, which may be simply cut and pasted into the `named.conf` file as shown in the following fragment:

```
// named.conf DLV fragment
...
options {
   ....
   dnssec-lookaside "." trusted-anchor "dlv.verisignlabs.com";
   ....
};
trusted-keys{
dlv.verisignlabs.com. 257 3 5
                 "AwEAAbw2HZErA6PpTSVdEbdvY1I1ly3gTFAhJPAsC7oa
                  tIr/P3hDqz7sUjDy4rVHQPNjKvQMv2vOAqTyrykryO2l
                  9WGmbKZjsXyK219AiAHvSC44TsiskIN8IP28KkM1CWg+
                  1O8FbPJGVbZ3OH1leRapnCCi2Z5qOdhecgFQWag/FupH
                  oqN7snieYsUdby/9ZO9dLDdQeL9xJn1CVtiMcxfB5/ju
                  KJ/V9bF7WIsdLKlootqniS42cjsyGGwxsfZxHQ3mH/GO
                  df1KnGs8ENBnpxSytJk4qOgYP5AkNAPTGOj1Kdma2f9v
                  i6wZAIYVkcQPKusBTYbUclFrIXnKGtPHH3Cny1s=";
dlv.verisignlabs.com. 257 3 5
                 "AQPAQR5KGn12Q/IPhkGMv6ZlAI57rw44/7csvZjuOvWD
                  bFuOGOCwwiRVa7FTh2MQIkCgUjqJ2ZTKTAyBMSadqFoV
                  Cc/CI6CFuQN+inmNNkGZsn5lE8qIoJkMIyl+/vO/OwhL
                  OCurFhObuyNJKouKqoO9wi3pOKrWQRkbnLWlcqeqfsAN
```

```
                    Gpxi27TveSm3x3pS8f9ZXHQvz5yFethXitHDQuYl+apF
                    ODsZ/TfXE9dl7+oR+5hzbzIMbPBByuqna4/ZFCcwJL2W
                    hEArHFQSpkzUaVX2ugBZ48HOM9XqG8aUCkElRAkxrawf
                    5x3bm6y3UmoQPvTQL8T71BZ6Cku84FyDGUoh";
};
```

The VeriSign DLV Pilot provides support to cover the whole hierarchy of the domain name to the root and therefore recommends a `dnssec-lookaside` statement with a `.` (root) domain as shown in the preceding example. The effect of this definition is that every secure zone for which there is no parent DS RR and no trusted anchor will incur a DLV query to the domain `dlv.verisignlabs.com`, which may be an unacceptable overhead. If the user wants to limit this process to only the `.com` domain, the following alternative statement could be used:

```
dnssec-lookaside ".com" trusted-anchor "dlv.verisignlabs.com";
```

In this case, only domain names ending with `.com` will incur a DLV lookup. Similarly, this could have been limited to `.at` or `.org` domains or multiple `dnsssec-lookaside` statements used to select only the `.de` and `.org` domains, depending on requirement. The trusted anchor name of `dlv.verisignlabs.com` is unique to the current VeriSign Pilot project and references entries (in the preceding case two) in the `trusted-keys` clause with the same name.

DLV Service

There is nothing magical about a DLV service. A DLV service uses a standard name server with a standard signed zone file and could be created for use by any affinity group as an alternative to multiple trusted anchors for each member of the group. To illustrate creation of a DLV service, assume an affinity group comprised of the domains `example.org`, `example.com`, and `example.net` decide to set up a DLV service that will hosted by `dlv.example.com`. In the absence of any special software as used by the VeriSign Labs Pilot, each member domain will create a DLV RR by the zone signing process described using a `-l dlv.example.com` argument. The DLV RRs are sent to the domain administrator for `dlv.example.com` by a process that will authenticate the sender, such as secure e-mail. A zone file comprising the supplied DLV RRs will be created as shown here:

```
; zone fragment for dlv.example.com
$TTL 1d ; zone default
$ORIGIN dlv.example.com.
@        IN SOA ns1.dlv.example.com. hostmaster.dlv.example.com. (
                   2005032902 ; serial
                   10800      ; refresh (3 hours)
                   15         ; retry (15 seconds)
                   604800     ; expire (1 week)
                   10800      ; minimum (3 hours)
                   )
           NS ns1.dlv.example.com.
           NS ns2.dlv.example.com.
ns1        A  192.168.254.2
ns2        A  192.168.254.3
; DLV RRs for affinity group
```

```
example.com.dlv.example.com. IN DLV 37558 5 1 (CCCCCCCCCCCC)
example.org.dlv.example.com. IN DLV 42134 5 1 (DDDDDDDDDDD)
example.net.dlv.example.com. IN DLV 02557 5 1 (EEEEEEEEEEEE)
....
```

A ZSK and KSK for the `dlv.example.com` zone will be created using the `dnssec-keygen` utility and added to the zone file as described earlier, and the zone will be signed with the `dnssec-signzone` utility using both KSK and ZSK as normal. The public key of the KSK for `dlv.example.com` is distributed to be used as a trusted anchor by all the members of the affinity group; thus a single trusted anchor is used to replace the alternative of three trusted anchors, which would otherwise be required.

The zone `dlv.example.com` would be delegated from `example.com` and an authoritative-only name server (see Chapter 7) created to support the service. Finally, each member would add the trusted anchor for `dlv.example.com` in a `trusted-keys` clause in their `named.conf file`, and to invoke the service each member would further add the following three lines to the `options` clause in the same `named.conf` file:

```
dnssec-lookaside "example.com" trusted-anchor "dlv.example.com";
dnssec-lookaside "example.net" trusted-anchor "dlv.example.com";
dnssec-lookaside "example.net" trusted-anchor "dlv.example.com";
```

The specification of `dnssec-lookaside` says that any domain *at or below* the defined domain name will use the lookaside zone defined in the `trusted-anchor` option, which means that only domain names ending with `example.com`, `example.org`, or `example.net` will incur a DLV lookup. However, the specification also says that the *deepest* domain name (which actually means the one with the most labels) defined in a `dnssec-lookaside` will be used for the lookaside query. So if a name server that included the previous three lines also wished to use, say, the VeriSign DLV Pilot service, it would *add* the following statement to invoke that DLV service:

```
dnsssec-lookaside "." trusted-anchor "dlv.verisignlabs.com";
```

The effect of this statement would be that any secure domain that does not end with `example.com`, `example.org`, or `example.net` would incur a DLV lookup to the `dlv.verisignlabs.com` lookaside domain, whereas only our three target domains, `example.com`, `example.net`, and `example.org`, would query the `dlv.example.com` lookaside domain. It is therefore possible to support a number of concurrent DLV services, each of which may target specific markets or affinity groups prior to the widespread availability of signed TLDs.

Summary

This chapter describes the theory and implementation of DNSSEC (colloquially known as DNSSEC.bis), which represents the second generation of standards used to ensure the authenticity and integrity of data supplied from a suitably configured authoritative name server to a security-aware requesting name server. DNSSEC standards use public key (asymmetric) cryptography to ensure that the data supplied in response to a query for, say, `www.example.com`, could only have come from the domain `example.com` (authenticity), that data received by the querying name server is the same as the data sent by the queried name server (data integrity), and that in the event `www.example.com` does not exist, it can be proven that such is the case (proof of nonexistence or denial of existence). Transaction security, used to secure operations such as DDNS or zone transfer, is covered in Chapter 10.

The chapter described the establishment of islands of security whereby single, unconnected zones may be secured, or a group of such isolated islands that are part of an affinity or common interest group, such as an enterprise network, may be secured. In this case, to get security coverage the zone requires a trusted anchor—the public key used to sign the secured zone—which is obtained by a secure process that authenticates the source and is then configured into all security-aware name servers that wish to validate responses for the zone using a `trusted-keys` clause. Securing the zone involves the use of a private key to digitally sign all the RRsets in the zone using an RRSIG RR type. Once established, secure zones can be linked together into chains of trust using their delegation points; thus if `example.com` is secured, it may be linked to the `.com` gTLD or it may be securely delegated to `sub.example.com`. This process is accomplished using the Delegated Signer RR, which is added to the parent domain and secures the delegation to the child domain. The public keys used in signing are defined in the zone file using DNSKEY RRs and are categorized as either a Zone Signing Key, which is used to sign the records within the zone file and a Key Signing Key, which is used to sign only the DNSKEY RRs used in the signing process and may be used externally as either a trusted anchor or referenced by a DS RR. While the standards allow a single DNSKEY RR to be used for both ZSK and KSK purposes, this not a recommended practice. Proof of nonexistence is provided by the NSEC RRs, which chain together all the RRs within the zone file. Cryptographic keys need to be changed either periodically to minimize risk or immediately in the case where a key is known to be compromised. This process, called key rollover, may use either a prepublish or double-signing strategy, both of which were described. Finally, examples illustrating the implementation of DNSSEC and covering all the preceding points were presented.

DNSSEC provides very positive benefits but does introduce new levels of discipline, particularly with regard to time—signatures (RRSIG RRs) in secured zone file have a finite validity period and thus require to be re-signed at periodic intervals. If the signatures are allowed to expire, the data from the zone will be marked as bogus by receiving security-aware servers.

The next chapter describes, with examples where appropriate, the statements and clauses used in `named.conf`, the configuration file that controls BIND's operational behavior.

PART 4

Reference

CHAPTER 12

BIND Configuration Reference

This chapter contains reference information about the daunting list of parameters that control the BIND 9 series of applications. The reference is split into two parts:

1. The BIND command-line *arguments* that control the operational environment at load time and *signals* that are accepted by a running daemon
2. The BIND configuration *parameters* defined in the `named.conf` file that control operational functionality

While the very number of configuration options may seem initially confusing, only a small subset is required for a typical BIND configuration. The normal set of parameters is illustrated in the examples in Chapter 7. This reference chapter is useful when control over a specific behavior is required, when checking the different forms of certain statements or clauses, and when considering a new design or implementation. The first section deals with command-line arguments and signals, and run-time control, and is then followed by the section on the `named.conf` parameter, which this book has separated into *clauses* (listed in Table 12-3) and *statements* (listed in Table 12-5) to provide a more logical access to the large number of configuration options available.

BIND Command Line

Table 12-1 describes the various command-line options used to control BIND.

Table 12-1. *BIND Command-Line Arguments*

Argument	Parameter	Description
`-c`	`/path/to/config-file`	Absolute path to the BIND configuration file (normally `named.conf`). This argument allows change of both the location and the name of the configuration file. The default depends on OS (typically Linux = `/etc/named.conf`, BSD = either `/etc/namedb/named.conf` or `/etc/local/etc/named.conf`, Windows = `c:\winnt\system32\dns\etc\named.conf`) and is defined by the `--sysconfdir` parameter to `configure`.
`-d`	`#debug`	See Table 12-2.
`-f`		Run in foreground, that is, do not run as daemon. Normally only used for debugging purposes.

Continued

Table 12-1. *Continued*

Argument	Parameter	Description
-g		Run in foreground, that is, do not run as daemon and log to `stderr` (console). Normally only used for debugging purposes.
-n	#cpus	Create `#cpus` threads to take advantage of multiple CPUs. If not specified, `named` will try to determine the number of CPUs present and create one thread per CPU. If it is unable to determine the number of CPUs, a single thread will be created.
-p	port-no	Listen on defined `port-no`. The default is `53`. Normally only used for debugging purposes.
-t	chroot/path	Use of this argument indicates that `named` will be run in a chroot jail (or sandbox). `chroot/path` defines the directory path of the chroot base and is conventionally set to `/var/named/chroot` (Linux) or `/var/named` (FreeBSD) but can be set to anything required. Must be used in conjunction with the `-u` argument to provide any meaningful security.
-u	UID	Cause BIND to `suid()` (change user name) to the defined `UID` after creating sockets on port 53 (which is in the privileged range of < 1024). If not present, runs as user root. Must be used with chroot options (see the `-t` entry), but many startup scripts now use a `-u named` or `-u bind` argument even if not chrooted, which means that log and PID files will have to have appropriate permissions set.
-v		Display the BIND version number to `stdout` (console) and exit.

There are two additional arguments (`-s` and `-x`) that should only be used by developers and therefore have been omitted.

BIND Debug Levels

Table 12-2 defines an incomplete list of the various debug levels that may be set using the `-d` command-line option or the `rndc trace log_level` option (see Chapter 9). For maximum logging, use 100.

Table 12-2. *BIND Debug Log Levels*

Debug Level	Coverage
0	No debugging—can also be set using `rndc notrace` (see Chapter 9).
1	Logs the basic name server operations: zone loading, maintenance (including SOA queries, zone transfers and zone expiration, and cache cleaning), `NOTIFY` messages, queries received, and high-level tasks dispatched.
2	Logs multicast requests.
3	Logs the low-level task creation, operation, and journal activity, such as when the name server writes a record of a zone change to the zone's journal file (`.jnl`) or when the name server applies a journal to a zone at startup. This level also logs the operation of the DNSSEC validator and checking of TSIG and SIG(0) signatures.
4	Logs when a master name server uses `AXFR` because the transferred zone's journal is not available.
5	Logs the view used while servicing a particular request.
6	Logs outbound zone transfer messages, including checks of the query that

Debug Level	Coverage
7	Logs the journals added and deleted, and a count of the number of bytes returned by a zone transfer.
8	Logs the following dynamic update messages: prerequisite checks, journal entries, and rollbacks. This level also logs low-level zone transfer messages and the resource records sent in a zone transfer.
10	Logs zone timer activity messages and client errors.
50	Logs internal event tracing.
90	Logs low-level operation of the BIND 9 task dispatcher.

BIND Signals

In general, BIND should be controlled using the `rndc` utility or the various startup scripts such as `/etc/rc.d/init.d/named` (for Linux) or `/etc/rc.d/named` for BSD. The following signals may be used:

- *SIGHUP (1)*: This signal is documented to reload the server but does not; it just terminates the server. To perform a reload or restart, see the upcoming text.
- *SIGINT (2)*: Terminate BIND.
- *SIGTERM (15)*: Terminate BIND.

To terminate BIND from the command line, the following command can be used:

```
killall named
```

or obtain the `named` PID using

```
ps ax|grep named
```

Then issue this command:

```
kill -2 named-pid
```

To perform a reload, either stop and start the server using an `rc.d` script, for instance, on Linux use `/etc/rc.d/init.d/named restart`, or issue the following command:

```
named.reload
```

The server will reload the `named.conf` file. The command `rndc reload` will perform the same operation if issued from the control channel (see the section "BIND `controls` Clause" later in this chapter).

BIND Configuration Overview

BIND uses a single configuration file called `named.conf`, which can contain a brutally long list of parameters to control its operation. The `named.conf` file can reside in a variety of places depending on your OS, for instance, for Linux (Fedora Core 2), in `/etc/named.conf`; for Windows (NT 4.0 and Windows 2000 only), `%SystemRoot%\system32\dns\etc\named.conf` (normally `c:\winnt\system32\dns\etc\named.conf`); and for FreeBSD, in `/etc/namedb/named.conf` or `/usr/local/etc/named/conf`. If BIND is being run in a sandbox (or chroot jail), the typical locations are as defined in Chapter 10.

■**Note** Older versions of BIND (the 4.x series) used a configuration file called `boot.conf`, which this book does not describe or mention other than in this note. Unless you are running a 4.x version of BIND version, ignore any references to this file in any documentation.

The `named.conf` file can contain three types of entries:

1. *Comments*: Comments can take one of three formats: C++ style, Unix shell style, or C style. Comments in the C++ style start with // and occupy a single line. This comment style can appear on its own on a line or can terminate any line. Comments in the Unix shell style start with # (hash or pound) and have the same single-line or line-terminating properties as the C++ style. Comments in the C style use /* to open a comment and */ to close the comment. C-style comments can occupy a single line or more than one line or even be used within a line. Examples:

```
/* C-style comment format needs opening and closing markers
** but allows multiple lines or */
/* single lines or */
zones /* in-line comment does not terminate line */ in {some zone statements};
// C++-style comments have single line format, no closing required
   some statement; // comment ends this line
# SHELL/PERL-style comments have single lines, no closing required
   some statement; # comment ends this line
```

2. *Clauses*: Clauses are used to group sets of statements. Clauses in some documentation are called *statements* or *options* and even *sections*, but this book uses the term *clause* throughout. Table 12-3 later in this chapter lists all the available clauses and their general content and scope.

3. *Statements*: Statements appear within clauses and control specific behaviors. Statements may have one or more parameters. Statements are called in some documentation *statements*, *clauses*, *phrases*, and even *options*, but this book uses the term *statement* throughout. Certain statements may appear in multiple clauses or clause types. The scope of the behavior depends on the clause in which the statement appears. Thus a statement that appears inside a `zone` clause has a scope for that zone—it affects behavior only for that zone. If the same statement appears inside an `options`

clause, it has global scope across all zones unless explicitly overridden by a statement in a `zone` clause. Table 12-6, shown later in this chapter, lists the statements and the clauses to which they apply. Table 12-5, also later in this chapter, lists each available statement with a brief description of its function. BIND releases include a list of the latest statements and clauses supported, which is available in `/usr/share/docs/bind-version/misc/options` (Fedora Core 2) or `/usr/src/contrib/bind/doc/` (FreeBSD). Windows users are not so lucky, as their distribution does not include such a file.

BIND provides a serious list of configuration statements, but only a small subset is necessary to create an operational configuration. Chapter 7 includes a number of sample files that use the minimum required statements and clauses.

■ **Note on Terminology** One of many reasons that users get confused is when documentation refers to the same entity by more than one term. This book has sought to use common and consistent terminology throughout. Accordingly, this book uses the term *clause* to define a grouping of statements, since Merriam-Webster (`www.m-w.com`) defines *clause* to be "a separate section of a discourse or writing; *specifically*: a distinct article in a formal document," which is good enough for us. *Statement* has an atomic meaning and is defined by Webster to be "a single declaration or remark."

Layout Styles

BIND is very picky about opening and closing brackets/braces, semicolons, and all the other separators defined in the formal syntaxes in later sections. There are many layout styles that can assist in minimizing errors as shown in the following examples:

```
// dense single-line style
zone "example.com" in{type slave; file "slave.example.com"; masters {10.0.0.1;};};
//  single-statement-per-line style
zone "example.com" in{
   type slave;
   file "slave.example.com";
   masters {10.0.0.1;};
   };
// spot the difference
zone "example.com" in{
   type slave;
   file "sec.slave.com";
   masters {10.0.0.1;}; };
```

The variations are simply attempts to minimize the chance of errors—they have no other significance. Experiment, then use the method you feel most comfortable with.

named-checkconf Is Your Friend

BIND releases contain a utility called named-checkconf that will do nothing except check your named.conf file and tell you what is wrong. To check any named.conf file for syntax errors, just issue the following command:

```
named-checkconf
```

This command will verify the named.conf file in the normal location for the distribution. If you are building a new file in another location or with a test file name, then issue a command something like the following:

```
named-checkconf /path/to/file.name
```

To check any master zone files referenced in the named.conf file, just add the -z argument:

```
named-checkconf /path/to/file.name -z
```

For a full list of named-checkconf arguments, see Chapter 9.

■**Note** In the normal sparse BIND style, if your test named.conf file is correct, named-checkconf will output nothing. Not even a courtesy "OK." Silence is indeed golden.

BIND Clauses

The named.conf file can contain a number of clauses. Clauses are used to group together sets of related statements. Table 12-3 defines the clauses available in named.conf with a brief description of their purpose and scope.

Table 12-3. *BIND Clause Summary*

Clause	Description
acl	Defines one or more access control lists, groups of hosts, or users identified by keys that may be referenced in view and other clauses or statements.
controls	Describes and manages access to the control channel used by the remote administrator when using the rndc utility.
include	This statement is unique and is documented here solely because it can be used to include clauses, parts of clauses, individual statements, or groups of statements. It obeys none of the normal rules and can appear anywhere in the named.conf file inside and outside of a clause. The include statement allows subsidiary files containing configuration clauses or statements to be included in-line. It is typically used for security or maintenance purposes.

Clause	Description
key	Defines shared-secret keys used to control and authenticate TSIG operations such as zone transfer, Dynamic DNS (DDNS), and the remote control channel (the controls clause).
logging	Defines the behavior and formatting of BIND's extensive logging feature.
lwres	Groups statements defining the behavior of BIND in lightweight resolver (lwres) mode.
masters	Defines a named set of masters for use by slave or stub zones to simplify maintenance in cases where multiple zones use common master servers.
options	Groups statements that control generic or global behavior and that have scope for all zones and views unless overridden within a zone, view, or other clause.
server	Defines the properties or behavior this name server will use when accessing or responding to a defined remote name server.
trusted-keys	Used to contain trusted anchors (an authenticated public key) used in DNSSEC.bis operations (see Chapter 11).
view	The view clause allows BIND to respond differently to different hosts, interfaces on the same server, or users. The view clause is unique in that each required zone must be defined within the view, thus allowing the defined zone to have completely different characteristics within any view. Any number of view clauses may be included.
zone	Contains statements defining the behavior for a specific zone. The scope of statements in a zone clause is limited to that zone only.

The whole named.conf file is parsed for completeness and correctness before use—this is a major change from previous releases of BIND. Prior to the availability of, or in the absence of, a valid logging clause, failure messages use syslogd and are, depending on your syslog.conf file, typically written to /var/log/messages; thereafter failures are written according to the logging clause definition. In the case of Windows, pre-logging clause failures are written to the Event Log. There are some modest rules defined for the order of clauses in BIND 9, and these are illustrated next. The general statement layout of a named.conf file is usually as follows:

```
// change log
// 1. changed by M.E. on 24th January
//   a. added something
acl "name" {...
             // acl clauses if present generally come first
             // to avoid forward references
};
key "name" {...
             // key clauses if present must appear
             // before being referenced
};

logging {
          // requires at least a file
          // statement unless using syslog
          // order not important with BIND 9
};
```

```
options {
         // other statements (as required)
};
// zones clauses including 'required' zones
zone {
....
};
zone {
....
};
```

If the view clause is being used, the order changes significantly as shown here:

```
// change log
// 1. changed by M.E. on 24th January
//   a. added something
acl "name" {...
             // acl clauses if present come first
             // to avoid forward references
};
key "name" {...
             // key clauses if present must appear
             // before being referenced
};
logging {.
          // usually requires at least a file statement
          // unless using the syslog
          // order not important with BIND 9
};
options {
         // global options
         // other statements as required
};
view "first" {
  // view specific statements
  // view specific zone clauses
  // including required zones such as hint or localhost
  zone {
        };
  .....
  zone {
       };
};    // end of view "first"
 view "second" {
  // view specific statements
  // view specific zone clauses
  // including required zones such as hint or localhost
  zone {
        };
  zone {
        };
};   // end of view "second"
```

BIND `address_match_list` Definition

Many statements and some clauses use the `address_match_list` construct as a basic and consistent building block from which complex matching conditions may be constructed. It is described here, somewhat out of order, simply because it is referenced and used so frequently. Rather than try and understand it now, you may be better advised to skim this section and get a general feel for what it contains, and then continue reviewing the clauses and statements until the need to understand this structure in detail becomes inevitable. The full syntax allows multiple variations:

```
address_match_list = element ; [ element; ... ]
```

An `address_match_list` is comprised of one or more elements, each of which has the following syntax:

```
element = [!] (ip [/prefix] | key key-name | "acl_name" | { address_match_list } )
```

The following are elements that make up an `address_match_list`:

- Optional negation (`!`) of an element
- An IP address (IPv4 or IPv6)
- An optional IP prefix (in the slash notation), for instance, 10.0.0.0/16 (or 10.0/16)
- A `key-name`, as defined in a `key` clause
- The name of an `address_match_list` previously defined with an `acl` clause or one of four predefined names (see Table 12-4)
- A nested `address_match_list` enclosed in braces

Table 12-4 shows the four predefined `address_match_list` names.

Table 12-4. *Predefined* `address_match_list` *Names*

Name	Description
`any`	Matches all hosts.
`none`	Matches no hosts.
`localhost`	Matches the IPv4 and IPv6 addresses of all network interfaces on the server. For instance, if the server has a single interface with an IP address of 192.168.2.3, then localhost will match 192.168.2.3 and 127.0.0.1 (the loopback address is always present).
`localnets`	Matches any host on an IPv4 or IPv6 network for which the server has an interface. That is, if the server has a single interface with an IP address of 192.168.2.3 and a netmask of 255.255.255.0 (or 192.168.2.3/24), then `localnets` will match 192.168.2.0 to 192.168.2.255 and 127.0.0.1 (the loopback is always assumed to be a single address).

One of the major uses for the `address_match_list` structure is with IP addresses for access control. When a given IP address is compared to an `address_match_list`, the list is traversed in order until an element matches, at which point processing stops. The action taken will depend on the context of the statement to which it is being applied as shown in the following example:

```
options {
     allow-transfer { !192.168.2.7;192.168.2.3/24;};
};
```

If the IP address 192.168.2.47 requests a transfer, it does not match the first element but matches the second element and the transfer is permitted. If, however, the IP 192.168.2.7 requests a transfer, it matches the first element that is negated, resulting in the transfer being denied. Because a match stops processing, the match order is significant. If the preceding were rewritten to reverse the order as shown in the following fragment, then 192.168.2.7 would always be permitted to transfer because the first item always matches:

```
options {
     allow-transfer {192.168.2.3/24; !192.168.2.7;};
};
```

The general rule may be expressed as follows: a nonnegated match *permits* the operation and a negated match *denies* the operation; if there is no match, the operation is *denied*.

An `address_match_list` can contain an `acl-name`. The following example shows the use of an `acl` clause to standardize an `address_ match_list`. By simply changing the contents of the `acl`, these changes are available to all users of the referenced `acl` clause:

```
acl "good-guys" {
     !192.169.2.5/28; // denies first 16 IPs
     192.168.2.5/24;  // allows rest of subnet
     localnets;   // allows our network
     2001:db8:0:1::/64; // allows this subnet only
};
options {
      allow-transfer {"good-guys";};
};
```

The `key-name` parameter allows the `address_match_list` to reference a `key` clause—the match in this case will occur if the incoming `key-name` in, say, a secure dynamic update transaction matches the `key-name` in a `key` clause.

Nesting is generally only used with the `topology` (not currently implemented in BIND 9), and the `sortlist` statement and the `address_match_list` behavior is slightly changed. Its use is described in the context of the `sortlist` statement.

■**Note** When using names in the named.conf file such as the address_match_list predefined name "none" shown previously or any user-defined name, they can be written with or without the quotation marks. However, if the name contains a space character, it must be enclosed in quotes. In general, and to avoid errors, this book uses quotes to enclose all names and will typically refer to them as *quoted strings*. While not always necessary, as just explained it is designed solely to prevent errors of omission.

BIND acl Clause

The acl (Access Control List) clause allows fine-grained control over which hosts may perform what operations on the name server. The acl clause can be used to hide complexity throughout the named.conf configuration. One or more acl clauses can contain complex sets of conditions, the address_match_list, just once in the named.conf file; thereafter, whenever the same conditions apply, the acl clause is simply referenced by name. The most common use of the acl clause is in conjunction with the view clause, but using it solely for this purpose undervalues the utility of this clause.

acl Clause Syntax

```
acl "acl-name" {
    address_match_list
};
```

The acl clause defines a named structure (acl-name) containing an address_match_list that may then be referenced from one or more statements and view clauses. The acl clause *must* be defined before it is referenced in any other statement or clause. For this reason, acls are usually defined first in the named.conf file. acl-name is an arbitrary, but unique, quoted string defining the specific name by which the address_match_list may be subsequently referenced. Any number of acl clauses may be defined. The following predefined or special acl-name values are built into BIND:

- none: Matches no hosts
- any: Matches all hosts
- localhost: Matches all the IP address(es) of the server on which BIND is running
- localnets: Matches all the IP address(es) and subnet masks of the server on which BIND is running

The special acl-name values and the full address_match_list structure are described in further detail in the section “BIND address_match_list Definition” earlier. The following examples show acl clauses being created and used including use of the special or predefined acl-names:

```
//defining acls
// simple ip address acl
acl "someips" {
  10.0.0.1; 192.168.23.1; 192.168.23.15;
};
// ip address acl with '/' format
 acl "moreips" {
  10.0.0.2;
  192.168.23.128/25; // 128 IPs
};
// nested acl
acl "allips" {
  "someips";
  "moreips";
};
// messy acl
acl "complex" {
  "someips";
  10.0.15.0/24;
  !10.0.16.1/24; // negated
  {10.0.17.1;10.0.18.2;}; // nested
 };
view "my stuff" {
    match-clients {"someips";};
    ....
};
// using acls
zone "example.com" in{
  type master;
  file "master.example.com";
  also-notify {"allips";};
};
zone "example.net" in{
  type slave;
  masters {192.168.2.3;192.168.2.4;};
  file "slave.example.net;
  allow-transfer {"none";}; // this is a special acl
};
```

BIND controls Clause

The `controls` clause is used to define access information when using remote administration services, specifically the `rndc` utility. The `controls` clause takes a single `inet` statement type, though more than one `inet` statement may be defined in a `controls` clause. The `inet` statement is defined in the later section "BIND `controls` Statements."

```
controls {
   inet inet_spec [inet_spec]  ;
};
```

A `controls` clause is *always defaulted* and generates a TCP `listen` operation on port 953 (the default control port) of the loopback address for either or both IPv4 and IPv6 (127.0.0.1 and/or ::1). If remote administration will not be used (that is, the `rndc` utility will not be used), this control interface can be explicitly disabled by defining an empty `controls` clause as shown here:

```
controls {};
```

The primary access control method for remote administration, `rndc` in BIND 9, is via the use of `keys` defined within the `inet` statement (see the following example). To retain compatibility with previous versions of BIND or to run without a user-generated key, a default key may be generated using the following command:

```
rndc-confgen -a
```

This command will create a file called `rndc.key` containing a default `key` clause with the name `"rndc-key"` (`rndckey` in Fedora) in the same directory as the `named.conf` file for the version of BIND being used; this file is used for subsequent access to the control channel. If this command is not executed before BIND is loaded, the following message will appear:

```
named [39248] none:0: open: /path/to/default/rndc.key: file not found
```

BIND will continue to run in this state, but the control channel will not be operable. For full configuration of the `inet` statement and examples of its use in the `controls` clause, see the section “BIND `controls` Statements” later in this chapter.

BIND include Statement

The `include` statement is unique in that it can appear anywhere in the `named.conf` file, either inside or outside a clause. It causes the specified file to be read at the point it is encountered and takes the following form:

```
include "file-name";
```

`file-name` is a quoted string and can be an absolute path, for instance, `/var/named/file.name`, or relative, for instance, `file.name`, in which case it will be assumed to be in the directory previously defined by a `directory` statement. In the absence of a `directory` statement, this will be the directory in which `named.conf` is located (defined by the `--sysconfdir configure` argument—see Chapter 6), which is normally `/etc` (or `/etc/namedb` for FreeBSD).

■**Note** The `include` statement is BIND specific and should not be confused with the RFC 1035 standard `$INCLUDE` directive used in zone files, though it has a similar function.

The `include` statement is typically used for three purposes:

1. *To simplify or distribute administration of* `named.conf` *file maintenance*: For example, zones may be administered independently by divisions of a company.

2. *To isolate and partition changes and updates*: For example, if `acl` clauses change frequently, it may be desirable to separate them into files that can be included, thus minimizing the need to edit the primary `named.conf` file.

3. *To control permissions*: It may be desirable to limit access using restricted permissions to files containing, for example, `key` clauses. Conversely, it may be used to loosen permissions on widely edited parts of the file.

The following example shows use of the `include` statement:

```
// include two acl clauses
include "/var/named/acl/private.acl"
include "/var/named/acl/public.acl"
options {
    // relative to named.conf directory
    include "some.options";
    directory "/var/named";
    // relative to 'directory'
    include "other.options"
};
//  using include for zones
...
// zones for chemical division - absolute path
include "/var/named/chemical/zone.files";
// zones for engineering division
include "/var/named/engineering/zone.files";
// these load from the path specified by 'directory' option
include "more-zone.files";
// housekeeping zones explicitly included
zone "64/27.23.168.192.in-addr.arpa" in{
   type master;
   file "192.168.23.rev";
};
```

The included files are simply the clauses or statements that would have been present in the `named.conf` were the `include` statement not present. To illustrate this principle, the included file `/var/named/acl/private.acl` referenced previously could look as shown here:

```
// included acl
 acl "private-acl" {
  10.0.0.1;
  192.168.23.128/25; // 128 IPs
};
```

Similarly, the `other.options` file could contain one or more statements as follows:

```
recursion yes;
allow-transfer {"none";};
```

BIND key Clause

The key clause is only used to contain a shared secret (symmetric) Message Authentication Code (MAC) algorithm used in a TSIG transaction (see Chapter 10) or with the rndc utility (see Chapter 9). Any keys used with a public key (asymmetric) algorithm are either stored in the zone file as KEY or DNSKEY RRs or, when used as a *trusted anchor* (DNSSEC), in a trusted-keys clause (see Chapter 11).

key Clause Syntax

```
key key-name {
   algorithm algorithm-name;
   secret "key-data";
};
```

The algorithm and secret statements are described later in the section "DNS BIND Security Statements." The data for the key clause may be generated by using the dnsssec-keygen or the rndc-confgen utilities (see Chapter 9).

The material contained in a key clause is a *shared secret* and therefore represents extremely sensitive information. By convention, the key clause or clauses are always placed in a separate file and the include statement used to embed them into the named.conf file. The included file can therefore have specific permissions applied to ensure limited visibility. The key clause must always appear in the named.conf file before it is referenced. The key-name field may be any suitable name that is used by both ends of the communication transaction, and consequentially the same key clause must be used by the peer application. For example, when used for TSIG operations during zone transfer, a key clause with the same key-name must be present in the corresponding slave name server, or if it is being used with the rndc application, a key clause with the same key-name must be present in the rndc.conf file. The key clause used in rndc.conf and named.conf is exactly the same (see Chapter 9 for information on rndc.conf). The examples in Chapter 10 in the section "Securing Zone Transfers" describe how a key clause is constructed from the output of the dnssec-keygen utility.

BIND logging Clause

The logging clause defines the extensive logging services available in BIND. Prior to BIND 9, the logging clause had to appear first in the named.conf file. This is no longer the case, and the logging clause may be placed anywhere convenient. BIND uses syslogd before a valid logging clause is available, so named.conf parse errors and other information will appear in /var/log/messages (depending on syslog.conf) prior to, or in the absence of, a valid logging clause. In the case of Windows, parse errors are written to the Event Log. Only one logging clause can be defined, but multiple channels may be defined to stream logs. The logging clause can be omitted, in which case a default one is assumed—this default is described in the later section "BIND logging Statements," since its functionality requires some understanding of the various statements used in a logging clause.

`logging` Clause Syntax

```
logging {
 [  channel channel_name { channel_spec }; ]
 [ category category_name { channel_name ; [ channel_name ; ... ] };
};
```

The example shows a minimal logging configuration that will work and generate modest log volumes.

```
// named.conf fragment
logging{
    channel single_log {
    file "/var/log/named/bind.log" versions 3 size 2m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    single_log;
    };
};
```

Further examples are shown in the section "`category` Statement" later in this chapter.

BIND `lwres` Clause

BIND provides two methods of running a resolver (called a *lightweight resolver* in the BIND jargon) that uses a simplified and nonstandard (BIND-only) UDP-based protocol. The first method uses a separate daemon called `lwresd`, which is described in Chapter 14, and the second uses the `lwres` clause within a normal BIND `named.conf` file. Using this latter method means that a single instance of BIND can provide both normal DNS processing and lightweight resolver support.

`lwres` Clause Syntax

```
lwres {
    // lwres clause statements
};
```

By default, the lightweight resolver provides service on port 921. The `lwres` clause can include the `listen-on`, `view`, `search`, and `ndots` statements, which are described in the section, "BIND Resolver Statements."

BIND `masters` Clause

The `masters` clause is a *named list* of zone masters that may be referenced from a `masters` statement in a `zone` clause. It is provided to simplify maintenance of situations in which common master servers are used for a number of zones.

`masters` Clause Syntax

masters "*masters_name*" [**port** *pg_num*] { ("*masters_list*" | *ip* [**port** *p_num*] ➥
[**key** *key-name*]) ; [...] } ;

The `masters_name` parameter (a quoted string) is the unique name by which this clause will be referenced. The `pg_num` parameter changes the port number used for zone transfers for all the listed servers (the default is port 53). The `p_num` parameter changes the port number for the specific IP address only. If present, `masters_list` references another list of masters defined in another `masters` clause. The optional `key-name` parameter defines the key to be used to authenticate the zone transfer and references a `key` clause with the same name. Any `masters` clause must be defined before it is referenced in a `masters` statement. The following example shows three masters for the zone, one of which will use port 1127 for zone transfers and one of which is an IPv6 address:

```
// defining masters
masters "common masters" {
    masters {192.168.2.7; 10.2.3.15 port 1127; 2001:db8:0:1::15;};
};
// using masters
zone "example.com" in{
  type slave;
  file "slave.example.com";
  masters {"common masters";};
};
zone "example.net" in{
  type slave;
  file "slave.example.net;
  masters {"common masters";};
};
```

BIND `options` Clause

The `options` clause is used to group statements that have global scope. The `options` clause may take a ferocious number of statements—see the list found in Table 12-6 later in this chapter.

`options` Clause Syntax

```
options {
    // options statements
};
```

The `options` clause has global scope, but many of the statements that can be used within an `options` clause can also appear within a `view` or `zone` clause, in which case they will override the statement in the `options` clause for the scope in which they appear (that is, for the whole view or the specific zone). The following example shows an `also-notify` statement (used to cause `NOTIFY` messages to be sent to servers other than the servers defined with NS RRs for the zone) being used globally but being overridden for a single zone.

```
// defining options clause
options {
    ....
    also-notify {192.168.2.3;192.168.2.4;};
    ....
};
// zones
zone "example.com" {
  // NOTIFY messages for this domain sent to global
  // also-notify list
  type master;
  file "master.example.com";
};
zone "example.net" {
  // NOTIFY messages NOT sent to global
  // also-notify list
  type master;
  file "master.example.net;
  also-notify {"none";};
};
```

BIND server Clause

The `server` clause defines the behavior BIND will use when accessing (incoming or outgoing) a *remote* server. It is typically used when the remote server has specific characteristics or protocol behavior, when it provides secure DNS (DNSSEC) services, or to stop handling requests from a specific server. The `server` clause can take a modest number of statements as defined in Table 12-6, which appears later in this chapter.

server Clause Syntax

```
server ip_address {
     // server statements
};
```

The `ip_address` parameter can be either IPv4 or IPv6. The `ip_address` will only accept a single address—it cannot take an IP prefix value (slash notation). If a group of servers have common behaviors, each one will require a separate `server` clause. If the remote server is a dual-stacked server (IPv4/IPv6), both addresses will be need to be defined using separate `server` clauses. `server` clauses can appear independently (a global `server` clause) or within a `view` clause. If they appear within a `view` clause, the defined behavior is limited to that `view` clause only. Outside the `view` clause, they will either use the behavior of the global `server` clause if it exists, or if none exists, they will take the default for any statements that can appear inside the `server` clause. The following example shows a dual-stacked server that will only accept single messages in each TCP block during transfers and that cannot support EDNS:

```
// named.conf fragment
....
// IPv4 server
server 10.2.3.15 {
    transfer-format one-answer;
    edns no;
};
// IPv6 addresses of same server
server 2001:db8:0:27::17 {
    transfer-format one-answer;
    edns no;
};
....
```

BIND `trusted-keys` Clause

The `trusted-keys` clause contains one or more public keys that have been obtained by a secure process for use as trusted anchors in DNSSEC.bis operations (see Chapter 11). The data defined in this clause will be the same as that of a DNSKEY RR defined at the apex or root of the domain or zone for which this is the trusted anchor and that has been used to sign the zone, most typically as a Key Signing Key (KSK). Thus, if the domain for which a trusted anchor is defined is `example.com`, then there *must* be a corresponding DNSKEY RR with a name of `example.com`. The public keys that appear in a `trusted-keys` clause must be obtained by a secure (non-DNS) procedure. While the key data contained in a `trusted-key` clause is public (and unlike a `key` clause requires no special protection), the reason it is obtained by a secure process lies with the need to authenticate the *source* of the data, not the data itself. If the DNSKEY RR was simply read from the DNS by an insecure query, it could have been spoofed in some way. Its presence in a `trusted-keys` clause indicates that it was received from a trusted (authenticated) source. Secure domains delegated from the domain for which this trusted anchor is defined, say `sub.example.com` (a child zone), will be authenticated by the presence of a DS RR at the delegation point in the domain `example.com` (the parent zone) and thus do not require a corresponding trusted anchor. The format of each trusted anchor in a `trusted-keys` clause is shown here:

```
"domain-name" flags proto algorithm "key-data"
```

The meaning and value of the `flags`, `proto`, and `algorithm` fields are as defined for the corresponding DNSKEY RR (see Chapter 13). The `"domain-name"` field is the `name` value from the DNSKEY RR, optionally enclosed in quotation marks (a quoted string), and must be the name of the domain that it will be used to verify. The `key-data` field is copied from the `key-data` field of the corresponding DNSKEY RR and is the base64 (RFC 3548) representation of the public key enclosed in quotation marks and terminated with the ubiquitous semicolon. The following example shows a DNSKEY RR with the zone signing and SEP (a.k.a. KSK) bits set, using the RSA-SHA-1 algorithm and the corresponding `trusted-keys` clause that would be derived from it:

```
example.com.        IN      DNSKEY 257  3  5 (
                          AQPSKmynfzW4kyBv015MUG2DeIQ3
                          Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                          kfzj31GajIQKY+5CptLr3buXA10h
                          WqTkF7H6RfoRqXQeogmMHfpftf6z
                          Mv1LyBUgia7za6ZEzOJBOztyvhjL
                          742iU/TpPSEDhm2SNKLijfUppn1U
                          aNvv4w==  )
```

The trusted-keys clause using the preceding DNSKEY RR would be as shown here:

```
trusted-keys {
"example.com"  257  3  5 "AQPSKmynfzW4kyBv015MUG2DeIQ3
                          Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                          kfzj31GajIQKY+5CptLr3buXA10h
                          WqTkF7H6RfoRqXQeogmMHfpftf6z
                          Mv1LyBUgia7za6ZEzOJBOztyvhjL
                          742iU/TpPSEDhm2SNKLijfUppn1U
                          aNvv4w==";
};
```

Any number of trusted anchors for different domains may be added to a trusted-keys clause. To allow for key-rollover procedures (see Chapter 11), it is permissible to have more than one trusted anchor with the same domain-name, thus in the preceding fragment it is permissible to have a second (third, fourth, etc.) entry with the name "example.com", each of which will contain a different public key (key-data).

BIND view Clause

The view clause allows the behavior of BIND to be based on any combination of the source IP address of the request, the destination address of the request, the recursive behavior of the request, or the keys used by the user. The view clause can take a vast number of statements as defined in Table 12-6, which appears later in this chapter.

view Clause Syntax

```
view "view_name" {
    // view statements
};
```

The view_name (optionally a quoted string) is an arbitrary name that uniquely identifies the view. A view clause matches when either or both of its match-clients and match-destinations statements match and when the match-recursive-only condition is met. If either or both of match-clients and match-destinations are missing, they default to any (all hosts match). The match-clients statement defines the address_match_list for the source IP address(es) of the *incoming* messages. The match-destination statement defines the address_match_list for the destination IP address of the *incoming* messages and may be used with multihomed servers or to differentiate, for example, localhost behavior from all other IP address sources. The match-recursive-only statement may be further used to qualify the view clause based on its query type (recursive or iterative). Both the `match-clients` and `match-destinations` statements can take an optional `key` parameter, which means that `view` selection can be based on a user rather than a physical IP address, or they can point to an `acl` clause, which defines the `address_match_list`. The `view` clause is unique in that all required `zone` clauses must be defined within each `view` clause such that a zone's behavior can be significantly different in each view. Any number of `view` clauses can be used. The following example shows a `view` clause being used based on the source addresses of the DNS transactions and the presence of recursive queries:

```
// named.conf fragment
view "recursive-external" {
    match-clients {!10.2.3.4/24;};
    match-recursive-only yes;
    // other view statements
    zone "example.com" in {
    ....
    };
};
view "internal" {
    match-clients {10.2.3.4/24;};
    // other view statements
    zone "example.com" in {
    ....
    };
};
```

In the preceding example, the second `view` clause is not strictly necessary since all conditions not satisfied by the first view will be defaulted to a zone definition outside the `view` clause. Many users, however, like to add the second `view` clause to avoid confusion. For further examples of the use of the `view` clause when used with Split or Stealth server configurations, see Chapter 7.

BIND zone Clause

The `zone` clause defines the characteristics of the zone. The `zone` clause may take a significant number of statements—see Table 12-6 later in this chapter for the full list.

zone Clause Syntax

```
zone "zone_name" [class] {
    // zone statements
};
```

The `zone_name` (optionally a quoted string) defines the name of the zone or domain being defined. The `class` parameter is optional, and, if not present, the default class IN (Internet) will be used. This book always defines the `class` parameter in examples to avoid confusion at the cost of two characters of typing per zone. `zone` clauses may be defined inside a `view` clause, in which case the scope of the zone definition is limited to the `view` clause. If the zone is to be supported in another `view` clause or outside any `view` clause, the `zone` clause must be repeated even if its operational characteristics remain the same. The following example shows a `zone` clause being used inside two `view` clauses and outside the `view` clause:

```
// named.conf fragment
view "recursive-external" {
    match-clients {10.2.4.4/24;};
    match-recursive-only yes;
    // other view statements
    zone "example.com" in {
    ....
    };
};
view "internal" {
    match-clients {10.2.3.4/24;};
    match-recursive-only yes;
    // other view statements
    zone "example.com" in {
    ....
    };
};
// definition of zone behavior outside the views
zone "example.com" in {
....
};
```

BIND Statements

BIND provides a daunting list of statements to control its behavior. For convenience, they are provided in alphabetic order in Table 12-5. Each statement is categorized into a generic category (for example, Transfers) that loosely describes its functionality and is then described in detail in each category section. It is hoped that you may find this more useful when browsing to find statements to control specific behaviors. Many statements can appear within more than one clause, and Table 12-6 lists each statement in alphabetic order and the clauses within which it may be used. A number of the statements use a generic structure called an `address_match_list` that was previously described. The general format of each statement's description is a brief summary of the statement's functionality followed by the syntax of each statement with an accompanying example. The syntax is then described in detail with additional examples as appropriate.

Table 12-5. *BIND Statement Summary*

Statement	Category	Summary
`additional-from-auth`	Queries	Used in conjunction with `additional-from-cache` to control whether BIND will follow CNAME (and DNAME) out-of-zone references. The default is to follow references.
`additional-from-cache`	Queries	Used in conjunction with `additional-from-auth` to control whether BIND will follow CNAME (and DNAME) out-of-zone references. The default is to follow references.
`algorithm`	Security	Defines the algorithm to be used in a `key` clause.

Statement	Category	Summary
`allow-notify`	Transfers	Applies to slave zones only and defines an `address_match_list` that is allowed to send `NOTIFY` messages for the zone in addition to those defined in the `masters` option for the zone. The default behavior is to allow `NOTIFY` messages only from the zone masters.
`allow-query`	Queries	An `address_match_list` defining which hosts are allowed to issue queries to the server. If not specified, all hosts are allowed to make queries.
`allow-recursion`	Queries	Defines an `address_match_list` that will be allowed to perform recursive queries.
`allow-transfer`	Transfers	Defines an `address_match_list` that is allowed to transfer the zone information from this server. The default behavior is to allow zone transfers to *any* host.
`allow-update`	Transfers	Defines an `address_match_list` that is allowed to submit dynamic updates for master zones. The default in BIND 9 is to disallow dynamic updates from all hosts.
`allow-update-forwarding`	Transfers	Defines an `address_match_list` that is allowed to submit dynamic updates to a slave server for onward transmission to a master. The default is to disallow update forwarding.
`allow-v6-synthesis`		Obsolete statement.
`also-notify`	Transfers	Applies to zone masters only. Defines one or more hosts that will be sent `NOTIFY` messages when zone changes occur.
`alt-transfer-source`	Transfers	Applies to slave zones only. Defines an alternative local IPv4 address(es) to be used for inbound zone transfers by the server if that defined by `transfer-source` fails and `use-alt-transfer-source` is enabled.
`alt-transfer-source-v6`	Transfers	Applies to slave zones only. Defines an alternative local IPv6 address(es) to be used for inbound zone transfers by the server if that defined by `transfer-source-v6` fails and `use-alt-transfer-source` is enabled.
`auth-nxdomain`	Queries	Controls whether the server will answer authoritatively on returning `NXDOMAIN` (domain does not exist) answers. Default behavior is not to answer authoritatively.
`avoid-v4-udp-ports`	Operations	Defines a list of IPv4 ports that BIND will not use when initiating queries. Used to avoid ports blocked by firewalls.
`avoid-v6-udp-ports`	Operations	Defines a list of IPv6 ports that BIND will not use when initiating queries. Used to avoid ports blocked by firewalls.
`blackhole`	Queries	Defines an `address_match_list` that the server will not respond to, or answer queries for. The default is none—all hosts are responded to.
`bogus`	Server	Defined in a `server` clause and allows a remote server to be ignored. The default is not to ignore.
`cache-file`		Developer-only option.
`category`	Logging	Controls the type of data logged to a particular `channel`.
`channel`	Logging	Defines a stream of data that may be independently logged.
`check-names`	Zones/ Operations	Restricts the character set of host names to those defined by RFC 952 and 1123. Has different syntax in the `view` and `options` clause from that used in the `zone` clause and is described separately.

Continued

Table 12-5. *Continued*

Statement	Category	Summary
`cleaning-interval`	Operations	The time in minutes when the server will remove expired resource records from the cache. The default is `60`.
`coresize`	Operations	Defines the maximum size of a core dump.
`database`	Operations	Only used with BIND Simple Database (sdb) API and specifies the driver name and any initial parameters.
`datasize`	Operations	Defines the maximum memory size the server may use.
`deallocate-on-exit`		Obsolete statement.
`delegation-only`	Queries	Applies to hint and stub zones only. Controls whether queries will always return a referral.
`dialup`	Operations	Optimizes the behavior of certain operations to minimize connect time for dial-up links.
`directory`	Operations	A quoted string defining the base directory used for zone and other files.
`disable-algorithms`	Security	Disables DNSSEC algorithms from a specific zone.
`dnssec-enable`	Security	Enables DNSSEC support in BIND. The default is not to support DNSSEC.
`dnssec-lookaside`	Security	Used with DNSSEC Lookaside Validation (DLV). Controls the method of validating DNSKEY RRs at the apex of a zone.
`dnssec-must-be-secure`	Security	Defines hierarchies that must/may not be secure (signed and validated).
`dual-stack-servers`	Operations	Only valid on dual-stacked (IPv4/IPv6) servers and defines a method of reaching a server using one of the stacks.
`dump-file`	Operations	A quoted string defining the absolute path where BIND dumps the cache in response to an `rndc dumpdb` command. If not specified, the default is `named_dump.db` in the location specified by a `directory` statement.
`edns`	Server	Controls use of the EDNS0 (RFC 2671) feature. The default is to support EDNS0.
`edns-udp-size`	Operations	Defines the size of the UDP packet advertised by the server when using EDNS0. The default is 4096.
`fake-iquery`		Obsolete statement.
`fetch-glue`		Obsolete statement.
`file`	Zone	Generic file name definition—used by master or slave zone files and in `logging` clauses.
`files`	Operations	Defines the maximum number of files the server may have open concurrently. The default is `unlimited`.
`forward`	Queries	Defines the order in which forwarding is to be performed. Always used in conjunction with the `forwarders` statement.
`forwarders`	Queries	Defines one or more hosts to which queries will be forwarded. Always used in conjunction with the `forward` statement.
`has-old-clients`		Obsolete—replaced with `auth-nxdomain` and `rfc2308-type1`.
`heartbeat-interval`	Operations	Only valid with the `dialup` statement. The server will perform zone maintenance tasks for all zones marked as `dialup` whenever this interval expires.

Statement	Category	Summary
`host-statistics`		Not implemented.
`hostname`	Operations	Only used with CHAOS (CH) class. The host name the server should report via a TXT query.
`inet`	Operations	Defines the control channel to be used for remote administration (`rndc`) of the server.
`interface-interval`	Operations	Defines when periodic checks and update of server interfaces is performed.
`ixfr-from-differences`	Transfers	Controls how `IXFR` transfers are calculated.
`ixfr-tmp-file`		Obsolete statement.
`key-directory`	Security	A quoted string defining the absolute path where the keys used in the dynamic update of secure zones may be found. Only required if this directory is different from that defined by a `directory` statement.
`keys`	Server	Specifies one or more `key-names`, defined within a `key` clause, to be used with a remote server.
`lame-ttl`	Operations	Defines the number of seconds to cache a lame server indication.
`listen-on`	Operations	Defines the port and IPv4 address(es) on which BIND will listen for incoming queries. The default is port 53 on all server interfaces. Multiple `listen-on` statements are allowed.
`listen-on-v6`	Operations	Defines the port and IPv6 address(es) on which BIND will listen for incoming queries. The default is port 53 on all server interfaces. Multiple `listen-on-v6` statements are allowed.
`maintain-ixfr-base`		Obsolete statement.
`masters`	Zone	Slave only. Defines one or more zone masters.
`match-clients`	Views	Controls the hosts that satisfy a `view` clause.
`match-destinations`	Views	Controls the hosts that satisfy a `view` clause.
`match-mapped-addresses`	Operations	Controls whether an IPv4 mapped address within an IPv6 address is used in an `address_match_list`.
`match-recursive-only`	Views	Controls the hosts that satisfy a `view` clause.
`max-cache-size`	Operations	Defines the maximum amount of memory to use for the server's cache in bytes.
`max-cache-ttl`	Operations	Defines the maximum time in seconds for which the server will cache positive answers.
`max-ixfr-log-size`		Obsolete statement.
`max-ncache-ttl`	Operations	Defines the maximum time in seconds for which the server will cache negative (`NXDOMAIN`) answers.
`max-ixfr-log-size`		Obsolete statement.
`max-journal-size`	Transfers	Controls the size of the journal files used in Dynamic DNS.
`max-refresh-time`	Transfers	Only valid for slave zones. The zone refresh time is normally defined by the SOA record `refresh` parameter. This statement will override the SOA and substitute the values defined.
`max-retry-time`	Transfers	Only valid for slave zones. The retry time is normally defined by the SOA record `retry` parameter. This statement will override the SOA and substitute the values defined.

Continued

Table 12-5. *Continued*

Statement	Category	Summary
`max-transfer-idle-in`	Transfers	Only valid for slave zones. Inbound zone transfers making no progress in the defined minutes will be terminated. The default is `60` (1 hour).
`max-transfer-idle-out`	Transfers	Only valid for master zones. Outbound zone transfers making no progress in the defined minutes will be terminated. The default is `120` (2 hours).
`max-transfer-time-in`	Transfers	Only valid for slave zones. Inbound zone transfers running longer than the defined minutes will be terminated. The default is `120` (2 hours).
`max-transfer-time-out`	Transfers	Only valid for master zones. Outbound zone transfers running longer than the defined minutes will be terminated. The default is `120` (2 hours).
`memstatistics-file`	Operations	The name of the file to which the server writes memory usage statistics on exit. If not specified, the default is `named.memstats`.
`min-refresh-time`	Transfers	Only valid for slave zones. The zone refresh time is normally defined by the SOA record `refresh` parameter. This statement will override the definition and substitute the values defined.
`min-retry-time`	Transfers	Only valid for slave zones. The retry time is normally defined by the SOA record `retry` parameter. This statement will override the definition and substitute the values defined.
`min-roots`		Not implemented.
`minimal-responses`	Queries	Controls whether the server will only add records to the authority and additional data sections when they are required (for instance, delegations, negative responses). This may improve the performance of the server.
`multi-master`	Transfers	Applies to slave servers only. Controls how multiple masters serial number errors are logged.
`multiple-cnames`		Obsolete statement.
`named-xfer`		Obsolete statement.
`ndots`	Resolver	Controls how queries are constructed in the lightweight resolver.
`notify`	Transfers	Controls whether `NOTIFY` messages are sent from a zone master on zone changes.
`notify-source`	Transfers	Only valid for master zones. Defines the IPv4 address (and optional port) to be used for outgoing `NOTIFY` messages.
`notify-source-v6`	Transfers	Only valid for master zones. Defines the IPv6 address (and optional port) to be used for outgoing `NOTIFY` messages.
`pid-file`	Operations	A quoted string defining where the Process Identifier (PID) used by BIND is written. If not present, it is distribution or OS specific, typically `/var/run/named/named.pid`.
`port`	Operations	Controls the port BIND will use to provide UDP or TCP services. The default is 53. This statement is intended primarily for testing.
`preferred-glue`	Operations	Controls the order of glue records in a response (A or AAAA).
`provide-ixfr`	Transfers	Controls whether a master will respond to an incremental (`IXFR`) zone request or will only respond with a full zone transfer (`AXFR`). The BIND 9 default is to use `IXFR` if possible.
`pubkey`		Obsolete statement.

Statement	Category	Summary
query-source	Queries	Controls the IPv4 address and port on which recursive queries are issued.
query-source-v6	Queries	Controls the IPv6 address and port on which recursive queries are issued.
querylog	Operations	Controls whether logging of queries is performed—overrides the logging clause category definition.
recursing-file	Operations	The file name used when the remote command rndc recursing is issued.
random-device	Security	The source of entropy to be used by the server for DNSSEC operations. If not specified, the default value is /dev/random (or equivalent) when present and none otherwise.
recursion	Queries	Defines whether recursion (caching) is allowed or not. The default is to provide recursive support.
recursive-clients	Queries	The maximum number of concurrent recursive queries the server may perform. The default is 1000.
request-ixfr	Transfers	Controls whether a server (acting as a slave or on behalf of a slave zone) will request an incremental (IXFR) zone transfer or will request a full zone transfer (AXFR). The BIND 9 default is to request IXFR.
rfc2308-type1	Queries	Not Implemented.
root-delegation-only	Queries	Used for root domains (gTLD and ccTLD) to indicate that all responses will be referrals (delegations).
rrset-order	Queries	Defines the order in which equal RRs (RRsets) are returned. Applies to all RR types.
search	Resolver	Controls the operation of the lightweight resolver.
secret	Security	A base64-encoded string containing a shared secret in a key clause.
serial-queries		Obsolete statement.
serial-query-rate	Transfers	Defines the number of queries per second that will be issued by the server on behalf of slave zones when querying the SOA RRs. The default is 20 per second.
server-id	Operations	The ID supplied by a server when interrogated under the CHAOS (CH) class.
sortlist	Queries	Controls the order in which equal RRs (RRsets) are returned to the client resolver. This is the client-side equivalent of the rrset-order statement.
stacksize	Operations	Controls the stack size used by the server.
statistics-file	Operations	The name of the file the server appends statistics to when instructed to do so using rndc stats. If not specified, the default is named.stats.
statistics-interval		Not implemented.
support-ixfr		Obsolete statement.
suppress-initial-notify		Not implemented.
sig-validity-interval	Security	Controls the time in days when Dynamic DNS signatures will expire. The default is 30 days.

Continued

Table 12-5. *Continued*

Statement	Category	Summary
`tcp-clients`	Operations	By default, DNS uses UDP port 53 for queries but allows both TCP and UDP. `tcp-clients` allows the user to define the maximum number of TCP connections that may be supported. The default is `100`.
`tcp-listen-queue`	Operations	Controls the number of outstanding TCP `listen` operations. The minimum value is `3`.
`tkey-dhkey`	Security	The Diffie-Hellman key used by the server to generate shared keys.
`tkey-domain`	Security	The domain appended to the names of all shared keys generated with `TKEY`.
`tkey-gssapi-credential`	Security	Used with TKEY operations. The GSSAPI and the credentials required are defined by RFC 2743 and its Kerberos form is defined in RFC 1964.
`topology`		Not implemented.
`transfer-format`	Transfers	Only used by master zones. Controls how many records are packed into a message during zone transfers.
`transfer-source`	Transfers	Only valid for slave zones. Defines which local IPv4 address(es) will be bound to TCP connections used to fetch zones transferred inbound by the server.
`transfer-source-v6`	Transfers	Only valid for slave zones. Defines which local IPv6 address(es) will be bound to TCP connections used to fetch zones transferred inbound by the server.
`transfers`	Server	Limits the number of concurrent zone transfers from any given server. If not present, the default for `transfers-per-ns` is used.
`transfers-in`	Transfers	Only used by slave zones. Controls the number of concurrent inbound zone transfers. The default is `10`.
`transfers-out`	Transfers	Only used by master zones. Controls the number of concurrent outbound zone transfers. The default is `10`.
`transfers-per-ns`	Transfers	Only used by slave zones. Defines the number of concurrent inbound zone transfers from any single name server. The default is `2`.
`treat-cr-as-space`		Obsolete statement.
`type`	Zone	Defines the characteristic of a zone for example master or hint.
`unix`		Not implemented.
`update-policy`	Transfers	Applies to master zones only. Controls the rules by which dynamic updates (DDNS) may be carried out. Mutually exclusive with `allow-update`.
`use-alt-transfer-source`	Transfers	Indicates whether `alt-transfer-source` and `alt-transfer-source-v6` can be used or not.
`use-id-pool`		Obsolete statement.
`use-ixfr`		Obsolete—use `provide-ixfr`.
`view`	Resolver	Used to define resolver characteristics.
`version`	Operations	Specifies the string that will be returned to a `version.bind` query when using the CHAOS (CH) class only. If not defined, the real BIND version number is returned.
`zone-statistics`	Operations	Controls whether the server will collect statistical data on all zones (unless specifically turned off on a per-zone basis by specifying `zone-statistics no;` in the `zone` clause). These statistics `rndc stats`.

Table 12-6 lists all statements and identifies in which clauses they may be used.

Table 12-6. *BIND Statements by Clause*

Statement	A	C	K	L	O	R	S	T	V	Z	X
`additional-from-auth`					x				x		
`additional-from-cache`					x				x		
`algorithm`			x								
`allow-notify`					x				x	x	
`allow-query`					x				x	x	
`allow-recursion`					x				x		
`allow-transfer`					x				x	x	
`allow-update`										x	
`allow-update-forwarding`					x				x	x	
`allow-v6-synthesis`											O
`also-notify`					x				x	x	
`alt-transfer-source`					x				x	x	
`alt-transfer-source-v6`					x				x	x	
`auth-nxdomain`					x				x		
`avoid-v4-udp-ports`					x						
`avoid-v6-udp-ports`					x						
`blackhole`					x						
`bogus`							x				
`cache-file`									x		
`category`				x							
`channel`				x							
`check-names`					x		x			x	
`cleaning-interval`					x				x		
`coresize`					x						
`database`										x	
`datasize`					x						
`deallocate-on-exit`											O
`delegation-only`										x	
`dialup`					x				x	x	
`directory`					x						

Continued

Key:

A = `acl` clause	**L** = `logging` clause	**S** = `server` clause	**Z** = `zone` clause
C = `controls` clause	**O** = `options` clause	**T** = `trusted-keys` clause	**X** = Obsolete (O) or not implemented (NI)
K = `keys` clause	**R** = `lwres` clause	**V** = `view` clause	

Table 12-6. *Continued*

Statement	A	C	K	L	O	R	S	T	V	Z	X
`disable-algorithms`					x				x		
`dnssec-enable`					x				x		
`dnssec-lookaside`					x				x		
`dnssec-must-be-secure`					x				x		
`dual-stack-servers`					x				x		
`dump-file`					x						
`edns`							x				
`edns-udp-size`					x				x		
`fake-iquery`					x						O
`fetch-glue`											O
`file`				x						x	
`files`					x				x		
`forward`					x				x	x	
`forwarders`					x				x	x	
`has-old-clients`					x						O
`heartbeat-interval`					x				x		
`host-statistics`					x						NI
`hostname`					x				x		
`inet`		x									
`interface-interval`					x						
`ixfr-from-differences`					x				x	x	
`ixfr-tmp-file`										x	O
`key-directory`					x				x	x	
`keys`							x				
`lame-ttl`					x				x		
`listen-on`					x	x					
`listen-on-v6`					x						
`maintain-ixfr-base`										x	O
`masters`										x	
`match-clients`									x		
`match-destinations`									x		
`match-mapped-addresses`					x						
`match-recursive-only`									x		

Key:

A = `acl` clause	**L** = `logging` clause	**S** = `server` clause	**Z** = `zone` clause
C = `controls` clause	**O** = `options` clause	**T** = `trusted-keys` clause	**X** = Obsolete (O) or not implemented (NI)
K = `keys` clause	**R** = `lwres` clause	**V** = `view` clause	

Statement	A	C	K	L	O	R	S	T	V	Z	X
max-cache-size					x				x		
max-cache-ttl					x				x		
max-ixfr-log-size					x		x			x	O
max-ncache-ttl					x				x		
max-ixfr-log-size					x						O
max-journal-size					x				x	x	
max-refresh-time					x				x	x	
max-retry-time					x				x	x	
max-transfer-idle-in					x				x	x	
max-transfer-idle-out					x				x	x	
max-transfer-time-in					x				x	x	
max-transfer-time-out					x				x	x	
memstatistics-file					x						
min-refresh-time					x				x	x	
min-retry-time					x				x	x	
min-roots					x				x		NI
minimal-responses					x				x		
multi-master					x				x	x	
multiple-cnames					x						O
named-xfer					x						O
ndots						x					
notify					x				x	x	
notify-source					x				x	x	
notify-source-v6					x				x	x	
pid-file					x						
port					x						
preferred-glue					x				x		
provide-ixfr					x		x	x			
pubkey											O
query-source					x				x		
query-source-v6					x				x		
querylog					x						
recursing-file					x						
random-device					x						
recursion					x				x		
recursive-clients					x						
request-ixfr					x		x	x			

Continued

Table 12-6. *Continued*

Statement	A	C	K	L	O	R	S	T	V	Z	X
`rfc2308-type1`					x			x			NI
`root-delegation-only`					x				x		
`rrset-order`					x				x		
`search`						x					
`secret`			x								
`serial-queries`					x						O
`serial-query-rate`					x						
`server-id`					x						
`sortlist`					x				x		
`stacksize`					x						
`statistics-file`					x						
`statistics-interval`					x						NI
`support-ixfr`							x				O
`suppress-initial-notify`					x		x				NI
`sig-validity-interval`					x				x	x	
`tcp-clients`					x						
`tcp-listen-queue`					x						
`tkey-dhkey`					x						
`tkey-domain`					x						
`tkey-gssapi-credential`					x						
`topology`											NI
`transfer-format`					x		x		x		
`transfer-source`					x				x	x	
`transfer-source-v6`					x				x	x	
`transfers`							x				
`transfers-in`					x						
`transfers-out`					x						
`transfers-per-ns`					x						
`treat-cr-as-space`					x						O
`type`										x	
`unix`		x									NI
`update-policy`										x	
`use-alt-transfer-source`					x				x	x	

Key:

A = `acl` clause	**L** = `logging` clause	**S** = `server` clause	**Z** = `zone` clause
C = `controls` clause	**O** = `options` clause	**T** = `trusted-keys` clause	**X** = Obsolete (O) or not implemented (NI)
K = `keys` clause	**R** = `lwres` clause	**V** = `view` clause	

Statement	A	C	K	L	O	R	S	T	V	Z	X
use-id-pool					x						O
use-ixfr					x						
view						x					
version					x						
zone-statistics					x				x	x	

BIND `controls` Statements

The `controls` clause permits only the `inet` statement type, though multiple such statements can appear inside the clause. A default `controls` clause is always assumed in the absence of any definition, which causes a TCP `listen` operation to be placed on port 953 of the loopback address for IPv4 and/or IPv6 (127.0.0.1 and ::1 respectively). If the `rndc` utility will not be used, the controls interface can be disabled by using an empty `controls` clause as shown here:

```
controls {};
```

`inet` Statement

The `inet` statement defines a method to control access to the `rndc` (remote administration) utility. More than one `inet` statement may be included in a `controls` clause.

`inet` Statement Syntax

```
inet inet_spec [inet_spec] ..;
inet * allow {192.168.254.2;} keys {"rndc-key";};
```

Each `inet_spec` parameter has the following format:

```
inet_spec = ( ip_addr | * ) [ port ip_port ] allow {  address_match_list  }
                keys {  key_list  };
```

The `ip_address` parameter defines the IP address of the local server interface on which `rndc` connections will be accepted. The wildcard value (*) will allow connection on any of the server's IPv4 addresses including the loopback address. The equivalent wildcard for IPv6 is `::`. The optional `ip_port` parameter allows a specific port to be nominated for use by `rndc` connections; if not present, the default port of 953 will be used. The `address_match_list` defines the permitted hosts that can connect to the `rndc` channel. The `key_list` parameter contains one or more `key-names` (defined in a `key` clause) containing the list of permitted users who are allowed access. While `address_match_lists` can include a `key` parameter, if one is present in the referenced `address_match_list`, it is ignored; only keys defined in the `key_list` of the `inet` statement are permitted access. The `key_list` can be omitted, in which case the file `rndc.key` in the same directory as `named.conf` that contains a default `key` clause with the name `"rndc-key"` (`rndckey` for Fedora) will be used to provide default access. The `rndc.key` file is created by running the following command:

```
rndc-confgen -a
```

The following example shows that a user on the loopback address can use the default key for access, while all other users must use the "rndc-remote" key. In all cases localhost will use port 953 (the default) and external connections will use port 7766. An acl clause is used as the source of the address_match_list:

```
// named.conf fragment
acl "rndc-users" {
     10.0.15.0/24;
     !10.0.16.1/24; // negated
     2001:db8:0:27::/64; // any address in subnet
 };
....
key "rndc-remote" {
     algorithm hmac-md5;
     secret "OmItW1lOyLVUEuvv+Fme+Q==";
};
controls {
     // local host - default key
     inet 127.0.0.1 allow {localhost;};
     inet * port 7766 allow {"rndc-users";} keys {"rndc-remote";};
};
```

Further examples of the inet statement are illustrated in the "rndc" section located in Chapter 9.

Caution For security reasons, the key clause earlier would normally be placed in a separate file, secured with read and write access only for the UID of BIND (the named daemon), which is typically either named or bind, and then included in the named.conf using an include statement.

BIND logging Statements

The logging clause takes two statements: the first defines the channel, one or more physical paths to the output stream, and the second defines the category or type of data that will be output to the channels. Multiple channel and category statements can exist in a logging clause. If no logging clause is defined in the named.conf file, then the following default definition is assumed:

```
logging {
     category default { default_syslog; default_debug; };
     category unmatched { null; };
};
```

The default means all categories (defined later in Table 12-9), with the exception of `queries` and `lame-servers`, will be written to `syslog` (`default_syslog`) and, if the debug level is nonzero, to a file called `named.run` (`default_debug`) in the location defined by a `directory` statement and that this file will grow to unlimited size unless manually deleted. The values in the preceding `logging` clause will only make complete sense after having read the `channel` and `category` descriptions that follow.

channel Statement

The `channel` statement is optional, and if not present, the four predefined `channel_name` values described later in Table 12-8 are always available. One or more `channel` statements define the output streams to which logging data will be written. `channel` statements can only be used in a `logging` clause.

channel Statement Syntax

```
channel channel_name { channel_spec };
channel secure_log {file "/var/log/named/dnssec.log" version 3 size 1m;
                   severity info;};
```

The `channel_name` is a unique name that is used to identify a channel definition and is used by the `category` statement as the destination for a particular type or category of log information. It is traditionally written as a nonspace string without quotes, but can be written as a quoted string. `channel_spec` defines the characteristics of the output stream and has the following format:

```
channel_spec = ( file "path-to-file"
                   [ versions ( number | unlimited ) ]
                   [ size size_in_bytes ]
                   | syslog syslog_facility | stderr | null );
              [ severity (critical | error | warning | notice |
                 info | debug [ level ] | dynamic ); ]
              [ print-category yes | no; ]
              [ print-severity yes | no; ]
              [ print-time yes | no; ]
              };
```

Table 12-7 describes the value of each parameter in the `channel_spec`.

Table 12-7. *Channel Statement Parameters*

Parameter	Values	Description
file	`path-to-file`	A quoted string defining the absolute or relative (to `directory` statement) path to the logging file, for instance, `/var/log/named/named.log`. From the preceding syntax, **file**, **syslog**, **stderr**, and **null** are mutually exclusive for a channel.
versions	`number|unlimited`	May take a `number` in the range `0` to `99` or `unlimited` (defaults to 99). This defines the number of file versions that should be kept by BIND. Versioned files are created by appending .0, .1, etc to the file name in the **file** parameter. Files are rolled (renamed or overwritten) so the base file name will contain the current log and .0 will contain the last log information prior to commencing the new log, .1 the next, and so on up to the limit defined by `number` or `unlimited`. Unless a **size** parameter is used, new log versions will only be rolled (or swapped) when BIND is restarted. If no **versions** statement is defined, a single log file of unlimited size is used and on restart new data is *appended* to the defined file. This can get to be a very big file, very quickly, and is not recommended.
size	`size_in_bytes`	Defines a size limit to the log file. May take the case-insensitive short forms `K`, `M`, or `G`, for example, `25m` = 25000000 (25 megabytes). **size** and **versions** are related as shown: **size** value and no **versions** parameter: When the size limit is reached, BIND will stop logging until the file size is reduced to below the threshold defined, that is, by manually deleting or truncating the file. **size** and a **versions** parameter: The log files will be rolled (renamed and overwritten as defined in the preceding **versions** description) when the size limit is reached. No **size**, only a **versions** parameter: The log files will be rolled (renamed and overwritten as defined in the **versions** description above) only when BIND is restarted.
syslog	`syslog_facility`	Uses `syslogd` to write output. The `syslog_facility` parameter is the facility definition to be used when writing to **syslog** and may take any valid value defined for **syslog** (see `man 3 syslog`) and its handing will be defined in `/etc/syslog.conf`. The default `syslog_facility` is `user`. When running under Windows, this setting will use the Event Log, Applications category. From the preceding syntax, **file**, **syslog**, **stderr**, and **null** are mutually exclusive for a channel.
stderr		Writes to the current standard error location (normally the console) and would typically only be used for debug purposes. From the preceding syntax, **file**, **syslog**, **stderr**, and **null** are mutually exclusive for a channel.
null		Writes to `/dev/null`—the bit bucket—such that all data is discarded. From the preceding syntax **file**, **syslog**, **stderr**, and **null** are mutually exclusive for a channel.
severity	`level`	Controls the logging level and may take one of the values defined in the preceding section "`channel` Statement Syntax." Logging will occur for any message equal to or higher than the level specified (=>); lower levels will not be logged. Various debug levels can be defined (see `-d` argument in Table 12-1 in the section "BIND Command Line" early in the chapter) and where level 0 is no debug information. The value `dynamic` means the value defined by either the `-d` command-line argument or by an `rndc trace debug_level` command.

Parameter	Values	Description
print-time	`yes|no`	Controls whether the date and time are written to the output channel (`yes`) or not (`no`). The default is `no`.
print-severity	`yes|no`	Controls whether the severity level is written to the output channel (`yes`) or not (`no`). The default is `no`.
print-category	`yes|no`	Controls whether the category value is written to the output channel (`yes`) or not (`no`). The default is `no`.

BIND provides four predefined `channel_name` definitions. If these are used in a category statement, they do not need to be defined using a channel statement—they just exist. Table 12-8 shows the predefined channels and their implicit definition.

Table 12-8. *Predefined Channels*

`channel_name`	Implicit Definition

`default_syslog`	`channel default_syslog {`
	`    syslog daemon;`
	`    severity info`
	`};`

`default_debug`	`channel default_debug {`
	`    file "named.run";`
	`    severity dynamic;`
	`};`

`default_stderr`	`channel default_stderr {`
	`    stderr;`
	`    severity info;`
	`};`

`null`	`channel null {`
	`   null;`
	`};`

If no `channel` statement is defined, the four predefined channels in Table 12-8 are available by default. The `default_debug` channel has the unique property that data is written to it *only* if the debug level (defined by a `category` statement, the `-d` command-line argument, or an `rndc trace debug_level`) is nonzero. This channel and the `default_syslog` channel are used in the default `logging` clause described at the beginning of this section.

`category` Statement

The `category` statement defines the type of log messages to be sent to a particular channel. More than one `category` statement may be included.

`category` Statement Syntax

```
category category_name { channel_name ; [ channel_name ; ... ] };
category dnssec {secure_log;};
```

The channel_name may refer to either one of the predefined channel_name values (default_syslog, default_debug, default_stderr, or null) or one defined in a channel statement. More than one channel_name may be defined for any given category statement, and in that case the category is written to all the defined channel_name values. The category_name parameter defines the type of output to be sent to the defined channel_name. This may take one of the values defined in Table 12-9.

Table 12-9. *Logging Category Types*

Value	Description
client	Logs processing of client requests.
config	Logs configuration file parsing and processing.
database	Logs messages relating to the databases used internally by the name server to store zone and cache data.
default	Logs all values that are not explicitly defined in category statements. If this is the only category defined, it will log all categories listed in this table with the exception of queries, which are not turned on by default and unmatched.
delegation-only	Logs queries that have returned NXDOMAIN as the result of a delegation-only zone type or a delegation-only statement in a hint or stub zone clause.
dispatch	Logs dispatches of incoming packets to the server modules where they are to be processed.
dnssec	Logs all DNSSEC, SIG(0), TKEY, and TSIG protocol processing.
general	Logs anything that is not classified in this table—catch-all category.
lame-servers	Logs all instances of lame servers (misconfiguration in the delegation of domains) discovered by BIND 9 when trying to obtain authoritative answers. If the volume of these messages is high, many users elect to send them to the null channel using, for instance, a category lame-servers {null;}; statement.
network	Logs all network operations.
notify	Logs all NOTIFY operations.
queries	Logs all query transactions. The querylog statement may be used to override this category statement. This entry can generate a substantial volume of data very quickly. This category is not turned on by default and hence the default type earlier will not log this information. This entry now logs whether a recursive query is requested (+ is recursive, - is iterative), whether it is EDNS0 (E), or whether it is signed (S).
resolver	Logs name resolution information including recursive lookups performed on behalf of clients by a caching name server.
security	Logs approval and denial of requests.
unmatched	Logs no matching view clause or unrecognized class value. A one-line summary is also logged to the client category. By default, this category is sent to the null channel.
update	Logs all DDNS transactions.
update-security	Logs approval and denial of update requests used with DDNS.
xfer-in	Logs details of zone transfers the server is receiving.
xfer-out	Logs details of zone transfers the server is sending.

The category statement is optional and if not present BIND will assume the following default:

```
category default { default_syslog; default_debug; };
```

This means that all categories except queries and unmatched will be logged to syslog (or Windows Event Manager, under the Applications category) *and* to the file named.run in the directory statement location (or its default), but only if the debug level is nonzero. The following example shows a simple logging clause using a single file:

```
// named.conf fragment
logging{
    channel single_log {
    file "/var/log/named/bind.log" versions 3 size 2m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    single_log;
    };
    category lame-servers{
    null;  // discard
    };
};
```

The following example shows streaming of NOTIFY and Dynamic DNS messages to separate log files. Assuming the view clause is being used, the unmatched category is also sent to stderr so the file can be quickly debugged.

```
// named.conf fragment
logging{
    channel main_log {
    file "/var/log/named/main.log" versions 3 size 2m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel notify_log {
    file "/var/log/named/notify.log" versions 3 size 1m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    channel ddns_log {
    file "/var/log/named/ddns.log" versions 3 size 1m;
    severity info;
    print-time yes;
    print-severity yes;
    print-category yes;
    };
    category default{
    main_log;
    };
    category lame-servers{
    null;  // discard
    };
    category notify{
    notify_log;
    };
    category update{
    ddns_log;
    };
    category update-security{
    ddns_log;
    };
    category unmatched {
    main_log; default_stderr;
    };
};
```

BIND Resolver Statements

This section describes the statements that may be included in the `lwres` (lightweight resolver) clause. The `listen-on` statement, which may also be included in the `lwres` clause, is described in the section "DNS BIND Operations" later in this chapter. If the `listen-on` statement is omitted in the `lwres` clause, it defaults to port 921 on localhost (127.0.0.1).

view

view "view-name";
view "good guys";

The `view` statement allows the resolver to use the characteristics defined by a `view` clause. If the statement is not present and no `view` clauses are defined, it uses a default (hard-coded) view within BIND. This statement can only appear in an `lwres` clause.

search

search {domain-name; [domain-name; ...]};
search {example.com; example.org;};

This statement has the same meaning as the equivalent named parameter in the `/etc/resolv.conf` file and defines the `domain-name` that will be added to any name supplied to the resolver. The `ndots` statement that follows can be used to control when this process is invoked. If more than one `domain-name` is present, they will be tried one after the other in the order they were defined. In the preceding example statement, if a name of `joe.example.net` was supplied and no `ndots` statement was present, the resolver will try `joe.example.net`, and if that fails, `joe.example.net.example.com`, and then `joe.example.net.example.org`. If the name `joe` was supplied, then the resolver would try first with `joe`, and if that fails, `joe.example.com`, and if that fails, `joe.example.org`. This statement can only appear in an `lwres` clause.

ndots

`ndots` `number;`
`ndots 2;`

This statement has the same meaning as the equivalent named parameter in the `/etc/resolv.conf` file and defines the minimum number of dots that must be present in a name before it used as an absolute name (it is assumed to be an FDQN). If there are fewer dots in the supplied name than `number`, each `domain-name` defined in a `search` statement will be added to the name. To illustrate the process, the `ndots 2;` from the example will be used together with the example defined in the `search` parameter earlier. If the resolver received the name `joe.example.net`, this has two dots in the name, and hence will be used in a query—only if this fails will the values defined in the `search` statement be appended to give `joe.example.net.example.com`, etc. If the name `joe` was supplied to the resolver, then it has no dots and hence will not be used directly in a query; but each value in the `search` statement will be appended to give `joe.example.com`, and if that fails, `joe.example.org`. This statement can only appear in an `lwres` clause.

BIND Transfer Statements

This section describes all the statements, in alphabetic order, that control or affect the behavior of zone transfers and Dynamic DNS updates.

allow-notify

`allow-notify` `{ address_match_list };`
`allow-notify { 10.2.3.2;10.2.3.7;192.168.2.0/24;};`

`allow-notify` applies to slave zones only and defines an `address_list_match` for hosts that are allowed to send `NOTIFY` messages to this slave in addition to those hosts defined in the `masters` statement for the zone. The default behavior is to allow zone `NOTIFY` messages only from the hosts defined in the `masters` statement. This statement may be defined in `zone` or `view` clauses or in a global `options` clause. Example:

```
// named.conf fragment
....
zone "example.com" in{
    type slave;
    masters {192.168.254.2;};
    file "slave.example.com";
    // allows NOTIFY message from the defined IPs
    allow-notify (192.168.0.15; 192.168.0.16; 10.0.0.1;);
};
zone "example.net" in{
    type slave;
    file "slave.example.net";
    masters {192.168.254.3;};
    // allows no NOTIFY messages
    allow-notify (none;);
};
```

The zone `example.com` can receive `NOTIFY` messages from 192.168.254.2 and the listed IPs; `example.net` can only accept `NOTIFY` messages from 192.168.254.3.

allow-transfer

`allow-transfer` `{ address_match_list };`
`allow-transfer { 192.168.2.7;};`

`allow-transfer` defines an `address_match_list` of hosts that are allowed to transfer the zone information *from* the server, master or slave, for the zone. The default behavior is to allow zone transfers to any host, which means that any host anywhere in the world can copy your zone file. While this may look excessively friendly, the assumption is that all zone data is public. If this is not the required behavior, it must be disabled explicitly as shown in the following example fragment. This statement may be specified in `zone` or `view` clauses or in a global `options` clause. The example shows zone transfers disabled for all zones by default, but the zone `example.com` has decided to allow transfers to any host for reasons best known to the domain owner.

```
options {
....
    allow-transfer {none;}; // none by default
....
};
....
zone "example.com" in{
....
    allow-transfer {any;}; // this zone only
....
};
```

allow-update

`allow-update` `{ address_match_list };`
`allow-update { !172.22.0.0/16;};`

`allow-update` defines an `address_match_list` of hosts that are allowed to submit dynamic updates for master zones, and thus this statement enables Dynamic DNS. The default in BIND 9 is to disallow updates from all hosts, that is, DDNS is *disabled* by default. This statement may be specified in zone clauses only. This statement is mutually exclusive with update-policy and applies to master zones only. The example shows DDNS for three zones: the first disables DDNS explicitly, the second uses an IP-based list, and the third references a key clause. The allow-update in the first zone clause could have been omitted since it is the default behavior. Many people like to be cautious in case the default mode changes.

```
// named.conf fragment
// key clause is shown only for illustration and would
// normally be included in the named.conf file
key "update-key" {
    ....
};
zone "example.net" in{
    type master;
    allow-update {none;}; // no DDNS by default
    ....
};
....
zone "example.com" in{
....type master;
    allow-update {10.0.1.2;}; // DDNS this host only
    ....
};
zone "example.org" in{
    type master;
    allow-update {keys "update-key";};
    ....
};
```

In the example.org zone, the reference to the key clause "update-key" implies that the application that performs the update, say nsupdate, is using TSIG and must also have the same shared secret *with the same key-name*. This process is described in Chapter 10.

allow-update-forwarding

```
allow-update-forwarding { address_match_list };
allow-update-forwarding { none;};
```

allow-update-forwarding defines an address_match_list of hosts that are allowed to submit dynamic updates to a slave server for onward transmission to a master. By default, this behavior is not allowed, that is, “none” is assumed as an address_match_list. This backdoor route to DDNS should be used with extreme caution, since if the allow-update on the master enables the zone slave to perform a DDNS update, this statement could expose the master to indirect attack. This statement applies to slave zones only and may be specified in zone or view clauses or in a global options clause.

also-notify

also-notify { ip_addr [**port** ip_port] ; [ip_addr [**port** ip_port] ; ...] };
also-notify { 10.0.3.7 port 1177;};

also-notify is applicable to master zones only and defines a list of IP address(es) and optional port numbers that will be sent a NOTIFY message when a zone changes, or a specific zone changes if the statement is specified in a zone clause. Any IP addresses are in addition to those listed in the NS RRs for the zone that will also be sent NOTIFY messages. The also-notify in a zone is *not* cumulative with any global also-notify statements. In addition, if a global notify no; statement is defined, this option may be used to override it for a specific zone, and conversely if the global options clause contains an also-notify list, setting notify no; in the zone will override the global option. This statement may be specified in a zone or view clause or in a global options clause.

```
options {
....
    also-notify {10.1.0.15; 172.28.32.7;}; // all zones
....
};
....
zone "example.com" in{
....
    also-notify {10.0.1.2;}; // only this host + those in NS RRs for zone
....
};
zone "example.net in{
....
    notify no; // no NOTIFY for zone
....
};
```

alt-transfer-source, alt-transfer-source-v6

alt-transfer-source (ipv4_address | *) [port (integer | *)];
alt-transfer-source-v6 (ipv6_address | *) [port (integer | *)];
alt-transfer-source 172.22.3.15; // assumed multihomed
alt-transfer-source-v6 2001:db8::2; // assumed multihomed

alt-transfer-source and alt-transfer-source-v6 apply to slave zones only. They define an alternative local IP address (on this server) to be used for inbound zone transfers by the server if that defined by transfer-source (transfer-source-v6) fails and use-alt-transfer-source is enabled. This address (and port) must also appear in the remote end's allow-transfer statement for the zone being transferred. This statement may be specified in zone or view clauses or in a global options clause.

ixfr-from-differences

```
ixfr-from-differences (yes | no);
ixfr-from-differences yes;
```

`ixfr-from-differences` defines how the name server calculates incremental zone changes. Normally, incremental zone transfers are only possible when used in conjunction with DDNS. `ixfr-from-differences` allows a zone master or slave to create incremental zone transfers for nondynamic zones. If set to `yes`, when the server receives (if a slave) or loads (if a master) a new version of a zone file, it will compare the new version to the previous one and calculate a set of differences. The differences are then logged in the zone's journal file (`.jnl` appended to zone file name) such that the changes can be transmitted to downstream slaves as an incremental zone transfer. This statement saves bandwidth at the expense of increased CPU and memory consumption. This statement may be used in a `zone`, `view`, or global `options` clause.

max-journal-size

```
max-journal-size size_in_bytes;
max-journal-size 50k;
```

`max-journal-size` sets a maximum size in bytes (may take the case-insensitive `K`, `M`, or `G` short forms) for each journal file. When the journal file approaches the specified size, some of the oldest transactions in the journal will be automatically removed. The default is unlimited size. Journal files are used by DDNS when modifying the Primary master zone file and when receiving `IXFR` changes on slave zones. The journal file uses a binary format, and its name is formed by appending the extension `.jnl` to the name of the corresponding zone file.

All changes made to a zone using dynamic update are written to the zone's journal file. The server will periodically flush the complete contents of the updated zone to its zone file; this happens approximately every 15 minutes. When a server is restarted after a shutdown or crash, it will replay the journal file to incorporate into the zone any updates that took place after the last zone file update.

If changes have to be made manually to a dynamic zone, then use the following sequence:

1. Disable dynamic updates to the zone using `rndc freeze zone`, which causes the zone file to be updated.

2. Edit the zone file.

3. Delete the `.jnl` file for the zone.

4. Run `rndc thaw (unfreeze) zone` to reload the changed zone and reenable dynamic updates. The current versions of BIND (9.3+) use the command `rndc thaw zone`; older versions use `rndc unfreeze zone`.

This statement may be used in a `zone`, `view`, or global `options` clause.

max-refresh-time, min-refresh-time

```
max-refresh-time seconds ;
min-refresh-time seconds ;
max-refresh-time 2w;
min-refresh-time 12h ;
```

`max-refresh-time` and `min-refresh-time` are only valid for slave or stub zones. The refresh time is normally defined by the SOA RR refresh parameter (defined in seconds). These statements allow the slave server administrator to override the definition and substitute the values defined, which are in seconds. The values may take the normal time shortcuts, for example, `35m` or `2d55m`. These statements may be specified in `zone` or `view` clauses or in a global `options` clause.

max-retry-time, min-retry-time

```
max-retry-time seconds ;
min-retry-time seconds ;
max-retry-time 3600 ;
min-retry-time 1800 ;
```

`max-retry-time` and `min-retry-time` are only valid for slave or stub zones. The retry time is normally defined by the SOA RR retry parameter. These statements allow the slave server administrator to override the definition and substitute the values defined. The values may take the normal time shortcuts, for example, `35m` or `2d55m`. These statements may be specified in `zone` or `view` clauses or in a global `options` clause.

max-transfer-idle-in

```
max-transfer-idle-in minutes ;
max-transfer-idle-in 10 ;
```

`max-transfer-idle-in` is only valid for slave zones. Inbound zone transfers making no progress in this many minutes will be terminated. The default is `60` (1 hour). The maximum value is `40320` (28 days). This statement may be specified in `zone` or `view` clauses or in a global `options` clause.

max-transfer-idle-out

```
max-transfer-idle-out minutes ;
max-transfer-idle-out 20;
```

`max-transfer-idle-out` is only valid for master zones. Outbound zone transfers running longer than this many minutes will be terminated. The default is `120` (2 hours). The maximum value is `40320` (28 days). This statement may be specified in `zone` or `view` clauses or in a global `options` clause.

max-transfer-time-in

```
max-transfer-time-in minutes ;
max-transfer-time-in 120;
```

`max-transfer-time-in` is only valid for slave zones. Inbound zone transfers running longer than this many minutes will be terminated. The default is `120` (2 hours). The maximum value is `40320` (28 days). This statement may be specified in `zone` or `view` clauses or in a global `options` clause.

max-transfer-time-out

```
max-transfer-time-out minutes ;
max-transfer-time-out 120;
```

`max-transfer-time-out` is only valid for master zones. Outbound zone transfers running longer than this many minutes will be terminated. The default is `120` (2 hours). The maximum value is `40320` (28 days). This statement may be specified in `zone` or `view` clauses or in a global `options` clause.

multi-master

```
multi-master ( yes | no) ;
multi-master yes ;
```

`multi-master` is relevant only when multiple masters are defined for a slave zone. It controls whether a log entry will be generated each time the serial number is *less* than that currently maintained by the slave (`no`) or not (`yes`). This situation can occur when the zone masters are out of sync with each other. The default is `no`. This statement may be specified in `zone` or `view` clauses or in a global `options` clause.

notify

```
notify ( yes | no | explicit );
notify explicit;
```

`notify` behavior is only applicable to master zones. If set to `yes` (BIND default), when zone information changes, `NOTIFY` messages are sent from zone masters to the slaves defined in the NS RRs for the zone (with the exception of the Primary master name server defined in the SOA RR) and to any IPs listed in `also-notify` options. If set to `no`, `NOTIFY` messages are not sent to any name server. If set to `explicit`, `NOTIFY` is only sent to those IP(s) listed in an `also-notify` statement.

If a global `notify` option is `no`, an `also-notify` statement may be used to override it for a specific zone, and conversely if the global `options` contain an `also-notify` list, setting `notify` to `no` in the zone will override the global option. This statement may be specified in `zone` or `view` clauses or in a global `options` clause. The following example illustrates that the zone `example.net` will *not* send `NOTIFY` messages to the name servers defined in its NS RRs but only those defined in the global `also-notify` statement.

```
options {
....
    also-notify {10.1.0.15; 172.28.32.7;}; // all zones
....
};
....
zone "example.com in{
....
    // NS RRs and global  also-notify
    // default behavior so could have been omitted
    notify yes;
....
};
zone "example.net in{
....
    // no NOTIFY to NS RRs
    // NOTIFY to global also-notify IPs
    notify explicit;
....
};
```

notify-source, notify-source-v6

`notify-source` `(ip4_addr | *) [`**`port`**` ip_port] ;`
`notify-source-v6` `(ip6_addr | *) [port ip_port] ;`

```
notify-source 192.168.254.3 ;
notify-source-v6 2001:db8:0:1::3 port 1178;
```

`notify-source` and `notify-source-v6` are only valid for master zones. `notify-source` defines the IP address and optionally UDP port to be used for *outgoing* `NOTIFY` messages. The value * means the IP of this server (default). This IP address must appear in the `masters` or `allow-notify` statement of the receiving slave name servers. Since neither the `masters` nor `allow-notify` statements take a port parameter, if an optional UDP port value other than `53` is used, a `transfer-source`, `transfer-source-v6`, `listen-on`, or `listen-on-v6` statement would be required on the slave. These statements are typically only used on a multihomed server and may be specified in `zone` or `view` clauses or in a global `options` clause. The example that follows shows an IPv6 address being used to send `NOTIFY` messages to a Global Unicast address.

```
options {
....
    notify-source-v6 {2001:db8:0:1::3;}; // all zones
....
};
```

provide-ixfr

provide-ixfr (yes| no) ;
provide-ixfr no ;

provide-ixfr only applies to master zones. The provide-ixfr option controls whether a master will respond to an incremental zone transfer request (IXFR)—parameter = yes—or will respond with a full zone transfer (AXFR)—parameter = no. The default is yes. This statement may be specified in server or view clauses or in a global options clause.

request-ixfr

request-ixfr (yes| no) ;
request-ixfr no;

request-ixfr applies only to slave zones. The request-ixfr option defines whether a server will request an incremental zone transfer (IXFR)—parameter = yes—or will request a full zone transfer (AXFR)—parameter = no. The default is yes. This statement may be specified in server or view clauses or in a global options clause.

serial-query-rate

serial-query-rate number;
serial-query-rate 5;

serial-query-rate applies to slave zones only and limits the number of simultaneous SOA queries to the number per second. The default is 20. This statement may only be used in a global options clause.

transfer-format

transfer-format (one-answer | many-answers);
transfer-format one-answer;

transfer-format is only used by master zones. This controls the format the server uses to transfer zones: one-answer places a single record in each message, and many-answers packs as many records as possible into a maximum-sized TCP message. The default is many-answers, which is only known to be supported by BIND, and if transferring zones to others servers, a transfer-format one-answer; statement *may* be required. This statement may be specified in server, zone, or view clauses or in a global options clause.

transfer-source, transfer-source-v6

transfer-source (ip4_addr | *) [**port** ip_port] ;
transfer-source-v6 (ip6_addr | *) [**port** ip_port] ;

```
transfer-source 172.15.2.3 port 1178;
transfer-source-v6 2001:db8::1;
```

transfer-source and transfer-source-v6 are only valid for slave zones on multihomed hosts (hosts with more than one IP address or interface). transfer-source defines which local IP address (on this server) will be bound to TCP connections used to fetch zones transferred inbound by this server. These statements also determine the source IP address, and optionally the UDP port, used for refresh queries and forwarded dynamic updates. If not set, it defaults to a value that will usually be the address of the interface “closest to” the remote end—generally the IP address on which the request arrived. This address must appear in the remote end’s `allow-transfer` option for the zone being transferred. These statements may be used in `zone` or `view` clauses or in a global `options` clause. The following example shows a multihomed server with IP addresses of 192.168.254.2 and 192.168.254.4 on which traffic normally arrives on 192.168.254.2:

```
// named.conf fragment
zone "example.com" in {
    type slave;
    ...
    // force transfers onto one interface
    transfer-source 192.168.254.4;
};
```

The master server for the zone must permit the transfer as shown here:

```
// named.conf fragment
zone "example.com" in {
    type master;
    ...
    // permit transfer
    allow-transfer 192.168.254.4;
};
```

transfers-in

`transfers-in` `number ;`
`transfers-in 5 ;`

`transfers-in` is only used by slave zones. This statement defines the number of concurrent inbound zone transfers. The default is `10`. This option may only be used in a global `options` clause.

transfers-per-ns

`transfers-per-ns` `number`
`transfers-per-ns 5`

`transfers-per-ns` is only used by slave zones. This statement determines the number of concurrent inbound zone transfers from any remote name server. The default is `2`. This option may only be specified in a global `options` clause.

transfers-out

```
transfers-out number ;
transfers-out 20 ;
```

`transfers-out` is only used by master zones. `transfers-out` defines the number of concurrent outbound zone transfers. The default is `10`. Zone transfer requests in excess of this limit will be refused. This option may only be specified in a global `options` clause.

update-policy

```
update-policy { update_policy_rule; [...] };
update-policy { grant fred.example.net name example.net MX;};
```

`update-policy` only applies to master zones. This statement defines the rules by which DDNS updates may be carried out. This statement may only be used with a key (TSIG or SIG(0)) and may be specified only in a `zone` statement. It is mutually exclusive with `allow-update`. `update_policy_rule` takes the following format:

```
permission identity matchtype tname [rr]
```

Table 12-10 describes the various fields used in the `update_policy_rule`.

Table 12-10. *Update Policy Rules*

Parameter	Description
`permission`	May be either `grant` or `deny`.
`identity`	A FQDN (ends with a dot) that refers to a RR in the zone file. This will typically be a KEY RR. Can also take a DNS wildcard value (*).
`matchtype`	Can take any of the following values: `name`, which matches the `tname` field exactly (that is, if `tname` is `joe.example.com.`, then it can only update the record `joe.example.com.`); `subdomain`, which matches anything containing the `tname` field (that is, if the `tname` is `example.com.`, it will match `bill.example.com` and `sheila.example.com` etc.); `self`, in which case the record being updated matches the `identity` field exactly—in this instance, `identity` will typically be set to the wildcard (*); `wildcard`, which indicates that the record being updated can be a valid DNS RR wildcard expansion.
`name`	An FQDN (ends with a dot) of the target or part of the target record (depending on the value of `matchtype`). Can take a DNS wildcard value (*).
`[rr]`	Optional. Defines the Resource Record types that may be updated including ANY (all RR types except NSEC). If omitted, the default allows all RR types except RRSIG, NSEC, SOA, and NS. Multiple entries may be defined using space-separated entries, for instance, `A MX PTR`.

The following example shows the use of `update-policy` whereby each host can update its own A RR but no others:

```
zone "example.com" in {
    type master;
     ....
    update-policy { grant * self * A;};
};
```

The policy says that any KEY RR (the first *) with the same name (`self`) as the A RR it is trying to update (the second *) will be allowed to do so (`grant`). Further examples of `update-policy` are described in "Securing Dynamic Updates" located in Chapter 10, including the necessary zone file entries.

use-alt-transfer-source

```
use-alt-transfer-source ( yes | no );
use-alt-transfer-source yes;
```

`use-alt-transfer-source` specifies whether the `alt-transfer-source` statements are allowed (`yes`) or not (`no`). The statement is typically defined in a `zone` clause to control specific behavior over a globally defined `alt-transfer-source` statement. If `view` clauses are used, this statement defaults to `no`; otherwise it defaults to `yes` (for BIND 8 compatibility). This statement may be specified in normal `zone` or `view` clauses or in a global `options` clause.

DNS BIND Operations

This section describes the statements that affect operation of the server.

avoid-v4-udp-ports, avoid-v6-udp-ports

```
avoid-v4-udp-ports { port; ... };
avoid-v6-udp-ports { port; ... };
avoid-v4-udp-ports { 1178; 1183;1188 };
avoid-v6-udp-ports { 7734; };
```

`avoid-v4-udp-ports` and `avoid-v6-udp-ports` define a list of port numbers that will *not* be used by BIND when initiating queries or zone transfers. This list may be used to avoid ports that are blocked by a firewall. This option can only be defined in the global `options` clause.

check-names

```
check-names ( master | slave | response ) ( fail | warn | ignore );
check-names response warn;
```

The `check-names` statement will check any host (owner) name in A, AAAA, or MX RRs and the domain names in SOA, NS, MX, and PTR RRs for the defined type (`master`, `slave`, or `response`) for compliance with RFC 821, 952, and 1123 and result in the defined action (`fail`, `warn`, or `ignore`). Care should be taken when using this statement because RFC 2181 greatly liberalized the rules for names (see the section "Resource Record Common Format" located in Chapter 13 for full details). The type of host name to be checked may be `master`, in which case the check only applies to master zones, `slave` applies only to slave zones, and `response` applies to names that arrive in response to a query from this server. The default is *not* to perform host name checks. `check-names` may be used in a `view` or `options` clause with the preceding syntax and in a `zone` clause, where it has a *different* syntax, shown here:

```
check-names ( fail | warn | ignore );
check-names warn;
```

cleaning-interval

```
cleaning-interval minutes;
cleaning-interval 12h;
```

`cleaning-interval` defines the time in minutes when all expired records will be deleted from the cache. The default is `60` (1 hour); if specified as `0`, *no* cleaning will be performed. The maximum value is `40320` (28 days). This statement does not affect the TTL interval but merely controls the size the cache may occupy on disk. This statement may be used in a `view` or global `options` clause.

coresize

```
coresize size_in_bytes;
coresize 2m;
```

The maximum size in bytes (may take the case-insensitive short forms `K`, `M`, or `G`) of a core dump if BIND crashes. This statement can only be used in a global `options` clause.

database

```
database "driver-name [param] [param] ..";
database "mysql param1 param2";
```

`database` defines information to be supplied to a database driver including using the Simple Database API. The data is enclosed in a quoted string and `driver-name` defines the name of the driver defined by the `dns_sdb_register()` function call (see Chapter 14). The optional `param` field may be any number of space-separated values that are passed as arguments (via `argc/argv`) to the included driver's `create()` callback to be interpreted in a way specific to the driver. This statement can only be used in a `zone` clause.

datasize

```
datasize size_in_bytes;
datasize 250m;
```

`datasize` specifies the maximum size in bytes (may take the case-insensitive short forms `K` or `M`) of memory used by the server. This is a hard limit and may stop the server from working. The statements `max-cache-size` and `recursive-clients` may also be used to limit memory usage. This statement can only be used in a global `options` clause.

dialup

```
dialup dialup_options;
dialup passive;
```

`dialup` optimizes behavior to minimize use of connect time on dial-up links. The default is `no`. This option can be defined in the `view`, `zone`, and `options` clauses.

The `dialup` statement's behavior concentrates activity into the `heartbeat-interval` and triggers `NOTIFY` and zone refresh operations based on the value of the `dialup_option` as defined in Table 12-11.

Table 12-11. *Dialup Statement Parameters*

`dialup_option`	**Normal Refresh**	**Heartbeat Refresh**	**Heartbeat Notify**
`no`	Yes	No	No
`yes`	No	Yes	Yes
`notify`	Yes	No	Yes
`refresh`	No	Yes	No
`passive`	No	No	No
`notify-passive`	No	No	Yes

directory

```
directory "path_name";
directory "/usr/local/var";
```

`directory` is a quoted string defining an absolute path, for instance, `/var/named`. All subsequent relative paths use this base directory. If no `directory` statement is specified, the directory from which the `named.conf` file was loaded is used (defined by `–sysconfdir` when BIND is configured). This option may only be used in a global `options` clause.

dual-stack-server

```
dual-stack-servers [ port pg_num ] { ( "host" [port p_num] |
              ipv4 [port p_num] | ipv6 [port p_num] ); ... };
dual-stack-servers port 1177 {192.168.2.3; "bill.example.net"};
```

`dual-stack-server` defines the IP address of one or more dual-stacked (IPv4/IPv6) servers that can be used by this server to resolve a query using a stack it does not support. In the preceding example, if only an AAAA (IPv6) RR is returned to a query, then this server, which is assumed to support only IPv4, can use the defined server or servers to resolve the query, since they support both stacks. On dual-stack servers, it is only effective if one of the stacks has been disabled on the command line. Using `pg_num` will act as a global port number for all subsequent server definitions, or they can be defined individually with the `p_num` field. The `host` field is a quoted string and is the FQDN of the host, which must be resolvable using the default protocol that is IPv4 in the preceding example. The `ipv4` and `ipv6` fields are the explicit IPv4 or IPv6 addresses that may be used as an alternative to the `host` format. This statement may be used in a `view` or global `options` clause.

dump-file

```
dump-file path_name;
dump-file "/var/cache/bind.cache";
```

`dump-file` is a quoted string defining the absolute path where BIND dumps the database (cache) in response to a `rndc dumpdb` (see Chapter 10). If not specified, the default is `named_dump.db` in the location specified by a `directory` option. This statement may only be used in a global `options` clause.

edns-udp-size

```
edns-udp-size size_in_bytes ;
edns-udp-size 1460;
```

`edns-udp-size` defines the `size_in_bytes` that the server will advertise for an EDNS UDP buffer. Valid values are `512` to `4096`; values outside this range will be silently adjusted. The default value is `4096`. EDNS is normally only used with DNSSEC transactions. This statement may be used in a `view` or global `options` clause.

files

```
files max_files ;
files 200 ;
```

`files` specifies the maximum number of files the server may have open concurrently. The default is `unlimited`. This statement may be used in a `view` or global `options` clause.

heartbeat-interval

```
heartbeat-interval minutes;
heartbeat-interval 2h;
```

`heartbeat-interval` defines the time in minutes when zones marked as `dialup` are updated. The default is `60` (1 hour); if specified as `0`, no updating will be performed. The maximum value is `40320` (28 days). This statement may be used in a `view` or `global` options clause.

hostname

```
hostname ( "host-name" | none );
hostname "myhost";
```

The `host-name` (a quoted string) the server should report when it receives a query of the name `hostname.bind` with type `TXT` and class CHAOS (CH). This defaults to the name found by `gethostname()` (the current host's name). While it may appear this statement is not relevant for normal non-CHAOS systems, however, using `dig` this information may be easily discovered; so if such information is sensitive, specifying `none` disables processing of the queries. This statement may be used in a `view` or global `options` clause.

interface-interval

```
interface-interval minutes;
interface-interval 0;
```

`interface-interval` controls the time in minutes when BIND scans alls interfaces on the server and will begin to listen on new interfaces (assuming they are not prevented by a `listen-on` option) and stops listening on interfaces that no longer exist. This statement is only required in a dynamic environment where IP addresses may be changing. The default is `60` (1 hour); if specified as `0`, no interface scan will be performed. The maximum value is `40320` (28 days). This option may only be specified in a global `options` statement.

lame-ttl

```
lame-ttl seconds;
lame-ttl 15m;
```

`lame-ttl` defines the number of seconds to cache lame delegations or lame servers, that is, servers that are defined as authoritative (appear in an NS RR) but do not respond as authoritative. The value `0` disables such caching and is *not* recommended. The default is `600` (10 minutes) and the maximum value is `1800` (30 minutes). This statement may be used in a `view` or `global` options clause.

listen-on

```
listen-on [ port ip_port ] { address_match_list };
listen-on { 192.168.254.2; };
```

`listen-on` defines the optional port and IP address(es) on which BIND will listen for incoming queries. The default is port 53 on all server interfaces. Multiple `listen-on` statements are allowed. This option may be used in a global `options` clause and an `lwres` clause, where if omitted it defaults to port 921 on localhost (127.0.0.1).

listen-on-v6

```
listen-on-v6 [ port ip_port ] { address_match_list };
listen-on-v6  port 1234 { any; };
```

`listen-on-v6` turns on BIND to listen for IPv6 queries. If this statement is not specified, the server will not listen for any IPv6 traffic—the default behavior. If the OS supports RFC 3493– and RFC 3542–compliant IPv6 sockets and the `address_match_list` uses the special `any` name, then a single `listen` is issued to the wildcard address. If the OS does not support this feature, a socket is opened for every required address and port. The port default is `53`. Multiple `listen-on-v6` statements are allowed. This option may only be used in a global `options` clause. The following examples show a number of definitions:

```
options {
....
    // turns on IPv6 for port 53
    listen-on-v6 {any;};
};
options {
....
    // turns off IPv6
    listen-on-v6 {none;};
};

options {
....
    // turns on IPv6 for port 53 for 16 IP range
    listen-on-v6 {2001:db8::/124;};
};
```

match-mapped-addresses

```
match-mapped-addresses ( yes | no ) ;
match-mapped-addresses yes  ;
```

If `yes`, `match-mapped-addresses` indicates that an `address_match_list` containing an IPv4 address will be checked against an IPv4-mapped IPv6 address (described in Chapter 5). This feature can incur significant CPU overheads and should be used as a workaround only where the OS software accepts such connections. This statement may only be used in a global `options` clause.

max-cache-size

```
max-cache-size size_in_bytes;
max-cache-size 50m;
```

`max-cache-size` defines the maximum amount of memory in bytes to use for the server's cache (case-insensitive short forms of `K`, `M`, or `G` are allowed). When the amount of data in the cache reaches this limit, the server will cause records to expire prematurely so that the limit is not exceeded. In a server with multiple views, the limit applies separately to the cache of each view. The default is `unlimited`, meaning that records are purged from the cache only when their TTLs expire. This statement may be used in a `view` or global `options` clause.

max-cache-ttl

```
max-cache-ttl seconds;
max-cache-ttl 3d2h5m;
```

`max-cache-ttl` sets the maximum time (in seconds) for which the server will cache positive answers and may be used to override (reduce) the actual TTL values on received RRs. Negative answer caching—`NXDOMAIN`—is defined by `max-ncache-ttl`. The default is one week (`604800` seconds). Standard BIND time short forms may be used. This statement may be used in a `view`

max-ncache-ttl

```
max-ncache-ttl seconds
max-cache-ttl 3h;
```

max-ncache-ttl sets the maximum time (in seconds) for which the server will cache negative (NXDOMAIN) answers (positive answers are defined by max-cache-ttl). The default max-ncache-ttl is 10800 (3 hours). max-ncache-ttl cannot exceed 7 days and will be silently truncated to 7 days if set to a greater value. This statement may be used in a view or global options clause.

memstatistics-file

```
memstatistics-file "file-name";
memstatistics-file "/var/stats/named/bind.mem";
```

memstatistics-file defines the file-name (a quoted string) to which BIND memory usage statistics will be written when it exits. This may be an absolute or relative (to directory) path. If the parameter is not present, the stats are written to named.memstats in the path defined by directory or its default. This statement may only be used in a global options clause.

pid-file

```
pid-file "path_name" ;
pid-file "bind.pid";
```

pid-file is a quoted string and defines where the Process Identifier used by BIND is written. It may be defined using an absolute path or path relative to the directory statement. If not present, it is distribution or OS specific, typically /var/run/named/named.pid. The appropriate permissions may be required to allow this file to be written. This option can only be defined in the global options clause.

port

```
port ip_port ;
port 1137;
```

ip_port defines on which port BIND will provide UDP and TCP services. The default is 53. This option is intended primarily for testing, and setting it to a nonstandard value will not allow the server to communicate with normal DNS systems. It can also be used in stealth configuration between and internal and external name servers to further disguise traffic that passes through a firewall (see Chapter 7). The option can only appear in the global options clause and must come before any other option that defines ports or IP addresses.

preferred-glue

```
preferred-glue A | AAAA;
preferrred-glue AAAA;
```

preferred-glue defines the order of preference in which glue records will be listed in the additional section of the response (see Chapter 15). If no order is specified, they will be listed in the order they appear in the zone file. This statement may be used in a view or global options clause.

querylog

```
querylog ( yes | no ) ;
querylog yes;
```

querylog may override the setting of the category statement of the logging clause and controls whether query logging should be started when named (BIND) starts. If querylog is not specified, then query logging is controlled by the rndc querylog command or the logging category queries. This statement may only be used in a global options clause.

recursing-file

```
recursing-file "file-name";
recursing-file "bind.stats";
```

recursing-file defines the file-name to which data will be written when the command rndc recursing is issued. May be an absolute or relative (to directory) path. If the parameter is not present, the information is written to the file named.recursing in the path defined by directory or its default. This statement may only be used in a global options clause.

server-id

```
server-id ( "id-string" | none |;
server-id "123";
```

server-id specifies the ID the server will return in response to a query for ID.SERVER with type TXT, under class CHAOS (CH). Specifying none disables processing of the queries; otherwise it will return id-string. The default is none. This statement may only be used in a global options clause.

stacksize

```
stacksize size_in_bytes;
stacksize 20k;
```

stacksize defines the maximum size in bytes (may take the case-insensitive short forms K, M, or G) of the stack memory used by the server. The default is no limit on stacksize. This statement may only be used in a global options clause.

statistics-file

```
statistics-file "file-name";
statistics-file "/var/stats/names/bind.stats";
```

statistics-file defines the file-name to which data will be written when the command rndc stats is issued. This may be an absolute or relative (to directory) path. If the parameter is not present, the information is written to the file named.stats in the path defined by directory or its default. This statement may only be used in a global options clause.

tcp-clients

```
tcp-clients number ;
tcp-clients 77;
```

By default, DNS uses UDP port 53 for queries, but allows both TCP and UDP. The `tcp-clients` statement allows the user to define the maximum number of TCP connections that may be supported. The BIND 9 default is 100. The option can only appear in the global `options` clause.

tcp-listen-queue

```
tcp-listen-queue number;
tcp-listen-queue 7;
```

`tcp-listen-queue` defines how many TCP `listen` operations are queued for incoming zone transfers. The default and minimum is `3`, and any value lower than this will be silently raised to `3`. Depending on OS features, this also controls how many TCP connections will be queued in kernel space waiting for some data before being passed to TCP `accept`. This statement may only be used in a global `options` clause.

version

```
version version_string ;
version "No Way";
```

The `version` statement defines the text that will be returned to a `version.bind` query for the CHAOS (CH) class only. The default is for BIND to return its real version number. This information, however, is easily discovered using the `dig` utility, so by adding `version_string` and a quoted string such as `"get lost"`, it may be possible to avoid exploitation of known weaknesses of specific software versions. This option can only be defined in the global `options` clause.

zone-statistics

```
zone-statistics ( yes | no ) ;
zone-statistics no;
```

`zone-statistics` defines whether zone statistics will be maintained. The default is `no`. The zone statistics may be accessed using `rndc stats`. This statement may be used in a `view`, `zone`, or global `options` clause.

DNS BIND Query Statements

This section describes all the statements available that relate to or control queries.

additional-from-auth, additional-from-cache

```
additional-from-auth ( yes | no ) ;
additional-from-cache ( yes | no ) ;
additional-from-auth yes ;
additional-from-cache no ;
```

`additional-from-auth` and `additional-from-cache` control the behavior when zones have additional (out-of-zone, sometimes called *out-of-bailiwick*) data or when following CNAME (or experimental DNAME) RRs. These options are used when configuring authoritative-only (noncaching) servers and are only effective if `recursion no` is specified in a global `options` or `view` clause. The default for both statements is `yes`. The statements may be defined in a `view` or global `options` clause. The behavior is defined by Table 12-12.

Table 12-12. `additional-from` *Statement Behavior*

auth	cache	BIND Behavior
yes	yes	BIND will follow out-of-zone records; for example, it will follow the MX record specifying `mail.example.net` in zone `example.com` for which it is authoritative (master or slave). This is the default behavior.
no	no	Cache disabled. BIND will *not* follow out-of-zone records even if it is in the cache; that is, it will not follow the MX record specifying `mail.example.net` for zone `example.com` for which it is authoritative (master or slave). It will return `REFUSED` for the out-of-zone record.
yes	no	Cache disabled. BIND will follow out-of-zone records, but since this requires the cache (which is disabled), the net result is the same—BIND will return `REFUSED` for the out-of-zone record.
no	yes	BIND will not follow out-of-zone records, but if they are available in the cache, they will be returned, else it will return `REFUSED` for the out-of-zone record.

allow-query

```
allow-query { address_match_list };
allow-query {!10.0.3.2/24;};
```

`allow-query` defines an `address_match_list` of hosts that are allowed to issue queries to this server. If not specified, all hosts are allowed to make queries. This statement may be used in a `view`, `zone`, or global `options` clause.

allow-recursion

```
allow-recursion { address_match_list };
allow-recursion { 192.168.2.3; !192.168.2.7; };
```

`allow-recursion` defines an `address_match_list` of hosts that are allowed to issue recursive queries to this server. If the answer to the query already exists in the cache, it will be returned irrespective of this statement. If not specified, all hosts are allowed to make recursive queries. This statement may be used in a `view` or global `options` clause.

auth-nxdomain

```
auth-nxdomain ( yes | no );
auth-nxdomain yes;]
```

If `auth-nxdomain` is `yes`, it allows the server to answer authoritatively (the AA bit is set) on returning `NXDOMAIN` (domain does not exist) answers. If `no` (the default), the server will not answer authoritatively. The current setting reverses the BIND 8 default. This statement may only be used in a global `options` clause.

blackhole

```
blackhole { address_match_list };
blackhole { none; };
```

`blackhole` defines an `address_match_list` of hosts that the server will *not* respond to nor answer queries for. This statement has the same effect as a series of `server` clauses with a `bogus yes;` statement but is significantly shorter! The default is `none` (all hosts are responded to). This statement may only be used in a global `options` clause.

delegation-only

```
delegation-only ( yes | no ) ;
delegation no;
```

`delegation-only` applies to hint and stub zones only, and if set to `yes`, indicates the zone will only respond with delegations (or referrals). (See the `type` statement for more information.) The default is `no`. This statement may only be used in a global `zone` clause.

forward

```
forward ( only | first );
forward only;
```

`forward` is only relevant in conjunction with a valid `forwarders` statement. If set to `only`, the server will only forward queries; if set to `first` (the default), it will send the queries to the forwarder (defined by the `forwarders` statement); and if not answered, it will issue queries directly. This statement may be used in a `zone`, `view`, or global `zone` clause.

forwarders

```
forwarders { ip_addr [port ip_port] ; [ ip_addr [port ip_port] ; ... ] };
forwarders { 10.2.3.4; 192.168.2.5;};
```

`forwarders` defines a list of IP address(es) (and optional port numbers) to which queries will be forwarded. It is only relevant if used with the `forward` statement. This statement may be used in a `zone`, `view`, or global `zone` clause. See also "Forwarding (a.k.a. Proxy, Client, Remote) DNS Server" in Chapter 7.

minimal-responses

```
minimal-responses ( yes | no ) ;
minimal-responses yes ;
```

If minimal-responses is set to yes, the server will only add records to the authority and additional data sections (see Chapter 15) when they are required by the protocol, specifically delegations and negative responses. Since the effect of this is to reduce the data volumes sent, it may improve the performance of the server. The BIND default is no. This statement may be used in a view or global zone clause.

query-source, query-source-v6

```
query-source [ address ( ip_addr | * ) ] [ port ( ip_port | * ) ];
query-source address 192.168.2.3 ;
query-source-v6 [ address ( ip_addr | * ) ] [ port ( ip_port | * ) ];
query-source-v6 address  * port 1188;
```

query-source and query-source-v6 define the IP address (IPv4 or IPv6) and optional port to be used as the source for *outgoing* queries from the server and are normally relevant only on multihomed servers (servers with multiple IP addresses or interfaces). The BIND default is any server interface IP address and a random unprivileged port. The optional port is only used to control UDP operations. avoid-v4-udp-ports and avoid-v6-udp-ports can be used to prevent selection of certain ports. This statement may be used in a view or global options clause.

recursion

```
recursion ( yes | no );
recursion no;
```

If recursion is set to yes (the default), the server will always provide *recursive query* behavior if requested by the client (resolver). If recursion is set to no, the server will only provide *iterative query* behavior. If the answer to the query already exists in the cache, it will be returned irrespective of the value of this statement. This statement essentially controls caching behavior in the server. The allow-recursion statement and the view clause provide fine-grained control over recursion services. This statement may be used in a view or global options clause.

recursive-clients

```
recursive-clients number;
recursive-clients 20;
```

Defines the number of simultaneous recursive lookups the server will perform on behalf of its clients. The default is 1000, that is, it will support 1000 simultaneous recursive lookup requests, which should be enough for most purposes! This statement may only be used in a global options clause.

root-delegation-only

```
root-delegation-only [ exclude { "domain_name"; ... } ];
root-delegation-only exclude { "com"; "net" };
```

If present, `root-delegation-only` indicates that all responses will be referrals or delegations. The optional exclude list consists of one or more `domain_name` (a quoted string) parameters. This statement is intended to be used for root and TLD domains (gTLDs and ccTLDs), but the `delegation-only` statement may be used to create the same effect for specific zones. This statement may be used in a `view` or global `options` clause.

rrset-order

```
rrset-order { order_spec ; [ order_spec ; ... ]
rrset-order { type A order cyclic; };
```

`rrset-order` defines the order in which RRsets—multiple records of the same type—are returned. This statement applies to any RR type in which the records are similar (their name, class, and type are the same). `rrset-order` is fully implemented from BIND 9.2.3. The default is defined to be a random-cyclic order, that is, the starting address is randomly chosen and thereafter cyclic order is followed, but experimentation shows the default to be purely cyclic. The `rrset-order` defines the order in which similar RRs are returned from the name server. The `sortlist` statement controls the order in which the RRs are returned to a client, for instance, a resolver. An `order_spec` is defined as follows:

```
[ class class_name ][ type type_name ][ name "domain_name"]
      order ordering
```

where `class_name` is the record class, for instance, IN (default is `any`); `type_name` is the RR type (defaults to `any`); and `domain_name` limits the statement to a specific domain suffix and defaults to root (all domains). `ordering` may take one of the following values: `fixed`—records are returned in the order they are defined in the zone file; `random`—records are returned in a random order; `cyclic`—records are returned in a round-robin fashion. Only one such statement may appear in any clause—the last defined will be used in the case of multiple statements. This statement may be used in a `view` or global `options` clause.

The following example shows that MX RRs for `example.com` only will be returned in `random` order; all others responses will use the default `cyclic` order.

```
rrset-order { type MX name "example.com" order random; order cyclic;};
```

sortlist

The `sortlist` statement is used to order *RRsets* (groups of RRs whose name, class, and type values are the same) for use by a resolver (a client). It is the client-side equivalent of the `rrset-order` statement and can work *against* the `rrset-order` statement when being used as part of a load-balancing configuration: `rrset-order` carefully delivers RRsets in its order of preference to a remote server that may then proceed to reorder them with a `sortlist` statement when responding to its client resolver. The `sortlist` statement attempts to order returned records based on the IP address of the client that initiated the request.

sortlist Statement Syntax

```
sortlist { address_match_list };
sortlist { {10.2/16; } ;};
```

The `address_match_list` is used very differently from the way it is used in all other statements and assumes that each element of the `address_match_list` is itself an `address_match_list`, that is, it is a nested `address_match_list` and is enclosed in braces. Processing depends on whether there is one or more than one element in the nested `address_match_list`. In the simple case of one element, as in the preceding example, if the client's IP address matches 10.2/16 (that is, lies in the range 10.2.0.0 to 10.2.255.255) and there are any IP addresses in the response in the same range, they will be the first records supplied in the response. Any remaining records will be sorted according to the `rrset-order` (default is cyclic). If no match is found, the records are returned in the order defined by the `rrset-order` or its default value (`cyclic`). If two elements are provided in the `address_match_list`, then the second element is assumed to be an ordered list of preferences. This is best illustrated by an example. Assume the zone `example.com` has a zone file with multiple A RRs for `lots.example.com`:

```
// zone file example.com
$ORIGIN example.com.
lots     IN  A  192.168.3.6
         IN  A  192.168.4.5
         IN  A  192.168.5.5
         IN  A  10.2.4.5
         IN  A  172.17.4.5
```

The client-side server has a `sortlist` statement, as shown here:

```
options {
    ....
    sortlist {
     192.168.4/24;  // 1st client IP selection
     {10.2/16;   // 1st preference
      172.17.4/24;  // 2nd preference
     };
     192.168.5/24;  // 2nd client IP selection
     {192.168.4/24;   // 1st preference
      172.18/24;  // 2nd preference
      10.2/16;  // 3rd preference
     };
    };
};
```

If the client, say a resolver, with an IP address of 192.168.5.33 issues an A query for `lots.example.com`, then the RRs will be returned in the following order:

```
192.168.4.5
10.2.4.5
192.168.3.6
192.168.5.5
172.17.4.5
```

The preceding order is computed using the following process: The top level of the `address_match_list` is searched against the client IP (192.168.5.33) address and matches the IP address in the `sortlist` statement with a comment of "2nd client IP selection"; the nested `address_match_list` is then treated as an ordered list for the A query result IPs (not the client IPs). The IP address in the `sortlist` statement with a comment of "1st preference" matches, so 192.168.4.5 becomes first in the returned list. The IP address in the `sortlist` statement with a comment of "2nd preference" does not match any of the returned IPs. The IP address in the `sortlist` statement with a comment of "3rd preference" matches, so 10.2.4.5 becomes second in the returned list. The remaining three RRs do not match, so they are returned according to the `rrset-order` statement or its default (`cyclic`) if not defined. The `sortlist` statement may be used in a `view` or global `options` clause.

DNS BIND Security Statements

This section describes all the statements that relate to or control security.

algorithm

`algorithm` `algorithm-name;`
`algorithm hmac-md5;`

The `algorithm` statement defines the shared secret algorithm being used and may only take the value `hmac-md5`. The algorithm statement is only used in a `key` clause.

disable-algorithms

`disable-algorithms` `domain {alg; [alg;]};`
`disable-algorithms example.net {hmac-md5; rsamd5;};`

The `disable-algorithms` statement may be used to disable specific cryptographic algorithms used with the defined `domain`. The `alg` field (one or more is allowed in each statement) may take the case-insensitive values `hmac-md5`, `rsamd5`, `rsasha1`, `dsa`, or `dh`. Multiple `disable-algorithm` statements may appear in a global `options` or `view` clause.

dnssec-enable

`dnssec-enable` `(yes | no);`
`dnssec-enable yes;`

BIND does not automatically enable secure DNS (DNSSEC) operations. The `dnssec-enable` statement is used to enable or disable (the default) any security feature. `dnssec-enable yes;` must be used if any secure (cryptographic) operation is being performed such as TSIG, TKEY, SIG(0), or DNSSEC.bis. Any name server with `dnssec-enable yes;` advertises its ability to support secure operations by include an OPT meta-RR (or pseudo-RR) in the additional section of any query and implicitly enables EDNS0 (RFC 2671) features. This statement may be used in a `view` or global `options` clause.

dnssec-lookaside

```
dnssec-lookaside domain trust-anchor dlv-domain
dnssec-lookaside .com trust-anchor dlv.verisignlabs.com;
```

The `dnssec-lookaside` statement is used with the experimental DNSSEC Lookaside Validation service and provides an alternative method for verifying a chain of trust using experimental DLV RRs. The objective of the DLV is to provide equivalent capabilities to a signed TLD zone *without* the registry operator having to sign the TLD zone. Any secure zone that lies at or below `domain` and that does not have a local `trusted-keys` clause may interrogate the `dlv-domain` to search for a DLV RR (which is similar in every respect to a DS RR). To verify the `dlv-domain`, a trusted anchor must be present (in a `trusted-keys` clause) for this `dlv-domain`. A pilot DLV service is currently being run by VeriSign, Inc. and the trusted anchor is available from the VeriSign DLV Registry Pilot experimental web site (`https://www.dlv.verisignlabs.com/trusted.html`). The `dlv-domain` may also be specified using the `-l` option to the `dnssec-signzone` command (see Chapter 9) to generate DLV RRs when the zone is signed. In the preceding example statement, any `.com` domain that does not have a configured trusted anchor will interrogate the domain `dlv.verisignlabs.com`. This statement may be used in a `view` or global `options` clause. The DLV system is explained in Chapter 11.

dnssec-must-be-secure

```
dnssec-must-be-secure domain (yes | no);
dnssec-must-be-secure example.com yes;
```

The `dnssec-must-be-secure` statement indicates whether `domain` must be secure or not. If the `yes` option is defined, then `domain` must be signed and must have a trusted anchor (in a local `trusted-keys` clause) or a verifiable chain of trust (through a DS RR at the parent), or `dnssec-lookaside` must be active at or above `domain`. The default is `no`. This statement may be used in a `view` or global `options` clause.

key-directory

```
key-directory "path_name";
key-directory "/var/named/keys";
```

`key-directory` is a quoted string defining the absolute path where the private keys used in the dynamic update (DDNS) of secure (signed) zones may be found. It is only required if this directory is different from that defined by a `directory` statement. This statement may be used in a `zone`, `view`, or global `options` statement.

random-device

```
random-device "path_to_device";
random-device "/dev/random";
```

`random-device` defines a source of randomness (or entropy) within the system and defaults to `/dev/random`. This device is needed for DNSSEC operations such as TKEY transactions and dynamic update of signed zones. Operations requiring entropy will fail when the specified source has been exhausted. The `random-device` option takes effect during the initial configuration load at server startup time and is ignored on subsequent reloads. This statement may only be used in a global `options` clause.

secret

```
secret key-data;
secret BLAHBLAHBLAH;
```

The `secret` statement can only appear in a `key` clause. The `key-data` field contains base64-encoded (RFC 3548) data, frequently referred to as *keying material*, which constitutes the *shared secret*. It is typically produced by the `dnssec-keygen` or `rndc-confgen` utilities (see Chapter 9). Chapter 10 shows how this statement is constructed from the `.private` file created when the `dnssec-keygen` utility is run. This statement contains *extremely sensitive* data, and for that reason the `secret` statement and its enclosing `key` clause is normally placed in a separate file (which has limited read and write permission), and the `include` statement is used to embed it into the `named.conf` file at run time.

sig-validity-interval

```
sig-validity-interval days ;
sig-validity-interval 30 ;
```

`sig-validity-interval` specifies the number of days into the future when DNSSEC signatures (using RRSIG RRs) automatically generated as a result of dynamic updates to signed zones will expire. The default is `30`. The maximum value is `3660` (10 years). The signature inception time is unconditionally set to one hour before Universal Coordinated Time (UTC) to allow for a limited amount of clock skew. All DNSSEC operations rely on a correct time zone value and network clock synchronization using `ntpd`. This statement may be used in a `zone`, `view`, or global `options` statement.

tkey-dhkey

```
tkey-dhkey "host-name" key-tag;
tkey-dhkey "fred.example.com" 45312;
```

The `tkey-dhkey` statement defines the file containing the Diffie-Hellman private key to be used in TKEY operations and must be located in the directory defined using a `directory` statement. In the preceding example, the key would be generated using the command shown here:

```
# dnssec-keygen -a dh -b 1024 -n host fred.example.com
Kfred.example.com.+002+45312
```

The `dnssec-keygen` utility outputs a single line identifying the files containing information—`Kfred.example.com.+002+4531` in the preceding example. The value `K` is a fixed identifier; `fred.example.com.` is the name of the host KEY RR reflected from the `dnssec-keygen` arguments (see Chapter 9); the number `002` indicates the Diffie-Hellman algorithm, and the number `45312` in the preceding example is known as the `key-tag` or *fingerprint* and is algorithmically generated to uniquely identify this key when the `dnssec-keygen` utility is run. The TKEY feature is not widely used. This statement may only be used in a global `options` clause.

tkey-domain

```
tkey-domain domain-name;
tkey-domain "example.com";
```

The `tkey-domain` statement defines the domain name that will be added to the names of all keys generated by a TKEY sequence. When a name server requests a TKEY exchange, it can optionally indicate the required name for the key. If present, the name of the shared key will be the client's supplied name with the `domain-name` appended to it; thus, if the client supplied a name of `fred` in the preceding example, the name server will return `fred.example.com`. If the client does not supply a name, a random series of hex digits will be used as the client part of the name. TKEY is not widely implemented. This statement can only be used in a global `options` clause.

tkey-gssapi-credential

```
tkey-gssapi-credential "credential";
```

`tkey-gssapi-credential` defines the credential associated with a Generic Security Services API (GSSAPI). The GSSAPI and the credentials required are defined by RFC 2743 and its Kerberos form is defined in RFC 1964. TKEY is not widely implemented at this time since the standards for TKEY mandate the preexistence of a shared secret (TSIG) or a public key (SIG(0)) to authenticate the initial exchange, which somewhat defeats the object of the Diffie-Hellman exchange used by TKEY. This statement may only appear in a global `options` clause and is not documented in the current (BIND 9.3.0) options list, but is still present in the source code.

DNS BIND `server` Statements

This section describes statements that may only be used in the `server` clause. The `server` clause can take more statements, and you should consult Table 12-6 for a complete list.

bogus

```
bogus ( yes | no );
bogus ( yes | no );
```

`bogus` indicates that traffic from this server should be ignored (`yes`), for instance, if known to be giving bad data, suffering a DoS attack, or some other reason. The same effect may be obtained using the `blackhole` statement. The default is `no`. This option can only be defined in the `server` clause.

edns

```
edns ( yes | no ) ;
edns no  ;
```

`edns` defines whether to use EDNS0 (RFC 2671) with a specific server (`yes`) or not (`no`). The default is `yes`. This statement may only be used in a `server` clause.

keys

```
keys "key-name";["key-name"; ...;];
keys "serv1-zone-transfer-key";
```

The `key-name` field of the `keys` statement references a `key` clause with the same `key-name` and mandates that transactions secured by TSIG (zone transfer or dynamic update) will use this key. In the case of zone transfers, the peer host must have an equivalent `key` clause with the same `key-name.` When used with `nsupdate`, `key-name` appears in the `-k` argument (see Chapter 9). This statement can only appear in a `server` clause and while the formal syntax allows for more than one `key-name` as of the current releases of BIND, only one `key-name` is supported per server. The section "Securing Zone Transfers" located in Chapter 10 shows the use of the `keys` statement in a `server` clause.

transfers

```
transfers number ;
transfers 5;
```

`transfers` limits the number of concurrent zone transfers from any given server. If not present, the default for `transfers-per-ns` is used (the default is `2`). This option may be used only in a `server` clause.

DNS BIND view Statements

This section describes statements that may only be used in the `view` clause. The `view` clause can take many more statements, and you should consult Table 12-6 for a complete list.

match-clients

```
match-clients { address_match_element; ... };
match-clients { 10.2.3.0/8;172.16.30.0/16;!192.168.0.0/16; };
```

A `view` clause matches when either or both of its `match-clients` and `match-destinations` statements match and when the `match-recursive-only` condition is met. If either `match-clients` or `match-destinations` or both are missing, they default to `any` (all hosts match). The `match-clients` statement defines the `address_match_list` for the source IP address of the incoming messages. Any IP address that matches will use the defined `view` clause. This statement may only be used in a `view` clause. An example showing the use of all three statements is described in the section "BIND `view` Clause" located earlier in this chapter.

match-destinations

```
match-destinations { address_match_element; ... };
match-destinations { 192.168.0.3; };
```

The match-destination statement defines the address_match_list for the destination address of the *incoming* message. It is one of three statements that can be used to match a view clause. The relationship between the statements is described under match-clients. This statement may only be used in a view clause.

match-recursive-only

```
match-recursive-only (yes | no);
match-recursive-only yes;
```

If an incoming query requests recursion and match-recursive-only is yes, then the condition is met. It is one of three statements that can be used to match a view clause. The relationship between the statements is described under the match-clients entry. This statement may only be used in a view clause.

DNS BIND zone Statements

This section describes the zone-only statements. The zone clause can take many more statements than described here, and you should consult Table 12-6 for a complete list.

check-names

```
check-names (warn|fail|ignore) ;
check-names fail;
```

The check-names statement may also appear in a view or global options clause where its syntax is different. The behavior controlled by this statement, which allows certain names to be limited to compliance with the name format defined in RFCs 821, 952, and 1123, is described under check-names in the "DNS BIND Operations" section located earlier in this chapter.

file

```
file "file_name";
file "slave.example.com";
```

file defines the file used by the zone in quoted string format, for example, "slave.example.com"—or whatever convention you use. The file entry is mandatory for master and hint and optional—but highly recommended—for slave and not required for forward zones. The file may be an absolute path or relative to the directory statement. The following example shows the use of the file statement:

```
// named.conf fragment
zone "example.com" in {
    type slave;
    // defines an optional file used to save slave zone data
    file "slave.example.com";
    ...
};
zone "example.net" in {
    type master;
    // defines a master zone file
    file "master.example.net";
    ....
};
```

masters

```
masters [ port pg_num ] { ( masters_list | ipv4
             [port p_num] | ipv6 [port p_num ] ) [ key "key-name" ]; ... };
masters {192.168.3.5;};
```

The `masters` statement is valid only with slave zones and defines one or more IP addresses and optional port numbers of servers that hold the master zone file. The slave will use the defined IP address(es) to update the zone file when the SOA RR `refresh` parameter is reached. The `pg_num` parameter changes the port number used for zone transfers for all the listed servers (the default is port 53). The `p_num` parameter changes the port number for the specific IP address only. `masters_list` may be used to reference a list of masters defined in a `masters` clause. The `key-name` field defines the key to be used to authenticate the zone transfers when using TSIG and references the name of the `key` clause; a corresponding `key` clause with the same `key-name` must be present in the master server for the zone. The following example shows three masters for the zone, one of which will use port 1127 for zone transfers and one of which is an IPv6 address:

```
// named.conf fragment
zone "example.com" in {
    type slave;
    file "slave.example.com";
    masters {192.168.2.7; 10.2.3.15 port 1127; 2001:db8:0:1::15;};
};
```

type

```
type zone_type;
type delegation-only;
```

The `type` statement defines the characteristics of the zone and may take one of the values defined in Table 12-13.

Table 12-13. *Type Statement Values*

Value	Description
`master`	The server has a master copy of the zone data (which is loaded from a local filestore) and provides authoritative answers for the zone.
`slave`	A slave zone is a replica of the master zone and obtains its zone data by zone transfer operations. The slave will respond authoritatively for the zone as long as it has valid (not timed out) zone data. The `masters` statement specifies one or more IP addresses of master servers that the slave contacts to refresh or update its copy of the zone data. When the TTL specified by the `refresh` parameter of the zone's SOA RR is reached or a `NOTIFY` message is received, the slave will query the SOA RR from the zone master. If the `sn` parameter (serial number) is greater than the current value, a zone transfer is initiated. If the slave cannot obtain a new copy of the zone data when the SOA `expiry` value is reached, then it will stop responding for the zone. Authentication of the master can also be done with per-server TSIG keys (see the entry for the `masters` statement earlier). By default, zone transfers are made using TCP on port 53, but this can be changed using the `masters` statement. If a `file` statement is defined, then the zone data will be written to this file whenever the zone is changed and reloaded from this file on a server restart. If no file statement is defined, then the slave will require a zone transfer from the zone master before it can start responding to queries for the zone.
`forward`	A zone of type `forward` is simply a way to configure forwarding, perhaps to a unique name server, on a per-domain or per-zone basis. To be effective, both a `forward` and `forwarders` statement should be included. If no `forwarders` statement is present or an empty list is provided, then no forwarding will be done for the domain, canceling the effects of any `forwarders` in the global `options` clause.
`hint`	The initial set of root-servers is defined using a hint zone. When the server starts up, it uses the hints zone file to find a root name server and get the most recent list of root name servers. If no hint zone is specified for class IN, the server uses a compiled-in default set of root servers. Classes other than IN have no built-in default hints. The hint zone is only required for a name server that provides recursive services.
`stub`	A stub zone is similar to a slave zone except that it replicates only the NS records of a master zone instead of the entire zone. Stub zones are not a standard part of the DNS—they are a feature specific to the BIND implementation and should not in general be used.
`delegation-only`	This indicates only referrals (or delegations) will be made for the zone and is recommended only for use with TLDs, *not* leaf (non-TLD) zones. The generation of referrals in leaf zones is determined by the use of the `delegation-only` statement and the RRs contained in the zone file, that is, a zone consisting of an SOA RR, NS RRs, and glue records will *only* be able to generate referrals (see also Chapter 9).

Summary

This chapter is a reference for the command-line options used when BIND is loaded and for all the entities used in a `named.conf` file—the file that controls the detailed behavior of BIND.

The `named.conf` file statements were defined to be of three types—comments, clauses, or statements. This book rigorously uses the term *clause* to refer to a collection or group of *statements* in the interest of clarity and consistency. Much BIND documentation uses a variety of terms such as *sections*, *clauses*, *statements*, *options*, and *phrases* to define the two entity types (apart from comments) contained in the `named.conf` file. Advanced readers may well be comfortable with different terms being applied to the same type of entity or, even worse (but depressingly frequent), the same term being applied to completely different entities. Such an environment, however, is neither edifying nor conducive to creating safe, error-free BIND configurations—the ultimate objective of this book. The terms were selected after consulting *Merriam-Webster OnLine* and BIND's source code.

The available clauses are listed alphabetically in Table 12-3. Statements are listed alphabetically in Table 12-5, together with very short descriptions and categorization. The individual statements are then described in detail in alphabetic order within each category, with a simple example in every case and some more complex examples where appropriate. It is hoped that such categorization will allow you to dip into the specific section required and also allow browsing of statements when looking to control or affect the behavior of similar types of operations, for instance, queries. Many statements can be used in more than one clause, and Table 12-6 lists each statement alphabetically and the clauses in which it can be used.

The next chapter contains reference material on zone files and the directives and Resource Records (RRs) that may be used in them.

CHAPTER 13

Zone File Reference

This chapter is intended to be a reference for zone file directives and Resource Records. Table 13-1 later in this chapter contains a list of all current RRs defined by IANA (`www.iana.org/assignments/dns-parameters`), their support status within BIND and Windows DNS software, the RFCs that define them, and a very brief description of the RR type. This provides you with a quick overview of the formidable list of RRs available and will enable you to browse them more effectively. This chapter features descriptions of the syntax for each directive and Resource Record, and in most cases their use is illustrated with one or more examples.

RRs have two representations: a textual form, in which they appear in a zone file as described in this chapter, and a binary format, also called the *wire format*, used when one or more RRs are transmitted in a query, query response, or similar network operation. The binary format of RRs is defined in Chapter 15. The following section reviews the zone file format rules and is then followed by material on the zone file directives and finally the Resource Records descriptions.

DNS Zone File Structure

Zone files describe a domain's characteristics, the hosts and services supported, in a form that may be used by DNS software. The files are textual and may be read or edited using any standard text editor. They can contain three types of entries:

1. *Comments*: All comments start with `;` (semicolon) and continue to the end of the line. Comments can occupy a single line or be added to any of the following record types.

2. *Directives*: All directives start with `$` and are used to control processing of the zone files.

3. *Resource Records*: RRs are used to define the characteristics, properties, or entities contained within the domain or zone. RRs are contained on a single line with the exception that entries enclosed in parentheses can spread across multiple lines.

The following is a zone file fragment that illustrates the preceding points and record types:

```
; this is a full-line comment
$TTL 12h    ; directive - comment terminates the line
$ORIGIN example.com.
; Start of Authority (SOA) record defining the zone (domain)
; illustrates an RR record spread over more than one line
; using the enclosing parentheses
```

```
@  IN  SOA  ns1.example.com. hostmaster.example.com. (
              2003080800 ; se = serial number
              3h         ; ref = refresh
              15m        ; ret = update retry
              3w         ; ex = expiry
              3h      ; min = minimum
              )
; single line RR
    IN  NS  ns1.example.com. ;with a comment
...
```

The preceding SOA RR could have been written on a single line, in which case there is no need for the parentheses:

```
@  IN  SOA  ns1.example.com. hostmaster.example.com. 2003080800 3h 15m 3w 3h
```

If parentheses are used, the `(` (open parenthesis) must appear on the first line.

DNS Directives

Zone file directives control the processing of zone files. There are three standardized directives: `$TTL`, `$ORIGIN`, `$INCLUDE` (RFC 1035). A fourth directive, `$GENERATE`, is supported by BIND but is not standardized.

The `$ORIGIN` Directive

The `$ORIGIN` directive was standardized in RFC 1035 and defines the domain name that will be *appended* to any name that appears in an RR and does not end with a dot—frequently called a *relative* or an *unqualified* name—to create a Fully Qualified Domain Name (FQDN). This process is called the "$ORIGIN substitution rule" throughout this book.

The `$ORIGIN` Substitution Rule

If a name appears in a Resource Record and does *not* end with a dot, then the value of the last, or only, `$ORIGIN` value will be appended to the name. If the name does end with a dot, then it is a Fully Qualified Domain Name, and nothing will be appended to the name. The terminating dot in an FQDN is interpreted as the root of the domain tree or hierarchy. An FQDN unambiguously defines a name to the root.

`$ORIGIN` Syntax

```
$ORIGIN domain-name
```

`domain-name` is always an FQDN—it always ends with a dot. `$ORIGIN` directives can appear anywhere in a zone file and will be used from the point they are defined onwards until replaced with another `$ORIGIN`.

```
$ORIGIN example.com.
; unqualified names from here will append example.com.
www             IN  A 192.168.2.2 ; unqualified
; www expands to www.example.com.
...
ftp.example.com. IN A 192.168.2.3 ; FQDN
...
$ORIGIN us.example.com.
; unqualified names from here will append us.example.com.
www             IN  A 192.168.254.2 ; unqualified
; www expands to www.us.example.com.
...
```

The `$ORIGIN` directive is not mandatory. If an `$ORIGIN` directive is not present, BIND will assume that the `$ORIGIN` value is the name of the `zone` clause that defines the zone file in `named.conf` (described in Chapter 12). This book always uses `$ORIGIN` directives in zone files for three reasons:

1. With the `$ORIGIN` directive present, a zone file is self-descriptive and self-contained—it requires no reference to any external information.
2. The `$ORIGIN` substitution rule (defined previously) is much less confusing. The value to be substituted is immediately apparent—the last `$ORIGIN` directive.
3. Not all software may use the same default assumptions about the `$ORIGIN` directive as does BIND. Zone files are more portable when the `$ORIGIN` directive is included.

■Tip For a further insight into the use of the `$ORIGIN` directive, have a look at a zone file on a slave server after the zone file has been transferred. There you will see that BIND constructs its zone files with an `$ORIGIN` directive at every level of the hierarchy.

The `$INCLUDE` Directive

The `$INCLUDE` directive allows inclusion in situ of an external file containing additional directives or RRs. It is typically used in maintenance of larger zone files; that is, individual parts of a single zone file can be modified by clients without exposing the global parameters or other client parts to either inspection or corruption. Alternatively, it can be used to add RRs to a zone file that were created externally such as KEY or DNSKEY RRs generated by the `dnssec-keygen` utility for use in secure DNS operations. Unlike the `include` statement used in the `named.conf` file, which is typically used to secure sensitive (private) keys, there is no corresponding need for the `$INCLUDE` in the zone file—any keys appearing in a zone file will always be public. This directive is standardized in RFC 1035. The RFC is silent on the topic of embedded `$INCLUDE` directives in the included files, and to err on the side of safety they should not be used.

$INCLUDE Syntax

```
$INCLUDE filename [domain-name]
```

The `filename` parameter may be an absolute path (for example, `/path/to/file`) or a relative path (for example, `relative/path/to/file`). If the relative path format is used, then the base directory is assumed to be the same location as the zone file. The optional `domain-name` parameter may be used to set an explicit `$ORIGIN` to be used in the included file; however, an included file can also contain one or more `$ORIGIN` directives as shown in the fragments that follow. The scope of `$ORIGIN` directives when used with an included file is limited to the included file only. On termination of the include operation, the value of `$ORIGIN` is restored to the value before the `$INCLUDE` directive.

The first zone file fragment shows an included file with no `$ORIGIN` directives, which will use the current `$ORIGIN` directive in operation at the point of inclusion:

```
$ORIGIN us.example.com.
...
mail       IN      A   192.168.35.12
; expands to mail.us.example.com.
$INCLUDE /var/named/zones/sub.example.com ; absolute path no $ORIGIN
ftp        IN      A   192.168.35.16
; expands to ftp.us.example.com.
```

The following fragment shows expansion of the `/var/named/zones/sub.example.com` `include` file:

```
; INCLUDE file statements
www       IN       A  192.168.23.15
; expands to www.us.example.com
...
; end of included file
```

The following fragments show the use of an explicit `$ORIGIN` on the `$INCLUDE` directive:

```
$ORIGIN us.example.com.
...
mail       IN      A   192.168.35.15
; expands to mail.us.example.com.
$INCLUDE sub.example.com uk.example.com. ; overrides current $ORIGIN
; $ORIGIN reverts to value before the $INCLUDE directive
ftp        IN      A   192.168.35.16
; expands to ftp.us.example.com
```

The included fragment in `sub.example.com` uses the explicit `$ORIGIN` on the `$INCLUDE` directive.

```
; INCLUDE file statements
www       IN       A  192.168.23.15
; expands to www.uk.example.com
...
; end of included file
```

The following fragments achieve the same result as the previous ones but may be less confusing because of the explicit use of an `$ORIGIN` directive in the included file:

```
$ORIGIN us.example.com.
...
mail       IN      A   192.168.35.15
; expands to mail.us.example.com.
$INCLUDE sub.example.com ; no $ORIGIN
; $ORIGIN reverts to value before the $INCLUDE directive
ftp        IN      A   192.168.35.16
; expands to ftp.us.example.com
```

The included fragment uses an explicit `$ORIGIN` directive:

```
; INCLUDE file statements
$ORIGIN uk.example.com.
www       IN       A  192.168.23.15
; expands to www.uk.example.com
...
; end of included file
```

The preceding fragment is self-contained and self-descriptive.

The $TTL Directive

Every Resource Record may take an *optional* Time to Live (TTL) value specified in seconds. The `$TTL` directive was standardized in RFC 2038 and defines the default TTL value applied to any RR that does not have an explicit TTL defined. TTL in the context of DNS means the time in seconds that a record may be cached (stored) by another name server or in some cases a resolver. (Caching is explained in Chapter 4.)

$TTL Syntax

```
$TTL time-in-seconds
```

The following shows a typical `$TTL` directive:

```
$TTL 172800 ; 2 days
```

BIND provides a short format to allow the time value to be written without resorting to a calculator or some strenuous mental arithmetic. The case-insensitive values are `m` = minutes, `h` = hours, `d` = days, `w` = weeks. This book uses the standard BIND short format throughout simply to make the time values used more quickly understood. If zone files are to be ported between BIND and other DNS software, the short forms should not be used. The preceding `$TTL` could be written in any of the following forms when using the BIND short format:

```
$TTL 2d
$TTL 48h
$TTL 2880m
$TTL 1d24h
```

The `time-in-seconds` value may be in the range `0`, which indicates the record should never be cached, to a maximum of `2147483647` (roughly 68 years). The current best practice recommendation (RFC 1912) suggests a value greater than one day, and on RRs that rarely change, longer values should be considered. This book typically uses a `$TTL` value of `172800` (2 days), which represents a reasonable balance between name server load and speed of change. In an attempt to reduce the possibly negative effects of caching on DNS-based load-balancing techniques (discussed in Chapter 8), there is an increasing trend toward *very* low (120 seconds or less) TTL values at the expense of DNS load.

The TTL determines two DNS operational parameters:

1. *Access load*: The lower the TTL, the more frequently queries occur and the higher the operational load on the zone DNS.
2. *Change propagation*: The TTL value represents the maximum time that any change will take to propagate from the zone master or slave to all name server caches.

The `$TTL` directive must appear before any RR to which it will be applied. BIND 9 will refuse to load a zone that does not have a valid `$TTL` directive.

■**Note** In older versions of BIND (prior to BIND 9), the default `$TTL` was defined in the `min` field of the SOA RR (described later in this chapter), which reflected the standards then in force. RFC 2308 defines both implementation of the `$TTL` directive and the revised use of the `min` field in the SOA RR.

The $GENERATE Directive

The `$GENERATE` directive is BIND specific and should not be used if zone files will be ported between BIND and other RFC-compliant DNS software.

`$GENERATE` is provided to ease generation of repetitive sequences of RRs. Only NS, PTR, A, AAAA, DNAME, and CNAME RRs are supported. The most obvious use for `$GENERATE` is when creating zone files used in delegation of reverse subnet maps. The reverse-map zone files involve a series of RRs that increment by a single value. The following fragment shows an extract from the reverse delegation zone file described in Chapter 8:

```
$ORIGIN 199.168.192.IN-ADDR.ARPA.
.....
65              IN  CNAME   65.64/26
66              IN  CNAME   66.64/26
67              IN  CNAME   67.64/26
....
125             IN  CNAME   125.64/26
126             IN  CNAME   126.64/26
```

The following `$GENERATE` directive would create the preceding full sequence.

```
$GENERATE 65-126 $ CNAME $.64/26
```

$GENERATE Syntax

```
$GENERATE start-stop[step] lhs type rhs
```

In the `$GENERATE` syntax, `start` is the starting value of the generated sequence and `stop` is the ending value. `step` is optional and indicates the value to be added on each iteration; if omitted, `1` is assumed. `lhs` indicates the value of the left-hand name. An `lhs` value of `$` indicates the current iteration value will be substituted as shown in the example. The `type` field is the RR type, and only CNAME, NS, A, AAAA, DNAME, and PTR are supported. `rhs` is the left-hand expression; again, `$` indicates the current iteration value will be substituted. The `rhs` and `lhs` values will have normal `$ORIGIN` substitution rules applied.

The corresponding PTR records used in normal reverse-map zone files will typically have unique host names that cannot be used with the `$GENERATE` directives; for example, `bill`, `fred`, `www`, etc. do not have an iterator relationship, but if host names were sequentially numbered, such as PC65 to PC126, the `$GENERATE` directive could be applied to them. Occasionally one wishes life was that simple!

DNS Resource Records

A large number of Resource Records have been defined over the 25 years of the DNS specification. These RRs are of two types: *real* RRs (for want of any better terminology) that appear in a zone file, and meta (or pseudo) RRs that only appear in the `QUESTION SECTION` or `ADDITIONAL SECTION` of queries (see Chapter 15). Table 13-2 later in this chapter describes the meta (or pseudo) RRs. Table 13-1 shows the currently assigned real Resource Records (they appear in zone files) from IANA (`www.iana.org/assignments/dns-parameters`) and their current support status in BIND and Windows DNS (Windows Server 2003). The Code column identifies the RR type, which is used only in the binary format when the RR is transmitted and does not appear in the text version of the RR—it is provided for information and cross-referencing purposes only. This table also shows the documentation status in this book (Reference column). The RRs are shown in alphabetic order for convenience.

Table 13-1. *Resource Record Status*

RR Name	Code	Reference	BIND	Windows	Specification	Notes
A	1	Yes	Yes	Yes	RFC 1035	Forward map. Host to IPv4 address.
A6	38	Yes	Yes	No	RFC 2874	Experimental. Forward map. Host to IPv6 address.
AAAA	28	Yes	Yes	Yes	RFC 3596	Forward map. Host to IPv6 address.
AFSDB	18	Yes	Yes	Yes	RFC 1183	Andrew File System Database location.
APL	42	Yes	Yes	No	RFC 3123	Experimental. Stands for Address Prefix Lists—supplies lists of IP addresses for any required purpose.

Continued

Table 13-1. *Continued*

RR Name	Code	Reference	BIND	Windows	Specification	Notes
ATMA	34	Yes	No	Yes	None	Private. Stands for ATM Address. Defined by the ATM forum (document reference `af-saa-0069.000.pdf`).
CERT	37	Yes	Yes	No	RFC 2538	CERT RRs define various security certificate formats, such as X.509, for storage in the DNS.
CNAME	5	Yes	Yes	Yes	RFC 1035	Stands for Canonical Name (Alias). Maps an alias name to another name.
DNAME	39	Yes	Yes	No	RFC 2672	Experimental. Used for reverse-map delegation—especially IPv6.
DNSKEY	48	Yes	Yes	No	RFC 4034	DNSKEY RRs define the public key used in DNSSEC (DNSSEC.bis) operations only. The KEY RR is used for all other public keys.
DS	43	Yes	Yes	No	RFC 4034	Delegation Signer RRs are only used in DNSSEC (DNSSEC.bis) operations and are placed in parent zones at the point of delegation to a child zone to create chains of trust.
EID	31	No	No	No	None	Private RR. Stands for Endpoint Identifier.
GPOS	27	No	Yes	No	RFC 1712	Stands for Geographical Position—made obsolete by LOC RR.
HINFO	13	Yes	Yes	Yes	RFC 1035	Textual host OS and hardware description.
IPSECKEY	45	Yes	No	No	RFC 4025	IPSECKEY RRs are used to define keys and other properties used in IPSec operations.
ISDN	20	Yes	Yes	Yes	RFC 1183	Maps a host to an ISDN E.164 address.
KEY	25	Yes	Yes	Yes	RFC 3445	KEY RRs define public keys for use in cryptographic security operation, such as SIG(0). The exception: DNSSEC (DNSSEC.bis), which uses the DNSKEY RR exclusively.
KX	36	Yes	Yes	No	RFC 2230	Stands for Key Exchanger. Returns an alternative host name.

RR Name	Code	Reference	BIND	Windows	Specification	Notes
LOC	29	Yes	Yes	No	RFC 1876	Experimental but widely used. Provides longitude, latitude, and altitude information for a name.
MB	7	Yes	Yes	Yes	RFC 1035	Experimental. Stands for Mailbox Name. Not widely used.
MD	3	No	No	No	RFC 1035	Mail Destination. Obsolete—replaced by MX.
MF	4	No	No	No	RFC 1035	Stands for Mail Forwarder. Obsolete—replaced by MX.
MG	8	Yes	Yes	Yes	RFC 1035	Experimental. Stands for Mail Group Member. Not widely used.
MINFO	14	Yes	No	Yes	RFC 1035	Experimental. Stands for Mail list information. Not widely used.
MR	9	Yes	Yes	Yes	RFC 1035	Experimental. Stands for Mail Rename. Not widely used.
MX	15	Yes	Yes	Yes	RFC 1035	Stands for Mail Exchanger. Defines the domain's incoming mail servers.
NAPTR	35	Yes	Yes	No	RFC 3403	Stands for Naming Authority Pointer. This is a general-purpose RR that defines rules to be applied to application data.
NIMLOC	32	No	No	No	None	Private. Stands for NIMROD Locator.
NS	2	Yes	Yes	Yes	RFC 1035	Name Server RRs define the name servers for the domain.
NSAP	22	Yes	Yes	No	RFC 1706	Maps a host to an NSAP (OSI address).
NSAP-PTR	23	No	No	No	RFC 1348	NSAP reverse map. Made obsolete in RFC 1706.
NSEC	47	Yes	Yes	No	RFC 4034	NSEC RRs are used in DNSSEC (DNSSEC.bis) operations to provide proof of nonexistence of names.
NULL	10	No	Yes	No	RFC 1035	Experimental. Cannot be defined in a master zone file.
NXT	30	No	Yes	Yes	RFC 3755	Stands for Next Domain. Made obsolete by RFC 3755.
PTR	12	Yes	Yes	Yes	RFC 1035	IP to host (reverse mapping) used by IPv4 and IPv6.

Continued

Table 13-1. *Continued*

RR Name	Code	Reference	BIND	Windows	Specification	Notes
PX	26	Yes	Yes	No	RFC 2163	X.400 to RFC 822 mail mapping.
RP	17	Yes	Yes	Yes	RFC 1183	Experimental. Stands for Responsible Person. Supplies textual information about a host or name.
RRSIG	46	Yes	Yes	No	RFC 4034	RRSIG RRs are used in DNSSEC (DNSSEC.bis) operations to contain the digital signatures of RRsets.
RT	21	Yes	Yes	Yes	RFC 1183	Experimental. Stands for Route Through. Defines the route to one host via another host.
SIG	25	Yes	Yes	Yes	RFC 2931	Stands for Security Signature. This RR is now limited to use as a meta (or pseudo) RR when securing public key transactions (SIG(0)) used in Dynamic DNS (DDNS).
SINK	40	No	No	No	None	Private RR.
SOA	6	Yes	Yes	Yes	RFC 1035/2308	Stands for Start of Authority. Defines global information about the domain.
SRV	33	Yes	Yes	Yes	RFC 2782	Stands for Services Record. Allows discovery of services provided by hosts.
SSHFP	44	Yes	Yes	No	None	Draft RFC (`draft-ietf-secsh-dns-05.txt`). Keys for use with Secure Shell (SSH).
TXT	16	Yes	Yes	Yes	RFC 1035	Arbitrary text associated with a domain. Also used for SPF antispam record.
WKS	11	Yes	Yes	Yes	RFC 1035	Deprecated. SRV provides more powerful features.
X25	19	Yes	Yes	Yes	RFC 1183	Maps a host to an X.25 address.

Table 13-2 lists meta (or pseudo) RRs and describes their use. Meta RRs do not appear in zone files but may appear in the `QUESTION SECTION`, `ANSWER SECTION`, or `ADDITIONAL SECTION` of a query (see Chapter 15). Meta RRs are defined in the IANA list (`www.iana.org/assignments/dns-parameters`).

Table 13-2. *Meta RRs*

RR Name	Code	Description
ANY	255	Appears in the `QUESTION SECTION` of a query and requests all records associated with the query name. If the associated name is the zone or domain name, then only those records having that name are supplied; for example, SOA, MX, NS RRs, not the entire zone file.
AXFR	252	Appears in the `QUESTION SECTION` of a query and requests a transfer of the entire zone.
IXFR	251	Appears in the `QUESTION SECTION` of a query and requests an incremental zone transfer, that is, only changed records.
MAILB	253	Appears in the `QUESTION SECTION` of a query and requests all MB, MG, and MR RRs for the associated name.
OPT	41	Appears in the `ADDITIONAL SECTION` of a query and response. Used to indicate EDNS0 (RFC 2671) is in use when either `dnssec-enable yes;` is set in the `options` clause or `edns yes;` is defined in a `server` clause of `named.conf`. The OPT meta RR format is described in the section "EDNS0 Transactions" located in Chapter 15 and is used among other things to negotiate a larger UDP block size.
SIG	25	Appears in the `ADDITIONAL SECTION`. See notes in Table 13-1.
TKEY	249	Appears only in the `ADDITIONAL SECTION` of a query or response. The Transfer KEY RR contains the computed Diffie-Hellman key exchange material.
TSIG	250	Appears only in the `ADDITIONAL SECTION` of a query or response. The Transfer SIG RR contains the Message Authentication Code (MAC) for use with either zone transfers or Dynamic DNS updates in shared secret transactions.

Resource Record Common Format

The first part of textual RRs is common to all types and the various fields are described in detail here to avoid repetition within the individual RR descriptions:

```
name    ttl   class   type  type-specific-data
```

The name Field

The `name` field, frequently called the *owner* name to differentiate it from names that can appear on the `type-specific-data` side of the RR, may take any of the following values:

- A Fully Qualified Domain Name—ends with a dot
- An unqualified name (does not end with a dot), in which case the `$ORIGIN` substitution rule is applied as described previously for the `$ORIGIN` directive
- A blank (tab) or space, in which case the name from the previous RR or `$ORIGIN` is substituted
- A single `@` character, in which case the current value of `$ORIGIN` is substituted

The `name` field may use a very liberal set of characters; the original specifications, RFCs 821, 952, and 1123, limited the character set to the following:

- Any upper- or lowercase alpha character—`a` to `z` and `A` to `Z`
- Any numeric value from the range `0` to `9`
- The `-` (dash or minus sign)

The preceding list is the safest set to use under all conditions (and can be enforced by using the `check-name` statement in BIND's `named.conf` file if required). The rule for the permissible character set was liberalized by RFC 2181, which essentially says any character, in context, is permissible, and it is up to the client application to validate the name format before using it. The justification for this change is that the DNS can be used for the storage of many types of data, not just domain names, each of which may need to use a unique character set. The biggest single effect of this RFC was to formally allow `_` (underscore), which is used in the SRV RR and the `/` (forward slash), which is used in the delegation of reverse subnet maps (see Chapter 8). There appears to be only two remaining hard limitations on names. First is the use of the terminating `.` (dot) in a name, the absence of which will invoke the `$ORIGIN` substitution rule. Second, a single `@` (commercial at sign) will explicitly substitute the `$ORIGIN` name.

Each dot-separated value in a name can be up to 63 characters in length and is called a *label*. In practice, to avoid reaching the 512-byte limit of UDP transactions in cases where multiple records are returned, smaller is better! A practical limit could be 10 to 20 characters per label and indeed, certain libraries limit host names to 32 characters, but as noted, up to 63 characters can be used where necessary. If the host name part is also used as a NETBIOS name, it should be limited to 15 characters or less. The sum of all labels in a name, including the separating dots, must not exceed 255 characters. RFC 3490 (clarified by RFC 3743) defines the rules for the use of Internationalized Domain Names (IDNs) in the DNS.

The `ttl` Field

The `ttl` field defines the time in seconds that the RR to which it applies may be cached. The field is optional, and if not present, the zone default (defined by the `$TTL` directive) is used. If the field is present, it will be used whether it is lower or higher than the zone default. The `ttl` field is an unsigned 32-bit integer and may take a value in the range `0` (do not cache) to `2147483647`. BIND allows its standard short format to be used in any `ttl` field. The case-insensitive values are `m` (minutes), `h` (hours), `d` (days), and `w` (weeks); for example, `3w2d1h5m5` is equivalent to `1991705` seconds. This book uses the BIND short format throughout because it is clearer, but if zone files are to be ported between BIND and other DNS applications, the short format should not be used. The value of the TTL field was clarified in RFC 2181.

The `class` Field

The `class` field may take the case-insensitive values of `IN` = Internet class, `CH` = CHAOS (an MIT LAN protocol), `HS` = Hesiod (an information service used at MIT). The latter two seem mostly of historic interest, but the use of the CHAOS class in a `dig` command (see Chapter 9) is the only way in which the BIND version number may be interrogated remotely (the `version` statement in `named.conf` may be used to reply with arbitrary information to disguise the version number).

The `type` Field

The `type` field designates the RR type, for example, AAAA. Each `type` is described in alphabetic order under its RR name in the "Resource Record Descriptions" section later in this chapter.

The `type-specific-data` Field

The `type-specific-data` field may consist of one or more parameters and is unique to the RR. The `type-specific-data` textual representations for each RR are described in the following sections. Chapter 15 defines the binary, or wire format, representation.

Bit Labels

RFC 2673 introduced a new *bit label,* or *bit-string label* field, that is optimized for the definition of IPv6 addresses when used in reverse-map delegation where the volume of textual data can be brutal. This bit label, which has currently EXPERIMENTAL status (its status was changed by RFC 3363), is described here for completeness, but is otherwise not used throughout this book. The bit-label field is designed to be used as a left-hand name field only. It cannot appear in a right-hand name expression.

Bit Label Syntax

```
\[string]
```

In the bit-label syntax, \ is a literal to indicate the beginning of a bit label. The characters [and] are used to enclose the bit-string label definition. The `string` field may be used to define a binary, octal, hexadecimal, or IPv4 format address field as shown here:

```
type-string[/length]
```

The `type-string` field begins with a literal that defines the string format and takes one of the following values:

- `x` indicates hexadecimal format and is followed by as many hexadecimal characters as required to enclose that part of the address being defined by the bit label. The `/length` field is mandatory and indicates the number of bits contained within the hexadecimal field. Any unused bits must be set to 0. The first hexadecimal character is assumed to begin the field, thus the hexadecimal format can only be used on 4-bit boundaries.
- `o` indicates octal format and is followed by as many octal characters as required to enclose that part of the address being defined by the bit label. The `/length` field is mandatory and indicates the number of bits contained within the octal field. Any unused bits must be set to zero. The first octal character is assumed to begin the field, thus the octal format can only be used on 3-bit boundaries.
- `b` indicates binary format and is followed by as many binary characters as required to enclose that part of the address being defined by the bit label. The `/length` field is not required with the binary format. The binary format can be used on any bit boundary.

- The absence of any literal defines that the field is in *dotted-quad* format (IPv4 address format) and must contain all four parts of the address. The /length field is mandatory and indicates the number of bits contained within the dotted-quad field. The dotted-quad format can only be used on 32-bit boundaries.

The following fragments show a *Global Unicast* IPv6 address, 2001:db8:3d::1, being fully delegated according to the hierarchy defined in Chapter 5. Each $ORIGIN directive is assumed to start a separate zone file; the DNAME RR used in the fragments also has EXPERIMENTAL status.

```
$ORIGIN IPV6.ARPA.
; first 16 bits
.....
\[x2001/16]             IN   DNAME   tla.example.org.
.....

$ORIGIN tla.example.org.
; next 13 bits
.....
\[x0db8/13]             IN   DNAME   nla.example.net.
.....

$ORIGIN nla.example.net.
; next 19 bits only possible with binary format
.....
\[b0010000000000111101]  IN   DNAME   ip6.example.com.
.....

$ORIGIN ip6.example.com.
; last 80 bits
.....
\[x000100000001/80]      IN   PTR   bill.example.com.
.....
```

Whether the preceding is more or less comprehensible than the normal IPv6 reverse mapping defined in Chapter 5 is for you to decide. Bit labels are fully supported by BIND (9.3+), but if used with DNS software that does not support them (including previous versions of BIND), such software will reject queries containing bit labels as invalid.

RRsets

RRs with the same name, class, and type are collectively called an RRset. By extension of this definition, a singleton RR is also an RRset! The following is an example of an RRset using MX RRs:

```
; zone file fragment
$TTL 2d ;172800 seconds
$ORIGIN example.com.
....
    3w  IN  MX 10 mail.example.com.
    4h  IN  MX  10 mail.example.com.
        IN  MX  20 mail.example.net.
....
```

The `type-specific-data` and `ttl` fields are explicitly excluded from the definition of an RRset. However, RFC 2181 does not allow RRsets to have different TTL values. If they are different, only one TTL, typically the lowest, will be used to cover the RRset. In the preceding example, the TTL values are not all the same, and the lowest (`4h` or `14400` seconds) would typically be used for the RRset.

Resource Record Descriptions

The following sections describe each RR type defined in Table 13-2. Examples are used where appropriate to illustrate the RR usage.

IPv4 Address (A) Record

The Address RR forward maps a host name to an IPv4 address. The IPv6 equivalent is an AAAA RR. The A RR is defined in RFC 1035. The only parameter is an IPv4 address in dotted decimal format.

A RR Syntax

```
name  ttl  class   rr     ipv4
joe        IN      A      192.168.254.3
```

If multiple addresses are defined with the same `name`, then BIND will respond to queries with all the addresses defined (an RRset), but the order may change depending on the value of the `rrset-order` statement in BIND's `named.conf` file. The default order is cyclic or round-robin. The same IP address may be defined with different names. IP addresses do not have to be in the same IP address class or range. The order in which A RRs are defined is not significant, but it may be easier to define them in either an ascending or descending order of IP address, as this can prevent unintentional duplicate definition of IP addresses. Since the `ipv4` field is an address, not a name, there is no terminating dot. The following zone file fragment illustrates various uses of the A RR:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
joe        IN      A      192.168.0.3  ; joe & www = same ip
www        IN      A      192.168.0.3
; could be rewritten as
; www.example.com.  A      192.168.0.3
```

```
fred  3600 IN     A      192.168.0.4  ; ttl overrides $TTL default
ftp        IN     A      192.168.0.5 ; round-robin with next
           IN     A      192.168.0.6
mail       IN     A      192.168.0.15  ; mail = round-robin
mail       IN     A      192.168.0.32
mail       IN     A      192.168.0.33
squat      IN     A      10.0.14.13  ; address in another range & class
```

In the preceding example, BIND will respond to queries for `mail.example.com` as follows—assume use of the `rrset-order {order cyclic;};` statement or no `rrset-order` statement, in which case it defaults to cyclic:

```
1st query 192.168.0.15, 192.168.0.32, 192.168.0.33
2nd query 192.168.0.33, 192.168.0.15, 192.168.0.32
3rd query 192.168.0.32, 192.168.0.33, 192.168.0.15
4th query 192.168.0.15, 192.168.0.32, 192.168.0.33
```

Multiple names may be used to define the same IP address as shown in the preceding example for `joe` and `www`. Many people prefer to use a CNAME RR (defined later in this chapter) to achieve the same result. There is no functional difference between the two definitions except that multiple A RRs are slightly faster since they involve less work when processing a query.

■**Note** This book uses both FQDN and unqualified name formats when defining left-hand and right-hand names to expose you to a variety of styles. It is recommended that a single style be used throughout zone files to avoid confusion; for example, FQDN in right-hand names, unqualified in left-hand names, or whatever style you find less confusing. Do not be tempted to adopt a style solely because it is shorter. If the style is short and less confusing—bliss.

Experimental IPv6 Address (A6) Record

The A6 RR is an experimental RR used to forward map host names to IPv6 addresses. RFC 3363 changed the status of the A6 RR, defined in RFC 2874, from a PROPOSED STANDARD to EXPERIMENTAL due primarily to performance and operational concerns. The current IETF recommendation is to use AAAA RRs to forward map IPv6 addresses. It is not clear at this time when (or if) the A6 RR will ever be restored to recommended usage by the IETF, even though it is fully supported by BIND. It is described here because it does significantly reduce the effort required to define an IPv6 address by recognizing the hierarchical nature of IPv6 addresses and allowing various parts of addresses to be defined in separate zone files or as separate parts of the same zone file. The default behavior of current BIND versions is to issue AAAA RR queries for IPv6 or A RR and AAAA RR queries when using dual-stack implementations. The only way to force use of A6 RRs at this time is to use BIND 9.2.1 or lower. IPv6 addresses are described in Chapter 5.

A6 RR Syntax

```
name    ttl  class  A6  prefix  ipv6 [next-name]
joe          IN     A6  64      ::1  subnet1.example.com.
```

The `prefix` field defines the number of bits (0 to 128) that are *not* defined by the A6 RR. In the preceding example, 64 bits are defined by this A6 RR, and 64 bits will be defined by another A6 RR. If the `prefix` is `0`, then no additional A6 RRs are required, the complete IPv6 address is defined in this RR, and the A6 RR has the same functionality as an AAAA RR.

The `ipv6` field contains that part of the IPv6 address that is defined by this A6 RR, which in the preceding example is 64 bits (128 – `prefix`). A full 128 bits must be defined in each A6 RR; bits that are not defined within any A6 RR should by convention be set to 0 as in the preceding example.

The optional `next-name` field defines the name of another A6 RR, which will define the remaining bits, 64 in the preceding example, of the IPv6 address. This field is mandatory if the `prefix` field is not 0; that is, the IPv6 address in this A6 RR is not completely defined.

A6 RRs may be chained such that an A6 record pointed to by one A6 RR may itself point to another A6 RR that describes the next part of the address. This process is illustrated in the following fragment, where the zone file separates the subnet definition (bits 48 to 63) from the interface ID and in turn defines an additional A6 pointer for a target IPv6 address of 2001:dba:ddef:1::1:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
joe          IN  A6  64      ::1  subnet1.example.com.
....
; the next A6 RR defines the subnet ID only (16 bits) and
; references a further A6 RR for the remaining 48 bits
subnet1      IN  A6  48      0:0:0:1:: example-com.example.net.
....
```

In the preceding fragment, the second A6 RR contains only 16 valid bits (128 – 64 – 48 = 16) and leaves a further 48 bits to be defined at the A6 RR with the name `example-com.example.net.`, which is contained in an external domain (`example.net`). The second A6 RR defines a full 128-bit address (`0:0:0:1::`) with the relevant part (the subnet ID of 1) in the correct position (bits 48 to 63); all other bits in the address are 0.

The A6 RR at `example-com.example.net` could define the remaining 48 bits or further chain to the Internet Registry that assigned the address blocks. In the following fragment, the remaining A6 RRs are shown in a series of zone files reflecting the IPv6 hierarchy—each `$ORIGIN` directive is assumed to be a separate zone file.

```
$ORIGIN example.net.
; NLA assigns 19 bits
example-com   IN  A6   29  0:2:ddef:: ipv6.example.org.

$ORIGIN example.org.
; SLA assigns remaining 29 bits
ipv6          IN  A6   0  2001:db8::
```

The resulting address of `joe.example.com` is `2001:dba:ddef:1::1`. Once established, the A6 RR chains should be stable, but the address values contained within them can be easily and readily changed, allowing network renumbering to be a fairly painless process.

The concern expressed by the IETF, which led to the A6 RR being relegated to EXPERIMENTAL status, is that a single address lookup can result in a significant number of DNS transactions, any one of which could fail; the chains can take some time to debug and are potentially error-prone; it is also possible to create A6 RR loops.

IPv6 Address (AAAA) Record

The AAAA RR is used to forward map hosts to IPv6 addresses and is the current IETF recommendation for this purpose. IPv6 is described in Chapter 5. The AAAA (colloquially referred to as Quad A) RR is functionally similar to the A RR used for IPv4 addresses and is defined in RFC 3596.

AAAA RR Syntax

```
name  ttl  class   rr     ipv6
joe        IN      A      2001:db8::1
```

If multiple addresses are defined with the same name, then BIND will respond to queries with a list of the addresses, but the order may change on successive queries depending on the value of the `rrset-order` statement in BIND’s `named.conf` file. The default order is cyclic or round-robin. The same IP may be defined with different names. IP addresses do not have to be in the same subnet or use the same global routing prefix. The order in which AAAA RRs are defined is not significant, but it may be easier to define them in either an ascending or descending order of IP address since this can prevent unintentional duplicate definition of IP addresses. Since the `ipv6` field is an address and not a name, there is no terminating dot. The following zone file fragment illustrates various uses of the AAAA RR:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
joe        IN      AAAA      2001:db8::3  ; joe & www = same ip
www        IN      AAAA      2001:db8::3
; functionally the same as the preceding record
www.example.com.   AAAA      2001:db8::3
fred  3600 IN      AAAA      2001:db8::4  ; ttl =3600 overrides $TTL default
ftp        IN      AAAA      2001:db8::5 ; round robin with next
           IN      AAAA      2001:db8::6
mail       IN      AAAA      2001:db8::7  ; mail = round robin
mail       IN      AAAA      2001:db8::32
mail       IN      AAAA      2001:db8::33
squat      IN      AAAA      2001:db8:0:0:1::13  ; address in another subnet
```

IPv6 and IPv4 RRs can be freely mixed in the zone file as shown the following fragment:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....

www         IN      A      192.168.0.3
mail        IN      A      192.168.0.32
www         IN      AAAA      2001:db8::3
mail        IN      AAAA      2001:db8::32
```

Blank name substitution can also be used in mixed configurations if this is more convenient or understandable:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
www         IN      A      192.168.0.3
            IN      AAAA   2001:db8::3
mail        IN      A      192.168.0.32
            IN      AAAA   2001:db8::32
```

In both preceding fragments, it is assumed that the hosts, `www.example.com` and `mail.example.com`, are running dual (IPv4/IPv6) IP stacks.

AFS Database (AFSDB) Record

The AFS Database RR defines a host that provides an AFS Database service (AFS was originally the Andrew File System). The purpose of this RR is to allow hosts within the domain to discover the host or hosts that provide both the AFS service and the type of service. The RR is not widely used, and its functionality could now be provided by the generic SRV RR. The AFSDB RR is experimental and is defined in RFC 1183.

AFSDB RR Syntax

```
name   ttl class AFSDB  sub-type host
joe        IN    RP  2          joe.people.example.com.
```

The `sub-type` field may be either `1` = the AFS version 3.0 of the service or `2` = the OSF DCE/NCA version of the AFS service. The `host` field defines the host name that provides the `sub-type` service. The following fragment shows the use of the AFSDB RR:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
...
@            IN  AFSDB  1  joe.example.com.
             IN  AFSDB  1  bill.example.com.
joe          IN  A       192.168.254.3
bill         IN  A       192.168.254.4
```

In the preceding fragment, multiple hosts providing the same `sub-type` are shown—the order of use is not defined in the RFC.

Note While it may appear that the AFSDB RR type is only relevant if the AFS file system is being used, this need not be the case. As long as the RR syntax is satisfied, the RR could be used for any purpose, as indeed is true for many other existing RR types. As an example, the AFSDB RR could be used to differentiate between two types of MySQL servers. The client application would clearly need to issue the appropriate AFSDB DNS query to obtain the required results from the name server. An alternative strategy would be to define a specific user-defined MySQL RR type. This process is described in the "User-defined RRs" section later in the chapter.

Address Prefix List (APL) Record

The Address Prefix List RR is an RR that may be used to define one or more IP addresses or IP address ranges for any required purpose. The APL RR is experimental and is defined in RFC 3132.

APL RR Syntax

```
name     class   ttl  rr  [!]af:address/prefix
router1  IN           APL 1:192.168.38.0/24 !1:192.168.38.0/26
```

The `!` field is optional and, if present, indicates negation; that is, the following address or address range is explicitly excluded. The negation feature can greatly reduce the number of entries required to define a given address range. To illustrate this point, in the preceding example, the first value (`192.168.38.0/24`) defines an IPv4 address range from 192.168.38.0 to 192.168.38.255. The negated part (`!192.168.38.0/26`) excludes 64 IPv4 addresses (192.168.38.0 to 192.168.38.63) that lie in this range. If the negation value was not used, then this definition would require all the positive ranges to be defined, which in the preceding case would result in 192.168.38.64/26, 192.168.38.128/26, and 192.168.38.192/26. The `af` field defines the address family as defined by IANA (`www.iana.org/assignments/address-family-numbers`); from this list, IPv4 = `1` and IPv6 = `2`. The `address/prefix` value is the IP address, whose format is defined by `af` value, written in the IP prefix (or slash) notation.

The following fragment shows the use of the APL RR to indicate the range of private addresses IPv4 used by a NAT-PT (IPv4 to IPv6) gateway and the corresponding public (Global Unicast) IPv6 addresses:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
...
nat-pt    IN  A       192.168.254.3
          IN  AAAA    2001:db8::17
          IN  APL     (
                       1:192.168.254.0/27  ; IPv4 = 32
                       2:2001:db8::0/122   ; IPv6 = 64
                       !2:2001:db8::37/128 ; excluding 1 IPv6
                      )
```

The APL RR does not define any specific application or requirement for the address lists. The preceding example, which is entirely fictitious, shows a possible use of the APL RR.

ATM Address (ATMA) Record

The Asynchronous Transfer Mode Address RR is the equivalent of an A RR for ATM endpoints. It associates an ATM address in either E.164 format or the AESA (ATM End System Address, defined in ISO8348/AD 2). The ATMA RR is a private RR type. It has been allocated an ID value by IANA, though it is not defined by an RFC, but by the ATM Forum (`www.atmforum.com/standards/approved.html`), document reference `af-saa-0069.000.pdf`, which is available at no charge. This specification defines both forward and reverse mapping. The ATMA RR is supported by the Windows Server 2003 DNS but is not supported by the current BIND release.

Certificate (CERT) Record

The Certificate RR may be used to store either public key certificates or Certificate Revocation Lists (CRL) in the zone file. The CERT RR is defined in RFC 2538.

CERT RR Syntax

```
name  ttl  class   rr   type key-tag algorithm cert-crl
joe        IN      CERT 1    12179       3      (
                        AQPSKmynfzW4kyBv015MUG2DeIQ3
                        Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                        kfzj31GajIQKY+5CptLr3buXA10h
                        WqTkF7H6RfoRqXQeogmMHfpftf6z
                        Mv1LyBUgia7za6ZEzOJBOztyvhjL
                        742iU/TpPSEDhm2SNKLijfUppn1U
                        aNvv4w==  )
```

For the CERT RR, the `type` field defines the certificate format and may take one of the following values:

`0` = Reserved

`1` = X.509 (RFC 3280)

`2` = SKPI (RFC 2693)

`3` = OpenPGP (RFC 2440)

`4–252` = Currently unassigned

`253` = Private URI (see the text that follows)

`254` = Private OID (see the text that follows)

`255` = Reserved

The value `253` specifies that the format of the `cert-crl` field will commence with a URI that defines the address, for instance, a host name of the location that may be interpreted by the recipient to define the format of the certificate. The URI must be followed by a single space and then a certificate whose representation format is defined by the URI.

The value 254 specifies that the format of the `cert-crl` field will commence with an Object Identifier (OID) that defines an object that may be interpreted by the recipient to define the format of the certificate. The OID must be followed by one or more spaces and then the certificate whose representation format is defined by the OID.

The `key-tag` field is generated by the `dnssec-keygen` utility to identify the key embedded in the certificate. The `algorithm` field defines the cryptographic algorithm being used and may take one of the following values:

`0` = Reserved

`1` = RSA-MD5—not recommended

`2` = Diffie-Hellman—optional

`3` = DSA—mandatory (RFC 2536)

`4` = Reserved for elliptic curve cryptography

`5` = RSA-SHA-1—mandatory (RFC 3110)

`6–251` = Currently unassigned

`252` = Indirect (see the section “Alternative Cryptographic Algorithms” later in this chapter)

`253` = Private URI (see the section “Alternative Cryptographic Algorithms” later in this chapter)

`254` = Private OID (see the section “Alternative Cryptographic Algorithms” later in this chapter)

`255` = Reserved

The `cert-crl` field contains the certificate or a Certificate Revocation List entry (see also text preceding this list if the `type` field is either 253 or 254) in the format defined by the relevant RFC associated with the `type` field. The CERT RR X.509 `cert-crl` field data may be generated using tools such as OpenSSL and GnuTLS and then edited into the RR.

Canonical Name (CNAME) Record

A Canonical Name record maps an alias to the real or *canonical* name that may lie inside or outside the current zone. Canonical simply means the expected or real name. The CNAME RR is defined in RFC 1035.

CNAME RR Syntax

```
name  ttl  class   rr     canonical-name
www        IN      CNAME  server1.example.com.
```

The most common use of CNAME RRs is where a host has more than one possible name; for example, assume a server has a real name of `server1.example.com` but also hosts a web and an FTP service, then both `www.example.com` and `ftp.example.com` must be *resolvable* (must translate) to the same IP address as `server1.example.com`. This can be done using multiple A records as shown previously, but many people elect to use an A RR for the real name, server1.example.com, and use CNAME RRs for www.example.com and ftp.example.com, both of which alias server1.example.com as shown in the following fragment:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
server1    IN      A      192.168.0.3
www        IN      CNAME  server1
ftp        IN      CNAME  server1
```

A name defined with a CNAME RR is only allowed to have a NSEC, KEY, DNSKEY, and RRSIG RR (used in DNSSEC only) using the same name, thus it is not permissible, for example, to define a TXT RR or an RP RR using the name assigned to the CNAME RR.

CNAME RRs incur performance overheads. The most common DNS query is for an A RR, or an AAAA RR if IPv6—the end system typically needs an address that is only defined with these RR types. In the preceding example, if a query for the address of www.example.com is received (an A or AAAA query), two lookup operations are performed on the master or slave server. The first lookup finds www, which is a CNAME; this is followed by a lookup for server1 to obtain the IP address, that is, the CNAME *chain* is followed to attempt to resolve the original request for an IP address. On low-volume name servers, the additional resources used are probably not significant, but on high-volume servers, the additional load can become nontrivial. The user must make a choice to balance what many see as the convenience of using CNAME RRs against the possible performance degradation involved.

While use of CNAME RRs with NS and MX records is widely implemented and normally generates a working configuration, it is theoretically not permitted (RFC 1034 section 3.6.2) since it can result in lost names. The fragment that follows illustrates a widely used but technically *invalid* configuration:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
           IN      MX  10  mail.example.com.
mail       IN      CNAME   server1
server1    IN      A      192.168.0.3
```

In the preceding configuration, when a query is issued for the A RR of mail.example.com, the result will return both the mail.example.com CNAME RR and the server1.example.com A RR. When the A RR is used by a mail application, the name associated with the CNAME can be lost; for instance, there may be a valid MX record referencing the host mail.example.com and elsewhere an A RR referencing server1.example.com, but nothing joining the two records. The following fragment, by reordering the RRs, will achieve the same result and allow a valid mapping of the MX name to the A RR name.

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
           IN      MX  10  mail.example.com.
server1    IN      CNAME   mail
mail       IN      A      192.168.0.3
```

For many users, the preceding simply feels uncomfortable because the *real* host name is `server1.example.com`, not `mail.example.com` but it is a perfectly legitimate definition that will cause no problems.

It is permitted for one CNAME RR to alias another CNAME RR, but this considered bad practice due to the additional lookup loads involved and because it can lead to CNAME loops (that is, a CNAME RR references a CNAME RR, which references a CNAME RR, and so on ad infinitum).

CNAME RRs are the only way to handle references to RRs that lie in another domain, sometimes referred to as an *out-of-bailiwick reference*. The following fragment shows that `ftp.example.com` is actually provided by `ftp.example.net`:

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
ftp        IN      CNAME   ftp.example.net.
```

The following fragment allows URLs of the form `www.example.com` and `example.com` to both access a web service, in this case one that also uses DNS load-balancing services:

```
; www.example.com and example.com access
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
; resolves example.com to an IP
@          IN      A      192.168.254.8
           IN      A      192.168.254.9
           IN      A      192.168.254.10
www        IN      CNAME  example.com.
```

Delegation of Reverse Names (DNAME) Record

The Delegation of Reverse Name RR is designed to assist the delegation of reverse mapping by reducing the size of the data that must be entered. The DNAME RR is designed to be used in conjunction with a bit label (described in the section “Bit Labels” earlier in the chapter) but does not strictly require a bit label. The DNAME without a bit label is equivalent to a CNAME when used in a reverse-map (or delegation) zone file, an example of which is provided in the section “Delegate Reverse Subnet Maps” located in Chapter 8. RFC 3363 changed the status of the bit label and the A6 RR from PROPOSED STANDARD to EXPERIMENTAL due to concerns over performance and that the new bit labels would require a change to all DNS software in the root and gTLD servers before the bit label could become active. Because the DNAME RR without bit labels is functionally equivalent to CNAME, its use is deprecated. The current IETF recommendation is to use text labels (names) with PTR records under the IP6.ARPA domain for the reverse mapping of IPv6 addresses. It not clear at this time when (or if) the DNAME RR and the bit label will ever be restored to recommended usage by the IETF, even though it is fully supported by BIND. The DNAME RR is defined in RFC 2672 and its syntax is shown next for completeness only.

DNAME RR Syntax

```
name            ttl  class  rr  next-name
1.0.0.0         IN     DNAME  ipv6.example.org.
```

An example of the use of DNAME RRs to delegate reverse mapping of IPv6 addresses is illustrated in the section "Bit Labels" earlier in the chapter.

DNSKEY Record

The DNSKEY RR describes the public key of a public key (asymmetric) cryptographic algorithm used with DNSSEC.bis (see Chapter 11). It is typically used to authenticate signed keys (if a Key Signing Key [KSK]) or zones (if a Zone Signing Key [ZSK]). The DNSKEY RR is typically generated by the dnssec-keygen utility (see Chapter 9) and is defined in RFC 4034.

DNSKEY RR Syntax

```
name  ttl  class   rr     flags proto algorithm key-data
example.com. IN    DNSKEY 256    3        5 (
                          AQPSKmynfzW4kyBv015MUG2DeIQ3
                          Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                          kfzj31GajIQKY+5CptLr3buXA10h
                          WqTkF7H6RfoRqXQeogmMHfpftf6z
                          Mv1LyBUgia7za6ZEzOJBOztyvhjL
                          742iU/TpPSEDhm2SNKLijfUppn1U
                          aNvv4w==  )
```

The flags field is the decimal representation of a 16-bit field that has the following bit-significant values:

- *Bits 0 to 6*: Currently unused and must be zero.
- *Bit 7*: If set, this indicates a Zone Signing Key, and the name field must be that of the apex or root of the zone being signed as shown in the preceding example. If not set, the key may not be used to sign zones, and the name field will typically reference a host record to which the DNSKEY applies.
- *Bits 8 to 14*: Currently unused and must be set to zero.
- *Bit 15*: If set, this is a Secure Entry Point key, and in this case, bit 7 must also be set to 1. This indicates for administrative purposes only that the key is a Key Signing Key, and as a consequence this bit is frequently referred to as the KSK bit.

The only valid hexadecimal combinations of the flags field are 0000, 0100, and 0101, which yield decimal values used in the RR of, respectively 0, 256, and 257.

The `proto` field can only take the value of `3` at this time; all other values are invalid.
The `algorithm` field may take one of the following values:

`0` = Reserved

`1` = RSA-MD5—not recommended (RFC 2537)

`2` = Diffie-Hellman (RFC 2539)

`3` = DSA-SHA-1—optional (RFC 2536)

`4` = Elliptic curve—not currently standardized

`5` = RSA-SHA-1—mandatory (RFC 3110)

`6` = `251` = Currently unassigned

`252` = Indirect (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`253` = Private URI (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`254` = Private OID (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`255` = Reserved

The `key-data` field is the base64 (RFC 3548) representation of the public key data. As shown in the example, if enclosed in the parentheses, whitespace is allowed for layout purposes.

■ **Note** RSA-MD5 is no longer recommended due to a number of discovered weaknesses published in February 2005. The weaknesses do not invalidate use of the algorithm.

Delegation Signer (DS) Record

The Delegation Signer RR is used in DNSSSEC (DNSSEC.bis—see Chapter 11) to create the chain of trust or authority from a signed parent zone to a signed child zone. The DS RR contains a hash (or digest) of a DNSKEY RR at the apex of the child zone. By convention, this DNSKEY RR has the SEP bit set (it has a `flags` field value of `257`), but this is not a requirement of the DNSSEC protocol. If a chain of trust is required for the zone `sub.example.com` (the child), the DS RR is added to the zone `example.com` (the parent) at the point of delegation—the NS RRs that point to `sub.example.com`. Both the parent and child zones must be signed. The DS RR is optionally generated by the `dnssec-signzone` utility (described in Chapter 9) and is defined in RFC 4034.

DS RR Syntax

```
name  ttl  class  rr  key-tag algorithm digest-type digest
joe        IN     DS  13245   5 1  (E0B4B11D0FCE00E3F
                              FA89FA873F40DC51281BF34)
```

The `key-tag` field is generated algorithmically by the `dnssec-keygen` utility and identifies the particular DNSKEY RR at the child zone—this is required because more than one DNSKEY RR may be present at the child zone apex either because separate KSK and ZSKs are used or due to key-rollover operations. The `algorithm` field defines the algorithm used by the `key-tag`-identified DNSKEY RR at the child zone, which is recommended to be a KSK, and may take one of the following values:

`0` = Reserved

`1` = RSA-MD5—not recommended (RFC 2537)

`2` = Diffie-Hellman (RFC 2539)

`3` = DSA-SHA-1—optional (RFC 2536)

`4` = Elliptic curve—not currently standardized

`5` = RSA-SHA-1—mandatory (RFC 3110)

`6–251` = Currently unassigned

`252` = Indirect (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`253` = Private URI (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`254` = Private OID (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`255` = Reserved

The `digest-type` field defines the digest algorithm being used and may take one of the following values:

`0` = Reserved

`1` = SHA-1—mandatory

`2–255` = Currently unassigned

The `digest` field is the base64 encoding of the digest of the KSK DNSKEY RR at the child zone.

The `dnssec-signzone` utility will optionally generate the DS RR with a file name of `dsset-zonename`; for example, if the zone being signed is `sub.example.com`, the resulting file is called `dsset-sub.example.com`.

As previously stated, the DS RR is included in the parent (signed) zone, which must then be re-signed following its addition. The experimental DNSSEC Lookaside Validation (DLV) system provides an alternative method of creating chains of trust using a DLV RR, which is functionally identical to the DS RR with the exception of the RR type code. DLV is described in Chapter 11.

System Information (HINFO) Record

The System Information RR allows the user to define the hardware type and operating system (OS) in use at a host. The HINFO RR was defined in RFC 1035. For security reasons, these records are rarely used on public servers.

HINFO RR Syntax

```
name  ttl  class   rr      hardware        OS
           IN      HINFO   PC-Intel-700mhz "Redhat Fedora Core 3"
```

If a space exists in either the `hardware` or `OS` field, that field must be enclosed in quotes. There must be at least one space between the `hardware` and `OS` fields. The preceding example illustrates that quotes are not required with the `hardware` field—the spaces have been replaced with `-` (hyphen)—but are required with the `OS` field, since it contains spaces within the field. No validation is performed on the field contents other than the space rules defined previously, which means this record can be used for any purpose; for instance, the fields could contain the name and phone number of technical support for the system. The following example shows the use of the HINFO RR:

```
; zone file fragment for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
....
www       IN      A      192.168.254.8
          IN     HINFO  "AMD 64 4.8GHZ 10TB" "FreeBSD 5.3"
```

The preceding HINFO record is associated with `www.example.com`.

Integrated Services Digital Network (ISDN) Record

The Integrated Services Digital Network RR is the equivalent of an A RR for ISDN Customer Premise Equipment (CPE). It associates the telephone number of the ISDN CPE to a host name. The ISDN RR has EXPERIMENTAL status and is defined in RFC 1183.

ISDN RR Syntax

```
name   ttl class RP  isdn-number   sa
joe        IN    ISDN   1441115551212 001
```

The `isdn-number` is in E.164 format (a telephone number). The telephone number is assumed to begin with the E.164 international dial sequence. There must be no spaces within the field.

The `sa` field defines an optional subaddress used with ISDN multidrop configurations and, if present, is separated from the `isdn-number` field by one or more spaces. If not used, it is omitted. Since the `isdn-number` is an address, not a name, there is no terminating dot.

IPSEC Key (IPSECKEY) Record

The IPSEC Key RR is used for storage of keys used specifically for IPSec operations. Originally, the KEY RR was designed to store such keys generically using an application subtype value.

RFC 3445 limited the KEY RR to DNS security uses only. Using this new RR type means that an application that wishes to establish a VPN (an IPSec service) to a specific host name can query the DNS for an IPSECKEY RR with the host name it wishes to connect to and obtain the relevant details such as the optional gateway and the cryptographic algorithm being used. The IPSECKEY RR is defined by RFC 4025.

IPSECKEY RR Syntax

```
name  ttl  class   rr   prec gwt algorithm gw key-data
joe        IN      IPSECKEY 256   1   2  192.168.2.1 (
                           AQPSKmynfzW4kyBv015MUG2DeIQ3
                           Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                           kfzj31GajIQKY+5CptLr3buXA10h
                           WqTkF7H6RfoRqXQeogmMHfpftf6z
                           Mv1LyBUgia7za6ZEzOJBOztyvhjL
                           742iU/TpPSEDhm2SNKLijfUppn1U
                           aNvv4w==  )
```

The `prec` (precedence) field is used the same way as the `preference` field of an MX RR to define the order of priority. Lower numbers take the highest precedence. Values may lie in the range `0` to `255` only.

The `gwt` field defines the type of gateway and may take one of the following values:

`0` = No gateway (the host supports the IPSec service directly).

`1` = An IPv4 gateway is defined that should be used to access this host.

`2` = An IPv6 gateway is defined that should be used to access this host.

`3` = A named host is present that should be used to access this host.

The `algorithm` field may take one of the values defined here:

`0` = No key is present.

`1` = DSA (RFC 2536).

`2` = RSA (RFC 3110).

`3–255` = Not assigned.

The `gw` field defines the gateway and may be either a single `.` (dot) if the `qwt` field = `0`, an IPv4 address if `qwt` = `1`, an IPv6 address if `qwt` = `2`, or a host name if `qwt` = `3`.

The `key-data` field contains the base64-encoded public key of the algorithm defined in the algorithm field.

Public Key (KEY) Record

The Public Key RR was originally defined in RFC 2535 to be used for the storage of public keys for use by multiple applications such as IPSec, SSH, etc., as well as for use by DNS security methods including the original DNSSEC protocol. RFC 3445 limits this RR to use in DNS security operations such as DDNS and zone transfer due to the difficulty of querying for specific uses—DNS queries operate on the RR type field, whereas the application functionality was defined in the `proto` field (described in the upcoming text) and was therefore not *directly* obtained by a query operation. IPSec (IPSECKEY) and SSH (SSHFP) both have new RR types that allow applications to directly query for the relevant RR.

KEY RR Syntax

```
name  ttl  class   rr     flags proto algorithm key-data
joe        IN      KEY     256    3        5 (
                          AQPSKmynfzW4kyBv015MUG2DeIQ3
                          Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                          kfzj31GajIQKY+5CptLr3buXA10h
                          WqTkF7H6RfoRqXQeogmMHfpftf6z
                          Mv1LyBUgia7za6ZEzOJBOztyvhjL
                          742iU/TpPSEDhm2SNKLijfUppn1U
                          aNvv4w==  )
```

The original definition of this RR was significantly reduced by RFC 3445 as noted previously. The definitions that follow reflect the current RFC 3445 status, and previous values where appropriate are also shown but noted as deprecated. The `flags` field consists of 16 bits in which only bit 7 is now used. In the textual format, this field is represented as a decimal value of either 0 (no flag bits set), in which case the key is used with the SIG(0) or TKEY meta RR to secure DDNS or zone transfer operations, or 256 (bit 7 = `1`), which allows it to still be used in zone signing or verification operations (see Chapter 11) though functionally replaced with the DNSKEY RR in DNSSEC.bis. All other values will be ignored by DNS systems. The `proto` field may only take the value `3`, all other values being deprecated. For historical reasons, previous versions may still exist and are defined here for completeness:

`0` = Reserved

`1` = TLS (deprecated by RFC 3445)

`2` = E-mail (deprecated by RFC 3445)

`3` = DNSSEC (only value allowed by RFC 3445)

`4` = IPSEC (deprecated—replaced by IPSECKEY RR)

`5–255` = Reserved

The `algorithm` field may take one of the following values:

`0` = Reserved

`1` = RSA-MD5—not recommended (RFC 2537)

`2` = Diffie-Hellman—optional, key only (RFC 2539)

`3` = DSA—mandatory (RFC 2536)

`4` = Elliptic curve—not currently standardized

`5` = RSA-SHA-1—mandatory (RFC 3110)

6–251 = Available for IANA allocation

252 = Reserved for indirect keys (see the section "Alternative Cryptographic Algorithms" later in this chapter)

253 = Private URI (see the section "Alternative Cryptographic Algorithms" later in this chapter)

254 = Private OID (see the section "Alternative Cryptographic Algorithms" later in this chapter)

255 = Reserved

■**Note** The original specification of the KEY RR (RFC 2535) only allowed algorithm types 1 to 4 defined previously and was not apparently revised; however, the `dnssec-keygen` utility allows algorithm 5 to be specified, and indeed this algorithm can be used in SIG(0) operations that use the KEY RR. It is therefore shown in the preceding supported list.

KEY RRs are typically generated by the `dnssec-keygen` utility (see Chapter 9), which creates an RR that may be included if appropriate (see Chapters 10 and 11), either directly in the zone file or through the `$INCLUDE` directive.

While various RFCs limit the use of this RR type in a variety of ways, there is in principle nothing to stop the user from using it, and the `dnssec-keygen` utility that creates it, as a general-purpose public key RR for specialized applications such as secure e-mail where the functionality is known to the application and the presence of a KEY RR with the same name as, say, an RP RR could provide some unique functionality.

Key Exchanger (KX) Record

The Key Exchanger RR is provided to allow a client to query a destination host and be provided with one or more alternative hosts. It is primarily intended for use in secure operations such as creation of an IPSec VPN or similar service, though its applicability is much wider. The destination host may not be capable of providing the particular service, but in its corresponding KX RR it can nominate another host that will support the service such as a secure gateway or router, which can be used to route packets to the target host. The IPSECKEY RR replaces many of the functions of this RR type for the particular example described in the defining RFC. The KX RR is defined in RFC 2230.

KX RR Syntax

```
name  ttl  class   rr     preference alt-host
joe        IN      KX     2    rt1.example.com.
```

The `preference` field has exactly the same meaning and use as in the MX RR. It may take a value in the range `0` to `65535`, with lower values being the most preferred. The `alt-host` field defines the host name where a VPN or some other service may be obtained for the current host.

Location (LOC) Record

The Location RR allows the definition of geographic positioning information associated with a host or service name. The LOC RR allows longitude, latitude, and altitude to be defined using the WGS-1984 (NAD-83) coordinate system—a US DoD standard for the definition of geographic coordinates. The LOC RR, which is experimental, was defined in RFC 1876 and is widely deployed, for instance, to allow geographic analysis of Internet backbones. The LOC RR can take a large number of parameters and most often uses the standard parentheses framing to allow them to be written on more than one line for clarity as shown in the following text. Location data may be acquired using GPS equipment or to varying degrees of accuracy from a number of websites such as for GEOnet Names Server (GNS—`http://earth-info.nga.mil/gns/html/index.html`), US Geological Survey's Geographic Names Information System (GNIS—`http://geonames.usgs.gov`), or the *Getty Thesaurus of Geographic Names Online* (`www.getty.edu/research/conducting_research/vocabularies/tgn/`).

LOC RR Syntax

```
name  ttl  class   rr  (
                          lat-d
                          [lat-m [lat-s]]
                          n-s
                          long-d
                          [long-m [long-s]]
                          e-w
                          alt["m"]
                          [size["m"] [hp["m"] [vp["m"]]]]
                          )
yahoo.com.   IN   LOC  37 23 30.900 N 121 59 19.000 W 7.00m 100m 100m 2m
```

The `lat-d` field defines the location latitude in degrees. `lat-m` and `lat-s` are optional fields defining the minutes (`lat-m`) and seconds (`lat-s`) and, if omitted, default to zero. The field `n-s` is mandatory and can take the value `N` (north) or `S` (south).

The `long-d` field defines the location longitude in degrees. `long-m` and `long-s` are optional fields defining the minutes (`long-m`) and seconds (`long-s`) and, if omitted, default to zero. The field `e-w` is mandatory and can take the value `E` (east) or `W` (west).

The `alt` field defines the location altitude and can be either positive or negative in the range `-100000.00` to `42849672.95` meters.

The `size` field is optional and is the diameter of the circle that encompasses the location, that is, it represents the positional accuracy. If omitted, `1m` is assumed.

The `hp` field is the optional horizontal accuracy and defaults to `10,000m` (meters). The `vp` field is the vertical accuracy and, if omitted, defaults to `10m` (meters). The defaults selected in these two parameters represent the typical size of zip/postal code data.

■**Note** The datum (base reference) used by the LOC record is WSG-1984 or NAD-83 (North American Datum) used by the GPS system. In some cases, geographic data uses NAD-27 as the datum, which is not the same—always verify the datum being used. Geographic data can be presented in decimal degrees. To convert decimal degrees to minutes and seconds, multiply the fractional part by 60 to get minutes and fractional minutes, and then multiply the fractional minutes by 60 to get seconds and fractional seconds.

The LOC record can be associated with any host or the domain. The following shows individual LOC RR examples using published records or publicly available data from the preceding sources and including a number of formats:

```
; yahoo.com LOC RR
yahoo.com.   IN   LOC  37 23 30.900 N 121 59 19.000 W 7.00m 100m 100m 2m
; Stamford, CT, US - Harbor Lighthouse
             IN   LOC  41 00 48 N 73 32 21 W 10m
; Kilmarnock, Scotland UK
             IN   LOC  (
                         55 ;latitude
                         38 ; seconds omitted
                         N
                         4 32 W ; longitude
                         100m  ; altitude - pure guess
                       )
```

The first example is a published RR. Since it is associated with the domain, it presumably references corporate headquarters, but each subdomain or host—possibly representing a separate geographic location—could publish LOC records. The second and third RRs were created using random locations from the databases referenced previously. There are, as far as the author knows, no registered domains for either the Stamford Harbor Lighthouse or the town of Kilmarnock in Scotland. Neither does either entity publish a LOC RR! The preceding databases typically do not provide altitude data, and while it is reasonable to suppose a lighthouse is close to sea level, the height of the town of Kilmarnock is entirely fictitious. The required accuracy of the data will depend on the reason for publishing an LOC RR, and in many cases the longitude and latitude may suffice to give location data.

Mailbox (MB) Record

The Mailbox RR defines the location of a given domain e-mail address. The MB RR has EXPERIMENTAL status and is defined in RFC 1035. The MB record is not widely deployed—the MX RR is the dominant mail record.

MB RR Syntax

```
name   ttl class rr  mailbox-host
joe        IN    MB  fred.example.com.
```

The `mailbox-host` field defines the host where the mailbox is located. The `mailbox-host` must have a valid A RR. The `name` field is the mailbox name written in the standard DNS format for mailboxes, that is, the first `.` (dot) is replaced with an `@` (commercial at sign) when constructing the e-mail address. The example fragment that follows illustrates that the mailbox for the domain administrator, `hostmaster.example.com.` (defined in the SOA record), is located on the host `bill.example.com`, whereas the normal mail host is `mail.example.com`. The mail address, when constructed, is the normal RFC 822 format, which is `hostmaster@example.com` in the following example:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN  MX     10  mail.example.com.
hostmaster    IN  MB         bill.example.com.
bill          IN  A          192.168.254.2
mail          IN  A          192.168.254.3
....
```

The preceding example requires the mail system to look for an appropriate MB record—almost none do. Most mail software looks for the presence of an MX RR and delivers mail to this specified host—`mail.example.com` in the preceding fragment. To achieve the same result in the preceding case, the mail system at `mail.example.com` would be configured to forward mail to the mailbox `hostmaster` on the host `bill.example.com`.

Mail Group (MG) Record

The Mail Group RR defines a group name and the mail boxes that are members of that group. The MG RR has EXPERIMENTAL status and is defined in RFC 1035.The MG record is not widely deployed—the MX RR is the dominant mail record.

MG RR Syntax

```
name   ttl class rr  mailbox-name
admins     IN    MG  fred.example.com.
```

The `mailbox-name` field defines the mailbox names that are part of the mail group. Mail sent to the group will be sent to each mailbox in the group. Each member of the mail group must be defined using an MB RR. The `mailbox-name` field is written in the standard DNS format for mailboxes, that is, the first . (dot) is replaced with an @ (commercial at sign) when constructing the e-mail address. The example fragment that follows illustrates that the mailbox for the domain administrator, hostmaster.example.com. (defined in the SOA record), is a mail group and will cause mail to be sent to phil.example.com (phil@example.com) and sheila.example.com (sheila@example.com), both of whose MB RRs define the final destination for the mail:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN  MX     10  mail.example.com.
hostmaster    IN  MG         phil.example.com.
              IN  MG         sheila.example.com.
phil          IN  MB         bill.example.com.
sheila        IN  MB         pc.example.com.
....
pc            IN  A          192.168.254.4
bill          IN  A          192.168.254.2
mail          IN  A          192.168.254.3
....
```

The preceding example needs the mail system to look for appropriate MG and MB RRs—almost none do. Most mail software looks for the presence of an MX RR and delivers mail to this specified host—mail.example.com in the preceding fragment. To achieve the same result in this case, the mail system at mail.example.com would have to be configured to forward mail for the mailbox hostmaster to both phil@example.com and sheila@example.com.

Mailbox Renamed (MR) Record

The Mailbox Renamed RR allows a mailbox name to be aliased (or forwarded) to another mailbox name. The MR RR has EXPERIMENTAL status and is defined in RFC 1035.The MB record is not widely deployed—the MX RR is the dominant mail record.

MR RR Syntax

```
name   ttl class rr  real-mailbox
joe        IN    MR  fred.example.com.
```

The real-mailbox field defines the aliased, or real, mailbox that must be defined with an MB RR. Mail sent to name will be forwarded to real-mailbox. The name and real-mailbox fields are the mailbox names written in the standard DNS format for mailboxes, that is, the first . (dot) is replaced with an @ (commercial at sign) when constructing the e-mail address. The example fragment that follows illustrates that the mailbox for the domain administrator, hostmaster.example.com. (defined in the SOA record), is forwarded to phil.example.com, located on the host bill.example.com, whereas the normal mail host is mail.example.com. The mail address when constructed is the normal format, which is hostmaster@example.com in the following example:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN  MX     10  mail.example.com.
hostmaster    IN  MR         phil.example.com.
phil          IN  MB         bill.example.com.
....
bill          IN  A          192.168.254.2
mail          IN  A          192.168.254.3
....
```

The preceding example needs the mail system to look for both MR and MB RRs—almost none do. Most mail software looks for the presence of an MX RR and delivers mail to this specified host—mail.example.com in the preceding fragment. To achieve the same result in this case, the mail system at mail.example.com will be configured to forward mail for the mailbox hostmaster@example.com to phil@example.com on the host bill.example.com.

Mailbox Mail List Information (MINFO) Record

The Mailbox Mail List Information RR defines the mailbox administrator for a mail list and optionally a mailbox to receive error messages relating to the mail list. The MINFO RR is experimental and is defined in RFC 1035.The MINFO RR is not widely deployed—the MX RR is the dominant mail record.

MINFO RR Syntax

```
name   ttl class rr     admin-mailbox     [error-mailbox]
users      IN   MINFO  fred.example.com. joe.example.com.
```

The admin-mailbox field defines the mailbox to which mail related to the mail list name will be sent. The optional error-mailbox will receive mail concerning errors relating to the mail list name. Both admin-mailbox and error-mail-box must be defined with an MB RR. The name, admin-mailbox, and error-mailbox fields are mailbox names written in the standard DNS format for mailboxes, that is, the first . (dot) is replaced with an @ (commercial at sign) when constructing the e-mail address.

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN  MX     10  mail.example.com.
people        IN  MINFO      admin.example.com. broken.example.com.
admin         IN  MB         bill.example.com.
broken        IN  MB         bill.example.com.
....
bill          IN  A          192.168.254.2
mail          IN  A          192.168.254.3
....
```

The preceding example needs the mail system to look for both MINFO and MB RRs—almost none do. Most mail software looks for the presence of an MX RR and delivers mail to this specified host—`mail.example.com` in the preceding fragment. Mail software typically treats mail list management as a separate or loosely coupled function.

Mail Exchange (MX) Record

The Mail Exchanger RR specifies the name and relative preference of mail servers (mail exchangers in the DNS jargon) for the zone. The MX record was defined in RFC 1035.

MX RR Syntax

```
name           ttl  class   rr  preference name
example.com.        IN      MX  10   mail.example.com.
```

The `preference` field is relative to any other MX record for the zone and may take the value in the range `0` to `65535`. Low values are *more* preferred. The common preferred value of `10` is just a convention and allows more preferred servers to be added without changing any other records; that is, if the most preferred server was set to `0` (again a common practice), then if an *even more preferred* server was introduced, two records would have to be changed! Any number of MX records may be defined with either different or equal preference values. The effect of using multiple MX RRs with an equal preference is described in the section "DNS Load Balancing" located in Chapter 8. If the mail host lies in the same zone, it requires an A RR. The right-hand name used in an MX RR should not point to a CNAME record (see the discussion of this point in the section "Canonical Name (CNAME) Record" earlier). MX records frequently use the wildcard * (asterisk) in the name field, which is described in Chapter 8. The example that follows shows a domain using three mail servers, two of which are hosted within the domain. The third and least preferred is hosted externally.

```
; zone fragment for example.com
; mail servers in the same zone
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN      MX     10  mail  ; unqualified name
; the line above is functionally the same as the line that follows
; example.com. IN       MX     10  mail.example.com.
; any number of mail servers may be defined
              IN      MX     20  mail2.example.com.
; an external back-up
              IN      MX     30  mail.example.net.
; the local mail servers need an A record
mail          IN      A      192.168.0.3
mail2         IN      A      192.168.0.3
```

The following fragment shows two mail servers, neither of which is located in the domain and hence do not require A RRs:

```
; zone fragment for example.com
; mail servers not in the zone
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. root.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
; mail servers not in zone - no A records required
               IN     MX     10  mail.example.net.
               IN     MX     20  mail.example.org.
```

Subdomain MX Records

Subdomains can be fully delegated, in which case the mail servers are defined in the subdomain zone files. This process is described in Chapter 8. This book uses the term *virtual* (or *pseudo*) subdomains, which use a single zone file to provide subdomain style addressing. The following example shows a *virtual* subdomain—the domain and all subdomain definitions are contained in a single zone file:

```
; zone fragment for example.com
; subdomain name servers
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN     SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               2h         ; refresh =  2 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
....
; mail server for main domain
              IN      MX 10  mail.example.com.
; A record for mail server earlier
mail          IN      A      192.168.0.5
; other domain level hosts and services
....
; subdomain definitions
$ORIGIN us.example.com.
              IN      MX 10  mail
; preceding record could have been written as
; us.example.com.   IN  MX 10 mail.us.example.com.
; optional - define the main mail server as backup
              IN      MX 20 mail.example.com.
; A record for subdomain mail server
mail          IN      A      10.10.0.29
; the preceding record could have been written as
; mail.us.example.com. A 10.10.0.29 if it is less confusing
....
; other subdomain definitions as required
```

An alternative way of defining the preceding that groups the MX records together is shown here:

```
; zone fragment for example.com
; subdomain mail servers
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN     SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               2h         ; refresh =  2 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
....
```

```
; mail server for main domain
              IN      MX 10  mail.example.com.
; mail server for subdomain 'us'
us            IN      MX 10  mail.us.example.com.
us            IN      MX 20  mail.example.com.
; A record for main mail server earlier
mail          IN      A      192.168.0.5
; other domain level hosts and services
....
; subdomain definitions
$ORIGIN us.example.com.
; A record for subdomain mail server
mail          IN      A      10.10.0.29
; the record above could have been written as
; mail.us.example.com. A 10.10.0.28 if it is less confusing
....
; other subdomain definitions as required
```

Naming Authority Pointer (NAPTR) Record

The Naming Authority Pointer Record RR is part of the Dynamic Delegation Discovery System (DDDS), which is defined in RFCs 3401, 3402, 3403, and 3404. The NAPTR RR is a generic record that defines a *rule* that may be applied to private data owned by a client application, for example, the ENUM telephony application, to yield a result that is meaningful to that application. The private client data is referred to as the *Application Unique String (AUS)*. Multiple NAPTR RRs may be present to create a *rule set*. NAPTR RRs are meaningful only in the context of the application that uses them. The example that follows illustrates the use of the rules in the context of a client application, ENUM telephony, to illustrate one use of the NAPTR RR. You are, however, cautioned that other client applications can and do exist and that appropriate documentation for the target application will describe how the result of applying the NAPTR rule will be used. It is further possible that the target application may redefine the use of certain fields within the NAPTR RR. The NAPTR RR is defined in RFC 3403.

NAPTR RR Syntax

```
name   ttl class rr     order pref flag svc regexp replace
users      IN  NAPTR  (
                        10  ; order
                        10  ; preference
                        "u" ; flag
                        "E2U+sip"  ; service
                        "!^.*$!sip:1234@sip.example.com!"  ; regexp
                        .  ; no replacement
                      )
```

The `order` field defines the order in which NAPTR RRs should be processed. It is a 16-bit unsigned value and may take the range 0 to 65565, with low values having the highest priority. If two NAPTR RRs have the same `order`, then the `pref` field is used to select the first NAPTR RR to be processed.

The pref field defines the preference within order to select the NAPTR RR to process first. It is a 16-bit field and may take the values 0 to 65535, the lowest value being the most preferred. The sense in which pref and order are used is that NAPTR RRs of higher order are not used until the client has examined all those of the lower order, and only if none is acceptable (for example, the protocol is not supported) should it use the higher order values, whereas pref indicates a user preference that the client is free to ignore if it wishes.

The optional flag field may be used to indicate an action to be taken by the application. The flag is a *quoted string* (it is enclosed in quotes) and may take any case-insensitive value from the set A to Z or 0 to 9; if no flag is present, an empty string ("") is used. The flag field's values are defined by the application and not by the NAPTR RR specification. The following values are *conventions* used by the ENUM (RFC 3761), SIP (RFC 3263), and URN (RFC 3404) applications and illustrate the functionality that may be provided by the flag field. The flag "a" defines a *terminal condition*—this NAPTR RR generates a complete result—and indicates that the result of the processing of this NAPTR rule will yield a name that can be used in a query for an address record (either A or AAAA RRs). The flag "s" defines a terminal condition where the result is a name that may be used to query for an SRV RR. The flag "u" defines a terminal condition where the result will be a URI. The flag "p" indicates that this is the last rule that obeys the NAPTR RR specification. On the surface this might imply a *terminal condition*, but the client application is free to continue processing using any private rules that lie outside the definition of the NAPTR RR, so it is terminal only as far as the NAPTR record is concerned, not necessarily the client application.

The optional svc field defines the service parameters used by the application. The parameters are contained within a quoted string, and their meaning is defined by the client application, not the NAPTR RR. If not present, an empty string ("") must be defined. As an example of the use of this field, the ENUM telephony application (RFC 3761) defines the svc field to be of the following format:

```
rs+protocol[+protocol]
```

where the rs field defines a *resolution service*, usually a mnemonic that indicates a transformation rule known to the client application and that is applied to the client data (the AUS). In the ENUM example that follows, the rs value is E2U (which defines the rules for ENUM-to-URI transformation). This field may be 1 to 32 alphanumeric characters and must start with an alphabetical character. The + (plus sign) is a separator and must be present. The protocol field may be any protocol known to the client application, for example, SIP, H323, or goobledeygook. This field may be a maximum of 32 alphanumeric characters and must start with an alphabetical character. More than one such protocol field may be present, each separated by a + (plus sign). To illustrate the point that this field is application defined, the URN application (RFC 3404) and SIP (RFC 3263) use the same format but *currently* reverse the order of the rs and protocol element!

The optional regexp field is a quoted string enclosing a POSIX Extended Regular Expression (ERE—defined in IEEE POSIX 1003.2 Section 2.8), augmented with a substitution expression defined in RFC 3402 (and loosely based on that used in the sed Unix utility), which is used to transform the client data (AUS). If the field is not present, an empty string ("") must be present. The formal grammar of the regexp field is shown here:

```
delim ere delim repl delim flag
```

where delim is a delimiting character (it may be / or !) used to separate parts of the field. The same delimiting character must be used throughout any single regexp field. The ere field is a valid Extended Regular Expression. repl is the replacement string. flag is optional and may take the value i to indicate a case-insensitive ere is to be used. The resulting repl field *may* also be normalized to lowercase as a consequence of using this flag. The regexp supports a back-reference feature whereby strings found within enclosing parentheses in the ere field may be substituted in the repl field by a numeric reference (1 to 9) indicating the order in which they were found. The following example illustrates this process. Assume the ere field contains the following:

```
(A(B(C)DE)(F)G)
```

The following back references in the repl field may be used to access the values:

```
\1  = ABCDEFG
\2  = BCDE
\3  = C
\4  = F
\5..\9  = error - no matching subexpression
```

The optional replace field is a domain name that will replace the client data. If the field is not used, a single . (dot) must be present.

The replace and regexp fields are mutually exclusive. It is an error for both to be present in the same NAPTR RR.

The NAPTR RR is a complex and powerful RR providing generic capability to any client application. Its functionality can only be understood in the context of the application that uses it. To illustrate how the NAPTR RR can be used, the following summarizes the order of processing of an NAPTR RR by the ENUM application:

1. The client application, say a VoIP SIP User Agent (UA), receives an E.164 telephone number (the client data or AUS—in this example, we assume +44-111-555-1212). The ENUM application within the SIP UA applies what is known as its *Well-Known First Rule* (a private rule known by the ENUM application), which in this case creates a domain name by stripping all nonnumeric values, reversing the number, and appending E164.ARPA to the end of it to create 2.1.2.1.5.5.5.4.4.E164.ARPA.
2. The client ENUM application issues a DNS query for NAPTR RRs with this domain name.
3. Zero or more NAPTR RRs may be returned.
4. Assuming at least one NAPTR RR is returned, the order and pref fields defined earlier are inspected to determine which NAPTR will be processed first.
5. If a regexp file is present, it will be applied to the private client data (the AUS).
6. The svc field will then be inspected and the E2U transformation algorithm applied to the results of the regexp output to create a URI. The resulting URI will then be used to find the target.
7. The protocol field within svc is used to indicate the protocol to be used to communicate with the target.

The zone fragment shown here illustrates the use of the NAPTR with the ENUM service (RFC 3761) and defines an NAPTR RR for the number +44-111-555-1212 (an E.164 format number) within a zone file describing the NXX (exchange code) 555, within area code (111), within the country code (44). As noted previously, the ENUM application transforms the number +44-111-555-1212 into a DNS query for `2.1.2.1.5.5.5.1.1.1.4.4.E164.ARPA`. The various fields are further described following the fragment.

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN 5.5.5.1.1.1.4.4.E164.ARPA.
....
2.1.2.1     NAPTR (
   10   ;order
   100  ; preference
   "u" ; flag - only one allowed by ENUM
   "E2U+sip" ;svc
   "!^\\+44111555(.#)$!sip:7\\1@sip.example.com!"
   . ; no replace field
   )
            NAPTR 10 101 "u" "E2U+pres" "!^.*$!mailto:sheila@example.com!" .
....
```

In the preceding example, all the `order` fields are the same, so the `pref` field will be used to determine which record is used first—in the preceding case, the NAPTR with a `pref` of 100. The `regexp` field `!^\\+44111555(.#)$!sip:7\\1@sip.example.com!`, when applied to 441115551212 (the AUS), will result in `sip:71212@sip.example.com`—in this case, a `7` is appended to the last four digits of the supplied phone number (using the extracted back-reference of `\1`), but it could be any appropriate algorithm. The application will then inspect the `svc` field (`E2U+sip` in the preceding example) and initiate a SIP session using the URI `sip:71212@sip.example.com` to contact the user. If this fails, it may decide to process the NAPTR RR with a `pref` of 101, which will result in a contact to `sheila@example.com` using a presence service of some kind.

■**Note** RFC 3403, which defines the NAPTR, describes an ENUM telephony example that has been updated by RFC 3761. The ENUM example shown is compatible with RFC 3761, whereby the order of resolution services (`rs` and `protocol`) have been reversed.

Name Server (NS) Record

Name Server RRs are used to list all the name servers that will respond authoritatively for the domain. NS RRs for a given zone are defined in two places: the child zone (where they are authoritative) and the parent zone (where they are not authoritative). Thus the zone `example.com` (the child zone) contains NS RRs defining the authoritative name servers for the zone, and the `.com` zone (the parent) has corresponding NS RRs, called the *delegation point*, that are used to create a referral to the authoritative name servers for the domain or zone. The requirement is that there are a minimum of two authoritative name servers and hence a minimum of two NS RRs for every zone. The NS RR was defined in RFC 1035.

NS RR Syntax

```
name           ttl  class   rr     name
example.com.        IN      NS      ns1.example.com.
```

NS RRs for the zone are defined at the zone apex or root, that is, they have the same name as the domain or zone as shown in the preceding example. By convention, name servers are defined immediately after the SOA record, but they can be defined anywhere convenient in the zone file. The name server defined in the SOA record, the so-called Primary Master, must have a corresponding NS RR. There is no requirement that any name servers, including the name server defined in the SOA RR for the zone, are contained within the domain for which they are authoritative. NS RRs define name servers that respond authoritatively for the zone; since both master (Primary) and slave (Secondary) servers perform this function, they are not differentiated in any way in NS RRs. The designation of master and slave is a purely operational decision. The NS RRs defined in the zone file (and in its parent) are publicly visible name servers. There is no need or requirement to define all the name servers in NS RRs for a specific zone file—it is possible to hide, say, a zone master name server while making only the slaves publicly visible as long as the requirement for two visible name servers is satisfied.

If the name server lies within the domain, it should have a corresponding A (or AAAA) record as would be defined for any host in the domain. The A RRs that define name servers that lie within the domain are frequently called *glue records*. Glue records are essential only for referrals from a parent zone. In practice, glue records are used for two purposes:

1. To speed up queries—and reduce DNS load—by providing the name and IP addresses (the glue) for all authoritative name servers, both within and external to the domain. The root and TLD servers, for example, provide this information in all referrals to remove the need for a subsequent query for an IP address of the name server. In the case of the TLD servers, the glue data is not obtained from the domain's zone file but from the Registrar when the domain name is registered.

2. To break the query deadlock for referrals that return name servers within the domain being queried. Assume a query for a domain, say the A RR for `www.example.com`, returns a referral containing the name but not the IP address of a name server, say `ns1.example.com`, which lies *within the domain* `example.com.` Since the IP address of the name server is not known, this will naturally result in a query for the A RR of `ns1.example.com`, which will return, again, a referral with the name but not the IP of `ns1.example.com`! When the glue record (an A or AAAA RR) is provided, both the name and the IP address are returned.

When dealing with any zone file, the A (or AAAA) RRs for the name servers that lie within the domain are not strictly glue records, they are conventional A RRs; but if a subdomain is being delegated from the zone file, the A (or AAAA) RRs for the subdomain name servers that lie inside the subdomain are glue records and are absolutely essential. This point is illustrated in the example fragments that follow.

■ **Note** It is worth stressing what may be to most readers an obvious point. The name servers referenced in NS records must all be zone masters or slaves. That is, they must respond *authoritatively* for the domain. They must have been positively configured to perform this function (for BIND this means `type slave` or `type master` in the zone clause) and must have a full copy of the zone file obtained from the local file-system (master or Primary) or via zone transfer (slave or Secondary). A caching server *cannot* perform this function. Name servers defined in NS RRs that do not respond authoritatively are said to be lame servers or the zone is said to have lame delegation and will generate lots of nasty log entries on DNS servers across the world—this configuration error is very visible to the DNS community.

The following zone file fragment shows two name servers, both of which lie within the domain:

```
; zone fragment for example.com
; name servers in the same zone
$TTL 2d ; default TTL is 2 days
$ORIGIN example.com.
example.com.  IN     SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN      NS     ns1  ; unqualified name
; the preceding line is functionally the same as the line that follows
; example.com. IN      NS     ns1.example.com.
; at least two name servers must be defined
              IN      NS     ns2.example.com. ;FQDN
; the in-zone name server(s) should have an A record
ns1           IN      A      192.168.0.3
ns2           IN      A      192.168.0.3
```

The next fragment shows two name servers, both of which lie outside the zone:

```
; zone fragment for example.com
; name servers not in the zone
$TTL 2d ; default TTL is 2 days
$ORIGIN example.com.
example.com.  IN     SOA   ns1.example.net. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
```

```
; name servers not in zone - no A records required
               IN      NS     ns1.example.net.
               IN      NS     ns2.example.net.
```

The following zone file delegates a subdomain `us.example.com` and shows the use of glue records:

```
; zone fragment for example.com
; name servers in the same zone
$TTL 2d ; default TTL is 2 days
$ORIGIN example.com.
@              IN      SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               2h         ; refresh =  2 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
; main domain name servers
              IN      NS     ns1.example.com.
              IN      NS     ns2.example.com.
; A records for name servers above - not glue records
ns1           IN      A      192.168.0.3
ns2           IN      A      192.168.0.4
....
; subdomain definitions
$ORIGIN us.example.com.
; two name servers for the subdomain
@             IN      NS     ns3.us.example.com.
; the record above could have been written as
; us.example.com. IN NS ns3.us.example.com.
; OR as simply
;      IN NS   ns3
; the next name server points to ns1 above
              IN      NS     ns1.example.com.
; address record for subdomain name server - essential glue record
ns3           IN      A      10.10.0.24 ; glue record
; the record above could have been written as
; ns3.us.example.com. A 10.10.0.24 if it is less confusing
```

In the preceding fragment, the NS RRs at the zone apex (the first two NS RRs) are part of the authoritative data for the zone `example.com`. The NS RRs for the subdomain `us.example.com` (the last two NS RRs shown) and the corresponding A RR is not part of the authoritative data for the zone `example.com`.

Network Service Access Point (NSAP) Record

The Network Service Access Point RR is the equivalent of an A RR for ISO's Open Systems Interconnect (OSI) system in that it maps a host name to an endpoint address. The NSAP is the OSI equivalent of the IP address and is hierarchically structured. The NSAP RR has informational status and is defined in RFC 1706. The NSAP address format is defined in ISO/IEC 8348 (`www.iso.org`). NSAP addresses are vaguely similar to IPv6 addresses in that they have a hierarchical organization, use a hexadecimal representation format, and are 128 bits long.

NSAP RR Syntax

```
name   ttl class rr  nsap-address
joe        IN    NSAP   0x47.0005.80.005a00.0000.0001.e133.ffffff000161.00
```

The `nsap-address` is the NSAP address of the end system. The NSAP address field begins with the literal string "`0x`", which will be familiar to C/C++ programmers and indicates the following field is hexadecimal. The dots within the `nsap-address` field are used for readability reasons only and do not appear in the binary representation. Since the `nsap-address` is an address, not a name, there is no terminating dot.

The following fragment shows a dual-stack (OSI/IP) host, `fred.example.com`, which is reachable by an IPv4 address and an NSAP address:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
              IN  MX    10  mail.example.com.
....
fred          IN  A     192.168.254.2
              IN  NSAP  0x47.0005.80.005a00.0000.0001.e133.ffffff000161.00
....
mail          IN  A     192/168.254.3
....
```

NSAPs may be reverse mapped using the domain `NSAP.INT` and normal PTR RRs. The reverse map is constructed in a similar manner to that defined for IPv6 (see Chapter 5 for full explanation) using a *nibble* format in which each character of the address is reversed, separated with a `.` (dot), and placed under the `NSAP.INT` domain. The example that follows shows a reverse-map fragment for the NSAP defined in the previous fragment:

```
; reverse zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN 3.3.1.e.1.0.0.0.0.0.0.0.0.0.a.5.0.0.0.8.5.0.0.0.7.4.NSAP.INT.
```

```
example.com. IN       SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               3h         ; refresh =  3 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
....
0.0.1.6.1.0.0.0.f.f.f.f.f.f  IN    PTR  fred.example.com.
```

In the example forward-mapping zone file, the host `fred.example.com` was shown as supporting a dual OSI/IP stack. The reverse maps for the IPv4 address and the NSAPs are constructed as separate zone files.

Next Secure (NSEC) Record

The Next Secure RR is part of the DNSSEC.bis revision (see Chapter 11) and is designed to provide two forms of what is called in quaint jargon *proof of nonexistence* or *denial of existence*. The first form allows a query to verify that a host name does *not* exist. Each host name has a corresponding NSEC RR that points to the next valid host name in the zone. The NSEC RRs provide a chain of valid host names—by implication anything not in this chain does not exist. In the second form, the NSEC RR contains a list of RR types that have the same name as the NSEC RR—again by implication, any RR type not in the list does not exist. NSEC RRs are generated automatically by the `dnssec-signzone` utility (described in Chapter 9). The NSEC RR is defined in RFC 4034.

NSEC RR Syntax

```
name  ttl  class   rr     next-name (rr-list)
joe        IN      NSEC     joes    (A TXT RRSIG NSEC)
```

The `next-name` field defines the next host name in the zone file. NSEC RRs are added during the `dnssec-signzone` process to each RR with a particular name to form a continuous chain through the zone file. If the RR to which the NSEC is added is the last in the file, the `next-name` points back to the SOA RR, thus creating a loop. Once the zone file is signed (see Chapter 11), it is possible to verify that any name does or does not exist in the zone file. The `rr-list` field defines all the RR types that exist with the same name as the NSEC RR. Since the NSEC RR is used only in DNSSEC.bis signed zones (see Chapter 11), the `rr-list` will always contain as a minimum the NSEC RR and its accompanying RRSIG RR. The `rr-list` makes it possible to verify that there is, say, an A RR for a host name but not, say, a KEY RR. The example that follows shows how the NSEC RR is used, including the loopback to the beginning of the zone file: if *a user-defined* RR exists at a particular host name (see "User-Defined RRs" later in the chapter), then it will be included in the list of RR types using the normal syntax, for example, TYPE6235.

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
mail        IN      A      192,168.2.3
            IN      AAAA   2001:db8::3
            IN      TXT "one upon a time"
            IN      KX   10 bill.example.com.
            IN      RRSIG
            IN      NSEC www  (A TXT KX AAAA NSEC RRSIG)
www         IN      AAAA   2001:db8::4
            IN      A      192.168.2.4
            IN      NSEC   @  (A AAAA) ; loops back to SOA
```

The NSEC RR is typically generated as part of a zone signing process using the utility `dnssec-signzone` (see Chapter 9). The NSEC RR is the subject of ongoing work in the standards committees, since by following the NSEC chain for a particular domain the entire domain may be enumerated fairly quickly. Without the NSEC RR, the domain can still be enumerated by exhaustive search, which can take some time and is more likely to be caught by intrusion detection systems. It is worth emphasizing, however, that data cannot be hidden in a publicly visible name server—after all, the point of it being in the DNS is that it can and will be made visible. If records need to be protected, then techniques such as Stealth servers must be used (see Chapter 4).

Pointer (PTR) Record

The Pointer RR is used to reverse map an IP address to a host name. PTR RRs are used for both IPv4 and IPv6 addresses, as well as others such as NSAP. Pointer records are the opposite of A RRs (or AAAA RRs for IPv6), which are used to forward map hosts to IP addresses. The PTR RR was defined in RFC 1035.

PTR RR Syntax

```
name ttl  class   rr     host-name
15          IN     PTR    www.example.com.
```

The left-hand `name` field in a PTR RR typically looks like a number but is treated as a name; that is, if it is not terminated with a dot, it is an unqualified name, and then `$ORIGIN` substitution takes place. The right-hand `host-name` field *must* be an FQDN; otherwise very bizarre results will occur—this is illustrated in the examples that follow. The `$ORIGIN` directive in a reverse-map zone file is essential if you wish to remain sane. The following fragment defines a reverse-map zone file for the IPv4 address range 192.168.23.0 to 192.168.23.255:

```
; Reverse map for 192.168.23.0
$TTL 12h
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@             IN      SOA   ns1.example.com. hostmaster.example.com. (
                              2003080800 ; serial number
                              3h         ; refresh
                              15m        ; update retry
                              3w         ; expiry
                              3h         ; minimum
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
....
2             IN      PTR     joe.example.com. ; right-hand FQDN names
; 2 is an unqualified name and could have been written as
; 2.23.168.192.IN-ADDR.ARPA. IN PTR  joe.example.com.
....
15            IN      PTR     www.example.com.
....
17            IN      PTR     bill.example.com.
....
254            IN      PTR     fred.mydomain.com.
```

In the preceding fragment, the IP address 192.168.23.2 will return the host name `joe.example.com` to a PTR query. As noted earlier, the right-hand name must be an FQDN (it must end with a dot) because of the `$ORIGIN`. If the dot were erroneously omitted, then `joe.example.com` would become `joe.example.com.23.168.192.IN-ADDR.ARPA.`—not the desired result. While it is good practice, is it not essential to define all IP addresses in the reverse-map zone file. The addresses `0` and `255` in the preceding example file (it is based on a Class C private address range) are designated the multicast (`0`) and broadcast (`255`) addresses for the class and are not defined in the reverse map.

IPv6 and IPv4 addresses cannot be mixed in the same file as they can for forward-map zone files. IPv6 addresses are mapped under the domain `IP6.ARPA`, whereas IPv4 addresses are mapped under the `IN-ADDR.ARPA` domain. IPv6 reverse maps use a nibble domain name format defined in Chapter 5. The following fragment illustrates the use of the PTR RR to reverse map the IPv6 addresses 2001:db8:0:1::1, 2001:db8:0:1::2, 2001:db8:0:2::1, and 2001:db8:0:2::1:

```
; reverse IPV6 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN 0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; sn = serial number
                        12h         ; refresh = refresh
                        15m        ; retry = update retry
                        3w         ; expiry = expiry
                        2h         ; min = minimum
                        )
```

```
; name servers Resource Recordsfor the domain
          IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to a host name
; hosts in subnet ID 1
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0         IN      PTR     ns1.example.com.
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0         IN      PTR     mail.example.com.
; hosts in subnet ID 2
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0         IN      PTR     joe.example.com.
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0         IN      PTR     www.example.com.
```

Chapter 5 defines alternative methods by which the IPv6 reverse maps may be organized to reduce the sheer size of the host addresses required.

X.400 to RFC 822 E-mail (PX) Record

The X.400 to RFC 822 E-mail RR allows mapping of ITU X.400 format e-mail addresses to RFC 822 format e-mail addresses using a MIXER-conformant gateway. The PX RR is defined in RFC 3163. The X.400 mail address format is defined by X.400 and X.402 (`www.itu.int`). X.400 uses an addressing scheme that ends with a country code and has no equivalent of a generic noncountry code entity such as `.com` or `.org`; the address mappings defined within the RFC are thus limited to country code–based domains (ccTLDs) or require an explicit mapping of the gTLD to a country code.

PX RR Syntax

```
name          ttl class rr  pref 822-domain x.400-name
*.example.com.    IN    PX  10   example.com. PRMD-example.ADMD-p400.C-nl.
```

The `pref` field is the same as used by the MX RR in that it takes the value `0` to `65535` and indicates the relative preference of an X.400 name. Lower values are the most preferred, that is, `10` is more preferred than `20`. The `822-domain` field is the domain name to which this PX RR applies. The `x.400-name` field defines the X.400 address to which mail will be sent by the MIXER gateway.

The following fragment sends all of `example.com`'s incoming mail to an X.400 mail system in Holland.

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
example.com.     IN  SOA   ns1.example.com. hostmaster.example.com. (
                   2003080800 ; serial number
                   3h         ; refresh =  3 hours
                   15M        ; update retry = 15 minutes
                   3W12h      ; expiry = 3 weeks + 12 hours
                   2h20M      ; minimum = 2 hours + 20 minutes
                 )
                 IN  NS      ns1.example.com.
                 IN  NS      ns2.example.com.
*.example.com.   IN  PX  10  example.com.  PRMD-example.ADMD-p400.C-nl.
ns1              IN  A       192.168.254.2
ns2              IN  A       192.168.254.3
www              IN  A       192.168.254.4
```

In the preceding example, the wildcard is used to map every name that does not have another record in the zone file to the X.400 gateway function. In the preceding zone file, every name except `ns1.example.com`, `ns2.example.com`, and `www.example.com` will be sent to the MIXER gateway. Because of the wildcard, the zone does not *require* an MX RR, but the sending mail system does need to be aware of, and explicitly request, the PX RR—most mail systems only check for an MX RR, which may significantly reduce the effectiveness of the PX RR. An alternative strategy would be for the domain to publish a normal MX RR and for the receiving MTA to send to the MIXER gateway via a local mapping or configuration option. A PX RR can be constructed to use a single mailbox mapping, rather than the wildcard mapping, as shown in the line that follows:

```
fred.example.com.  IN  PX  10  fred.example.com.  O-ab.PRMD-net2.ADMDb.C-nl.
```

Responsible Person (RP) Record

The Responsible Person RR allows an e-mail address and some optional human-readable text to be associated with a host. The RP RR is experimental and is defined in RFC 1183. Due to privacy and spam considerations, RP records are not widely used on public servers but can provide very useful contact data during diagnosis and debugging network problems.

RP RR Syntax

```
name   ttl class RP  email txt-rr
joe        IN    RP  fred.example.com. joe.people.example.com.
```

The `email` field is constructed in the normal method for e-mail addresses within the DNS where the first `.` (dot) is replaced with a `@` (commercial at sign) when constructing the mail address; that is, in the preceding example `fred.example.com` would result in the e-mail address of `fred@example.com`. This format is used because `@` has a special significance in the RR (it is a short form for the `$ORIGIN`).

The `text-rr` field defines the name of an optional TXT RR that may contain human-readable text, for example, a name and phone number. If no TXT is present, the `text-rr` field is replaced with a single dot. Multiple RP records may be associated with any host. The following fragment shows the use of the RP RR:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
...
www          IN  A     192.168.254.2
             IN  A     192.168.254.3
             IN  RP    bill.example.com. bill.people.example.com.
```

```
; could have been written as
;            IN  RP    bill bill.people
; line that follows uses an e-mail external to the domain
; and has no corresponding text record (replaced with a single dot)
             IN  RP    fred.example.net. .
...
; all people records organized under people
bill.people  IN  TXT "Bill Someone - pager = 111-1111"
....
```

The line beginning `bill.people` does not strictly define a subdomain structure, but in this case is used simply as a convenient method of grouping people records in the zone file for the organization.

Resource Record Signature (RRSIG) Record

The Resource Recordset Signature RR is a DNSSEC.bis (see Chapter 11) record that contains the digital signature of the RRset being signed. RRSIG RRs operate on RRsets—defined as being any record whose `name`, `class`, and `rr` type fields are the same—not individual RRs. The RRSIG RRs (the digital signatures) for the zone's RRsets are generated automatically by the `dnssec-signzone` utility (described in Chapter 9) during zone signing using the private key whose public key is stored in a DNSKEY RR defined at the zone apex or root. The RRSIG RR is defined in RFC 4034.

RRSIG RR Syntax

```
name  ttl  class   rr     (type algorithm labels ottl expire
                           start key-tag signer signature)
joe    2d  IN      A       192.168.22.22
joe        IN      RRSIG  (A    ; rr type covered
                           5  ; algorithm (RSA-SHA-1)
                           3  ; labels at this name
                           2d ; original ttl of RRs covered
                           20050414000000 ;expiry time
                           20050314000000 ; start time
                           24567 ; key tag
                           example.com. ; signer
                           blahblahblah) ; signature data
```

In the preceding example, both the RR being signed (the A RR) as well as the RRSIG RR (its digital signature) are shown to clarify the use of the RRSIG record.

The `type` field defines the RRset type being signed; in the preceding example an RRset comprising a single A RR is shown, but any number of such RRs could have been included in the RRset.

The `algorithm` field may take one of the values defined here:

`0` = Reserved

`1` = RSA-MD5—not recommended (RFC 2537)

`2` = Diffie-Hellman (RFC 2539)

`3` = DSA-SHA-1—optional (RFC 2536)

`4` = Elliptic curve—not currently standardized

`5` = RSA-SHA-1—mandatory (RFC 3110)

`6–251` = Currently unassigned

`252` = Indirect (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`253` = Private URI (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`254` = Private OID (see the section "Alternative Cryptographic Algorithms" later in this chapter)

`255` = Reserved

The `labels` field defines the number of labels in the FQDN version of the `name` field, excluding any wildcard values. In the preceding example, the number is 3 since the FQDN corresponding to `joe` will be assumed to be `joe.example.com`. If the name had been `*` (the wildcard value), then the value of the label field would have been `2`, thus excluding the wildcard from the label count. This allows verification software to know whether the RRSIG was or was not synthesized and thus re-create the conditions by which successful verification can take place.

The `ottl` field defines the TTL of the RRset being covered. In the preceding example, this is show explicitly as `2d` (`172800` seconds) in the A RR, but if not present would have been taken from the last `$TTL` directive in the zone file.

The `expire` field defines the time at which the RRSIG is no longer valid, and the `start` field indicates when the RRSIG record becomes valid. In their textual form, as shown in the example, both have the format YYYYMMDDHHMMSS, where YYYY is a four-digit year number, MM a two-digit month number, DD a two-digit day within a month number, HH a two-digit hour within a day, MM a two-digit minute within an hour, and SS a two-digit second within an hour.

The `key-tag` field identifies the DNSKEY RR used to generate the digital signature. Since multiple DNSKEY RRs may be present with the same name in a zone file, this field is used to find the correct key. The `key-tag` field is generated by the `dnssec-keygen` utility and uses a variant on the zone's complement checksum algorithm and can thus be rapidly reproduced by verification software to find the correct key.

The `signer` field is the name of the DNSKEY RR whose private key was used to generate the signature—in the example, a DNSKEY RR with a name of `example.com`.

The `signature` field is the base64 (RFC 3548) representation of the digital signature. In the example shown, the digital signature is generated using the digest function SHA-1, which is then encrypted with the RSA algorithm using a private key whose public key is defined in the DNSKEY RR with a host name of `example.com`.

The RRSIG RR is unique in that it does *not* form an RRset; otherwise recursive processing would occur when signing a zone.

Route Through (RT) Record

The Route Through RR defines an intermediate host through which all datagrams should be routed. The intermediate host would typically be a gateway or protocol converter. The RT RR is experimental and is defined in RFC 1183. The RT RR is not widely used.

RT RR Syntax

```
name   ttl class RT  preference intermediate
joe        IN    RT  10         bill.example.com.
```

The `preference` field is a value in the range `0` to `65535` and is used in a similar way to the MX record. The lower the value, the more preferred the route. The `intermediate` field defines the host name to which datagrams destined for `name` should be sent. The following fragment shows how the RT RR is used:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
...
fred         IN  A       192.168.254.2
joe          IN  A       192.168.254.3
bill         IN  A       192.168.254.4
             IN  RT  10  fred.example.com.
             IN  RT  20  joe.example.com.
....
```

In the preceding fragment, to reach `bill.example.com`, `fred.example.com.` would be used, and only if not available would `joe.example.com.` be used.

Signature (SIG) Record

The Signature RR was defined as part of the first generation of DNSSEC (RFC 2535). It is no longer used for this purpose, but is now limited to specific use as a meta (or pseudo) RR containing the digital signature when securing transactions such as dynamic update using public key (asymmetric) cryptographic techniques. The equivalent RR for shared secret transaction security is TSIG—another meta RR. The revised use of what is now called SIG(0) is defined in RFC 2931.

SIG RR Syntax

```
name  ttl  class   rr  (type algorithm labels ottl
                         expire start key-tag signature)
joe        IN      SIG   (0    ; identifies SIG(0)
                           5  ; algorithm (RSA-SHA-1)
                           3  ; labels at this name
                           2d ; original ttl of RRs covered
                           20050414000000 ;expiry time
                           20050314000000 ; start time
                           24567 ; key tag
                           example.com. ; signer
                           blahblahblah) ; signature data
```

The field values and meaning of the SIG RR are exactly the same as those of the RRSIG RR defined earlier with the exception of the `type` field, which in this usage is always set to `0`—hence this RR type is commonly referred to as a SIG(0) RR.

The SIG(0) RR is generated at run time (it is a meta or pseudo RR) by the name server and is added to the `ADDITIONAL SECTION` (see Chapter 15) of the query or its response to carry the digital signature that both authenticates and ensures the integrity of the transaction. The public key used by SIG(0) is stored in the zone file using a KEY RR.

Start of Authority (SOA) Record

The Start of Authority RR describes the global properties for the zone (or domain). There is only one SOA record allowed in a zone file, and it must be the first RR entry. The SOA RR was defined in RFC 1035, and the use of the `min` field of the RR was redefined in RFC 2308.

SOA RR Syntax

```
name        ttl class rr    name-server admin-mailbox  sn refresh retry expiry min
@               IN    SOA   ns.example.com. hostmaster.example.com. (
                              2003080800 ; se = serial number
                              43200      ; refresh = 12h
                              900        ; retry = 15m
                              1209600    ; expiry = 2w
                              3600       ; min = negative cache= 1h
                              )
```

The SOA RR is certainly the most important RR and takes one of the largest numbers of fields of any RR. To assist in readability, it is usually written using the standard parentheses method to enable the various fields to be written one per line for clarity only as shown in the preceding example. Table 13-3 describes the fields unique to this record—the common fields were described previously.

Table 13-3. *SOA RR Fields*

Field	Description
`name-server`	This is a name server for the domain and is referred to as the Primary Master, which has a meaning only in the context of Dynamic DNS (described in Chapter 3) and designates the server that can be updated by DDNS transactions. If DDNS is not being used, it may be *any* suitable name server that will answer authoritatively for the domain. The name server may lie within the domain or in an external or foreign domain. The name server referenced, however, must be defined using an NS RR. The `name-server` is most commonly written as an FQDN (ends with a dot). If the `name-server` is an external server (does not lie in this zone), it *must* be an FQDN. In the DNS jargon, this field is called the `MNAME` field, which is why this book uses the term `name-server`.
`admin-mailbox`	The e-mail address of the person responsible for this zone. In the jargon, this is called the `RNAME` field, which is why this book calls it the `admin-mailbox`. It is the e-mail address of a suitable administrator or technical contact for the domain. By convention (in RFC 2412), it is suggested that the reserved mailbox `hostmaster` be used for this purpose, but any sensible and stable e-mail address can be used. The format is `mailbox-name.domain`, for example, `hostmaster.example.com`, using a `.` (dot), not the more normal `@` (commercial at sign), which has other uses in the zone file. When e-mail is sent to the `admin-mailbox`, the normal format of `hostmaster@example.com` is used. There is no requirement that this mailbox lie inside the domain—it can use any suitable mail address such as `hostmaster.example.org`. It is most commonly written as an FQDN (ends with a dot), but if the e-mail address lies in the domain, it can be written as simply `hostmaster` without the dot.
`sn` (serial number)	Unsigned 32-bit value in range `1` to `4294967295` with a maximum increment of `2147483647`. In BIND implementations, this is defined to be a ten-digit field. The `sn` value must be incremented (must be greater) when any Resource Record in the zone file is updated. By convention, a date format is used to simplify the task of updating the `sn` value—the most popular date convention is YYYYMMDDSS, where YYYY is the four-digit year number, MM is the two-digit month, DD is the two-digit day, and SS is a two-digit sequence number, starting from 00, used when the zone file changes more than once in the day. Using this date format, the value `2005061504` means the last update was done on 15 June 2005 and it was the fifth update that day. The date format is just a convention, not a requirement; consequentially, no validation is performed on this value, so it is easy to use incorrect date values. Extreme care should be taken when working with this number. Chapter 8 describes how to fix out-of-sequence errors. The value, range, and arithmetic operations performed on the serial number is defined in RFC 1982.
`refresh`	Signed 32-bit time value in seconds. Indicates the time after which the slave will try to refresh the zone from the master. RFC 1912 recommends a range of `1200` to `43200`, `1200` (20 minutes) if the data is volatile or `43200` (12 hours) if it is not. If `NOTIFY` (described in Chapter 3), the BIND default, is being used, this can be set to a much higher value, for example, one or more days or greater than `86400`. When using BIND, the normal time short format can be used.
`retry`	Signed 32-bit value in seconds. It defines the time between retries if the slave (Secondary) fails to contact the master after `refresh` has expired. Values will depend upon local knowledge of the network speed and reliability of the master (Primary) servers. Typical values would be `180` (2 minutes) to `900` (15 minutes) or higher. When using BIND, the normal time short format can be used.

Continued

Table 13-3. *Continued*

Field	Description
expiry	Signed 32-bit value in seconds. Indicates when the zone data is no longer authoritative. This field applies to slave (Secondary) servers only. In BIND, slaves stop responding to queries for the zone when this time has expired and no contact has been made with the master. Thus when the refresh value expires, the slave will attempt to read the SOA record for the zone—and request a zone transfer (AXFR) if the sn field has changed. If contact is made, the expire and refresh values are reset and the cycle begins again. If the slave *fails* to contact the master, it will retry the operation every retry interval, but it will continue to supply authoritative data for the zone until the expiry value is reached, at which point it will stop responding to queries for the domain. RFC 1912 recommends 1209600 to 2419200 (2 to 4 weeks) to allow for major outages of the master. When using BIND, the normal time short format can be used.
min (minimum)	Signed 32-bit value in seconds. RFC 2308 (implemented by BIND 9) redefined this value to be the negative caching time—the time a NXDOMAIN (no name) record is cached. The maximum value allowed by BIND 9 for this parameter is 10800 (3 hours). This field *was* the zone TTL default (in BIND versions 4 and 8). RFC 2308 makes the $TTL directive mandatory in a zone file and consequentially allows the min field to be reused for negative caching time. Older documentation or zone file configurations may reflect the old usage and have time values greater than 3 hours in this field. In this case, BIND will log a nasty error message when the zone is loaded but continue with a default value. When using BIND, the normal time short format can be used.

The following zone file fragment illustrates that one or all name servers may be external to the domain:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
@      IN      SOA   ns.example.net. hostmaster.example.com. (
               2003080800 ; serial number
               1d12h      ; refresh =  1 day 12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
        IN     NS    ns.example.net. ;name servers
        IN     NS    ns.example.org.
```

The e-mail address in the preceding example is inside the domain, so it could have been rewritten to use the unqualified name form as shown here:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
@      IN      SOA   ns.example.net. hostmaster (
               2003080800 ; serial number
               1d12h      ; refresh =  1 day 12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
        IN     NS    ns.example.net. ;name servers
        IN     NS    ns.example.org.
```

The following fragment shows use of two name servers, one inside the domain, the other external:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
@      IN      SOA   ns.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               1d12h      ; refresh =  1 day 12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
       IN      NS  ns.example.com.
       IN      NS  ns.example.net.
...
; A record required for internal name server
ns     IN      A   192.168.2.1
```

The following fragment rewrites the preceding fragment and uses unqualified names wherever possible:

```
; zone file fragment for example.com
$TTL 2d ; zone TTL default = 2 days or 172800 seconds
$ORIGIN example.com.
@      IN      SOA   ns  hostmaster (
               2003080800 ; serial number
               1d12h      ; refresh =  1 day 12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
       IN      NS  ns
       IN      NS  ns.example.net.
...
; A record required for internal name server
```

Services (SRV) Record

The Services RR allows a service to be associated with a host name. A user or application that wishes to discover where a service is located can interrogate for the relevant SRV RR that describes the service. The result of a successful SRV query will be one or more host names, the port that provides the service, and two values that can be used to select the relative priority and performance of the service. Having obtained the host name, a further A (or AAAA) query will be required to obtain the IP address of the selected service. The SRV RR is being increasingly supported as the means by which the location of a service at a particular domain may be discovered, notably with VoIP and LDAP applications. OpenLDAP (`www.openldap.org`) in particular supports the SRV record (and publishes an SRV RR) to discover the location of the LDAP service at a domain. The SRV RR is defined in RFC 2782.

SRV RR Syntax

```
srvce.prot.name  ttl  class   rr  pri  weight port target
_http._tcp            IN      SRV 0    5      80   www.example.com.
```

Table 13-4 describes the various fields unique to the SRV RR.

Table 13-4. *SRV RR Fields*

Field	Description
`srvce`	The `srvce` field defines the *symbolic service name*. Standard symbolic service name values are listed by IANA (under the port number list at `www.iana.org/assignments/port-numbers`), but there is a specific SRV list currently being maintained (see note that follows) outside of IANA. Service names are case insensitive and are always prepended with _ (underscore). Common values are `_http` for web service, `_ftp` for File Transfer Protocol, `_sip` for Session Initiation Protocol, and `_ldap` for LDAP service. This `srvce` field may also take a *local* value—its scope is local to the user and therefore may take any desired value that does not conflict with the IANA list earlier. The IANA list also defines the port assigned to the service, but the `port` field within the SRV RR allows this port number to be changed for the particular service instance if required.
`prot`	The `prot` field defines the case-insensitive protocol name (see `www.iana.org/assignments/service-names`) prepended with a _ (underscore). Common values are `_tcp` for the TCP protocol and `_udp` for the UDP protocol.
`name`	The `name` field is optional. If not present, then normal `$ORIGIN` substitution rules will occur. See the examples that follow.
`pri`	The `pri` field defines the relative priority of this service (range `0` to `65535`). Lower numbers are higher priority as in the MX RR type.
`weight`	The `weight` field is used when more than one service with same priority is available. `weight` is a 16-bit unsigned integer in the range `0` to `65535`. The value `0` indicates no weighting should be applied. If the weight is `1` or greater, it is a relative number in which the highest is most frequently delivered; that is, given two SRV records, both with a priority of, say, `10`, one with a weight of `1`, the other a weight of `6`, the one with weight `6` will have its RR delivered first six times out of seven by the name server.
`port`	The `port` field defines the port number that delivers the service on the target (see the `target` entry). This would normally be the port assigned to the symbolic service (`srvce` field), but this is not a requirement; for instance, it is permissible to define an `_http` service with a port number of 8100 rather than the more normal port 80.
`target`	The `target` field defines the name of the host that will provide this service and may require a query to obtain the IP address (A or AAAA RR query). The `target` host may lie within this domain or in an external or foreign domain.

The following fragment shows use of the priority and weight fields to define a web service with load balancing:

```
; zone file fragment for example.com.
$TTL 2d ; zone TTL default = 2 days
$ORIGIN example.com.
@               SOA server.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               1d12h      ; refresh =  1 day 12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
...
_http._tcp      SRV 10 1 80 slow.example.com.
                SRV 10 3 80 fast.example.com.
; if neither slow or fast available, switch to
; an external backup web server but use port 8100 not port 80
                SRV 20 0 8100 backup.example.net.
slow            A   192.168.254.3
fast            A   192.168.254.4
```

In the preceding fragment, both `fast.example.com` and `slow.example.com` have equal priorities; the weight values are `1` and `3`, respectively, which will result in `fast.example.com` being returned three times to every one return of `slow.example.com`. Thus `fast.example.com` will theoretically receive 75% of the load. If neither `fast` nor `slow` is available, the externally hosted `backup.example.net` should be used with port 8100, not the more normal HTTP port of 80. The following fragment shows use of the SRV RR to discover the host for the LDAP service at `example.com`:

```
; zone file fragment for example.com.
$TTL 2d ; zone TTL default = 2 days
$ORIGIN example.com.
....
; defines an ldap service available at the host jim.example.com
_ldap._tcp.example.com.  IN  SRV 0 0 389 ldap.example.com.
; the preceding record could have been written as
; _ldap._tcp             IN  SRV 0 0 389 ldap
....
ldap                     IN  A   192.168.254.2
....
```

To discover whether an LDAP service is available at `example.com`, an SRV query would be sent for `_ldap_.tcp.example.com`, which in the preceding case would return `0 0 389 ldap.example.com`; `ldap.example.com` would then be queried for its A RR (or AAAA RR if IPv6), and communication could commence.

■Note IANA is not currently maintaining a registry of SRV symbolic names, and as a consequence a number of impromptu web sites are springing up to try and maintain such a registry, the objective being to hand it over to IANA at the appropriate time. One such site is maintained at`www.dns-sd.org/ServiceTypes.html`.

SSH Key Fingerprint (SSHFP) Record

The SSH Key Fingerprint RR allows a host to obtain the fingerprint (hash or digest) of the public key for use in an SSH session by a DNS query. This functionality was originally provided using the subtype of the KEY RR but subsequently removed by RFC 3445. The SSHFP RR is defined in `draft-ietf-secsh-dns-05.txt`, and you are reminded that there may be changes prior to publication of the RFC.

■Note Draft RFCs may be obtained from `http://datatracker.ietf.org`.

SSHFP RR Syntax

```
name  ttl  class   rr     alg fpt  (fingerprint)
joe        IN      SSHFP   2   1 (123456
                                  789abcdef67890123456
                                  789abcdef67890)
```

The `alg` field defines the SSH algorithm and may take one of the following values:

0 = Reserved

1 = RSA

2 = DSS (DSA)

The `fpt` field is a message-digest algorithm used to create the fingerprint of the SSH key and may take one of the following values:

0 = Reserved

1 = SHA-1

The `fingerprint` field is the base64 (RFC 3548) material created by the digest (hash) algorithm.

```
; zone fragment for example.com
$TTL 2d ; zone default = 2 days or 172800 seconds
$ORIGIN example.com.
....
www        IN      A      192.168.0.3
           IN      SSHFP  1 1 (AABB12AA334477
                               CD1234a57890)
....
```

In the preceding example, the SSH key for the host `www.example.com` uses the RSA algorithm and may be computed by decoding the fingerprint using the SHA-1 message digest.

The current (BIND 9.3+) versions of BIND support the SSHFP RR type natively. The SSHFP RR may be created using the `ssh-keygen` utility (OpenSSH 3.6.1+) using a command such as the following:

```
ssh-keygen -f /etc/ssh/ssh_host_rsa_key.pub \
-r host.example.com. >> master.example.com
```

The \ indicates the line has been broken solely for presentation reasons and should appear as a single line when entered on the command line. The preceding command generates the SSHFP RR from the default OpenSSH (`www.openssh.com`) RSA key (the `-f` argument) with a name of `host.example.com` (the `-r` argument) and appends it to the zone file `master.example.com`.

The `ssh-key` utility is also capable of generating an SSHFP RR using the generic RR format (see "User-defined RRs" later in the chapter) if the BIND version being used does not support the SSHFP RR (any release prior to 9.3). In this case, the following command will create a TYPE44 RR of the correct format by using the `-g` argument:

```
ssh-keygen -g -f /etc/ssh/ssh_host_rsa_key.pub \
-r host.example.com. >> master.example.com
```

The \ indicates the line has been broken solely for presentation reasons and should appear as a single line when entered on the command line.

Text (TXT) Record

The Text RR provides the ability to associate arbitrary text with a name; for example, it can be used to provide a description of the host, service contacts, or any other required information. The TXT RR was defined in RFC 1035.

TXT RR Syntax

```
name  ttl  class   rr     text
joe        IN      TXT    "Located in a black hole"
```

The `text` field may be any arbitrary text and is enclosed in quotes. The TXT RR is also used to define the Sender Policy Framework (SPF) information record that may be used to validate legitimate e-mail sources from a domain, as described in Chapter 8, and in this case, the content and format of the text field are defined by the SPF specification. The following example shows the use of a TXT record to contain truly meaningful data for a host:

```
; zone file fragment for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
....
```

```
@           IN     A      192.168.254.8
            IN     A      192.168.254.9
            IN     A      192.168.254.10
            IN     TXT "Example.com web service is here"
www         IN     CNAME  example.com.
```

The text "Example.com web service" is associated with `example.com` and will be returned on a TXT query for `example.com`. The associated CNAME RR as noted in the section "Canonical Name (CNAME) Record" does not allow TXT RRs to be defined with the same name.

Well-Known Service (WKS) Record

The Well-Known Service RR is used to define the services and protocols supported by a host. Clients can discover the location of the desired service by inspecting the WKS for the domain. The WKS RR was defined in RFC 1035. Its use is deprecated and replaced by the SRV RR, which provides a more general-purpose mechanism.

WKS RR Syntax

```
name  class  ttl  rr   ipv4         proto svc1 svc2 ...
@     IN          WKS  192.168.0.1  TCP   telnet http
```

The `ipv4` field is the IPv4 address to which the following list of services applies. The `proto` field defines the protocol supported by the following services and may take the case-insensitive value `tcp` or `udp`. The `svc1` and `svc2` fields are an arbitrary long list of the services provided at the `ipv4` address. The services list may use any name from the IANA registered port numbers list (`www.iana.org/assignments/port-numbers`). This list may be enclosed in the standard parentheses notation if it extends over more than a single line as illustrated in the fragments that follow.

WKS RRs are normally defined at the domain level such that a domain query for the WKS RRs will return all the available records and the client must then find the appropriate host to provide the service. The fragment that follows shows a number of services provided on two separate hosts. The RFC is silent on the topic of selecting a host when two hosts provide the same service.

```
; zone file fragment for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
....
@           IN   WKS  192.168.254.2 TCP telnet http
            IN   WKS  192.168.254.3 {
                   telnet
                   smptp   ;some comment about this service
                   ftp
                 }
....
bill        IN   192.168.254.2
fred        IN   192.168.254.2
...
```

The first preceding WKS RR could be replaced with two SRV RRs as shown here:

```
_telnet._tcp    IN   SRV 10 0 23  bill.example.com.
_http._tcp      IN   SRV 10 0 80  bill.example.com.
```

X.25 Address (X25) Record

The X.25 Address RR is the equivalent of an A RR for an X.25 (packet-switched network) address. It associates the address of an endpoint (DTE) in an X.25 network with a given host name. The X25 RR has EXPERIMENTAL status and is defined in RFC 1183.

X25 RR Syntax

```
name   ttl class x.25-address
joe        IN    X25  311061700956
```

The x.25-address field is the numeric Packet-Switched Data Network (PSDN) address in X.121 format. It must start with the four-character Data Network Identification Code (DNIC—similar to the regional code in a phone number). Since x.25-address is an address, not a name, there is no terminating dot.

Alternative Cryptographic Algorithms

The cryptographic RR types that define or reference cryptographic algorithms, specifically CERT, DNSKEY, DS, KEY, and RRSIG, allow for additional algorithms other than those specified within the RFC that defines each RR type. This is accomplished using the algorithm field values 252, 253, and 254.

The value 252 denotes an indirect value where the key-data or signature field of the RR is located elsewhere. As of the publication of this book, no further definition of this field has been introduced, and it must currently be regarded as unused. The value 253 denotes that the key-data or signature field will commence with a host name that will be used by the recipient to interpret the content of the space-separated key-data or signature field. The value 254 denotes that the key-data or signature field will start with an Object Identifier. The OID is assumed to define the cryptographic algorithm being used and allows the recipient to interpret the following space-separated key-data or signature fields. The following example shows a standard DNSKEY RR using the RSA-SHA-1 algorithm (5):

```
example.com.        IN      DNSKEY 256    3        5 (
                          AQPSKmynfzW4kyBv015MUG2DeIQ3
                          Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                          kfzj31GajIQKY+5CptLr3buXA10h
                          WqTkF7H6RfoRqXQeogmMHfpftf6z
                          Mv1LyBUgia7za6ZEzOJBOztyvhjL
                          742iU/TpPSEDhm2SNKLijfUppn1U
                          aNvv4w==  )
```

The following shows the same RR using an OID (254) format:

```
example.com.       IN      DNSKEY 256    3        254 (1.3.6.1.4.1.X.22.55.4.3
                         AQPSKmynfzW4kyBv015MUG2DeIQ3
                         Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                         kfzj31GajIQKY+5CptLr3buXA10h
                         WqTkF7H6RfoRqXQeogmMHfpftf6z
                         Mv1LyBUgia7za6ZEzOJBOztyvhjL
                         742iU/TpPSEDhm2SNKLijfUppn1U
                         aNvv4w==  )
```

The OID shown in the preceding example (1.3.6.1.4.1.X.22.55.4.3) represents one possible format and is comprised of two parts. The first part, 1.3.6.1.4.1, is the base OID of the private enterprise group assigned by IANA (`www.iana.org`). The value `X` would be replaced by an enterprise unique number that may be obtained from IANA (`www.iana.org/cgi-bin/enterprise.pl`). This is followed by the enterprise-assigned number (22.55.4.3) that would define the algorithm to be used.

User-Defined RRs

It is possible to extend the DNS with user-defined RRs. User-defined RRs may be used to add a new RR type not defined in the current IANA list (`www.iana.org/assignments/dns-parameters`), to define a standardized RR that is not currently supported by the name server software, or to supply the normal `type-specific data` in an alternative format. Such RRs can be stored in zone files, transferred to slaves, and queried by clients. The method by which DNS software handles user-defined RRs is specified by RFC 3597.

The standard syntax of an RR is defined as follows:

```
name   ttl  class   type  type-specific-data
```

The `class`, `type`, and `type-specific-data` fields may all be defined using the mechanism described here:

The `class` field may be user defined by using the word "CLASS" immediately followed by the decimal value of the class type being defined; for instance, CLASS15 defines a new `class` type that will have a decimal value of 15. No whitespace is allowed between CLASS and the decimal number. Existing classes may be represented using this format, for example, CLASS1 = `IN` (or Internet class).

The `type` field may be user defined by using the word "TYPE" immediately followed by the decimal value of the type being defined, for example, TYPE555 defines a new type that will have a decimal value of 555. No whitespace is allowed between TYPE and the decimal number. Existing RRs types may be represented using this format, for example, TYPE1 is an A RR.

User-defined `type-specific-data` is indicated by using the literal sequence `\#`, followed by whitespace, followed by the number of octets in the field. The fields are written as two hexadecimal characters per octet. If an RR does not have any data, it must be written with a data length of zero. If whitespace is required for clarity, the values must be enclosed in (and) (parentheses). The following example illustrates the possible definitions:

```
$ORIGIN example.com.
....
a       CLASS32     TYPE731         \# 6 abcd (
                                               ef 01 23 45 )
b       HS          TYPE62347       \# 0  ; no data format
e       IN          A               \# 4 0A000001 ;hex version of IP address
f       CLASS1      TYPE1           10.0.0.2 ; A RR
....
```

If a known RR is defined using the TYPEx format, for example, TYPE1 for an A RR, or the data with a known type is defined using the \# format, then those formats are used for the purposes of converting the data to a *binary format*. Thereafter, the RR is treated as normal for that type, that is, it becomes a *known RR type*. It is not possible to alter the *operational* treatment of an existing RR using the user-defined RR textual syntax earlier.

Summary

This chapter has provided a reference with examples where appropriate for all zone file directives and most RRs defined in the current IANA list (`www.iana.org/assignments/dns-parameters`). The RRs not described are either privately defined (NIMLOC, EID, and SINK), formally obsolete (GPOS, MD, MF, and NSAP-PTR), or not useful in a production environment (NULL, which is experimental and cannot be defined in a master zone file). Definitive RFC references are provided for all RRs.

The RRs supported by both the current version of BIND (9.3.0) and Windows (Windows 2003 Server) are presented in Table 13-1.

The current IETF policy regarding IPv6 addresses, forward mapping using AAAA RR, and reverse-mapping using PTR RRs, is documented, and the experimental A6 RR is shown with examples. The NAPTR RR is illustrated with some examples. You are reminded that this brutally complicated RR only makes sense when read in conjunction with the application that will make use of the NAPTR RR.

The descriptions of the RRs concerned with DNS security (DS, DNSKEY, KEY, NSEC, and RRSIG) should be read in conjunction with Chapters 10 and 11 as appropriate. The SSHFP RR description was based on a draft RFC and may be subject to change when finally published.

A number of the RRs described are rarely used, and you are cautioned that those defined as experimental may be withdrawn or changed at any time. RFC 3597 defines a method by which user-defined DNS RR types may be added to DNS zone files and queried by clients. Using this procedure, it is possible for users to extend DNS capabilities using standard software. BIND 9.3+ supports the user-defined RR capability.

The next chapter describes the use of DNS application program interfaces (APIs) and the various libraries available with BIND releases. It is intended primarily for programmers and others wishing to either extend the basic capabilities of the DNS or utilize its existing capabilities from within other software.

PART 5

Programming

CHAPTER 14

BIND APIs and Resolver Libraries

This chapter is primarily intended for programmers and designers who wish to modify the basic functionality of BIND, need to interface to the libraries available with BIND, or need to interface to the standard DNS-related POSIX calls supported by `libc`. Reasonable knowledge of the C language is required to make sense of most of the information in this chapter.

BIND API Overview

BIND provides two APIs. One is an Advanced Database API (we'll call it *adb* for convenience), which has been available since BIND version 8. It allows user-written routines to replace BIND's internal database function for both nominated and all zones. Only a brief synopsis of the adb is included in this chapter. From BIND version 9.1 onward, a Simple Database API, termed *sdb*, has also been provided. It allows a user-written driver to supply zone data either from alternate data sources (for instance, a relational database), or using specialized algorithms (for instance, for load-balancing). A complete description of this API, including an example driver, is presented in the chapter. Neither the adb nor the sdb APIs allow zones to be added or deleted dynamically.

Caution Before contemplating the use of either of these APIs, it is important to be aware that they are statically linked. Simply put, this means the BIND source files and `Makefile.in` are edited and BIND is rebuilt to include the user-written source and header files. If anything goes wrong with the added routines, it is likely that BIND will crash and stop serving DNS queries. No supervisory functionality is provided by the BIND API functions. This is unlike, say, Apache, where modules can be dynamically loaded, and in the event of an error in the loaded module, the basic server will likely continue to function.

Advanced Database API (adb)

Use of the adb is a nontrivial task because all the DNS protocol functionality that is required must be supported by user-written software routines. These may include zone transfer, Dynamic Update (DDNS), and DNSSEC, as well as the basic service of providing zone data. The adb API provides a total of 36 functions to implement the capability of the interface.

The definitive reference for the adb is the commented C header file `db.h`. It's located in the directory `bind-release/lib/dns/include/dns/db.h`, where `bind-release` should be replaced with the location and version number where you unpacked the source distribution (for example, `/usr/src/bind-9.3.0`). Although this source file is well documented, it does not contain enough information to implement fully all the capability required to support this interface. Any potential developer needs to spend time with the BIND source to understand the subject and all its nuances fully before starting any implementation. The adb API is not described further in this book.

Simple Database API (sdb)

The sdb is a relatively simple abstraction, consisting of five callbacks and a small handful of RR writing functions. It is optimized to enable zone data to be supplied via a user-written driver from alternative data sources (for instance, a relational database or a Lightweight Directory Access Protocol [LDAP] service), or to manipulate zone data in a user-defined way (for instance, to allow load balancing of A RRs or MX RRs). Information for the sdb API is documented, in typical minimalist style, in the C header file `bind-release/lib/dns/include/dns/sdb.h`, where `bind-release` should be replaced with the location and version number where you unpacked the source distribution (for example, `/usr/src/bind/9.3.0`). A number of sdb API examples are also provided in the directory `bind-release/contrib/sdb`, covering PostgreSQL, LDAP, and Berkeley Database (BDB). The sdb API provides the following functionality:

- Responses may be in textual RR format (though binary alternates are provided), and hence are optimized for user-interface–style databases.
- Parameters can be passed to the sdb driver when the zone is initialized (defined in the `database` statement of `named.conf`—see Chapter 12—and illustrated in the example later in this chapter).
- The sdb driver may register up to five callback types, which cover zone initialization (`create()`), zone termination (`destroy()`), zone transfer (`allnodes()`), zone authority information (`authority()`), and zone query (`lookup()`).
- Zones that use the sdb driver interface cannot also be dynamically updated.

The functionality of the sdb interface is described with an illustrative example in the following sections.

The Simple Database API (sdb)

The functionality of the sdb API is illustrated with a sample driver that simply reads a standard zone file, whose name is supplied as an initialization parameter, into a memory buffer that it subsequently uses to respond to queries. The comments in the code fragments and the accompanying notes indicate the kind of functionality that could be provided at each callback. The sample driver used in this book is contained in a single module called `example.c`; has an accompanying header file called, surprisingly, `example.h`; and has a driver name of `"example"`. The code for the sample driver is shown in extracted fragments as required. The listing for both the C module and the header file are shown in the section "sdb Sample Driver."

Before starting, it is assumed that a copy of the latest tarball for BIND has been downloaded from the ISC site (`www.isc.org`) and unpacked into a suitable location. The following sample assumes that version 9.3.0 was downloaded and unpacked into `/usr/src`. Thus, the base directory of BIND (called `bind-release` from here on) is `/usr/src/bind-9.3.0`.

■**Note** The sample application is a viable, if not very useful, driver application that reads a standard zone file and serves it via the sdb API. Its primary purpose is to illustrate the functionality of the interface, not the back-end file-system interface, which is inevitably user specific. The code sample should be viewed entirely from this perspective, and by no means as a real-world, ready-to-go driver application. The author pleads brevity as the sole justification for any egregious shortcuts.

Callback Overview

The sdb API provides five callback functions, only one of which is mandatory; these callback functions are introduced in this section.

create()

The `create()` callback function is optional and is invoked when the zone is initialized by BIND. It is therefore called for each instance of the driver. The use of a specific sdb driver is defined using the `database` statement in a `zone` clause of `named.conf`. Any mixture of `zone` clauses may be configured to support one or more sdb drivers, as shown in the following fragment:

```
// named.conf fragment
....
zone "example.com" in {
   // invokes the driver named "example" with one argument
   database "example master.example.com";
};
zone "example.net" in{
   type master;
  // normal zone definition - no driver used
   file "master.example.net";
};
zone "example.org" in {
   // uses another driver for this zone
   database "another-driver";
};
....
```

In the preceding fragment, the zone `example.com` uses the sample driver, whose name is `"example"`, and is passed the `master.example.com` parameter (any number of which may be supplied, each being space-separated) on the `create()` callback in a standard command-line `argc/argv` structure. The `example.net` zone is a normal master zone that does not use any sdb driver functionality. The example.org zone uses another (fictitious) sdb driver called "another-driver", which does not require any parameters to be passed.

destroy()

The destroy() callback is optional and is invoked when the zone is unloaded. It is typically used to perform any tidy-up functions, such as terminating database connections, closing files used, freeing memory allocated, and so on.

lookup()

The lookup() callback is mandatory and is invoked on receipt of a query for the domain for which the driver has been registered. The driver supplies results to be included in the ANSWER SECTION of the query (see Chapter 15) by using either the dns_sdb_putrr (a text RR) or the dns_sdb_putrdata (a binary RDATA section only; see Chapter 15) functions. Both of these functions are described in the section "Returning RRs" later in the chapter. If the driver always returns SOA and NS RRs for all queries at the zone apex, the authority() callback function is not required. If for some reason the driver maintains these records in a separate structure, then an authority() callback is required. The lookup() callback does not request a type of record (for instance, an A or AAAA RR); instead, all RRs for the queried name must be returned.

authority()

The authority() callback is optional, and is invoked for a received query at the zone apex when the preceding lookup() callback did not return the zone SOA and NS RRs. It requests the SOA and NS RRs for the zone apex used to populate the AUTHORITY SECTION (see Chapter 15) of the query response. The authority() callback returns the required NS and SOA using either the dns_sdb_putrr() or dns_sdb_putrdata() functions, or the SOA RR may be optionally returned using the dns_sdb_putsoa() function. Each function is described later in the chapter.

allnodes()

The allnodes() callback is optional, and is invoked when a transfer zone request (AXFR) is received by the name server for the zone for which the driver is registered. It may not be appropriate depending on the application type. Each RR in the zone is returned using the dsn_sdb_putnamedrr() or dns_sdb_putnamedrdata() functions, or in the case of the SOA, only the dns_sdb_putsoa() function.

Registering the Callbacks

Registering the callbacks involves calling the dns_sdb_register() function from a driver-initialization function, which in turn is invoked by a manual edit to the bind-release/bin/named/main.c BIND module. To keep matters as simple as possible, our example.c and example.h files reside in the same directory as main.c (bind-release/bin/named). The following fragment from the example.c module contains example_init(), which registers four of the five callback interfaces for illustration purposes. However, recall that only the lookup() callback is mandatory. The fragment also shows the example_clear() function, which performs the corresponding dns_sdb_unregister() function:

```
// example.c fragment
....
// list of callback functions in dns_sdbmethods_t structure
PRIVATE dns_sdbmethods_t example_callbacks = {
   example_lookup, // lookup callback function - mandatory
   NULL, // authority callback function - optional
   example_allnodes, // allnodes callback function - optional
   example_create, // create callback function - optional
   example_destroy // destroy callback function - optional
};
// pointer to handle allocated by BIND and supplied to dns_sdb_unregister
PRIVATE dns_sdbimplementation_t *namedhandle = NULL;

....

/********************
*
* example_init
* register callbacks for the example driver
* Note: In this driver the DNS_SDBFLAG_RELATIVERDATA flag
* is not strictly needed and is used only to illustrate the use of multiple flags
* the variable 'directory' is used to illustrate that
* parameters may be supplied to this function
* equally the function may or may not return a value
********************/
isc_result_t  example_init (char directory[])
{
   // initialize flags
   unsigned int flags = DNS_SDBFLAG_RELATIVEOWNER |
                        DNS_SDBFLAG_RELATIVERDATA;
   return (dns_sdb_register(DRIVERNAME,&example_callbacks,
           directory,flags,ns_g_mctx, &namedhandle));
};
/********************
*
* example_clear
* unregister callbacks for this driver
*
********************/

void example_clear(void)
{
   if (namedhandle != NULL){
      dns_sdb_unregister(&namedhandle);
   };
};
```

The significant point to note in the preceding fragment is that because the calls to initialize and terminate any driver are under control of the driver developer, zero or more parameters can be supplied, and a value can be optionally returned that may be tested in the `main.c` code. The initialization function typically performs global (driver-level) initialization (for example, open one or more database connections). The termination function typically performs any global (driver-level) clean-up processing.

`dns_sdb_register()` Function

The `dns_sdb_register()` function is called once from the driver initialization function to register all the supported callback functions. In the sample driver, `dns_sdb_register()` is called from the function `example_init()`. The prototype statement is shown here:

```
isc_result_t dns_sdb_register(const char *drivername,
    const dns_sdbmethods_t *methods,
    void *driverdata, unsigned int flags, isc_mem_t *mctx,
    dns_sdbimplementation_t **sdbimp);
```

As you can see, this function accepts numerous parameters. Let's introduce each:

- `isc_result_t` is the standard return code, whose values are described in the section "`isc_result_t` Return Codes" later in the chapter.
- `drivername` is a standard null-terminated string containing the name of the driver, and corresponds to that defined in the `database` statement in the `zone` clause of `named.conf`. In the sample driver, this is `"example"`.
- `methods` is the `dns_sdbmethods_t` structure used to contain the address of the callback functions in the order `lookup()`, `authority()`, `allnodes()`, `create()`, and `destroy()`. Only the `lookup()` callback is mandatory. Any of the others may be set to NULL to indicate it is not supported.
- `driverdata` is an optional driver-allocated parameter. The type and value are determined by the driver. This value is returned on the `create()` and `destroy()` callbacks. This parameter is driver specific, not zone specific. An equivalent zone (or instance) parameter (`dbdata`) provides a similar function and may be supplied using the optional `create()` callback when the zone is initialized. The driver-level parameter is a way of passing parameters to each zone's `create()` callback. This parameter could, as an example, store a time value that would be used to compute elapsed time in all zones handled by the driver. In the sample driver, a directory-string variable is used to illustrate its usage. If not required, the variable should be set to NULL.
- `flags` defines any required flags, which may take one or more of the following values:
    - `_SDBFLAG_RELATIVEOWNER`: If present, indicates that the `lookup()` and `authority()` callbacks will be called with relative domain names. If not present, the callbacks will use absolute names (FQDN). If the zone is `example.com` and the `DNS_SDBFLAG_RELATIVEOWNER` flag is present, then the name `joe` would be supplied on a lookup to indicate `joe.example.com`. If the `DNS_SDBFLAG_RELATIVEOWNER` flag were not present, the same call would need to supply a name of `joe.example.com.` (with the dot).

- NS_SDBFLAG_RELATIVERDATA: If present, indicates that responses in a dns_sdb_putnamedrdata() or a dns_sdb_putrdata() function may contain labels with relative names in the binary string supplied (see the section "NAME Field Format," located in Chapter 15). If this flag is not present, only fully expanded names (FQDNs) are allowed in RDATA fields.
- NS_SDBFLAG_THREADSAFE: If present, indicates the driver is capable of handling multiple parallel requests. If not present, only one request is sent at a time, and the next request is issued only when the driver has returned from that callback.

- mctx is a pointer to a BIND memory context and should take the value ns_g_mctx in all drivers. This pointer is also used in the memory management functions isc_mem_free(), isc_mem_put(), isc_mem_get(), and isc_mem_strdup() (see the later section "Memory Management for Drivers").
- sdbimp is a pointer to a *handle pointer* allocated by BIND when the dns_sdb_register() function is invoked, and must be returned by the driver when it issues the dns_sdb_unregister() function. The driver is responsible for defining the storage location to contain this pointer in a dns_sdb_implementation_t structure. It should be initialized as shown in the preceding sample driver fragment.

dns_sdc_unregister() Function

This function is called to unregister the driver's callback functions. In the sample driver, this is done from the example_clear() function, which in turn is called from BIND's main.c module on termination of BIND. example_clear() is always called after all destroy() callback functions. The prototype is shown here:

```
void dns_sdb_unregister(dns_sdbimplementation_t **sdbimp);
```

The sdbimp pointer is the same as the one supplied by the driver when the dns_sdb_register() function was invoked, and is the handle used by BIND to recognize this driver.

isc_result_t Return Codes

The following return codes may be used with all the driver functions—including callbacks—to indicate the return type:

- ISC_R_SUCCESS: Good return.
- ISC_R_FAILURE: The function failed for some unspecified reason.
- ISC_R_NOTFOUND: The specified hostname or authority RRs were not found.
- ISC_R_NOMEMORY: Memory allocation failure—see the later section "Memory Management for Drivers."

Adding the Driver to BIND

The functions example_init() and example_clear() must be triggered from BIND's main.c module and the driver's header file added to support the calls. The following example uses

BIND 9.3.0 main.c (in bind-release/bin/named/main.c). Finally, all the modules that comprise the driver must be added to the BIND Makefile.in to be included in the final build.

Header File Insertion

The driver header file should be kept to an absolute minimum to minimize nesting complexity, and should contain only prototypes and necessary definitions to support the calls from this module. The header file should be placed in the bind-release/bin/named directory. The insertion point for headers is well documented (line 70 in main.c for 9.3.0), as shown in the following code:

```
#include <named/lwresd.h>
#include <named/main.h>

/*
 * Include header files for database drivers here.
 */
/*
#include "example.h"  // header for example driver inserted
```

Initialization Function Insertion

The initialization function example_init() must now be inserted to call the driver. Any number of drivers may be included, each of which is added to the function setup() in a well-documented location before the call to ns_server_create() (line 651 in main.c for 9.3.0), as shown in the following code:

```
   ns_builtin_init();
   /*
    * Add calls to register sdb drivers here.
    */
   /*
   example_init("/var/named/zones/");  // call to example to register the callbacks
   ns_server_create(ns_g_mctx, &ns_g_server);
```

Termination Function Insertion

Finally, the termination function example_clear() needs to be added to main.c in the cleanup() function, after the call to ns_server_destroy(). Again, the location is well documented in the source (line 658 in 9.3.0), as shown in the following code:

```
   ns_server_destroy(&ns_g_server);
   ns_builtin_deinit();
   /*
   * Add calls to unregister sdb drivers here.
   */
/*
example_clear();  // unregister callback function
```

`Makefile.in` Insertion

The `Makefile.in` file (`bind-release/bin/named/Makefile.in`) must be modified to include all the driver modules during the BIND build. The insertion point for source (`DBDRIVER_SRCS`) and object (`DBDRIVER_OBJS`) is well documented (line 25 in 9.3.0), as shown in the following code:

```
#
# Add database drivers here.
#
DBDRIVER_OBJS = example.@O@
DBDRIVER_SRCS = example.c
DBDRIVER_INCLUDES =
DBDRIVER_LIBS =
```

The sample driver has no special requirements for either library or include locations.

The Callback Functions

As defined previously, there are five callback functions: `lookup()` (mandatory), `authority()` (optional), `allnodes()` (optional), `create()` (optional), and `destroy()` (optional). In the following sections, each callback function is described and illustrated using the sample application.

`create()` Callback Function

The `create()` callback is called for each zone in which a `database` statement exists for this driver. Thus, it is called to create an *instance* of the driver, and provides the opportunity to initialize any zone-specific data. For instance, the `create()` callback could allocate memory, initialize SQL queries, and so on. The prototype is as follows:

```
isc_result_t (*dns_sdbcreatefunc_t)(const char *zone, int argc, char **argv,
       void *driverdata, void **dbdata);
```

Where

- `isc_result_t` is the result code—see the section “`isc_result_t` Return Codes” earlier in the chapter.
- `zone` is a null-terminated string containing the zone name.
- `argc` is the standard count of arguments supplied in the `database` statement of this zone clause. In the example `database` statement used previously (`database "example master.example.com";`), the count is 2.
- `argv` is a null-terminated string array with one entry for each of the supplied arguments. In the example `database` statement (`database "example master.example.com";`), `arg[0]` contains `"example"` and `argv[1]` contains `"master.example.com"`.
- `driverdata` is the optional value supplied in the `dns_sdb_register` function that may be used as a driver global.
- `dbdata` is optional, and allows this instance of the driver to create unique information that is returned with every `allnodes()`, `authority()`, `destroy()`, and `lookup()` callback. It is used in the sample driver to contain the zone data in a memory location that has been dynamically allocated.

The following is an extracted fragment from the sample driver application showing some of the possible functionality of the `create()` callback:

```
// sdb driver fragment
/*********************
*
* example_create()
* handle create callback for the example driver
* 1. call example_read_zone
* 2. update dbdata if OK
* 3. driverdata contains /var/named/zones/
* 4. on good exit dbdata contains memory based zone structure
*********************/
PRIVATE isc_result_t example_create(const char *zone, int argc, char **argv,
       void *driverdata, void **dbdata)
{
  isc_result_t result = ISC_R_FAILURE;
  if (argc != 2){
    isc_log_write(ns_g_lctx, NS_LOGCATEGORY_GENERAL,NS_LOGMODULE_SERVER,
 ISC_LOG_ERROR, "Example Driver: No file defined for %s", zone);
        return result;
       };
  *dbdata = example_read_zone(driverdata, argv[1],zone);
  if(*dbdata == NULL){ // failed
    return result;
  };
  return ISC_R_SUCCESS;
};
```

The sample driver uses a file name argument (`argv[1]`) supplied as part of the `create()` callback and defined in the `database` statement for the zone. The `driverdata` variable that was set up during the `dns_sdb_register()` function call is used as a parameter to `example_read_zone()`. The function uses the logging service (see the section "Logging for Drivers" later in the chapter) to log an error if the required file name parameter is not present. The function `example_read_zone()` allocates a block of memory in which it stores the complete parsed zone file (in a structure called `RRSET`). This memory structure is returned as the `dbdata` value, which is subsequently returned on all `lookup()`, `authority()`, `allnodes()`, and `destroy()` callbacks for this zone.

`destroy()` Callback Function

The `destroy()` function is called when the zone is closed, and provides the opportunity to release memory, close files, terminate database connections, and so on. The prototype is as follows:

```
void (*dns_sdbdestroyfunc_t)(const char *zone, void *driverdata, void **dbdata);
```

Where

- `zone` is the null-terminated string containing the zone name.
- `driverdata` is the global data that was optionally supplied in the `dns_sdb_register()` function call.
- `dbdata` is the instance-specific data that was optionally supplied on the `create()` callback.

The following fragment shows use of the `destroy()` function:

```
/*********************
*
* example_destroy
* handle destroy callback for the example driver
* deallocate memory allocated at create
*********************/
PRIVATE void example_destroy(const char *zone, void *driverdata, void **dbdata)
{
  isc_mem_free(ns_g_mctx, dbdata);
  return;
};
```

In the sample driver, the `dbdata` parameter is a dynamic memory chunk allocated during the `create()` callback (in the function `example_read_zone()`) and is freed in the `example_destroy()` function.

`lookup()` Callback Function

The `lookup()` callback is invoked on receipt of a query for any zone that uses the nominated driver; that is, it has a `database` statement in the `zone` clause that references the `"name"` of this driver. The `lookup` prototype is shown here:

```
isc_result_t  (*dns_sdblookupfunc_t)(const char *zone, const char *name,
                void *dbdata, dns_sdblookup_t *lookup);
```

Where

- `isc_result_t` is the result code—see the section "`isc_result_t` Return Codes" earlier in the chapter.
- `zone` is a null-terminated string containing the zone name.
- `name` is the required hostname in the query. This value may be a relative (unqualified) name if the `DNS_SDBFLAG_RELATIVEOWNER` flag was present in the `dns_sdb_register()` function call (the relative name of the zone apex is represented as `"@"`). If this flag was not present, then an FQDN will be supplied; for instance, `joe.example.com.` (with a dot). The driver should return *all* RRs, using the `dsn_sdb_putrr()`, `dns_sdb_putrdata()`, or `dns_sdb_putsoa()` functions, with the supplied `name`. It is left to BIND to select and return the appropriate RRs.
- `dbdata` is the value that was optionally supplied in the `create()` callback, and may be NULL.
- `lookup` is a `dns_sdblookup_t` structure that is used to return the results of the `lookup()` callback.

The `lookup` (`dns_sdblookup_t`) structure is referenced in any `dns_sdb_putrr()`, `dns_sdb_putsoa()`, or `dns_sdb_putrdata()` calls used to return RRs, as shown in the following fragment:

```
/********************
*
* example_lookup
* handle lookup callback for the example driver
* trivial exhaustive scan of the whole memory structure
*
********************/
isc_result_t  example_lookup(const char *zone, const char *name,
                void *dbdata, dns_sdblookup_t *lookup)
{
  RRSET_ID rrs = dbdata;
  isc_result_t result = ISC_R_NOTFOUND; // default
  while(*(&rrs->owner[0]) != '~') // ~ is EOZ marker
  {
    if(strcmp(name,rrs->owner))
    {
      result = ISC_R_SUCCESS;
      result = dns_sdb_putrr(lookup,rrs->type,rrs->ttl, rrs->rdata);
      if (result != ISC_R_SUCCESS)
      {
        return result;  // error exit
      };
    };
    ++rrs;
  };
  return result;
};
```

In the sample driver, each RR is returned using `dns_sdb_putrr()`, which is a purely textual interface. This includes the SOA and NS RRs, hence in this driver the `authority()` callback is not required.

`authority()` Callback Function

The `authority()` callback is optional. It is invoked on receipt of a query for the zone apex if the SOA and NS RRs were not returned with the preceding `lookup()` callback. The `authority()` prototype is shown in the following code:

```
isc_result_t  (*dns_sdbauthorityfunc_t)(const char *zone, void *dbdata,
    dns_sdblookup_t *authority);
```

Where

- `isc_result_t` is the result code—see the section “`isc_result_t` Return Codes” earlier in the chapter.
- `zone` is a null-terminated string containing the zone name.
- `dbdata` is a user-created value that was optionally supplied in the `create()` callback. This value is returned on all `lookup()`, `allnodes()`, `authority()`, and `destroy()` calls.

- authority is a dns_sdblookup_t structure that contains the RRs returned by the authority() callback, and is referenced in any dns_sdb_putrr(), dns_sdb_putrdata(), or dns_sdb_putsoa() calls used to return RRs. Each function is described in the section “Returning RRs” later in the chapter.

allnodes() Callback Function

The allnodes() callback is optional. It is invoked on receipt of a zone transfer request for the zone. The allnodes() prototype is shown here:

```
isc_result_t (*dns_sdballnodesfunc_t)(const char *zone, void *dbdata,
                    dns_sdballnodes_t *allnodes);
```

Where

- isc_result_t is the result code—see the section “isc_result_t Return Codes” earlier in the chapter.
- zone is a null-terminated string containing the zone name.
- dbdata is a user-created value that was optionally supplied in the create() callback. This value is returned on all lookup(), allnodes(), authority(), and destroy() calls.
- allnodes is a dns_sdblookup_t structure that contains the RRs returned by the allnodes() callback.

The allnodes dns_sdblookup_t structure is referenced in any dns_sdb_putnamedrr() or dns_sdb_putnamedrdata() calls used to return RRs, as shown in the following fragment:

```
/*********************
*
* example_allnodes
* handle allnodes callback for the example driver
*
*********************/
PRIVATE isc_result_t example_allnodes(const char *zone, void *dbdata,
                    dns_sdballnodes_t *allnodes)
{
  RRSET_ID rrs = dbdata;
  isc_result_t result = ISC_R_SUCCESS; // default
  while(*(&rrs->owner[0]) != '~'){ // ~ is EOZ marker
    result = dns_sdb_putnamedrr(allnodes,rrs->owner,rrs->type,rrs->ttl, rrs->rdata);
    if (result != ISC_R_SUCCESS){
      return result;  // error exit - error code from dns_sdb_putrr
    };
    ++rrs;
  };
  return result;
};
```

The sample driver's `allnodes()` function simply iterates through the memory version of the zone file and returns all RRs. In the event of any error, it returns the error code returned by the failing function, which all use the "`isc_result_t` Return Codes" values defined earlier in the chapter.

Returning RRs

Five functions are supplied for returning RRs; these are described in this section.

`dns_sdb_putrr()` Function

This function may be used to return RRs by either the `lookup()` or `authority()` callbacks when the data is purely textual. The prototype is as follows:

```
isc_result_t  dns_sdb_putrr(dns_sdblookup_t *lookup, const char *type,
              dns_ttl_t ttl, const char *data);
```

Where

- `isc_result_t` is the result code—see the section "`isc_result_t` Return Codes" earlier in the chapter.
- `lookup()` is the address of the `dns_sdblookup_t` structure supplied in either the `lookup()` or `authority()` callback.
- `type` is a null-terminated string containing the returned RR type.
- `ttl` is the binary TTL associated with the RR.
- `data` is a single null-terminated string containing the `RDATA` for the RR type. Thus, if the RR type being returned is an MX RR, this field would contain both the *preference* field and the *name* (for example, "10 mail" or "10 mail.example.com."). Either a relative name or an FQDN is acceptable, and does not depend on the setting of the `DNS_SDBFLAG_RELATIVEOWNER`, which refers to the owner (or left-hand name only). This function may be used to return the SOA; alternatively, the optimized function `dns_sdb_putsoa()` may be used. In either case, it is important that the serial number is incremented if zone data has changed since the last callback. This function is used to return all the RRs in the sample application.

`dns_sdb_putrdata()` Function

This function may be used to return RR records to the `lookup()` or `authority()` callbacks in binary format. The prototype is as follows:

```
isc_result_t  dns_sdb_putrdata(dns_sdblookup_t *lookup,
        dns_rdatatype_t type, dns_ttl_t ttl,
        const unsigned char *rdata, unsigned int rdlen);
```

Where

- `isc_result_t` is the result code—see the section "`isc_result_t` Return Codes" earlier in the chapter.

- lookup is the address of the dns_sdblookup_t structure supplied in either the lookup() or authority() callback.
- type is the binary RR number.
- ttl is the TTL associated with the RR.
- rdata is a binary representation of the RDATA field for the RR type. Relative names (containing labels with the top two bits set to 11; see "NAME Field Format," located in Chapter 15) can only appear in this binary field if the DNS_SDBFLAG_RELATIVERDATA flag that was supplied in the dns_sdb_register() function has been set. Otherwise, it is assumed that all names are labels (top two bits set to 00) and must be FQDNs (end with a dot).

dns_sdb_putsoa() Function

This function is optimized to simplify returning SOA RRs and may be used by either the lookup() or authority() callbacks. The prototype is as follows:

```
isc_result_t  dns_sdb_putsoa(dns_sdblookup_t *lookup, const char *mname,
        const char *rname, isc_uint32_t serial);
```

Where

- isc_result_t is the result code—see the section "isc_result_t Return Codes" earlier in the chapter.
- lookup is the address of the dns_sdblookup_t structure supplied in either the lookup() or authority() callback.
- mname is a null-terminated string containing the MNAME field—the name of the primary-master server for the zone.
- rname is a null-terminated string containing the RNAME field (the e-mail address of the zone administrator), and by convention uses hostmaster.example.com.
- serial is the binary serial number for the zone.

All the other fields in the SOA RR are defaulted; that is, refresh, expiry, and so on. The sample driver does not use this function call, but the following fragment shows how it could be used in the lookup() callback function:

```
dns_sdb_putsoa(lookup, "ns1.example.com", "hostmaster.example.com", 2005042900);
```

dns_sdb_putnamedrr() Function

This function may be used when returning text RRs during an allnodes() (zone transfer) callback. The prototype is as follows:

```
isc_result_t dns_sdb_putnamedrr(dns_sdballnodes_t *allnodes, const char *name,
    const char *type, dns_ttl_t ttl, const char *data);
```

The fields used in this function are in every respect the same as those used in the dns_sdb_putrr() function described previously.

`dsn_sdb_putnamedrdata()` Function

This function may be used when returning binary (wire-format) RRs during an `allnodes()` (zone transfer) callback. The prototype is as follows:

```
isc_result_t dns_sdb_putnamedrdata(dns_sdballnodes_t *allnodes, const char *name,
      dns_rdatatype_t type, dns_ttl_t ttl,
      const void *rdata, unsigned int rdlen);
```

The fields used in this function are in every respect the same as those used in the `dns_sdb_putrdata()` function call, and are described previously.

Memory Management for Drivers

Memory for use in driver instances should be allocated via BIND's memory allocation functions to assist in any debug problems and to allow BIND to manage all memory usage. Memory management functions are defined in `bind-release/lib/isc/include/isc/mem.h`, and this header should be included into the driver module if these functions are required. Although a number of functions are provided, the following may be used to obtain memory and to free memory.

`isc_mem_get()` Function

This BIND function may be used to allocate any required memory for use by the driver:

```
void *isc_mem_get(isc_mem_t *context, size_t size);
```

Where

- `void *` is the start address of the allocated memory. If set to NULL on return it means that no memory is available, and the requesting callback function should return with a status of `ISC_R_NOMEMORY`.
- `context` is the name-server memory context and should be set to `ns_g_mctx`.
- `size` is the size in bytes of the memory required.

`isc_mem_free()` Function

This BIND function deallocates the memory obtained by the corresponding `isc_mem_get()` function:

```
void isc_mem_free(isc_mem_t context, void *memory);
```

Where

- `context` is the name-server memory context and should be set to `ns_g_mctx`.
- `memory` is the address of the start of memory allocated by an `isc_mem_get()` function.

Logging for Drivers

Drivers may invoke BIND's logging services. The available functions are accessed via the header `named/log.h`, which must be included in the driver module if any logging function is used. The following describes `isc_log_write()`, the most useful generic log function, which is used in the sample code.

`isc_log_write()` Function

This `isc_log_write()` function may be used to write a log entry. The prototype, contained in `bind-release/lib/isc/include/isc/log.h`, is as follows:

```
void isc_log_write(isc_log_t *lctx, isc_logcategory_t *category,
      isc_logmodule_t *module, int level,
      const char *format, ...);
```

Where

- `lctxt` indicates a BIND logging context and should be set to `ns_g_lctx`.
- `category` is the log category as defined in the `category` statement (see the section "BIND `logging` Statements," located in Chapter 12) of a `logging` clause. It takes the following values:
  - `NS_LOGCATEGORY_GENERAL`: General—default
  - `NS_LOGCATEGORY_CLIENT`: Client
  - `NS_LOGCATEGORY_NETWORK`: Network
  - `NS_LOGCATEGORY_UPDATE`: Update
  - `NS_LOGCATEGORY_UPDATE_SECURITY`: Update security
  - `NS_LOGCATEGORY_QUERIES`: Query
  - `NS_LOGCATEGORY_UNMATCHED`: Unmatched

  The most useful logging value is `NS_LOGCATEGORY_GENERAL`, which is written to the default logging category. However, to assist in debugging, a seldom-used category such as *unmatched* could be used with appropriate `category` and `channel` statements in the `logging` clause.
- `module` defines the module being used and should be set to `NS_LOGMODULE_SERVER`.
- `level` defines the log level number and should be set to `ISC_LOG_ERROR`.
- `format` is a field that accepts normal `printf` arguments.

The following shows an example of a log message:

```
isc_log_write(ns_g_ltcx, NS_LOGCATEGORY_GENERAL,
   NS_LOGMODULE_SERVER, ISC_LOG_ERROR,
    "Example zone %s: Failed status=%d", zone,status);
```

In the preceding code, `zone` would be defined as a null-terminated string variable and `status` as an integer variable.

Testing the Driver

The following line test compiles the `example.c` module. The `-I` argument is necessary to pick up various BIND header files, the `-o` argument defines the output file name, and the `-c` argument limits the operation to compile only:

```
# gcc example.c -o example.o -c -I include
```

To test compile BIND's `main.c` module in isolation, use the following line (the `-I` and `-D` directives suppress BIND environment errors):

```
# gcc main.c -o main.o -c -I include -I unix/include -I ../../lib/bind \
-I ../../lib/isc/include \
-D NS_SYSCONFDIR="" -D NS_LOCALSTATEDIR="" -D VERSION=__VERSION__
```

The "\" splits the line for presentation purposes only, and the command should appear on a single line.

Building BIND

The following command sequence configures and makes BIND in a test location of `/var/etc` for the `named.conf` file. This test location keeps it separate from any current installation version of `named.conf`:

```
# make distclean
# ./configure --prefix=/usr --sysconfdir=/var/etc --localstatedir=/var \
--disable-threads  --with-openssl
# make
```

The line ending with "\" is split for presentation reasons only, and should appear on a single line.

If an existing version of BIND 9 has been installed on the system, there is no need to run `make install` until the new software has been fully tested. The following command line runs BIND from the `bind-release` directory (assumed to be `/usr/src`):

```
# /usr/src/bind-9.3.0/bin/named/named -u named
```

The command-line arguments should be the same as those used on the existing BIND installation. Only when the software is production ready should it be built with the final locations, and installed as shown in the following code:

```
# make distclean
# ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var \
--disable-threads  --with-openssl
# make
# make install
```

The line ending with "\" is split for presentation reasons only and should appear on a single line.

sdb Sample Driver

The listings for the sample application driver are shown in the following sections.

Source Module (`example.c`)

The following listing is the complete source module for `example.c`. Fragments of this module were used to illustrate various functions throughout the preceding sdb API function descriptions. It is presented here for completeness only.

```
// example sdb driver for BIND
// reads and parses zone file into ram structure
// INCLUDES
// add any required std function includes used by driver
#include <stdio.h> // for fopen etc.

// BIND includes
#include <isc/mem.h> // required for isc_mem_t structure
#include <isc/result.h> // result codes
#include <dns/sdb.h> // std headers for all sdb functions
#include <named/globals.h> // BIND globals
#include <named/log.h> // for isc_log_write
// driver includes
#include "example.h"  // header for sample driver

// DEFINES and MACROS
#define DRIVERNAME "example"
#define PRIVATE static
#define EXAMPLE_ZONE_SIZE (2 * 1024) // fixed memory allocation
#define FILENAME_SIZE 50
#define BUFFER_SIZE 200
// STRUCTURES
/* trivial structure to hold RRs */
typedef struct rrset_tag{
  char         owner[30];  // owner name
   unsigned int ttl;       // TTL
   char         type[10];  // RR type
   char         rdata[50];  // rdata
}RRSET, *RRSET_ID;
// PRIVATE PROTOTYPES
PRIVATE isc_result_t example_create(const char *zone, int argc, char **argv,
       void *driverdata, void **dbdata);
PRIVATE void example_destroy(const char *zone, void *driverdata, void **dbdata);
PRIVATE isc_result_t  example_lookup(const char *zone, const char *name,
                void *dbdata, dns_sdblookup_t *);
```

```
PRIVATE isc_result_t example_allnodes(const char *zone, void *dbdata,
                    dns_sdballnodes_t *allnodes);
PRIVATE void * example_read_zone(char *directory, char *file,const char* zone);
// PRIVATE VARIABLES
// list of callback functions in dns_sdbmethods_t structure
PRIVATE dns_sdbmethods_t example_callbacks = {
   example_lookup, // lookup callback function - mandatory
   NULL, // authority callback function - optional
   example_allnodes, // allnodes callback function - optional
   example_create, // create callback function - optional
   example_destroy // destroy callback function - optional
};
// pointer to handle allocated by BIND and supplied to dns_sdb_unregister
PRIVATE dns_sdbimplementation_t *namedhandle = NULL;

// PRIVATE FUNCTIONS
/*********************
*
* example_create()
* handle create callback for the example driver
* 1. call example_read_zone
* 2. update dbdata if OK
* 3. driverdata contains /var/named/zones/
* 4. on good exit dbdata contains memory based zone structure
*********************/
PRIVATE isc_result_t example_create(const char *zone, int argc, char **argv,
       void *driverdata, void **dbdata)
{
  isc_result_t result = ISC_R_FAILURE;
  if (argc != 2){
    isc_log_write(ns_g_lctx, NS_LOGCATEGORY_GENERAL,
    NS_LOGMODULE_SERVER, ISC_LOG_ERROR,
      "Example Driver: No file defined for %s", zone);
       return result;
      };
  *dbdata = example_read_zone(driverdata, argv[1],zone);
  if(*dbdata == NULL){ // failed
    return result;
  };
  return ISC_R_SUCCESS;
};
/*********************
*
* example_read_zone
* 1. read zone file
* 2. allocate fixed memory chunk (2K)
* 3. add count of RRs
* 4. populate RR structure
* return NULL = error else address of buffer containing zone file
```

```
*********************/
PRIVATE void * example_read_zone(char *directory, char *file, const char* zone)
{
  char filename[FILENAME_SIZE];  // zone file name
  char buffer[BUFFER_SIZE];   // zone file line buffer
  FILE *fp;
  RRSET_ID rrs;
  strcpy(filename, directory);
  strcpy(&filename[0] + strlen(directory), file);
  if(!fopen(filename,"r")){
    isc_log_write(ns_g_lctx, NS_LOGCATEGORY_GENERAL,
      NS_LOGMODULE_SERVER, ISC_LOG_ERROR,
      "Example Driver: Zone %s File %s does not exist", zone, filename);
    return NULL;
  }
  // allocate fixed memory for file
  // very crude for example purposes only
  rrs = (RRSET_ID)isc_mem_get(ns_g_mctx,EXAMPLE_ZONE_SIZE);
  if (rrs == NULL){ // failed
  isc_log_write(ns_g_lctx, NS_LOGCATEGORY_GENERAL,
      NS_LOGMODULE_SERVER, ISC_LOG_ERROR,
      "Example Driver: isc_mem_get fail");
    return NULL;
  }
  while(fgets(buffer, BUFFER_SIZE, fp) != NULL){
    /* read file discard comments
    *  populate structure rrs
    * all zone apex RRs will have @ in owner name */
  };
  ++rrs; // point to next structure entry
  // add dummy end record
  *(&rrs->owner[0]) = '~'; // special end name
  fclose(fp);  // close file
  return rrs;  // return dbdata containing zone file
};
/*********************
*
* example_lookup
* handle lookup callback for the example driver
* trivial exhaustive scan of the whole memory structure
*
*********************/
isc_result_t  example_lookup(const char *zone, const char *name,
                void *dbdata, dns_sdblookup_t *lookup)
```

```
{
  RRSET_ID rrs = dbdata;
  isc_result_t result = ISC_R_NOTFOUND; // default
  while(*(&rrs->owner[0]) != '~')
  {
    if(strcmp(name,rrs->owner))
    {
      result = ISC_R_SUCCESS;
      result = dns_sdb_putrr(lookup,rrs->type,rrs->ttl, rrs->rdata);
      if (result != ISC_R_SUCCESS)
      {
        return result;  // error exit
      };
    };
    ++rrs;
  };
  return result;
};

/*********************
*
* example_allnodes
* handle allnodes callback for the example driver
*
*********************/
PRIVATE isc_result_t example_allnodes(const char *zone, void *dbdata,
                    dns_sdballnodes_t *allnodes)
{
  RRSET_ID rrs = dbdata;
  isc_result_t result = ISC_R_SUCCESS; // default
  while(*(&rrs->owner[0]) != '~'){
    result = dns_sdb_putnamedrr(allnodes,rrs->owner,rrs->type,rrs->ttl, rrs->rdata);
    if (result != ISC_R_SUCCESS){
      return result;  // error exit
    };
    ++rrs;
  };
  return result;
};
/*********************
*
* example_destroy
* handle destroy callback for the example driver
* deallocate memory allocated at create
*********************/
PRIVATE void example_destroy(const char *zone, void *driverdata, void **dbdata)
```

```
{
  isc_mem_free(ns_g_mctx, dbdata);
  return;
};

// PUBLIC FUNCTIONS
/*********************
*
* example_init
* register callbacks for the example driver
* Note: In this driver the DNS_SDBFLAG_RELATIVERDATA flag
* is not strictly needed and is used only to illustrate the use of multiple flags
* the variable 'directory' is used to illustrate that
* parameters may be supplied on this call
* equally the function may or may not return a value
*
*********************/
isc_result_t  example_init (char directory[])
{
   // initialize flags
   unsigned int flags = DNS_SDBFLAG_RELATIVEOWNER | DNS_SDBFLAG_RELATIVERDATA;
   return (dns_sdb_register(DRIVERNAME,&example_callbacks,
      directory,flags,ns_g_mctx, &namedhandle));
};
/*********************
*
* example_clear
* unregister callbacks for this driver
*
*********************/

void example_clear(void)
{
   if (namedhandle != NULL){
      dns_sdb_unregister(&namedhandle);
   };
};
```

Header File (`example.h`)

`example.h` follows:

```
// example sdb driver for BIND
// reads and parses zone file into ram structure
// header file
isc_result_t example_init (char directory[]);
void example_clear(void);
```

Resolver Libraries

A confusing number of library functions are available with BIND. They fall into three categories:

- The traditional so-called `res_` library set has been available for some time, but it has been updated with newer BIND 9 and DNSSEC.bis features. These library functions are controlled by `resolv.conf` (normally `/etc/resolv.conf`). This library is primarily concerned with providing programmatic access to the construction of DNS queries. This library, for the sake of convenience, will be called the RES Library, even though not all of its functions start with `res_`.
- An `lwres` library set is new with BIND 9. This library is not yet widely implemented and is not described further in this book.
- The standard POSIX functions, for instance `gethostaddr()`, are called the POSIX DNS Library. The current status of POSIX calls is summarized briefly in the next section.

Many of the classic operations have corresponding functions in each library. However, functions can be mixed and matched from all the available libraries. Although all the samples and most of the code descriptions use C, there are a number of language wrappers (for example, Python, Java, and Ruby). A number of these language wrappers are listed at the author's web site for this book at `www.netwidget.net/books/apress/dns`.

POSIX Library Status

The POSIX DNS calls are defined in IEEE 1003.1 - 2001. However, because IETF documents are freely available, unlike the IEEE documents, they are also described in RFC 3493, which has INFORMATIONAL status and defers to the IEEE specification as being definitive. The available POSIX DNS functions are listed with brief notes about their status, as follows:

- `gethostbyname()`: Name to Address translation. IPv4 only. Not thread safe.
- `gethostbyaddr()`: Number to Name (reverse-map) translation. IPv4 only. Not thread safe.
- `getaddrinfo()`: Name to Address translation. IPv4 and IPv6. Thread safe.
- `freeaddrinfo()`: Free resources used by `getaddrinfo()`. Thread safe.
- `getnameinfo()`: Address to Name translation. IPv4 and IPv6. Thread safe.

The functions `gethostbyname2()` and `gethostbyaddr2()` were tactical updates to enable a basic IPv6 service, and are now deprecated in favor of `getaddrinfo()` and `getnameinfo()`. The functions `gethostbyname_r()` and `gethostbyaddr_r()` were, again, tactical implementations to provide thread safe calls and were never defined formally. The functions `getipnodebyname()` and `getipnodebyaddr()` were defined in RFC 2553 and are deprecated in favor of `getaddrinfo()` and `getnameinfo()`, respectively. `getnameinfo()` and `gethostinfo()` should be used for all new implementations (IPv4 or IPv6).

The RES Library Set

The functionality of this set of library calls is controlled by resolv.conf (normally /etc/resolv.conf), whose format and use is described in the section "RES Library Functions" later in the chapter. Additionally, the behavior may be modified by an available structure called _res, which is accessible by the library user.

Invoking the RES Library

Each function in this library is contained in libc (GLIBC on FreeBSD), and requires the following #includes in all modules:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
```

The _res Structure

Although most of the time the default behavior is acceptable, it is possible to modify the behavior of certain operations by manipulating the _res structure, which is contained in resolve.h (normally /usr/include/resolv.h), and is shown in the following code:

```
struct __res_state {
 int    retrans;  /* retransmission interval */
 int    retry;  /* no. of retransmits */
 u_long options;  /* option flags */
 int    nscount;    /* number of name servers */
 struct sockaddr_in nsaddr_list[MAXNS];
/* address of name server */
#define nsaddr nsaddr_list[0]
/* for backward compatibility */
 u_short id;  /* current message id */
 char *dnsrch[MAXDNSRCH+1];
/* components of domain to search */
 char defdname[256]; /* default domain (deprecated) */
 u_long pfcode; /* RES_PRF_ flags - see below. */
 unsigned ndots:4;
/* threshold for initial abs. query */
 unsigned nsort:4; /* number of elements in sort_list[] */
 char unused[3];
 struct {
  struct in_addr addr;
  u_int32_t mask;
 } sort_list[MAXRESOLVSORT];
 char pad[72]; /* on an i386 this means 512b total */
};
```

```
/* for INET6 */
/*
* replacement of __res_state, separated to
* keep binary compatibility.
*/
struct __res_state_ext {
 struct sockaddr_storage nsaddr_list[MAXNS];
  struct {
  int af;  /* address family for addr, mask */
  union {
  struct in_addr ina;
  struct in6_addr in6a;
  } addr, mask;
 } sort_list[MAXRESOLVSORT];
};
```

Various flags may be manipulated in the `__res_state.options` field to control system behavior:

- `RES_INIT`: Set when the structure has been initialized; that is, when `res_init()` has been called.
- `RES_DEBUG`: Display debug information.
- `RES_AAONLY`: Accept authoritative answers only. When this option is set, `res_send()` should continue until it finds an authoritative answer or finds an error. Currently, this is not implemented.
- `RES_USEVC`: Use TCP instead of UDP for queries.
- `RES_STAYOPEN`: Used with `RES_USEVC` to keep the TCP connection open between queries.
- `RES_IGNTC`: Unused currently. Ignore truncation errors; that is, don't retry with TCP.
- `RES_RECURSE`: Set the recursion-desired bit in queries. This is the default for `res_query()`, `res_search()`, and `res_send()`. If this bit is not set, the functions do not perform iterative queries, and expect the initiator to handle all referrals.
- `RES_DEFNAMES`: If set, `res_search()` appends the default domain name (from `resolv.conf`) to unqualified names (those that do not contain a dot). This option is enabled by default but is only used by `res_search()`.
- `RES_DNSRCH`: If this option is set, `res_search()` searches for host names in the current domain and in parent domains. This option is enabled by default.
- `RES_NOALIASES`: This option turns off the user-level aliasing feature controlled by the `'HOSTALIASES'` environment variable. Network daemons should set this option.
- `RES_USE_INET6`: Enables support for IPv6-only applications. This causes IPv4 addresses to be returned as an IPv4 mapped address. For example, 10.1.1.1 is returned as `::ffff:10.1.1.1` (see Chapter 5).
- `RES_USE_EDNS0`: Enables support for the OPT meta (pseudo) RR for the EDNS0 extension. With the option set, an OPT pseudo RR is added to the `ADDITIONAL SECTION` of all queries.

RES Library Functions

The RES Library provides a number of functions, each of which is described in the following sections.

dn_comp Function

The dn_comp() function converts an ASCII name to a form in which it can be used in a query. Depending on the calling parameters it may also compress a domain name by removing suffixes.

The prototype is as follows:

```
int dn_comp(const char *exp_dn, u_char *comp_dn, int length,
      u_char **dnptrs, u_char **lastdnptr);
```

Where

- The function returns the length of the compressed name, or -1 if an error occurs.
- exp_dn is a pointer to the name that needs to be compressed.
- comp_dn is a pointer to a buffer to contain the compressed name.
- length is the length of the comp_dn buffer.
- dnptrs is a pointer to an array of pointers that contain previous names in the query, which we'll call the *compressed list array* for ease of explanation. If dnptrs is NULL, then only conversion to uncompressed label format is performed (see the section "NAME Field Format" in Chapter 15). However, if the value of dnptrs and lastdnptr is initialized to the address of the *compressed list array,* and the initial entry in the compressed list array is set to NULL, then the function dn_comp() will use the *compressed list array* as a list of name pointers that may be examined when searching for suffix compression possibilities. As each name is added, the dn_comp() function adds its pointer to the *compressed list array,* and lastdnptr is updated to point to the last entry in the *compressed list array.*
- lastdnptr is an address pointer that points to the last valid entry in the dnptrs array, and is updated on termination of the function. If lastdnptr is set to NULL, then the *compressed list array* is not updated.

dn_expand Function

The dn_expand() entry expands a compressed domain name returned in a query message.

The prototype is as follows:

```
int dn_expand(const u_char *msg, const u_char *eomorig,
      const u_char *comp_dn, char *exp_dn, int length);
```

Where

- The function returns the length of the expanded name supplied, or -1 if an error occurs.
- msg points to the buffer that contains the messages returned from a res_send(), res_query(), or res_search() function.
- eomorig is a pointer to the end of the query message, whose start is pointered to by msg, and which contains the name to be expanded.

- `comp_dn` is a pointer to the beginning of the name to be expanded.
- `exp_dn` is a pointer to a buffer that contains the expanded name on termination of the function.
- `length` is the size of the buffer `exp_dn`, and should be big enough to contain the largest possible expanded name.

dn_skipname Function

The `dn_skipname()` function skips over a compressed domain name in a query response and can be used to step through a message. By adding the returned value to a current pointer, an application can move through, or parse, a returned query.

The prototype is as follows:

```
int dn_skipname(const u_char *comp_dn, const u_char *eom);
```

Where

- The size of the compressed name is returned, or `-1` if there is an error.
- `comp_dn` is a pointer to the beginning of the name to be skipped.
- `eom` is a pointer to the end of the query message containing the name to be skipped.

ns_get16 Function

The `ns_get16()` function gets a 16-bit quantity from a buffer pointed to by `src`.

The prototype is as follows:

```
u_int ns_get16(const u_char *src);
```

ns_get32 Function

The `ns_get32()` function gets a 32-bit quantity from a buffer pointed to by `src`.

The prototype is as follows:

```
u_long ns_get32(const u_char *src);
```

ns_put16 Function

The `ns_put16()` function puts a 16-bit quantity, `src`, into the buffer pointed to by `dst`.

The prototype is as follows:

```
void ns_put16(u_int src, u_char *dst);
```

ns_put32 Function

The `ns_put32()` function puts a 32-bit quantity, `src`, into the buffer pointed to by `dst`.

The prototype is as follows:

```
void ns_put32(u_long src, u_char *dst);
```

res_init() Function

res_init() reads resolv.conf to obtain the default domain name, search list, and the IP address(es) of the local name server(s). This function call should be issued first to initialize all necessary variables. However, if this is not done, it is called automatically by the res_send(), res_query(), or res_search() functions. On completion of this function RES_INIT is set.

The prototype is as follows:

```
int res_init();
```

The function takes no parameters, but returns 0 on success or -1 if there was insufficient memory or any other error.

res_mkquery Function

The res_mkquery() function constructs a standard query message in a user-supplied buffer, but does not send it. Instead, the res_send() function must be called to dispatch the query.

The prototype is as follows:

```
int res_mkquery(int op, const char *dname, int class,
      int type, const u_char *data, int datalen,
      const u_char *newrr_in, u_char *buf, int buflen);
```

Where

- The function returns the length of the created query or -1 if an error occurs.
- op is the type of operation required, and is normally set to QUERY (see Chapter 15).
- dname is a pointer to a null-terminated string containing the fully qualified name (FQDN) to be used in the query; for instance, "www.example.com."
- class is the binary value of the class type; for example, the value 1 is IN (Internet class).
- type is the binary value of the required RR type, including ANY (type = 255).
- data: Historical use with inverse queries (now obsoleted) should be set to NULL.
- datalen: Historical use with inverse queries should be set to 0.
- newrr_in is intended for use with DDNS, and is not currently supported.
- buf is a pointer to a buffer that contains the created query.
- buflen is the length of the query buffer, and should be long enough to contain the largest possible query.

res_query Function

The res_query() function takes the supplied parameters, automatically constructs a query, and awaits the reply. It bypasses any rules defined by RES_DEFNAMES and RES_DNSRCH. Thus, all supplied names should be fully qualified (FQDNs). It calls res_init() if RES_INIT is not set.

The prototype is as follows:

```
int res_query(const char *dname, int class, int type,
     u_char *answer, int anslen);
```

Where

- The function returns the length of the response or -1 if an error occurs.
- dname is a pointer to a null-terminated string containing the fully qualified name to be used in the query; for instance, www.example.com.
- class is the binary value of the class type; for example, the value 1 is IN (Internet class).
- type is the binary value of the required RR type, including ANY (type = 255).
- answer is a pointer to a buffer that contains the query response.
- anslen is the length of the answer buffer, and should be long enough to contain the largest possible response.

res_search Function

The res_search() function takes the supplied parameters and automatically constructs a query and awaits the reply, like res_query(). However, in addition, it implements the default name and search rules controlled by the RES_DEFNAMES and RES_DNSRCH options. It calls res_init() if RES_INIT is not set. It returns the first successful reply.

The prototype is as follows:

```
int res_search(const char *dname, int class, int type,
     u_char *answer, int anslen);
```

Where

- The function returns the length of the response, or -1 if an error occurs.
- dname is a pointer to a null-terminated string containing the FQDN or an unqualified name to be used in the query; for instance, "www".
- class is a binary value of the class type. For example, the value 1 is IN (Internet class).
- type is the binary value of the required RR type, including ANY (type = 255).
- answer is a pointer to a buffer to contain the response.
- anslen is the length of the answer buffer, and should be long enough to contain the largest possible response.

res_send Function

The res_send() function sends a preformatted query, typically created with res_mkquery(), and returns an answer. It calls res_init() if RES_INIT is not set, sends the query to the local name server, and handles timeouts and retries.

The prototype is as follows:

```
int res_send(const u_char *msg, int msglen, u_char *answer,
      int anslen);
```

Where

- The function returns the length of the reply message, or `-1` if there were errors.
- `msg` is a pointer to the buffer containing the query.
- `msglen` is the length of the query in bytes.
- `answer` is a pointer to a buffer that contains the answer.
- `anslen` is the length of the buffer allocated to contain the answer, and should be large enough to contain the maximum size of the expected response.

Summary

BIND provides two APIs to allow the user to add functionality to the basic name server. Both interfaces are statically linked and require manual editing of core BIND modules. Unlike most dynamically loaded extensions, such as those used in Apache, any failure in the user-supplied functions will likely result in a failure of the named daemon and a consequent loss of DNS service.

The adb interface allows complete replacement of the core zone-processing functionality of BIND. adb must support all the required capabilities of the production system, including zone transfers, dynamic updates, and DNSSEC.bis operations. Use of this API requires a significant outlay of time and resources, with 36 functions being supported. This interface is documented in `bind-release/lib/dns/include/dns/db.h`. The adb API is not described in this book.

The sdb API is an abstracted interface that allows one or more zones to be replaced with one or more drivers. These drivers supply the appropriate data in response to DNS queries for the zones to which the driver applies. The interface operates using up to five callback functions, of which only one, the `lookup()` callback, is mandatory. The sdb interface, as well as additional interfaces that may be used for logging and allocating memory, is described with a sample driver used to illustrate key points.

The next chapter describes the DNS binary or wire-format messages. The chapter is appropriate for those doing serious debugging or for those using the binary version when returning RRs via the sdb interface.

CHAPTER 15

DNS Messages and Records

This chapter describes the binary messages and Resource Record (RR) formats that pass between DNS servers. These messages comprise what is sometimes called the *wire format* because it is the format sent across the network, or *wire*, in the understated slang of the network professional. While it is primarily a reference section with copious descriptions of bits and bytes, this chapter is also intended to allow the naturally curious reader to understand just what is a referral, or when a `dig` command is issued just what are the `ANSWER SECTION` and `AUTHORITY SECTION` that are returned, as well as why are there only 13 root-servers (because of packet size considerations). The chapter is laid out in two sections. The first section details the layout and format of the binary data that passes between servers. The second section defines the binary format of each Resource Record as it appears in a message. In most cases, the fastest and most convenient way to analyze the format of these messages is to use a network or protocol sniffer. These applications typically capture raw network frames and most will provide differing levels of protocol interpretation—translating the various messages into a somewhat understandable form. There are many excellent packages available from a variety of sources, including the superb Open Source Ethereal network analyzer (`www.ethereal.com`), which runs on Linux, Unix, and Windows platforms.

DNS messages depend on whether or not the server initiating the transaction is using Extended DNS (EDNS0—RFC 2671) features. EDNS is used by security transactions such as TSIG and SIG(0), and security-aware servers in DNSSEC.bis transactions (see Chapter 11). In the interest of clarity, EDNS and normal messages are clearly separated. If you are not running secure transactions (secure zone transfers, secure DDNS, or DNSSEC.bis), the "EDNS0 Transactions" section later in this chapter is not relevant. There are three situations where the information contained in this chapter may be of more than superficial interest:

1. First, during network debugging when name servers from multiple vendors or even differently configured name servers from the same source may be experiencing interworking problems. In this situation, it is important to identify which server is, or may be, causing the problem. Having a general understanding of normal traffic is an essential prerequisite to accomplish this analysis.

2. Second, during testing of new or beta software where sniffer analysis software may not be available to perform interpretation of the various formats being used.

3. And finally, the DNS specifications allow new RRs to be created for any user-defined purpose using the special syntax described in the section "User-Defined RRs" located in Chapter 13. These RRs will most certainly not have sniffer analysis support, and verification of the correct format during testing will of necessity be an entirely manual process.

To understand this chapter, you should be reasonably familiar with hexadecimal, binary and decimal representations. The sidebar "Binary, Decimal, and Hexadecimal" is provided as a quick refresher.

BINARY, DECIMAL, AND HEXADECIMAL

The contents of any 8-bit byte (an octet) may be expressed in decimal (base 10), yielding a value in the range 0 to 255; binary (base 2), yielding a value in the range 0000 0000 to 1111 1111; or hexadecimal (base 16), yielding two hex characters in the range 00 to FF. Each hexadecimal character may take values from 0 to 9 and A to F, allowing a total of 16 values to be represented The following table shows a number of arbitrary values represented in all three bases:

Decimal	Hexadecimal	Binary
0	00	0000 0000
65	41	0100 0001
187	BB	1011 1011
255	FF	1111 1111

To convert a dotted decimal IP, for instance, 192.168.0.5 to hexadecimal, take each dotted decimal value and convert it using a hex calculator (the standard Windows calculator, gcalc on Gnome 2 and kcalc on KDE, will all do the job when scientific mode is selected). Thus 192.168.0.5 will yield C0.A8.0.5; when representing a hexadecimal number, it is normal to show both hexadecimal characters, and so this value would be written as C0.A8.00.05 The separating dots are just an easy way to make the number more readable but otherwise have no significance—the value could have been written as C0A80005. To convert from hexadecimal to dotted decimal, simply reverse the process and omit any leading zeros; thus 005 or 05 would normally be written simply as 5 in decimal mode.

Bit Numbering

Bit numbering can be very confusing, with various standard bodies adopting different conventions. The following are all valid, and used, bit-numbering conventions for describing an 8-bit byte.

Left-to-right base 0 (IETF)	0	1	2	3	4	5	6	7
Left-to-right base 1	1	2	3	4	5	6	7	8
Right-to-left base 1 (ITU)	8	7	6	5	4	3	2	1
Power of 2	7	6	5	4	3	2	1	0

Always check what convention is used on any specification. The convention used by the IETF and used in this book is a left-to-right number starting from 0.

DNS Message Formats

When a dig is issued, it generates a corresponding DNS query and typically responds with a lot of data. The response is essentially a translation of the binary message—the answer—that is received to the dig command's question. The following dig command is a simple query for the IPv4 address of www.example.com using the recursive server at 192.168.254.2 (the full syntax of the dig command is described in Chapter 9).

```
# dig @192.168.254.2 example.com any

; <<>> DiG 9.3.0 <<>> @192.186.254.2 www.example.com a
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1947
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.com.       IN   A

;; ANSWER SECTION:
www.example.com.      172800   IN   CNAME   joe.example.com.
joe.example.com.      172800   IN   A       192.168.200.4
;; AUTHORITY SECTION:
example.com.      172800   IN   NS   ns2.example.net.
example.com.      172800   IN   NS   ns1.example.com.

;; ADDITIONAL SECTION:
ns1.example.com.     172800   IN   A   192.168.200.8
ns2.example.net.     172800   IN   A   192.168.254.10

;; Query time: 312 msec
;; SERVER: 192.168.254.2 #53(192.168.254.2)
;; WHEN: Wed Mar 09 22:17:44 2005
;; MSG SIZE  rcvd: 124
```

The response is divided into four parts, or sections:

- The QUESTION SECTION reflects the original query that is being answered, which in this case was a query for the A RR of www.example.com.
- The ANSWER SECTION provides two answers to the query. The first indicates that www.example.com is a CNAME, and since the server follows the CNAME chain because the dig requested an A RR (not a CNAME RR), it supplies the A RR of the canonical (or real) host—in this case, the A RR of joe.example.com.
- The AUTHORITY SECTION provides the name of the servers that are authoritative for the domain example.com.
- The ADDITIONAL SECTION provides information that may be useful to the server; in this case, it is the A RRs of the name servers.

■**Note** When used with DNSSEC, a fifth section titled `OPT PSEUDOSECTION` is displayed. This is simply a reformatting of the OPT meta (pseudo) RR used in DNSSEC transactions from the `ADDITIONAL SECTION`.

The next `dig` command is the same A query for `www.example.com`, but in this case the target server is 128.8.10.90, which is one of the 13 root-servers (`D.ROOT-SERVERS.NET` in this case).

```
# dig @128.8.10.90 www.example.com A

; <<>> DiG 9.3.0 <<>> @128.8.10.90 www.example.com a
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 14

;; QUESTION SECTION:
;www.example.com.         IN   A

;; AUTHORITY SECTION:
com.     172800   IN   NS   A.GTLD-SERVERS.NET.
com.     172800   IN   NS   G.GTLD-SERVERS.NET.
com.     172800   IN   NS   H.GTLD-SERVERS.NET.
com.     172800   IN   NS   C.GTLD-SERVERS.NET.
com.     172800   IN   NS   I.GTLD-SERVERS.NET.
com.     172800   IN   NS   B.GTLD-SERVERS.NET.
com.     172800   IN   NS   D.GTLD-SERVERS.NET.
com.     172800   IN   NS   L.GTLD-SERVERS.NET.
com.     172800   IN   NS   F.GTLD-SERVERS.NET.
com.     172800   IN   NS   J.GTLD-SERVERS.NET.
com.     172800   IN   NS   K.GTLD-SERVERS.NET.
com.     172800   IN   NS   E.GTLD-SERVERS.NET.
com.     172800   IN   NS   M.GTLD-SERVERS.NET.

;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.     172800   IN   A  192.5.6.30
G.GTLD-SERVERS.NET.     172800   IN   A  192.42.93.30
H.GTLD-SERVERS.NET.     172800   IN   A  192.54.112.30
C.GTLD-SERVERS.NET.     172800   IN   A  192.26.92.30
I.GTLD-SERVERS.NET.     172800   IN   A  192.43.172.30
B.GTLD-SERVERS.NET.     172800   IN   A  192.33.14.30
D.GTLD-SERVERS.NET.     172800   IN   A  192.31.80.30
L.GTLD-SERVERS.NET.     172800   IN   A  192.41.162.30
F.GTLD-SERVERS.NET.     172800   IN   A  192.35.51.30
J.GTLD-SERVERS.NET.     172800   IN   A  192.48.79.30
K.GTLD-SERVERS.NET.     172800   IN   A  192.52.178.30
```

```
E.GTLD-SERVERS.NET.     172800   IN   A  192.12.94.30
M.GTLD-SERVERS.NET.     172800   IN   A  192.55.83.30
A.GTLD-SERVERS.NET.     172800   IN   AAAA  2001:503:a83e::2:30

;; Query time: 46 msec
;; SERVER: 128.8.10.90#53(128.8.10.90)
;; WHEN: Wed Mar 09 22:28:53 2005
;; MSG SIZE  rcvd: 492
```

In this case, the `QUESTION SECTION` is a simple reflection of the `dig` command—the query or question that was sent. There is no `ANSWER SECTION`, because this query is for a user domain (`example.com`), and root-servers do not provide recursive query support. There is, however, an `AUTHORITY SECTION` indicating the next closest name servers that may be able to answer the query—in this case, the `.com` gTLD servers. Finally, the `ADDITIONAL SECTION` provides the A RRs and in one case an AAAA (IPv6) RR of the name servers listed in the `AUTHORITY SECTION`. These RRs are the glue records that will save an additional transaction, since without this information the next step would have to find the A RR of one or more of the supplied servers. This response is a *referral*—it contains no errors (`NOERR` status) and no `ANSWER SECTION`, but information in the `AUTHORITY SECTION` and usually—from the root and TLD servers always—information in the `ADDITIONAL SECTION`.

■ **Note** The AAAA RR is perfectly reasonable in the preceding list, since as we saw in Chapter 13 zone files can mix AAAA and A RRs freely. In the particular case shown, the test server was dual stacked with IPv4 and IPv6, but an IPv4-only server would also have received the AAAA RR.

DNS Message Overview

Message formats are defined in RFC 1035 and were extended by RFC 2671 (EDNS0). To help explain this section, the following `dig` command was issued:

```
# dig @192.168.235.2 www.example.com A
```

The command returned the following text response:

```
; <<>> DiG 9.3.0 <<>> @192.168.235.2 www.example.com a
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 146
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 2, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.com.     IN   A
```

```
;; ANSWER SECTION:
www.example.com.    86400   IN   A   10.1.2.1
www.example.com.    86400   IN   A   192.168.3.1
www.example.com.    86400   IN   A   192.168.4.1
www.example.com.    86400   IN   A   172.16.2.1
www.example.com.    86400   IN   A   172.17.2.1
www.example.com.    86400   IN   A   192.168.2.1

;; AUTHORITY SECTION:
example.com.    86400   IN   NS   ns1.example.com.
example.com.    86400   IN   NS   ns2.example.net.

;; ADDITIONAL SECTION:
ns1.example.com.    86400   IN   A   192.168.2.6

;; Query time: 0 msec
;; SERVER: 192.168.235.2 #53(192.168.235.2)
;; WHEN: Thu Mar 10 15:47:05 2005
;; MSG SIZE  rcvd: 192
```

This answer indicates a successful response (NOERR), with an ANSWER SECTION—in this case many A RRs—and both AUTHORITY and ADDITIONAL SECTIONs. A packet sniffer namely Ethereal was used to capture the question packet content as shown in Figure 15-1. The content shown *excludes* all IP and UDP packet framing and is just the raw DNS message.

```
0000  05 5e 01 00 00 01 00 00 00 00 00 00 03 77 77 77 .............www
0001  07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002  01                                              .
```

Figure 15-1. *DNS query question*

The corresponding answer packet is shown in Figure 15-2 and was again captured using a packet sniffer. This content excludes all IP and UDP packet framing.

```
0000  05 5e 85 80 00 01 00 06 00 02 00 01 03 77 77 77 .............www
0001  07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002  01 c0 0c 00 01 00 01 00 01 51 80 00 04 0a 01 02 ................
0003  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 03 ................
0004  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 04 ................
0005  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 10 02 ................
0006  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 11 02 ................
0007  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 02 ................
0008  01 c0 10 00 02 00 01 00 01 51 80 00 11 03 63 73 ..............ns
0009  32 07 65 78 61 6d 70 6c 65 03 6e 65 74 00 c0 10 2.example.net...
000a  00 02 00 01 00 01 51 80 00 06 03 6e 73 31 c0 10 ...........ns1..
000b  c0 aa 00 01 00 01 00 01 51 80 00 04 c0 a8 02 06 ................
```

Figure 15-2. *DNS query response*

These two messages will be used throughout to illustrate the explanations that follow.

Note The data highlighted in gray in Figures 15-1 and 15-2 indicates the message header, which is explained in the next section.

DNS Message Format

The good news is that each message has the same generic format with five sections, as shown in Table 15-1.

Table 15-1. *DNS Message Format*

Section	Meaning/Use
Section 1	Message header
Section 2	The `QUESTION SECTION`: the DNS query for which a response is being sought
Section 3	The `ANSWER SECTION`: the Resource Record(s) that answer the question
Section 4	The `AUTHORITY SECTION`: the Resource Record(s) that point to the domain authority
Section 5	The `ADDITIONAL SECTION`: the Resource Record(s) that may hold additional information

Not every section is present in every message, and this is indicated in the message header this chapter covers next.

DNS Message Header

The message header is present in all messages. It contains various flags and values that control the transaction. Figure 15-3 shows the format of the message header and uses the standard IETF bit numbering convention throughout.

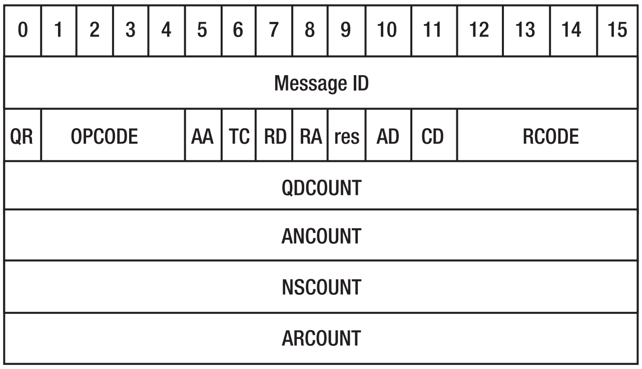


Figure 15-3. *DNS message header*

Table 15-2 defines the field values listed in Figure 15-3.

Table 15-2. *DNS Message Header Values*

Section	Bits	Explanation
`Message ID`	16	The message ID supplied by the requestor (the questioner) and reflected back unchanged by the responder (answerer). Identifies the transaction. Appears as a value of `055e` in Figures 15-1 and 15-2.
`QR`	1	Query-Response bit. Set to `0` by the questioner (query) and to `1` in the response (answer). Not set in Figure 15-1 and set in Figure 15-2.
`OPCODE`	4	Identifies the request/operation type. Currently assigned values are `0` = `QUERY`. Standard query. `1` = `IQUERY`. Inverse query. Made obsolete by RFC 3425. `2` = `STATUS`. DNS status request. `3` = Reserved. `4` = `NOTIFY`. `5` = DDNS update. `6–15` = Unused. Available for assignment. In a response this field reflects the user's request. In Figures 15-1 and 15-2, this field is `0` (a query).
`AA`	1	Authoritative Answer. Valid in responses only. Set if the response was received from a zone master or slave. It is also set the first time the response is received from a master or slave by a caching server, but when subsequently read from the cache the AA bit is not set. Because of aliases (CNAME RRs), multiple owner names may exist, so the AA bit corresponds to the name in the `ANSWER SECTION` that matches the query name. Set in Figure 15-2 to indicate an authoritative response.
`TC`	1	Truncation. Specifies that this message was truncated due to length greater than that permitted on the transmission channel. Set on all truncated messages except the last one. Not set in either Figures 15-1 and 15-3.
`RD`	1	Recursion Desired. This bit may be set in a query and is copied into the response if recursion is supported. If rejected, the response (answer) does not have this bit set. Recursive query support is optional. Set in both Figures 15-1 and 15-2.
`RA`	1	Recursion Available. This bit is valid in a response (answer) and denotes whether recursive query support is available (`1`) or not (`0`) in the name server. Set in Figure 15-2.
`res`	1	Reserved by IANA for future use.
`AD`	1	Authenticated Data. Used by DNSSEC. Indicates that the data was reliably authenticated. A chain of trust was verified (see Chapter 11). Not set in either Figures 15-1 or 15-2, and since the target server used in the example was a security-aware name server, it means the zone was not *signed*.
`CD`	1	Checking Disabled. Used by DNSSEC. If set, it means the initiator of the request (either a server or a resolver) will take responsibility for all security processing. It disables checking at the receiving name server which will pass back all the necessary information such as RRSIG and DNSKEY RRs to the resolver to allow it to perform the transaction validation.

Section	Bits	Explanation
RCODE	4	Identifies the response type to the query. Ignored on a request (question). Currently assigned values include the following: 0 = NOERR. No error condition. 1 = FORMERR. Format error—the name server was unable to interpret the query. 2 = SERVFAIL. Server failure—the name server was unable to process this query either due to a problem with the name server or a requested feature cannot be satisfied such as a recursive request to an authoritative-only name server. 3 = NXDOMAIN. Name error—meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist. 4 = NOTIMP. Not implemented (versions of BIND prior to 9.3 would respond with NOTIMPL)—the name server does not support the requested operation. 5 = REFUSED. The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation, for instance, zone transfer for a particular zone. 6 = YXDomain. Name exists when it should not (RFC 2136). 7 = YXRRSet. RR set exists when it should not (RFC 2136). 8 = NXRRSet. RR set that should exist does not (RFC 2136). 9 = NotAuth. Server not authoritative for zone (RFC 2136). 10 = NotZone. Name not contained in zone (RFC 2136). 11–15 = Unused. Available for assignment. Extended RCODE values with EDNS0 only (see "EDNS0 Transactions" later in this chapter): 16 = BADVERS. Bad OPT version number (not 0) (RFC 2671). 16 = BADSIG. TSIG signature failure (RFC 2845). 17 = BADKEY. Key not recognized. (RFC 2845). 18 = BADTIME. Signature out-of-time window (RFC 2845). 19 = BADMODE. Bad TKEY mode (RFC 2930) or invalid key name. 20 = BADNAME. Duplicate key name (RFC 2930). 21 = BADALG. Algorithm not supported (RFC 2930). 22–3840 = Not used. Available for assignment. 3841–4095 = Private use. 4096–65535 = Not used and available for assignment. The RCODE is 0 (NOERR) in Figure 15-2.
QDCOUNT	16	Defines the number of entries in the QUESTION SECTION. This field is 1 in both Figures 15-1 and 15-2.
ANCOUNT	16	Defines the number of resource records in the ANSWER SECTION. May be 0, in which case no answer records are present in the message. This field is 0 in Figure 15-1 and 6 in Figure 15-2.
NSCOUNT	16	Defines the number of resource records in the AUTHORITY SECTION. May be 0, in which case no authority records are present in the message. This field is 0 in Figure 15-1 and 2 in Figure 15-2.
ARCOUNT	16	Defines the number of resource records in the ADDITIONAL SECTION. May be 0, in which case no additional records are present in the message. This field is 0 in Figure 15-1 and 1 in Figure 15-2.

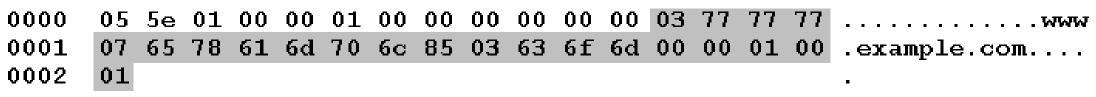
DNS QUESTION SECTION

It is permissible to have only one question per message (defined by QDCOUNT earlier). A question has the generic format defined in Table 15-3.

Table 15-3. *DNS Question Format*

Field Name	Meaning/Use
QNAME	The domain name being queried such as `www.example.com`.
QTYPE	The RR type being requested—values are defined by IANA (`www.iana.org/assignments/dns-parameters`) such as A, ANY, or NAPTR.
QCLASS	The RR class being requested, for instance, Internet, CHAOS, etc.

Figure 15-4 shows the highlighted `QUESTION SECTION`.



```
0000   05 5e 01 00 00 01 00 00 00 00 00 00 03 77 77 77  ............www
0001   07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00  .example.com....
0002   01                                               .
```

Figure 15-4. `QUESTION SECTION`

Each field has the format defined in Table 15-4.

Table 15-4. *DNS Question Fields*

Name	Explanation

QNAME	Defines the name being queried. The name being queried is split into *labels* by removing the separating dots. Each label is represented as a length (one octet) followed by a variable number of characters—the label string—defined by `length`:
	`length`: A single octet defining the number of characters in the label that follows. The top 2 bits of this number must be `00` (to indicate the label format is being used), which gives a maximum label name length of 63 bytes (octets). A value of `0` indicates the end of the `name` field.
	`label string`: A string containing the characters in the label.
	In Figure 15-1, the question name is comprised of three labels terminated with a `00` value, as shown here:
	`03 77 77 77 07 65 78 61 6D 70 6C 65 03 63 6F 6D 00`
	`   w  w  w     e  x  a  m  p  l  e     c  o  m`
	The final `00` (a zero label length) indicates the end of the name.

QTYPE	Unsigned 16-bit value. The RR type being requested. These values are assigned by IANA (`www.iana.org/assignments/dns-parameters`). The values are also listed in Chapter 13, Table 13-1. In Figures 15-1 and 15-2 the value is `1` (indicating an A RR).
QCLASS	Unsigned 16-bit value. The class of resource records being requested, for instance, Internet, CHAOS, etc. These values are assigned by IANA. The currently assigned values are `1` = `IN` or Internet `2` = Obsolete `3` = `CH` (CHAOS) `4` = `HS` (HESIOD) In Figures 15-1 and 15-2 this value is 1 (Internet).

DNS ANSWER, AUTHORITY, and ADDITIONAL SECTIONS

The ANSWER, AUTHORITY, and ADDITIONAL SECTIONs contain RR records that all share the same format. Which section the RR appears in is determined solely by the count of records in each section that is contained in the message header. Thus an A RR can appear in an ANSWER or an ADDITIONAL SECTION. So far, this stuff has been relatively straightforward if messy—take a deep breath before reading on. The generic binary or wire format of RR records is shown in Table 15-5.

Table 15-5. ANSWER, AUTHORITY, *and* ADDITIONAL SECTION *RR Format*

Field Name	Explanation
NAME	The name being returned, for instance www or ns2.example.net. If the name is in the same domain as the question, then typically only the host part (label) is returned and a pointer record used to construct an FQDN; if not, then an FQDN is present. This process is described and illustrated later in this section.
TYPE	The RR type being returned, for instance, NS or AAAA.
CLASS	The RR class being returned, for instance, Internet, CHAOS, etc.
TTL	The TTL of the RR being returned in seconds, for instance, 2800.
RDLENGTH	The length in octets of the RDATA field being returned.
RDATA	The RR-specific data length defined by RDLENGTH, for instance, 192.168.254.2.

NAME Field Format

The NAME field has one of three formats depending on the value of the top 2 bits. Table 15-6 shows the meaning and layout of the three types based on the value of the top 2 bits.

Table 15-6. NAME *Field Format*

Value	Size of Field	Explanation
00	6	This indicates the label format described for the QUESTION SECTION earlier and comprises a series of variable strings whose length is indicated by the low 6 bits of each octet (see Figure 15-4 earlier). The sequence is always terminated with a zero length value. The remaining fields of this record format are defined later in Table 15-7.
11	14	The pointer format. The following 14 bits are assumed to be the offset from the start of the message of a name that must be in standard label format. Figure 15-5 shows a highlighted ANSWER SECTION commencing with a pointer format with an offset of x0c (12 octets) into the message that points to the label record for www.example.com. The remaining fields of this record format are defined later in Table 15-7.
01	6	This denotes an EDNS0 format message (RFC 2671). The low order 6 bits of this field contains an extended TYPE field that, together with the rest of the record format, is described in the section "EDNS0 Transactions."

The format of the subsequent data is determined by the top 2 bits and is described next for non-EDNS0 values (top 2 bits either 00 or 11) and EDNS0 format data (top 2 bits are 01).

```
0000  05 5e 85 80 00 01 00 06 00 02 00 01 03 77 77 77 .............www
0001  07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002  01 c0 0c 00 01 00 01 00 01 51 80 00 04 0a 01 02 ................
0003  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 03 ................
0004  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 04 ................
0005  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 10 02 ................
0006  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 11 02 ................
0007  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 02 ................
0008  01 c0 10 00 02 00 01 00 01 51 80 00 11 03 63 73 ..............ns
0009  32 07 65 78 61 6d 70 6c 65 03 6e 65 74 00 c0 10 2.example.net...
000a  00 02 00 01 00 01 51 80 00 06 03 6e 73 31 c0 10 ...........ns1..
000b  c0 aa 00 01 00 01 00 01 51 80 00 04 c0 a8 02 06 ................
```

Figure 15-5. *Highlighted* ANSWER SECTION

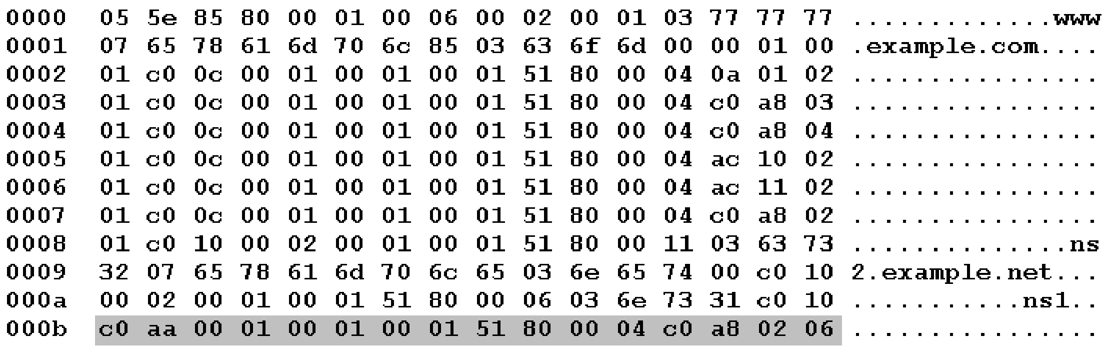
Non-EDNS0 Record Format

The format of a non-EDNS response record—one that has the top 2 bits set to either `00` or `11`—is described in Table 15-7.

Table 15-7. *Non-EDNS Record Format*

Name	Explanation
`TYPE`	Unsigned 16-bit value. The RR type that determines the content of the `RDATA` field (see `RDATA` entry). These values are assigned by IANA (`www.iana.org/assignments/dns-parameters`). In Figure 15-5, the highlighted record has a value of `00 01` = A type RR.
`CLASS`	Unsigned 16-bit value. The class of RR, for instance, Internet, CHAOS, etc. These values are assigned by IANA. In Figure 15-5, the highlighted record has a value of `00 01` = Internet class.
`TTL`	Unsigned 32-bit value. The time in seconds that the record may be cached. A value of `0` indicates the record should not be cached. In Figure 15-5, the highlighted record has a value of `00 01 51 80` (hex) = 86,400 seconds (2 days).
`RDLENGTH`	Unsigned 16-bit value that defines the length in bytes (octets) of the `RDATA` record. In Figure 15-5, the highlighted record has a value of `00 04`, meaning the following record has a length of 4 octets.
`RDATA`	Each RR type has a specific `RDATA` format that is defined in the section "DNS Binary RR Format" later in this chapter. In Figure 15-5, the highlighted record has a value of `0a 01 02 01` (hex) = 10.1.2.1. Since this is an A RR, it defines a 4-octet (32-bit) IPv4 address.

All records create FQDNs, sometimes using chained (pointer) constructs to minimize the amount of data returned. Figure 15-6 shows a highlighted additional section containing an A RR.



```
0000  05 5e 85 80 00 01 00 06 00 02 00 01 03 77 77 77 .............www
0001  07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002  01 c0 0c 00 01 00 01 00 01 51 80 00 04 0a 01 02 ................
0003  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 03 ................
0004  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 04 ................
0005  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 10 02 ................
0006  01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 11 02 ................
0007  01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 02 ................
0008  01 c0 10 00 02 00 01 00 01 51 80 00 11 03 63 73 ..............ns
0009  32 07 65 78 61 6d 70 6c 65 03 6e 65 74 00 c0 10 2.example.net...
000a  00 02 00 01 00 01 51 80 00 06 03 6e 73 31 c0 10 ...........ns1..
000b  c0 aa 00 01 00 01 00 01 51 80 00 04 c0 a8 02 06 ................
```

Figure 15-6. *A RR using chained pointers*

The preceding record starts with a pointer format (top 2 bits are 11) containing an offset of `00 aa` (decimal 170) that points to a label name type containing `03 6e 73 31` (for `ns1`), which is followed by a pointer type (`c0 10`), which in turn points to `example.com`. The last label in the chain is zero length, which stops the label generation phase. The record contains type = `00 01` (A), class = `00 01` (IN), ttl = `00 01 51 80` (28,600 seconds—2 days), an `RDLENGTH` of `00 04` octets and `RDATA` of `c0 a8 02 06`, which being an A RR represents an IP of 192.168.2.6, resulting in the A RR that follows:

```
ns1.example.com.  28600  IN  A   192.168.2.6
```

This is the A RR in the `ADDITIONAL SECTION` of the `dig` result shown in the section "DNS Message Overview" earlier.

EDNS0 Transactions

EDNS0 is normally used only in security transactions, but it can be forced by defining a `server` clause with the statement `edns yes;` in BIND's `named.conf` file (see Chapter 12). The server will advertise its ability to participate in EDNS transactions by sending an OPT pseudo RR in the `ADDITIONAL SECTION` field of a query but is displayed by `dig` under the title `OPT PSEUDOSECTION` (see the section "Verifying the Signed Zone" located in Chapter 11 for an example). If the receiving server cannot support such a service or does not recognize the OPT RR, it will respond with a failure in the `RCODE` field of the message header (`NOTIMP`, `FORMERR`, or `SERVFAIL`). In this case, the initiating server may continue without using EDNS services. Figure 15-7 shows the original `dig` command presented earlier (Figure 15-1) after the server was configured to use EDNS services through the use of the following `named.conf` fragment:

```
// named.conf fragment
options {
     ....
     forward only;
     forwarders {192.168.2.3;};
};
server 192.168.2.3 {
     edns yes;
};
....
```

The captured query packet is shown in Figure 15-7.

```
0000   01 51 01 00 00 01 00 00 00 00 00 01 03 77 77 77 .............www
0001   07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002   01 00 00 29 10 00 00 00 80 00 00 00             ............
```

Figure 15-7. *EDNS0 query*

The query differs from that shown in Figure 15-1 by having a count of 1 in the `ADDITIONAL SECTION` of the message header and contains an OPT pseudo RR whose format is defined in Table 15-8.

The corresponding response packet also contains an OPT pseudo RR as shown in Figure 15-8.

```
0000   05 5e 85 80 00 01 00 06 00 02 00 02 03 77 77 77 .............www
0001   07 65 78 61 6d 70 6c 85 03 63 6f 6d 00 00 01 00 .example.com....
0002   01 c0 0c 00 01 00 01 00 01 51 80 00 04 0a 01 02 ................
0003   01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 03 ................
0004   01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 04 ................
0005   01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 10 02 ................
0006   01 c0 0c 00 01 00 01 00 01 51 80 00 04 ac 11 02 ................
0007   01 c0 0c 00 01 00 01 00 01 51 80 00 04 c0 a8 02 ................
0008   01 c0 10 00 02 00 01 00 01 51 80 00 11 03 63 73 ..............ns
0009   32 07 65 78 61 6d 70 6c 65 03 6e 65 74 00 c0 10 2.example.net...
000a   00 02 00 01 00 01 51 80 00 06 03 6e 73 31 c0 10 ...........ns1..
000b   c0 aa 00 01 00 01 00 01 51 80 00 04 c0 a8 02 06 ................
000c   00 00 29 10 00 00 00 80 00 00 00                ...........
```

Figure 15-8. *EDNS0 response message*

The number of `ADDITIONAL SECTION` records is now 2 (it is 1 in Figure 15-2) and the OPT pseudo RR is present as highlighted. Rather than modifying the `named.conf` file, the same result may be obtained using the `dig` command with the option `+dnssec`, which turns on DNSSEC services that always use the EDNS0 service:

```
dig @192.168.235.2 www.example.com A +dnssec
```

OPT Pseudo RR Format

The OPT pseudo RR is created dynamically by the server and does not appear in a zone file. Its format uses the standard RR format defined in Table 15-5 earlier, but redefines the use of each field, as shown in Table 15-8.

Table 15-8. *OPT RR Format*

Field Name	Explanation
NAME	Always 00 (root).
TYPE	16 bits unsigned. The OPT RR type = 29 (41 decimal) in Figures 15-7 and 15-8.
CLASS	16 bits unsigned. The maximum size of a UDP message that can be accepted by this server. This is defined to be a minimum of 1220 octets and its default size is 4096 (Figures 15-7 and 15-8 both show 10 00 = 4096 decimal).
TTL	32 bits unsigned. This field is laid out as follows: *Field 1*: 8 bits unsigned. Extended RCODE (values defined in Table 15-2 earlier). *Field 2*: 8 bits unsigned. Version (must be 0). *Field3*: 16 bits unsigned. Flags as shown here: Bit 0 = DO (DNSSEC OK). Bits 1–15 = Unused.
RDLENGTH	The length in octets of the RDATA field being returned.
RDATA	The RDATA may be used to carry any number of extended optional data sets, each of which has the following format: *Field 1*: 16 bits unsigned. OPTION-CODE (none currently assigned). *Field 2*: 16 bits unsigned. OPTION-LENGTH—length in octets of option data. *Field 3*: Option data—length defined by OPTION-LENGTH and format defined by OPTION-CODE.

EDNS0 allows for extended label formats by setting the top 2 bits of the NAME field (see Table 15-6) to the value 01. The low 6 bits of this field contain an extended label code. The binary or bit label type (defined in RFC 2673 and discussed in Chapter 13) uses an extended label code value of 1 (binary 00 0001). This label type was changed to *experimental* status by RFC 3363 and is not discussed further. The value 63 (binary 11 1111) has been reserved for further extensions to the EDNS format.

DNS Binary RR Format

Each RR type has an RR-specific RDATA content. Table 15-9 shows the binary format (RDATA) of each RR type described in Chapter 13.

Table 15-9. `RDATA` *Field*

RR Name	Type Code	Specification	RDLENGTH	RDATA
A	1	RFC 1035	4	*Field 1*: Unsigned 32-bit integer. IPv4 address.
A6	38	RFC 2874	Var	May contain two or three fields defined by `RDLENGTH` and the value of the `Prefix Length`—if 0, the third (`Prefix Name`) field is not present; if `128`, the `Address Suffix` field is not present. *Field 1*: `Prefix Length`. Unsigned 8-bit integer with a value between 0 and 128 inclusive. Defines the number of bits not included in this record. If 0, the third field is not present and the second field defines the full IP address. *Field 2*: `Address Suffix`. Optional. There must be exactly enough octets in this field to contain a number of bits equal to 128 minus `Prefix Length`, with 0 to 7 leading pad bits to make this field an integral number of octets. Pad bits, if present, must be 0. Length in range 0 to 16 defined by `Prefix Length`. *Field 3*: `Prefix Name`. Optional. Present if `Prefix Length` field is nonzero. The domain name of the prefix (the A6 record that defines the next part of the address).
AAAA	28	RFC 3596	16	*Field 1*: 128 bits (16 octets). IPv6 address.
AFSDB	18	RFC 1183	Var	*Field 1*: 16-bit integer—subtype. *Field 2*: Variable-length host name in label format.*
APL	42	RFC 3123	Var	*Field 1*: 16-bit address family (`www.iana.org/assignments/address-family-numbers`). *Field 2*: 7-bits length of prefix (address family specific). *Field 3*: 1-bit negation. `1` = negated. *Field 4*: Address type determined by Field 1. If IPv4 (1), it is 32 bits unsigned; if IPv6 (2), it is 16 octets (128 bits). Fields 1 to 4 may be repeated any number of times to allow for multiple address ranges and is defined by `RDLENGTH`.

** See Table 15-6.*

RR Name	Type Code	Specification	RDLENGTH	RDATA
CERT	37	RFC 2538	Var	*Field 1*: 16 bits unsigned. Type of certificate. Values: 0 = Reserved 1 = PKIX (X.509 as per PKIX) 2 = SPKI cert 3 = PGP cert 4–252 = Available for IANA assignment 253 = Private URI 254 = Private OID (ASN.1) 255–65534 = Available for IANA assignment 65535 = Reserved *Field 2*: 16-bit unsigned key-tag. *Field 3*: 8-bit unsigned algorithm type. Values (RFC 4034): 0 = Reserved 1 = RSA-MD5 (recommended) 2 = Diffie-Hellman optional, key only 3 = DSA 4 = Reserved for elliptic curve cryptography 5–251 = Available for IANA assignment 252 = Reserved for indirect keys 253 = Private URI 254 = Private OID (ASN.1) 255 = Reserved *Field 4*: Base64-encoded key string (for format, see RFC 3548).
CNAME	5	RFC 1035	Var	*Field 1*: Variable-length host name in label format.*
DNAME	39	RFC 2672	Var	*Field 1*: Variable-length host name in label format.*
DNSKEY	48	RFC 3755	Var	*Field 1*: 16-bit unsigned flags (see Chapter 13). *Field 2*: 8-bit unsigned protocol (must be 3 per RFC 3445). *Field 3*: 8-bit unsigned algorithm (see the section "DNSKEY (DNSKEY) Record" located in Chapter 13). *Field 4*: Public key data—format depends on algorithm.†
DS	43	RFC 3658	Var	*Field 1*: 16-bit unsigned key-tag. *Field 2*: 8-bit unsigned algorithm (see the section "Delegation Signer (DS) Record" located in Chapter 13). *Field 3*: 8-bit unsigned digest. Values: 0 = Reserved 1 = SHA-1 2–255 = Unassigned *Field 4*: Digest data.

Continued

** See Table 15-6.*
† See the section "Security Algorithm Formats" later in this chapter

Table 15-9. *Continued*

RR Name	Type Code	Specification	RDLENGTH	RDATA
HINFO	13	RFC 1035	Var	*Field 1*: Variable-length hardware description (nominal) in label format.* *Field 2*: Variable-length OS description (nominal) in label format.*
IPSECKEY	45	RFC 4025	Var	*Field 1*: 8-bit unsigned preference. *Field 2*: 8-bit unsigned gateway type—defines the contents and format of Field 4. Values: 0 = No gateway is present. 1 = A 4-byte IPv4 address is present in Field 4. 2 = A 16-byte IPv6 address is present in Field 4. 3 = Domain name in Field 4 in label format.* *Field 3*: 8-bit unsigned algorithm type. Values: 0 = No key is present. 1 = A DSA key is present (defined in RFC 2536). 2 = A RSA key is present (defined in RFC 3110). *Field 4*: Variable gateway—content defined by Field 2. *Field 5*: Variable-length base64-encoded data defined by Field 3.
ISDN	20	RFC 1183	Var	*Field 1*: Variable-length ISDN E.164 number in label format.* *Field 2*: Variable-length ISDN subaddress in label format.*
KEY	25	RFC 2535	Var	*Field 1*: 16-bit unsigned flags (see the section "Public Key (KEY) Record" located in Chapter 13). *Field 2*: 8-bit unsigned protocol (must be 3 per RFC 3445). *Field 3*: 8-bit unsigned algorithm (see the section "Public Key (KEY) Record" located in Chapter 13). *Field 4*: Public key data—format depends on algorithm.†
KX	36	RFC 2230	Var	*Field 1*: 16-bit unsigned preference. *Field 2*: Variable-length of exchange host name in label format.*

** See Table 15-6.*
† See the section "Security Algorithm Formats" later in this chapter

RR Name	Type Code	Specification	RDLENGTH	RDATA
LOC	29	RFC 1876	28	*Field 1*: 8-bit unsigned version (must be 0). *Field 2*: 8-bit `size`. The diameter of a sphere enclosing the described entity, in centimeters, expressed as a pair of 4-bit unsigned integers, each ranging from 0 to 9; the most significant 4 bits represent the base and the second number represents the power of 10 by which to multiply the base. *Field 3*: 8-bit horizontal precision of the data, in centimeters, expressed using the same representation as `size` (earlier). This is the diameter of the horizontal "circle of error." *Field 4*: 8-bit vertical precision of the data, in centimeters, expressed using the same representation as `size` (earlier) This is the total potential vertical error. *Field 5*: Unsigned 32-bit integer. The latitude of the center of the sphere described by the `SIZE` field in thousandths of a second of arc. 2^31 represents the equator; numbers above that are northern latitude. *Field 6*: Unsigned 32-bit integer. The longitude of the center of the sphere described by the `SIZE` field, in thousandths of a second of arc, rounded away from the prime meridian. 2^31 represents the prime meridian; numbers above that are eastern longitude. *Field 7*: Unsigned 32-bit integer. The altitude of the center of the sphere described by the `SIZE` field, in centimeters, from a base of 100,000m below the (WGS 84) reference spheroid used by GPS (semimajor axis `a` = 6378137.0, reciprocal flattening `rf` = 298.257223563).
MB	7	RFC 1035	Var	*Field 1*: Variable-length mailbox name in label format.*
MG	8	RFC 1035	Var	*Field 1*: Variable-length group mailbox name in label format.*
MINFO	14	RFC 1035	Var	*Field 1*: Variable-length mailbox name responsible for mailbox or group in label format.* *Field 2*: Variable-length error mailbox name in label format.* If not used, contains a single zero length label.
MR	9	RFC 1035	Var	*Field 1*: Variable-length mailbox name in label format.*

Continued

** See Table 15-6.*

Table 15-9. *Continued*

RR Name	Type Code	Specification	RDLENGTH	RDATA
MX	15	RFC 1035	Var	*Field 1*: 16-bit unsigned preference. *Field 2*: Variable-length mail host name in label format.*
NAPTR	35	RFC 3403	Var	*Field 1*: 16-bit unsigned order. *Field 2*: 16-bit unsigned preference. *Field 3*: Variable-length alphanumeric flags in label format.* Flag values are defined by the application (see the section “Naming Authority Pointer (NAPTR) Record” located in Chapter 13). *Field 4*: Variable-length service name in label format.* Services values are defined by the application (see Chapter 13). *Field 5*: Variable-length regular expression in label format.* *Field 6*: Variable-length replacement host name in label format.*
NS	2	RFC 1035	Var	*Field 1*: Variable-length name server in label format.*
NSAP	22	RFC 1706	Var	*Field 1*: Variable-length binary-encoded NSAP.
NSEC	47	RFC 3755	Var	*Field 1*: Variable-length next host name (or name of SOA RR if this is the last in the zone) in label format.* *Field 2*: Bitmap of RR present at the host name of the NSEC RR (for format, see “NSEC Bitmap Format” later).
OPT	41	RFC 2671	Var	The OPT is a pseudo RR—it does not appear in a zone file—but is created by the server. Its format is defined in Table 15-8 earlier.
PTR	12	RFC 1035	Var	*Field 1*: Variable-length host name in label format.*
PX	26	RFC 2163	Var	*Field 1*: Unsigned 16-bit preference value. *Field 2*: Variable-length mailbox (RFC 822 format) name in label format.* *Field 3*: Variable-length X.400 name in label format.*
RP	17	RFC 1183	Var	*Field 1*: Variable-length mailbox name in label format.* *Field 2*: Variable-length name of Text RR containing additional information in label format.* If not used, contains a single zero length label.

** See Table 15-6.*

RR Name	Type Code	Specification	RDLENGTH	RDATA
RRSIG	46	RFC 3755	Var	*Field 1*: Unsigned 16-bit type covered (the RR type being signed). *Field 2*: Unsigned 8-bit algorithm (see the section "Resource Record Signature (RRSIG) Record" located in Chapter 13). *Field 3*: Unsigned 8-bit labels (number of labels in the host name excluding root and wildcard). *Field 4*: Unsigned 32-bit original TTL. *Field 5*: Unsigned 32-bit signature expiration. *Field 6*: Unsigned 32-bit signature inception (when valid). *Field 7*: Unsigned 16-bit key tag. *Field 8*: Variable-length of the name of the DNSKEY RR used to sign the RRSIG. In label format.* *Field 9*: Signature data—format depends on algorithm.†
RT	21	RFC 1183	Var	*Field 1*: Unsigned 16-bit preference value. *Field 2*: Variable-length host name in label format.*
SIG	25	RFC 2931/2535	Var	A meta (pseudo) RR used in SIG(0) transactions. *Field 1*: Unsigned 16-bit type covered (the RR type being signed). Must be `0`. *Field 2*: Unsigned 8-bit algorithm (see Chapter 13). *Field 3*: Unsigned 8-bit labels (number of labels in the host name excluding root and wildcard). *Field 4*: Unsigned 32-bit original TTL. *Field 5*: Unsigned 32-bit signature expiration. *Field 6*: Unsigned 32-bit signature inception (when valid). *Field 7*: Unsigned 16-bit `key-tag`. *Field 8*: Variable length of the name of the DNSKEY RR used to sign the RRSIG. In label format.* *Field 9*: Signature data—format depends on algorithm.†
SOA	6	RFC 1035/2308	Var	*Field 1*: Variable-length name of the primary name server (MNAME) in label format.* *Field 2*: Variable-length group mailbox name (RNAME) in label format.* *Field 3*: Unsigned 32-bit serial number. *Field 4*: Unsigned 32-bit refresh value. *Field 5*: Unsigned 32-bit retry value. *Field 6*: Unsigned 32-bit expiry value. *Field 7*: Unsigned 32-bit negative response (`NXDOMAIN`) TTL.

Continued

** See Table 15-6.*
† See the section "Security Algorithm Formats" later in this chapter

Table 15-9. *Continued*

RR Name	Type Code	Specification	RDLENGTH	RDATA
SRV	33	RFC 2782	Var	*Field 1*: Unsigned 16-bit priority. *Field 2*: Unsigned 16-bit weight. *Field 3*: Unsigned 16-bit port number. *Field 4*: Variable-length target host name in label format.*
SSHFP	44	Draft	Var	*Field 1*: Unsigned 8-bit algorithm (values as for RRSIG RR; see the section "Resource Record Signature (RRSIG) Record" located in Chapter 13). *Field 2*: Unsigned 8-bit fingerprint type. Values: 0 = Reserved 1 = SHA-1 *Field 3*: Public key data—format depends on algorithm.†
TXT	16	RFC 1035	Var	*Field 1*: Variable-length text in label format.*
WKS	11	RFC 1035	Var	Deprecated (use SRV): *Field 1*: Unsigned 32-bit IPv4 address. *Field 2*: Unsigned 8-bit protocol (`www.iana.org/assignments/protocol-numbers`). *Field 3*: Variable-length field that contains a single bit for all the ports used by the protocol, where bit 0 = port 0, etc. Must be an integral number of octets. Length is defined by `RDLENGTH`—length of Field 1 and Field 2 (5).
X25	19	RFC 1183	Var	*Field 1*: Variable-length X.25 PSDN address (X.121) in label format.*

** See Table 15-6.*
† See the section "Security Algorithm Formats" later in this chapter

Security Algorithm Formats

Each security algorithm used in the RRSIG, DS, DNSKEY, SIG, and KEY RRs has an identifying value as defined here:

0 = Reserved

1 = RSA-MD5 (RFC 2537)—not recommended by IETF

2 = Diffie-Hellman (RFC 2539)

3 = DSA-SHA-1 [Z] (RFC 2536)—optional

4 = Reserved for elliptic curve—not currently defined

5 = RSA-SHA-1 [Z] (RFC 3110)—mandatory

6–251 Available for assignment by IETF/IANA

252 = Indirect

253 = Private DNS [Z]—optional

254 = Private OID [Z]—optional

255 = Reserved

The public key data area is encoded uniquely for each type as defined by the RFC listed earlier: the algorithms marked [Z] may be used as Zone Signing Keys (see Chapter 11). The mandatory algorithm (type 5) is defined later for convenience since this is expected to be used most frequently.

The algorithm value 253 contains a host name in label format (see "NAME Field Format" earlier). This entry is assumed to describe the actual algorithm and encoding format being used. The algorithm 254 is an ASN.1 OID and starts with a single octet length followed by a BER-encoded (ITU X.690) ASN.1 OID that describes the algorithm and key encoding format.

Algorithm 5 (RSA-SHA-1)

The binary part of the RDATA field containing the key data is encoded as shown here:

- LENGTH: Length of the exponent—may be either 1 or 3 octets
- EXPONENT: Defined by length
- MODULUS: Remaining size of RDATA = RDLENGTH – (other fields in RDATA + LENGTH + EXPONENT)

The LENGTH field is one octet if the exponent length is in the range 1 to 255. If greater than 255, the first octet is 0 and octets 2 and 3 define the length. Both the EXPONENT and MODULUS fields have a maximum length of 4096 bits (1024 octets) and are unsigned numbers (binary string).

NSEC Bitmap Format

The NSEC RR uses a bitmap format in its binary RDATA field to minimize data volume, and its format is described in this section. The NSEC RR requires a list of all the RR types with the same host name. The RR TYPE field is an unsigned 16-bit integer allowing 65,535 possible values—the vast majority of which will be unused. In order to remove as much redundant information as possible, the RDATA Field 2 format is defined as shown here:

```
window | length | bitmap1 [ | bitmapn]
```

The total available TYPE field space (65,535) is divided into 256 (0 to 255) windows, each of 256-bit values (32 octets), each bit representing a record type and numbered from 0 to 255. Thus the first window (window 0) will describe RRs from 0 to 255, the second window (window 1) will describe RRs from 256 to 511, and so on. Only those windows that contain any RRs are required to be present. If there are no RRs in window 1, then it need not be present. Each window has up to 32 octets containing the bitmap for the 256 values. The number of octets may be truncated at the last nonzero octet of the bitmap and the length field used to indicate the number of octets available in the window description. The following is a simple example containing an A RR (type value = 1) an AAAA RR (type value = 28), an RRSIG RR (type value = 46), and an NSEC RR (type value = 47) to illustrate the mechanism:

```
bill     IN    NSEC  next.example.com (A AAAA RRSIG NSEC)
```

The hexadecimal representation of Field 2 only of this RR is shown here:

```
00 06 40 00 00 08 00 03
```

where the first 00 is the window (window 0 covering RR types 0 to 255). The length is 06, indicating only 6 octets (of the possible 32) are present. The first octet of the bitmap represents RR types from 0 to 7 and is 40, indicating type 1 (bit 1) is present (an A RR), and the fourth octet represents RR types 24 to 31 and is 08, indicating type 28 (bit 28) is present (an AAAA RR). Similarly, the RRSIG and NSEC RRs occupy the relevant bit positions in the sixth octet, which describes types 40 to 47. Since this is the last RR type in this record, all other values are omitted. The next example shows a more complex type using the user-defined RR syntax described in Chapter 14 to define a TYPE517 RR:

```
bill     IN    NSEC  next.example.com (A AAAA RRSIG NSEC TYPE517)
```

The hexadecimal representation of Field 2 only of this RR is shown here:

```
00 06 40 00 00 08 00 03
02 01 04
```

The first line is the same as the previous example and is not described further. The second line indicates window 2 (RRs from 512 to 767). Window 1 has no entries and has been omitted. The length value is 01, indicating only a single octet is present. The octet represents RR types 512 to 519 and has bit 517 set—indicating the TYPE517 RR.

Summary

This chapter described the protocol messages that pass between DNS servers. This is sometimes called the wire format. In most cases the message, or wire, format can be interpreted using a packet sniffer—there are times, however, when even the best tools either don't support the latest version or provide less-than-complete interpretation in which the user has to resort to tried and trusted manual methods. Each message has the same format comprising a message header followed by QUESTION, ANSWER, AUTHORITY, and ADDITIONAL SECTIONs. EDNS0 message formats add further complexity to the wire format but are only used with security transactions such as TSIG, SIG(0), TKEY, and DNSSEC.

Index