BIND 9 Administrator Reference Manual

BIND 9.11.36 (Extended Support Version)

Copyright (C) 2000-2021 Internet Systems Consortium, Inc. ("ISC")

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at http://mozilla.org/MPL/2.0/.

Internet Systems Consortium, Inc.
PO Box 360
Newmarket, NH 03857
USA
https://www.isc.org/

Contents

1 Introduction

The Internet Domain Name System (DNS) consists of the syntax to specify the names of entities in the Internet in a hierarchical manner, the rules used for delegating authority over names, and the system implementation that actually maps names to Internet addresses. DNS data is maintained in a group of distributed hierarchical databases.

1.1 Scope of Document

The Berkeley Internet Name Domain (BIND) implements a domain name server for a number of operating systems. This document provides basic information about the installation and care of the Internet Systems Consortium (ISC) BIND version 9 software package for system administrators.

This version of the manual corresponds to BIND version 9.11.

1.2 Organization of This Document

In this document, *Chapter 1* introduces the basic DNS and BIND concepts. *Chapter 2* describes resource requirements for running BIND in various environments. Information in *Chapter 3* is *task-oriented* in its presentation and is organized functionally, to aid in the process of installing the BIND 9 software. The task-oriented section is followed by *Chapter 4*, which contains more advanced concepts that the system administrator may need for implementing certain options. *Chapter 5* describes the BIND 9 lightweight resolver. The contents of *Chapter 6* are organized as in a reference manual to aid in the ongoing maintenance of the software. *Chapter 7* addresses security considerations, and *Chapter 8* contains troubleshooting help. The main body of the document is followed by several *appendices* which contain useful reference information, such as a *bibliography* and historic information related to BIND and the Domain Name System.

1.3 Conventions Used in This Document

In this document, we use the following general typographic conventions:

To describe:	*We use the style:*
a pathname, filename, URL, hostname, mailing list name, or new term or concept	`Fixed width`
literal user input	**`Fixed Width Bold`**
program output	`Fixed Width`

The following conventions are used in descriptions of the BIND configuration file:

To describe:	*We use the style:*
keywords	`Fixed Width`
variables	`Fixed Width`
Optional input	[Text is enclosed in square brackets]

1.4 THE DOMAIN NAME SYSTEM (DNS)

This document explains the installation and upkeep of the BIND (Berkeley Internet Name Domain) software package. We begin by reviewing the fundamentals of the Domain Name System (DNS) as they relate to BIND.

DNS Fundamentals

The Domain Name System (DNS) is a hierarchical, distributed database. It stores information for mapping Internet host names to IP addresses and vice versa, mail routing information, and other data used by Internet applications.

Clients look up information in the DNS by calling a *resolver* library, which sends queries to one or more *name servers* and interprets the responses. The BIND 9 software distribution contains a name server, **named**, and a resolver library, **liblwres**.

Domains and Domain Names

The data stored in the DNS is identified by *domain names* that are organized as a tree according to organizational or administrative boundaries. Each node of the tree, called a *domain*, is given a label. The domain name of the node is the concatenation of all the labels on the path from the node to the *root* node. This is represented in written form as a string of labels listed from right to left and separated by dots. A label need only be unique within its parent domain.

For example, a domain name for a host at the company *Example, Inc.* could be `ourhost.example.com`, where `com` is the top level domain to which `ourhost.example.com` belongs, `example` is a subdomain of `com`, and `ourhost` is the name of the host.

For administrative purposes, the name space is partitioned into areas called *zones*, each starting at a node and extending down to the "leaf" nodes or to nodes where other zones start. The data for each zone is stored in a *name server*, which answers queries about the zone using the *DNS protocol*.

The data associated with each domain name is stored in the form of *resource records* (RRs). Some of the supported resource record types are described in Section 6.3.

For more detailed information about the design of the DNS and the DNS protocol, please refer to the standards documents listed in Section C.2.

Zones

To properly operate a name server, it is important to understand the difference between a *zone* and a *domain*.

As stated previously, a zone is a point of delegation in the DNS tree. A zone consists of those contiguous parts of the domain tree for which a name server has complete information and over which it has authority. It contains all domain names from a certain point downward in the domain tree except those which are delegated to other zones. A delegation point is marked by one or more *NS records* in the parent zone, which should be matched by equivalent NS records at the root of the delegated zone.

For instance, consider the `example.com` domain which includes names such as `host.aaa.example.com` and `host.bbb.example.com` even though the `example.com` zone includes only delegations for the `aaa.example.com` and `bbb.example.com` zones. A zone can map exactly to a single domain, but could also include only part of a domain, the rest of which could be delegated to other name servers. Every name in the DNS tree is a *domain*, even if it is *terminal*, that is, has no *subdomains*. Every subdomain is a domain and every domain except the root is also a subdomain. The terminology is not intuitive and we suggest reading RFCs 1033, 1034, and 1035 to gain a complete understanding of this difficult and subtle topic.

Though BIND is called a "domain name server", it deals primarily in terms of zones. The "primary" and "secondary" declarations in the `named.conf` file specify zones, not domains. When BIND asks some other site if it is willing to be a secondary server for a *domain*, it is actually asking for secondary service for some collection of *zones*.

Authoritative Name Servers

Each zone is served by at least one *authoritative name server*, which contains the complete data for the zone. To make the DNS tolerant of server and network failures, most zones have two or more authoritative servers, on different networks.

Responses from authoritative servers have the "authoritative answer" (AA) bit set in the response packets. This makes them easy to identify when debugging DNS configurations using tools like **dig** (Section 3.3).

The Primary Server

The authoritative server where the main copy of the zone data is maintained is called the *primary* (or **master**) server, or simply the *primary*. Typically it loads the zone contents from some local

file edited by humans or perhaps generated mechanically from some other local file which is edited by humans. This file is called the *zone file* or *master file*.

In some cases, however, the zone file may not be edited by humans at all, but may instead be the result of *dynamic update* operations.

Secondary Servers

The other authoritative servers, called the *secondary* (or **slave**) servers, load the zone contents from another server using a replication process known as a *zone transfer*. Typically the data is transferred directly from the primary master, but it is also possible to transfer it from another secondary. In other words, a secondary server may itself act as a primary to a subordinate secondary server.

Periodically, the secondary server must send a refresh query to determine whether the zone contents have been updated. This is done by sending a query for the zone's Start of Authority (SOA) record and checking whether the SERIAL field has been updated; if so, a new transfer request is initiated. The timing of these refresh queries is controlled by the SOA REFRESH and RETRY fields, but can be overridden with the **max-refresh-time**, **min-refresh-time**, **max-retry-time**, and **min-retry-time** options.

If the zone data cannot be updated within the time specified by the SOA EXPIRE option (up to a hard-coded maximum of 24 weeks), the secondary zone expires and no longer responds to queries.

Stealth Servers

Usually, all of the zone's authoritative servers are listed in NS records in the parent zone. These NS records constitute a *delegation* of the zone from the parent. The authoritative servers are also listed in the zone file itself, at the *top level* or *apex* of the zone. Servers that are not in the parent's NS delegation can be listed in the zone's top-level NS records, but servers that are not present at the zone's top level cannot be listed in the parent's delegation.

A *stealth server* is a server that is authoritative for a zone but is not listed in that zone's NS records. Stealth servers can be used for keeping a local copy of a zone, to speed up access to the zone's records, or to make sure that the zone is available even if all the "official" servers for the zone are inaccessible.

A configuration where the primary server itself is a stealth server is often referred to as a "hidden primary" configuration. One use for this configuration is when the primary is behind a firewall and is therefore unable to communicate directly with the outside world.

Caching Name Servers

The resolver libraries provided by most operating systems are *stub resolvers*, meaning that they are not capable of performing the full DNS resolution process by themselves by talking directly to the authoritative servers. Instead, they rely on a local name server to perform the resolution on their behalf. Such a server is called a *recursive* name server; it performs *recursive lookups* for local clients.

To improve performance, recursive servers cache the results of the lookups they perform. Since the processes of recursion and caching are intimately connected, the terms *recursive server* and *caching server* are often used synonymously.

The length of time for which a record may be retained in the cache of a caching name server is controlled by the Time-To-Live (TTL) field associated with each resource record.

Forwarding

Even a caching name server does not necessarily perform the complete recursive lookup itself. Instead, it can *forward* some or all of the queries that it cannot satisfy from its cache to another caching name server, commonly referred to as a *forwarder*.

Forwarders are typically used when an administrator does not wish for all the servers at a given site to interact directly with the rest of the Internet. For example, a common scenario is when multiple internal DNS servers are behind an Internet firewall. Servers behind the firewall forward their requests to the server with external access, which queries Internet DNS servers on the internal servers' behalf.

Another scenario (largely now superseded by Response Policy Zones) is to send queries first to a custom server for RBL processing before forwarding them to the wider Internet.

There may be one or more forwarders in a given setup. The order in which the forwarders are listed in `named.conf` does not determine the sequence in which they are queried; rather, **named** uses the response times from previous queries to select the server that is likely to respond the most quickly. A server that has not yet been queried is given an initial small random response time to ensure that it is tried at least once. Dynamic adjustment of the recorded response times ensures that all forwarders are queried, even those with slower response times. This permits changes in behavior based on server responsiveness.

Name Servers in Multiple Roles

The BIND name server can simultaneously act as a primary for some zones, a secondary for other zones, and a caching (recursive) server for a set of local clients.

However, since the functions of authoritative name service and caching/recursive name service are logically separate, it is often advantageous to run them on separate server machines. A server that only provides authoritative name service (an *authoritative-only* server) can run with recursion disabled, improving reliability and security. A server that is not authoritative for any zones and only provides recursive service to local clients (a *caching-only* server) does not need to be reachable from the Internet at large and can be placed inside a firewall.

2 BIND Resource Requirements

2.1 Hardware requirements

DNS hardware requirements have traditionally been quite modest. For many installations, servers that have been retired from active duty have performed admirably as DNS servers.

However, the DNSSEC features of BIND 9 may be quite CPU-intensive, so organizations that make heavy use of these features may wish to consider larger systems for these applications. BIND 9 is fully multithreaded, allowing full utilization of multiprocessor systems for installations that need it.

2.2 CPU Requirements

CPU requirements for BIND 9 range from i386-class machines, for serving static zones without caching, to enterprise-class machines to process many dynamic updates and DNSSEC-signed zones, serving many thousands of queries per second.

2.3 Memory Requirements

Server memory must be sufficient to hold both the cache and the zones loaded from disk. The **max-cache-size** option can limit the amount of memory used by the cache, at the expense of reducing cache hit rates and causing more DNS traffic. If additional section caching (Section 6.2) is enabled, the **max-acache-size** option can be used to limit the amount of memory used by the mechanism. It is still good practice to have enough memory to load all zone and cache data into memory; unfortunately, the best way to determine this for a given installation is to watch the name server in operation. After a few weeks, the server process should reach a relatively stable size where entries are expiring from the cache as fast as they are being inserted.

2.4 Name Server-Intensive Environment Issues

For name server-intensive environments, there are two configurations that may be used. The first is one where clients and any second-level internal name servers query a main name server,

which has enough memory to build a large cache; this approach minimizes the bandwidth used by external name lookups. The second alternative is to set up second-level internal name servers to make queries independently. In this configuration, none of the individual machines need to have as much memory or CPU power as in the first alternative, but this has the disadvantage of making many more external queries, as none of the name servers share their cached data.

2.5 SUPPORTED OPERATING SYSTEMS

ISC BIND 9 compiles and runs on many Unix-like operating systems and on Microsoft Windows Server 2012 R2, 2016 and Windows 10. For an up-to-date list of supported systems, see the PLATFORMS.md file in the top-level directory of the BIND 9 source distribution.

3 Name Server Configuration

In this chapter we provide some suggested configurations, along with guidelines for their use. We suggest reasonable values for certain option settings.

3.1 SAMPLE CONFIGURATIONS

A Caching-only Name Server

The following sample configuration is appropriate for a caching-only name server for use by clients internal to a corporation. All queries from outside clients are refused using the **allow-query** option. The same effect can be achieved using suitable firewall rules.

```
// Two corporate subnets we wish to allow queries from.
acl corpnets { 192.168.4.0/24; 192.168.7.0/24; };
options {
     // Working directory
     directory "/etc/namedb";

     allow-query { corpnets; };
};
// Provide a reverse mapping for the loopback
// address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
     type master;
     file "localhost.rev";
     notify no;
};
```

An Authoritative-only Name Server

This sample configuration is for an authoritative-only server that is the primary server for "`example.com`" and a secondary server for the subdomain "`eng.example.com`".

```
options {
     // Working directory
     directory "/etc/namedb";
     // Do not allow access to cache
```

```
     allow-query-cache { none; };
     // This is the default
     allow-query { any; };
     // Do not provide recursive service
     recursion no;
};

// Provide a reverse mapping for the loopback
// address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
     type master;
     file "localhost.rev";
     notify no;
};
// We are the primary server for example.com
zone "example.com" {
     type master;
     file "example.com.db";
     // IP addresses of secondary servers allowed to
     // transfer example.com
     allow-transfer {
          192.168.4.14;
          192.168.5.53;
     };
};
// We are a secondary server for eng.example.com
zone "eng.example.com" {
     type slave;
     file "eng.example.com.bk";
     // IP address of eng.example.com primary server
     masters { 192.168.4.12; };
};
```

3.2 LOAD BALANCING

A primitive form of load balancing can be achieved in the DNS by using multiple records (such as multiple A records) for one name.

For example, assuming three HTTP servers with network addresses of 10.0.0.1, 10.0.0.2, and 10.0.0.3, a set of records such as the following means that clients will connect to each machine one-third of the time:

Name	TTL	CLASS	TYPE	Resource Record (RR) Data
www	600	IN	A	10.0.0.1
	600	IN	A	10.0.0.2
	600	IN	A	10.0.0.3

When a resolver queries for these records, BIND rotates them and responds to the query with the records in a different order. In the example above, clients randomly receive records in the order 1, 2, 3; 2, 3, 1; and 3, 1, 2. Most clients use the first record returned and discard the rest.

For more detail on ordering responses, check the **rrset-order** sub-statement in the **options** statement, see RRset Ordering.

3.3 NAME SERVER OPERATIONS

Tools for Use With the Name Server Daemon

This section describes several indispensable diagnostic, administrative, and monitoring tools available to the system administrator for controlling and debugging the name server daemon.

Diagnostic Tools

The **dig**, **host**, and **nslookup** programs are all command-line tools for manually querying name servers. They differ in style and output format.

dig

dig is the most versatile and complete of these lookup tools. It has two modes: simple interactive mode for a single query, and batch mode, which executes a query for each in a list of several query lines. All query options are accessible from the command line.

`dig` [`@server`] *domain* [*query-type*] [*query-class*] [`+query-option`] [`-dig-option`] [`%comment`]

The usual simple use of **dig** takes the form

dig @server domain query-type query-class

For more information and a list of available commands and options, see the **dig** man page.

host

The **host** utility emphasizes simplicity and ease of use. By default, it converts between host names and Internet addresses, but its functionality can be extended with the use of options.

`host` [-aCdlnrsTwv] [-c *class*] [-N *ndots*] [-t *type*] [-W *timeout*] [-R *retries*] [-m *flag*] [-4] [-6] *hostname* [*server*]

For more information and a list of available commands and options, see the **host** man page.

nslookup

nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains, or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

`nslookup` [-option...] [*host-to-find* | - [server]]

Interactive mode is entered when no arguments are given (the default name server is used) or when the first argument is a hyphen ("-") and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

Due to its arcane user interface and frequently inconsistent behavior, we do not recommend the use of **nslookup**. Use **dig** instead.

Administrative Tools

Administrative tools play an integral part in the management of a server.

named-checkconf
The **named-checkconf** program checks the syntax of a `named.conf` file.

`named-checkconf` [-jvz] [-t *directory*] [*filename*]

named-checkzone
The **named-checkzone** program checks a zone file for syntax and consistency.

`named-checkzone` [-djqvD] [-c *class*] [-o *output*] [-t *directory*] [-w *directory*] [-k *(ignore|warn|fail)*] [-n *(ignore|warn|fail)*] [-W *(ignore|warn)*] *zone* [*filename*]

named-compilezone
This tool is similar to **named-checkzone,** but it always dumps the zone content to a specified file (typically in a different format).

rndc
The remote name daemon control (**rndc**) program allows the system administrator to control the operation of a name server. If **rndc** is run without any options, it displays a usage message as follows:

`rndc` [-c *config*] [-s *server*] [-p *port*] [-y *key*] *command* [*command*...]

See rndc(8) for details of the available **rndc** commands.

rndc requires a configuration file, since all communication with the server is authenticated with digital signatures that rely on a shared secret, and there is no way to provide that secret other than with a configuration file. The default location for the **rndc** configuration file is `/etc/rndc.conf`, but an alternate location can be specified with the `-c` option. If the configuration file is not found, **rndc** also looks in `/etc/rndc.key` (or whatever `sysconfdir` was defined when the BIND build was configured). The `rndc.key` file is generated by running **rndc-confgen -a** as described in Section 6.2.

The format of the configuration file is similar to that of `named.conf`, but is limited to only four statements: the **options**, **key**, **server**, and **include** statements. These statements are what associate the secret keys to the servers with which they are meant to be shared. The order of statements is not significant.

The **options** statement has three clauses: **default-server**, **default-key**, and **default-port**. **default-server** takes a host name or address argument and represents the server that is contacted if no `-s` option is provided on the command line. **default-key** takes the name of a key as its argument, as defined by a **key** statement. **default-port** specifies the port to which **rndc** should connect if no port is given on the command line or in a **server** statement.

The **key** statement defines a key to be used by **rndc** when authenticating with **named**. Its syntax is identical to the **key** statement in `named.conf`. The keyword **key** is followed by a key name, which must be a valid domain name, though it need not actually be hierarchical; thus, a string like "**rndc_key**" is a valid name. The **key** statement has two clauses: **algorithm** and **secret**. While the configuration parser accepts any string as the argument to **algorithm**, currently only the strings "**hmac-md5**", "**hmac-sha1**", "**hmac-sha224**", "**hmac-sha256**", "**hmac-sha384**", and "**hmac-sha512**" have any meaning. The secret is a Base64-encoded string as specified in RFC 3548.

The **server** statement associates a key defined using the **key** statement with a server. The keyword **server** is followed by a host name or address. The **server** statement has two clauses: **key** and **port**. The **key** clause specifies the name of the key to be used when communicating with this server, and the **port** clause can be used to specify the port **rndc** should connect to on the server.

A sample minimal configuration file is as follows:

```
key rndc_key {
     algorithm "hmac-sha256";
     secret
       "c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K ↩
           ";
};
options {
     default-server 127.0.0.1;
     default-key    rndc_key;
};
```

This file, if installed as `/etc/rndc.conf`, allows the command:

```
$ rndc reload
```

to connect to 127.0.0.1 port 953 and causes the name server to reload, if a name server on the local machine is running with following controls statements:

```
controls {
        inet 127.0.0.1
            allow { localhost; } keys { rndc_key; };
};
```

and it has an identical key statement for `rndc_key`.

Running the **rndc-confgen** program conveniently creates an `rndc.conf` file, and also displays the corresponding **controls** statement needed to add to `named.conf`. Alternatively, it is possible to run **rndc-confgen -a** to set up an `rndc.key` file and not modify `named.conf` at all.

Signals

Certain Unix signals cause the name server to take specific actions, as described in the following table. These signals can be sent using the **kill** command.

SIGHUP	Causes the server to read `named.conf` and reload the database.
SIGTERM	Causes the server to clean up and exit.
SIGINT	Causes the server to clean up and exit.

4 Advanced DNS Features

4.1 NOTIFY

DNS NOTIFY is a mechanism that allows primary servers to notify their secondary servers of changes to a zone's data. In response to a **NOTIFY** from a primary server, the secondary checks to see that its version of the zone is the current version and, if not, initiates a zone transfer.

For more information about DNS **NOTIFY**, see the description of the **notify** option in Section 6.2 and the description of the zone option **also-notify** in Section 6.2. The **NOTIFY** protocol is specified in RFC 1996.

> NOTE
>
> As a secondary zone can also be a primary to other secondaries, **named**, by default, sends **NOTIFY** messages for every zone it loads. Specifying **notify primary-only;** causes **named** to only send **NOTIFY** for primary zones that it loads.

4.2 DYNAMIC UPDATE

Dynamic Update is a method for adding, replacing, or deleting records in a primary server by sending it a special form of DNS messages. The format and meaning of these messages is specified in RFC 2136.

Dynamic update is enabled by including an **allow-update** or an **update-policy** clause in the **zone** statement.

If the zone's **update-policy** is set to **`local`**, updates to the zone are permitted for the key `local-ddns`, which is generated by **named** at startup. See Section 6.2 for more details.

Dynamic updates using Kerberos-signed requests can be made using the TKEY/GSS protocol, either by setting the **tkey-gssapi-keytab** option, or by setting both the **tkey-gssapi-credential**

and **tkey-domain** options. Once enabled, Kerberos-signed requests are matched against the update policies for the zone, using the Kerberos principal as the signer for the request.

Updating of secure zones (zones using DNSSEC) follows RFC 3007: RRSIG, NSEC, and NSEC3 records affected by updates are automatically regenerated by the server using an online zone key. Update authorization is based on transaction signatures and an explicit server policy.

The Journal File

All changes made to a zone using dynamic update are stored in the zone's journal file. This file is automatically created by the server when the first dynamic update takes place. The name of the journal file is formed by appending the extension `.jnl` to the name of the corresponding zone file, unless specifically overridden. The journal file is in a binary format and should not be edited manually.

The server also occasionally writes ("dumps") the complete contents of the updated zone to its zone file. This is not done immediately after each dynamic update, because that would be too slow when a large zone is updated frequently. Instead, the dump is delayed by up to 15 minutes, allowing additional updates to take place. During the dump process, transient files are created with the extensions `.jnw` and `.jbk`; under ordinary circumstances, these are removed when the dump is complete, and can be safely ignored.

When a server is restarted after a shutdown or crash, it replays the journal file to incorporate into the zone any updates that took place after the last zone dump.

Changes that result from incoming incremental zone transfers are also journaled in a similar way.

The zone files of dynamic zones cannot normally be edited by hand because they are not guaranteed to contain the most recent dynamic changes; those are only in the journal file. The only way to ensure that the zone file of a dynamic zone is up-to-date is to run **rndc stop**.

To make changes to a dynamic zone manually, follow these steps: first, disable dynamic updates to the zone using **rndc freeze *zone***. This updates the zone file with the changes stored in its `.jnl` file. Then, edit the zone file. Finally, run **rndc thaw *zone*** to reload the changed zone and re-enable dynamic updates.

rndc sync *zone* updates the zone file with changes from the journal file without stopping dynamic updates; this may be useful for viewing the current zone state. To remove the `.jnl` file after updating the zone file, use **rndc sync -clean**.

4.3 INCREMENTAL ZONE TRANSFERS (IXFR)

The incremental zone transfer (IXFR) protocol is a way for secondary servers to transfer only changed data, instead of having to transfer an entire zone. The IXFR protocol is specified in RFC 1995. See Proposed Standards.

When acting as a primary server, BIND 9 supports IXFR for those zones where the necessary change history information is available. These include primary zones maintained by dynamic update and secondary zones whose data was obtained by IXFR. For manually maintained primary zones, and for secondary zones obtained by performing a full zone transfer (AXFR), IXFR is supported only if the option **ixfr-from-differences** is set to **yes**.

When acting as a secondary server, BIND 9 attempts to use IXFR unless it is explicitly disabled. For more information about disabling IXFR, see the description of the **request-ixfr** clause of the **server** statement.

4.4 SPLIT DNS

Setting up different views of the DNS space to internal and external resolvers is usually referred to as a *split DNS* setup. There are several reasons an organization might want to set up its DNS this way.

One common reason to use split DNS is to hide "internal" DNS information from "external" clients on the Internet. There is some debate as to whether this is actually useful. Internal DNS information leaks out in many ways (via email headers, for example) and most savvy "attackers" can find the information they need using other means. However, since listing addresses of internal servers that external clients cannot possibly reach can result in connection delays and other annoyances, an organization may choose to use split DNS to present a consistent view of itself to the outside world.

Another common reason for setting up a split DNS system is to allow internal networks that are behind filters or in RFC 1918 space (reserved IP space, as documented in RFC 1918) to resolve DNS on the Internet. Split DNS can also be used to allow mail from outside back into the internal network.

Example Split DNS Setup

Let's say a company named *Example, Inc.* (`example.com`) has several corporate sites that have an internal network with reserved Internet Protocol (IP) space and an external demilitarized zone (DMZ), or "outside" section of a network, that is available to the public.

Example, Inc. wants its internal clients to be able to resolve external hostnames and to exchange mail with people on the outside. The company also wants its internal resolvers to have access to certain internal-only zones that are not available at all outside of the internal network.

In order to accomplish this, the company sets up two sets of name servers. One set is on the inside network (in the reserved IP space) and the other set is on bastion hosts, which are "proxy" hosts in the DMZ that can talk to both sides of its network.

The internal servers are configured to forward all queries, except queries for `site1.internal`, `site2.internal`, `site1.example.com`, and `site2.example.com`, to the servers in the DMZ. These internal servers will have complete sets of information for `site1.example.com`, `site2.example.com`, `site1.internal`, and `site2.internal`.

To protect the `site1.internal` and `site2.internal` domains, the internal name servers must be configured to disallow all queries to these domains from any external hosts, including the bastion hosts.

The external servers, which are on the bastion hosts, are configured to serve the "public" version of the `site1.example.com` and `site2.example.com` zones. This could include things such as the host records for public servers (`www.example.com` and `ftp.example.com`) and mail exchange (MX) records (`a.mx.example.com` and `b.mx.example.com`).

In addition, the public `site1.example.com` and `site2.example.com` zones should have special MX records that contain wildcard ("*") records pointing to the bastion hosts. This is needed because external mail servers do not have any other way of looking up how to deliver mail to those internal hosts. With the wildcard records, the mail is delivered to the bastion host, which can then forward it on to internal hosts.

Here's an example of a wildcard MX record:

```
*   IN MX 10 external1.example.com.
```

Now that they accept mail on behalf of anything in the internal network, the bastion hosts need to know how to deliver mail to internal hosts. The resolvers on the bastion hosts need to be configured to point to the internal name servers for DNS resolution.

Queries for internal hostnames are answered by the internal servers, and queries for external hostnames are forwarded back out to the DNS servers on the bastion hosts.

For all of this to work properly, internal clients need to be configured to query *only* the internal name servers for DNS queries. This could also be enforced via selective filtering on the network.

If everything has been set properly, *Example, Inc.*'s internal clients are now able to:

- Look up any hostnames in the `site1.example.com` and `site2.example.com` zones.
- Look up any hostnames in the `site1.internal` and `site2.internal` domains.
- Look up any hostnames on the Internet.
- Exchange mail with both internal and external users.

Hosts on the Internet are able to:

- Look up any hostnames in the `site1.example.com` and `site2.example.com` zones.
- Exchange mail with anyone in the `site1.example.com` and `site2.example.com` zones.

Here is an example configuration for the setup just described above. Note that this is only configuration information; for information on how to configure the zone files, see Section 3.1.

Internal DNS server config:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };

options {
    ...
    ...
    forward only;
    // forward to external servers
    forwarders {
        bastion-ips-go-here;
```

```
    };
    // sample allow-transfer (no one)
    allow-transfer { none; };
    // restrict query access
    allow-query { internals; externals; };
    // restrict recursion
    allow-recursion { internals; };
    ...
    ...
};

// sample primary zone
zone "site1.example.com" {
  type master;
  file "m/site1.example.com";
  // do normal iterative resolution (do not forward)
  forwarders { };
  allow-query { internals; externals; };
  allow-transfer { internals; };
};

// sample secondary zone
zone "site2.example.com" {
  type slave;
  file "s/site2.example.com";
  masters { 172.16.72.3; };
  forwarders { };
  allow-query { internals; externals; };
  allow-transfer { internals; };
};

zone "site1.internal" {
  type master;
  file "m/site1.internal";
  forwarders { };
  allow-query { internals; };
  allow-transfer { internals; }
};

zone "site2.internal" {
  type slave;
  file "s/site2.internal";
  masters { 172.16.72.3; };
  forwarders { };
  allow-query { internals };
  allow-transfer { internals; }
};
```

External (bastion host) DNS server config:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };
```

```
options {
  ...
  ...
  // sample allow-transfer (no one)
  allow-transfer { none; };
  // default query access
  allow-query { any; };
  // restrict cache access
  allow-query-cache { internals; externals; };
  // restrict recursion
  allow-recursion { internals; externals; };
  ...
  ...
};

// sample secondary zone
zone "site1.example.com" {
  type master;
  file "m/site1.foo.com";
  allow-transfer { internals; externals; };
};

zone "site2.example.com" {
  type slave;
  file "s/site2.foo.com";
  masters { another_bastion_host_maybe; };
  allow-transfer { internals; externals; }
};
```

In the `resolv.conf` (or equivalent) on the bastion host(s):

```
search ...
nameserver 172.16.72.2
nameserver 172.16.72.3
nameserver 172.16.72.4
```

4.5 TSIG

TSIG (Transaction SIGnatures) is a mechanism for authenticating DNS messages, originally specified in RFC 2845. It allows DNS messages to be cryptographically signed using a shared secret. TSIG can be used in any DNS transaction, as a way to restrict access to certain server functions (e.g., recursive queries) to authorized clients when IP-based access control is insufficient or needs to be overridden, or as a way to ensure message authenticity when it is critical to the integrity of the server, such as with dynamic UPDATE messages or zone transfers from a primary to a secondary server.

This section is a guide to setting up TSIG in BIND. It describes the configuration syntax and the process of creating TSIG keys.

named supports TSIG for server-to-server communication, and some of the tools included with BIND support it for sending messages to **named**:

- nsupdate(1) supports TSIG via the `-k`, `-l`, and `-y` command-line options, or via the **key** command when running interactively.

- dig(1) supports TSIG via the `-k` and `-y` command-line options.

Generating a Shared Key

TSIG keys can be generated using the **tsig-keygen** command; the output of the command is a **key** directive suitable for inclusion in `named.conf`. The key name, algorithm, and size can be specified by command-line parameters; the defaults are "tsig-key", HMAC-SHA256, and 256 bits, respectively.

Any string which is a valid DNS name can be used as a key name. For example, a key to be shared between servers called *host1* and *host2* could be called "host1-host2.", and this key can be generated using:

```
$ tsig-keygen host1-host2. > host1-host2.key
```

This key may then be copied to both hosts. The key name and secret must be identical on both hosts. (Note: copying a shared secret from one server to another is beyond the scope of the DNS. A secure transport mechanism should be used: secure FTP, SSL, ssh, telephone, encrypted email, etc.)

tsig-keygen can also be run as **ddns-confgen**, in which case its output includes additional configuration text for setting up dynamic DNS in **named**. See ddns-confgen(8) for details.

Loading a New Key

For a key shared between servers called *host1* and *host2*, the following could be added to each server's `named.conf` file:

```
key "host1-host2." {
        algorithm hmac-sha256;
        secret "DAopyf1mhCbFVZw7pgmNPBoLUq8wEUT7UuPoLENP2HY=";
};
```

(This is the same key generated above using **tsig-keygen**.)

Since this text contains a secret, it is recommended that either `named.conf` not be world-readable, or that the **key** directive be stored in a file which is not world-readable and which is included in `named.conf` via the **include** directive.

Once a key has been added to `named.conf` and the server has been restarted or reconfigured, the server can recognize the key. If the server receives a message signed by the key, it is able to verify the signature. If the signature is valid, the response is signed using the same key.

TSIG keys that are known to a server can be listed using the command **rndc tsig-list**.

Instructing the Server to Use a Key

A server sending a request to another server must be told whether to use a key, and if so, which key to use.

For example, a key may be specified for each server in the **masters** statement in the definition of a secondary zone; in this case, all SOA QUERY messages, NOTIFY messages, and zone transfer requests (AXFR or IXFR) are signed using the specified key. Keys may also be specified in the **also-notify** statement of a primary or secondary zone, causing NOTIFY messages to be signed using the specified key.

Keys can also be specified in a **server** directive. Adding the following on *host1*, if the IP address of *host2* is 10.1.2.3, would cause *all* requests from *host1* to *host2*, including normal DNS queries, to be signed using the **host1-host2.** key:

```
server 10.1.2.3 {
        keys { host1-host2. ;};
};
```

Multiple keys may be present in the **keys** statement, but only the first one is used. As this directive does not contain secrets, it can be used in a world-readable file.

Requests sent by *host2* to *host1* would *not* be signed, unless a similar **server** directive were in *host2*'s configuration file.

Whenever any server sends a TSIG-signed DNS request, it expects the response to be signed with the same key. If a response is not signed, or if the signature is not valid, the response is rejected.

TSIG-Based Access Control

TSIG keys may be specified in ACL definitions and ACL directives such as **allow-query**, **allow-transfer**, and **allow-update**. The above key would be denoted in an ACL element as **key host1-host2.**

Here is an example of an **allow-update** directive using a TSIG key:

```
allow-update { !{ !localnets; any; }; key host1-host2. ;};
```

This allows dynamic updates to succeed only if the UPDATE request comes from an address in **localnets**, *and* if it is signed using the **host1-host2.** key.

See Section 6.2 for a discussion of the more flexible **update-policy** statement.

Errors

Processing of TSIG-signed messages can result in several errors:

- If a TSIG-aware server receives a message signed by an unknown key, the response will be unsigned, with the TSIG extended error code set to BADKEY.

- If a TSIG-aware server receives a message from a known key but with an invalid signature, the response will be unsigned, with the TSIG extended error code set to BADSIG.
- If a TSIG-aware server receives a message with a time outside of the allowed range, the response will be signed but the TSIG extended error code set to BADTIME, and the time values will be adjusted so that the response can be successfully verified.

In all of the above cases, the server returns a response code of NOTAUTH (not authenticated).

4.6 TKEY

TKEY (Transaction KEY) is a mechanism for automatically negotiating a shared secret between two hosts, originally specified in RFC 2930.

There are several TKEY "modes" that specify how a key is to be generated or assigned. BIND 9 implements only one of these modes: Diffie-Hellman key exchange. Both hosts are required to have a KEY record with algorithm DH (though this record is not required to be present in a zone).

The TKEY process is initiated by a client or server by sending a query of type TKEY to a TKEY-aware server. The query must include an appropriate KEY record in the additional section, and must be signed using either TSIG or SIG(0) with a previously established key. The server's response, if successful, contains a TKEY record in its answer section. After this transaction, both participants have enough information to calculate a shared secret using Diffie-Hellman key exchange. The shared secret can then be used by to sign subsequent transactions between the two servers.

TSIG keys known by the server, including TKEY-negotiated keys, can be listed using **rndc tsig-list**.

TKEY-negotiated keys can be deleted from a server using **rndc tsig-delete**. This can also be done via the TKEY protocol itself, by sending an authenticated TKEY query specifying the "key deletion" mode.

4.7 SIG(0)

BIND partially supports DNSSEC SIG(0) transaction signatures as specified in RFC 2535 and RFC 2931. SIG(0) uses public/private keys to authenticate messages. Access control is performed in the same manner as with TSIG keys; privileges can be granted or denied in ACL directives based on the key name.

When a SIG(0) signed message is received, it is only verified if the key is known and trusted by the server. The server does not attempt to recursively fetch or validate the key.

SIG(0) signing of multiple-message TCP streams is not supported.

The only tool shipped with BIND 9 that generates SIG(0) signed messages is **nsupdate**.

4.8 DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security (*DNSSEC-bis*) extensions, defined in RFC 4033, RFC 4034, and RFC 4035. This section describes the creation and use of DNSSEC signed zones.

In order to set up a DNSSEC secure zone, there are a series of steps which must be followed. BIND 9 ships with several tools that are used in this process, which are explained in more detail below. In all cases, the `-h` option prints a full list of parameters. Note that the DNSSEC tools require the keyset files to be in the working directory or the directory specified by the `-d` option.

There must also be communication with the administrators of the parent and/or child zone to transmit keys. A zone's security status must be indicated by the parent zone for a DNSSEC-capable resolver to trust its data. This is done through the presence or absence of a `DS` record at the delegation point.

For other servers to trust data in this zone, they must be statically configured with either this zone's zone key or the zone key of another zone above this one in the DNS tree.

Generating Keys

The **dnssec-keygen** program is used to generate keys.

A secure zone must contain one or more zone keys. The zone keys will sign all other records in the zone, as well as the zone keys of any secure delegated zones. Zone keys must have the same name as the zone, have a name type of **ZONE**, and be usable for authentication. It is recommended that zone keys use a cryptographic algorithm designated as "mandatory to implement" by the IETF; currently the only one is RSASHA1.

The following command generates a 768-bit RSASHA1 key for the `child.example` zone:

```
dnssec-keygen -a RSASHA1 -b 768 -n ZONE child.example.
```

Two output files are produced: `Kchild.example.+005+12345.key` and `Kchild.example.+005+12345.private` (where 12345 is an example of a key tag). The key filenames contain the key name (`child.example.`), the algorithm (3 is DSA, 1 is RSAMD5, 5 is RSASHA1, etc.), and the key tag (12345 in this case). The private key (in the `.private` file) is used to generate signatures, and the public key (in the `.key` file) is used for signature verification.

To generate another key with the same properties but with a different key tag, repeat the above command.

The **dnssec-keyfromlabel** program is used to get a key pair from a crypto hardware device and build the key files. Its usage is similar to **dnssec-keygen**.

The public keys should be inserted into the zone file by including the `.key` files using **$INCLUDE** statements.

Signing the Zone

The **dnssec-signzone** program is used to sign a zone.

Any `keyset` files corresponding to secure sub-zones should be present. The zone signer generates `NSEC`, `NSEC3`, and `RRSIG` records for the zone, as well as `DS` for the child zones if `-g` is specified. If `-g` is not specified, then DS RRsets for the secure child zones need to be added manually.

By default, all zone keys which have an available private key are used to generate signatures. The following command signs the zone, assuming it is in a file called `zone.child.example`:

```
dnssec-signzone -o child.example zone.child.example
```

One output file is produced: `zone.child.example.signed`. This file should be referenced by `named.conf` as the input file for the zone.

dnssec-signzone also produces keyset and dsset files. These are used to provide the parent zone administrators with the `DNSKEYs` (or their corresponding `DS` records) that are the secure entry point to the zone.

Configuring Servers for DNSSEC

To enable **named** to respond appropriately to DNS requests from DNSSEC-aware clients, **dnssec-enable** must be set to yes. (This is the default setting.)

To enable **named** to validate answers from other servers, the **dnssec-enable** option must be set to **yes**, and the **dnssec-validation** option must be set to **yes** or **auto**.

If **dnssec-validation** is set to **auto**, then a default trust anchor for the DNS root zone is used. If it is set to **yes**, however, then at least one trust anchor must be configured with a **trusted-keys** or **managed-keys** statement in `named.conf`, or DNSSEC validation will not occur. The default setting is **yes**.

trusted-keys are copies of DNSKEY RRs for zones that are used to form the first link in the cryptographic chain of trust. All keys listed in **trusted-keys** (and corresponding zones) are deemed to exist and only the listed keys are used to validate the DNSKEY RRset that they are from.

managed-keys are trusted keys which are automatically kept up-to-date via RFC 5011 trust anchor maintenance.

trusted-keys and **managed-keys** are described in more detail later in this document.

BIND 9 does not verify signatures on load, so zone keys for authoritative zones do not need to be specified in the configuration file.

After DNSSEC is established, a typical DNSSEC configuration looks something like the following. It has one or more public keys for the root, which allows answers from outside the organization to be validated. It also has several keys for parts of the namespace that the organization controls. These are here to ensure that **named** is immune to compromised security in the DNSSEC components of parent zones.

```
managed-keys {
        /* Root Key */
        "." initial-key 257 3 3 "BNY4wrWM1nCfJ+ ↩
            CXd0rVXyYmobt7sEEfK3clRbGaTwS
                                JxrGkxJWoZu6I7PzJu/ ↩
                                    E9gx4UC1zGAHlXKdE4zYIpRh
```

```
                                aBKnvcC2U9mZhkdUpd1Vso/ ↩
                                    HAdjNe8LmMlnzY3zy2Xy
                                4klWOADTPzSv9eamj8V18PHGjBLaVtYvk/ ↩
                                    ln5ZApjYg
                                hf+6fElrmLkdaz MQ2OCnACR817DF4BBa7UR/ ↩
                                    beDHyp
                                5iWTXWSi6XmoJLbG9Scqc7l70KDqlvXR3M/ ↩
                                    lUUVRbke
                                g1IPJSidmK3ZyCllh4XSKbje/45 ↩
                                    SKucHgnwU5jefMtq
                                66gKodQj+ ↩
                                    MiA21AfUVe7u99WzTLzY3qlxDhxYQQ20FQ
                                97S+LKUTpQcq27R7AT3/ ↩
                                    V5hRQxScINqwcz4jYqZD2fQ
                                dgxbcDTClU0CRBdiieyLMNzXG3";
};

trusted-keys {
        /* Key for our organization's forward zone */
        example.com. 257 3 5 "AwEAAaxPMcR2x0HbQV4WeZB6oEDX+r0QM6
                              5KbhTjrW1ZaARmPhEZZe3Y9ifgEuq7vZ/z
                              GZUdEGNWy+JZzus0lUptwgjGwhUS1558Hb
                              4JKUbbOTcM8pwXlj0EiX3oDFVmjHO444gL
                              kBOUKUf/mC7HvfwYH/Be22GnClrinKJp1O
                              g4ywzO9WglMk7jbfW33gUKvirTHr25GL7S
                              TQUzBb5Usxt8lgnyTUHs1t3JwCY5hKZ6Cq
                              FxmAVZP20igTixin/1LcrgX/KMEGd/biuv
                              F4qJCyduieHukuY3H4XMAcR+xia2nIUPvm
                              /oyWR8BW/hWdzOvnSCThlHf3xiYleDbt/o
                              1OTQ09A0=";

        /* Key for our reverse zone. */
        2.0.192.IN-ADDRPA.NET. 257 3 5 "AQOnS4xn/IgOUpBPJ3bogzwc
                                        xOdNax071L18QqZnQQQAVVr+i
                                        LhGTnNGp3HoWQLUIzKrJVZ3zg
                                        gy3WwNT6kZo6c0tszYqbtvchm
                                        gQC8CzKojM/W16i6MG/eafGU3
                                        siaOdS0yOI6BgPsw+YZdzlYMa
                                        IJGf4M4dyoKIhzdZyQ2bYQrjy
                                        Q4LB0lC7aOnsMyYKHHYeRvPxj
                                        IQXmdqgOJGq+vsevG06zW+1xg
                                        YJh9rCIfnm1GX/KMgxLPG2vXT
                                        D/RnLX+D3T3UL7HJYHJhAZD5L
                                        59VvjSPsZJHeDCUyWYrvPZesZ
                                        DIRvhDD52SKvbheeTJUm6Ehkz
                                        ytNN2SN96QRk8j/iI8ib";
};

options {
        ...
        dnssec-enable yes;
        dnssec-validation yes;
};
```

NOTE

None of the keys listed in this example are valid. In particular, the root key is not valid.

When DNSSEC validation is enabled and properly configured, the resolver rejects any answers from signed, secure zones which fail to validate, and returns SERVFAIL to the client.

Responses may fail to validate for any of several reasons, including missing, expired, or invalid signatures, a key which does not match the DS RRset in the parent zone, or an insecure response from a zone which, according to its parent, should have been secure.

NOTE

When the validator receives a response from an unsigned zone that has a signed parent, it must confirm with the parent that the zone was intentionally left unsigned. It does this by verifying, via signed and validated NSEC/NSEC3 records, that the parent zone contains no DS records for the child.

If the validator *can* prove that the zone is insecure, then the response is accepted. However, if it cannot, the validator must assume an insecure response to be a forgery; it rejects the response and logs an error.

The logged error reads "insecurity proof failed" and "got insecure response; parent indicates it should be secure".

4.9 DNSSEC, DYNAMIC ZONES, AND AUTOMATIC SIGNING

Converting from insecure to secure

A zone ca be changed from insecure to secure in two ways: using a dynamic DNS update, or via the **auto-dnssec** zone option.

For either method, **named** must be configured so that it can see the `K*` files which contain the public and private parts of the keys that are used to sign the zone. These files are generated by **dnssec-keygen**, and they should be placed in the key-directory, as specified in `named.conf`:

```
zone example.net {
  type master;
  update-policy local;
  file "dynamic/example.net/example.net";
```

```
    key-directory "dynamic/example.net";
};
```

If one KSK and one ZSK DNSKEY key have been generated, this configuration causes all records in the zone to be signed with the ZSK, and the DNSKEY RRset to be signed with the KSK. An NSEC chain is generated as part of the initial signing process.

Dynamic DNS Update Method

To insert the keys via dynamic update:

```
% nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7  ↩
    AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y I1m/ ↩
    SAQBxIqMfLtIwqWPdgthsu36azGQAX8=
> update add example.net DNSKEY 257 3 7 AwEAAd/7odU/64 ↩
    o2LGsifbLtQmtO8dFDtTAZXSX2+X3e/UNlq9IHq3Y0 XtC0Iuawl/qkaKVxXe2lo8Ct+ ↩
    dM6UehyCqk=
> send
```

While the update request completes almost immediately, the zone is not completely signed until **named** has had time to "walk" the zone and generate the NSEC and RRSIG records. The NSEC record at the apex is added last, to signal that there is a complete NSEC chain.

To sign using NSEC3 instead of NSEC, add an NSEC3PARAM record to the initial update request. The OPTOUT bit in the NSEC3 chain can be set in the flags field of the NSEC3PARAM record.

```
% nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7  ↩
    AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y I1m/ ↩
    SAQBxIqMfLtIwqWPdgthsu36azGQAX8=
> update add example.net DNSKEY 257 3 7 AwEAAd/7odU/64 ↩
    o2LGsifbLtQmtO8dFDtTAZXSX2+X3e/UNlq9IHq3Y0 XtC0Iuawl/qkaKVxXe2lo8Ct+ ↩
    dM6UehyCqk=
> update add example.net NSEC3PARAM 1 1 100 1234567890
> send
```

Again, this update request completes almost immediately; however, the record does not show up until **named** has had a chance to build/remove the relevant chain. A private type record is created to record the state of the operation (see below for more details), and is removed once the operation completes.

While the initial signing and NSEC/NSEC3 chain generation is happening, other updates are possible as well.

Fully Automatic Zone Signing

To enable automatic signing, add the **auto-dnssec** option to the zone statement in `named.conf`. **auto-dnssec** has two possible arguments: `allow` or `maintain`.

With **auto-dnssec allow**, **named** can search the key directory for keys matching the zone, insert them into the zone, and use them to sign the zone. It does so only when it receives an **rndc sign <zonename>**.

auto-dnssec maintain includes the above functionality, but also automatically adjusts the zone's DNSKEY records on a schedule according to the keys' timing metadata. (See dnssec-keygen(8) and dnssec-settime(8) for more information.)

named periodically searches the key directory for keys matching the zone; if the keys' metadata indicates that any change should be made to the zone - such as adding, removing, or revoking a key - then that action is carried out. By default, the key directory is checked for changes every 60 minutes; this period can be adjusted with `dnssec-loadkeys-interval`, up to a maximum of 24 hours. The **rndc loadkeys** forces **named** to check for key updates immediately.

If keys are present in the key directory the first time the zone is loaded, the zone is signed immediately, without waiting for an **rndc sign** or **rndc loadkeys** command. Those commands can still be used when there are unscheduled key changes.

When new keys are added to a zone, the TTL is set to match that of any existing DNSKEY RRset. If there is no existing DNSKEY RRset, the TTL is set to the TTL specified when the key was created (using the **dnssec-keygen -L** option), if any, or to the SOA TTL.

To sign the zone using NSEC3 instead of NSEC, submit an NSEC3PARAM record via dynamic update prior to the scheduled publication and activation of the keys. The OPTOUT bit for the NSEC3 chain can be set in the flags field of the NSEC3PARAM record. The NSEC3PARAM record does not appear in the zone immediately, but it is stored for later reference. When the zone is signed and the NSEC3 chain is completed, the NSEC3PARAM record appears in the zone.

Using the **auto-dnssec** option requires the zone to be configured to allow dynamic updates, by adding an **allow-update** or **update-policy** statement to the zone configuration. If this has not been done, the configuration fails.

Private Type Records

The state of the signing process is signaled by private type records (with a default type value of 65534). When signing is complete, these records with a non-zero initial octet have a non-zero value for the final octet.

If the first octet of a private type record is non-zero, the record indicates either that the zone needs to be signed with the key matching the record, or that all signatures that match the record should be removed. Here are the meanings of the different values of the first octet:

```
algorithm (octet 1)
key id in network order (octet 2 and 3)
removal flag (octet 4)
complete flag (octet 5)
```

Only records flagged as "complete" can be removed via dynamic update; attempts to remove other private type records are silently ignored.

If the first octet is zero (this is a reserved algorithm number that should never appear in a DNSKEY record), the record indicates that changes to the NSEC3 chains are in progress. The rest of the record contains an NSEC3PARAM record, while the flag field tells what operation to perform based on the flag bits:

```
0x01 OPTOUT
0x80 CREATE
0x40 REMOVE
0x20 NONSEC
```

DNSKEY Rollovers

As with insecure-to-secure conversions, DNSSEC keyrolls can be done in two ways: using a dynamic DNS update, or via the **auto-dnssec** zone option.

Dynamic DNS Update Method

To perform key rollovers via dynamic update, the `K*` files for the new keys must be added so that **named** can find them. The new DNSKEY RRs can then be added via dynamic update. **named** then causes the zone to be signed with the new keys; when the signing is complete, the private type records are updated so that the last octet is non-zero.

If this is for a KSK, the parent and any trust anchor repositories of the new KSK must be informed.

The maximum TTL in the zone must expire before removing the old DNSKEY. If it is a KSK that is being updated, the DS RRset in the parent must also be updated its TTL allowed to expire. This ensures that all clients are able to verify at least one signature when the old DNSKEY is removed.

The old DNSKEY can be removed via UPDATE, taking care to specify the correct key. **named** cleans out any signatures generated by the old key after the update completes.

Automatic Key Rollovers

When a new key reaches its activation date (as set by **dnssec-keygen** or **dnssec-settime**), and if the **auto-dnssec** zone option is set to `maintain`, **named** automatically carries out the key rollover. If the key's algorithm has not previously been used to sign the zone, then the zone is fully signed as quickly as possible. However, if the new key replaces an existing key of the same algorithm, the zone is re-signed incrementally, with signatures from the old key replaced with signatures from the new key as their signature validity periods expire. By default, this rollover completes in 30 days, after which it is safe to remove the old key from the DNSKEY RRset.

NSEC3PARAM Rollovers via UPDATE

The new NSEC3PARAM record can be added via dynamic update. When the new NSEC3 chain has been generated, the NSEC3PARAM flag field is set to zero. At that point, the old

NSEC3PARAM record can be removed. The old chain is removed after the update request completes.

Converting From NSEC to NSEC3

To do this, an NSEC3PARAM record must be added. When the conversion is complete, the NSEC chain is removed and the NSEC3PARAM record has a zero flag field. The NSEC3 chain is generated before the NSEC chain is destroyed.

Converting From NSEC3 to NSEC

To do this, use **nsupdate** to remove all NSEC3PARAM records with a zero flag field. The NSEC chain is generated before the NSEC3 chain is removed.

Converting From Secure to Insecure

To convert a signed zone to unsigned using dynamic DNS, delete all the DNSKEY records from the zone apex using **nsupdate**. All signatures, NSEC or NSEC3 chains, and associated NSEC3PARAM records are removed automatically. This takes place after the update request completes.

This requires the **dnssec-secure-to-insecure** option to be set to **yes** in `named.conf`.

In addition, if the **auto-dnssec maintain** zone statement is used, it should be removed or changed to **allow** instead; otherwise, it will re-sign).

Periodic Re-signing

In any secure zone which supports dynamic updates, **named** periodically re-signs RRsets which have not been re-signed as a result of some update action. The signature lifetimes are adjusted to spread the re-sign load over time rather than all at once.

NSEC3 and OPTOUT

named only supports creating new NSEC3 chains where all the NSEC3 records in the zone have the same OPTOUT state. **named** supports UPDATES to zones where the NSEC3 records in the chain have mixed OPTOUT state. **named** does not support changing the OPTOUT state of an individual NSEC3 record; if the OPTOUT state of an individual NSEC3 needs to be changed, the entire chain must be changed.

4.10 DYNAMIC TRUST ANCHOR MANAGEMENT

BIND is able to maintain DNSSEC trust anchors using RFC 5011 key management. This feature allows **named** to keep track of changes to critical DNSSEC keys without any need for the operator to make changes to configuration files.

Validating Resolver

To configure a validating resolver to use RFC 5011 to maintain a trust anchor, configure the trust anchor using a **managed-keys** statement. Information about this can be found in Section 6.2.

Authoritative Server

To set up an authoritative zone for RFC 5011 trust anchor maintenance, generate two (or more) key signing keys (KSKs) for the zone. Sign the zone with one of them; this is the "active" KSK. All KSKs which do not sign the zone are "stand-by" keys.

Any validating resolver which is configured to use the active KSK as an RFC 5011-managed trust anchor takes note of the stand-by KSKs in the zone's DNSKEY RRset, and stores them for future reference. The resolver rechecks the zone periodically; after 30 days, if the new key is still there, the key is accepted by the resolver as a valid trust anchor for the zone. Anytime after this 30-day acceptance timer has completed, the active KSK can be revoked, and the zone can be "rolled over" to the newly accepted key.

The easiest way to place a stand-by key in a zone is to use the "smart signing" features of **dnssec-keygen** and **dnssec-signzone**. If a key exists with a publication date in the past, but an activation date which is unset or in the future, **dnssec-signzone -S** includes the DNSKEY record in the zone but does not sign with it:

```
$ dnssec-keygen -K keys -f KSK -P now -A now+2y example.net
$ dnssec-signzone -S -K keys example.net
```

To revoke a key, use the command **dnssec-revoke**. This adds the REVOKED bit to the key flags and regenerates the `K*.key` and `K*.private` files.

After revoking the active key, the zone must be signed with both the revoked KSK and the new active KSK. Smart signing takes care of this automatically.

Once a key has been revoked and used to sign the DNSKEY RRset in which it appears, that key is never again accepted as a valid trust anchor by the resolver. However, validation can proceed using the new active key, which was accepted by the resolver when it was a stand-by key.

See RFC 5011 for more details on key rollover scenarios.

When a key has been revoked, its key ID changes, increasing by 128 and wrapping around at 65535. So, for example, the key "`Kexample.com.+005+10000`" becomes "`Kexample.com.+005+10128`".

If two keys have IDs exactly 128 apart and one is revoked, the two key IDs will collide, causing several problems. To prevent this, **dnssec-keygen** does not generate a new key if another key which may collide is present. This checking only occurs if the new keys are written to the same directory that holds all other keys in use for that zone.

Older versions of BIND 9 did not have this protection. Exercise caution if using key revocation on keys that were generated by previous releases, or if using keys stored in multiple directories or on multiple machines.

It is expected that a future release of BIND 9 will address this problem in a different way, by storing revoked keys with their original unrevoked key IDs.

4.11 PKCS#11 (CRYPTOKI) SUPPORT

Public Key Cryptography Standard #11 (PKCS#11) defines a platform-independent API for the control of hardware security modules (HSMs) and other cryptographic support devices.

BIND 9 is known to work with three HSMs: The AEP Keyper, which has been tested with Debian Linux, Solaris x86 and Windows Server 2003; the Thales nShield, tested with Debian Linux; and the Sun SCA 6000 cryptographic acceleration board, tested with Solaris x86. In addition, BIND can be used with all current versions of SoftHSM, a software-based HSM simulator library produced by the OpenDNSSEC project.

PKCS#11 makes use of a "provider library": a dynamically loadable library which provides a low-level PKCS#11 interface to drive the HSM hardware. The PKCS#11 provider library comes from the HSM vendor, and it is specific to the HSM to be controlled.

There are two available mechanisms for PKCS#11 support in BIND 9: OpenSSL-based PKCS#11 and native PKCS#11. When using the first mechanism, BIND uses a modified version of OpenSSL, which loads the provider library and operates the HSM indirectly; any cryptographic operations not supported by the HSM can be carried out by OpenSSL instead. The second mechanism enables BIND to bypass OpenSSL completely; BIND loads the provider library itself, and uses the PKCS#11 API to drive the HSM directly.

Prerequisites

See the documentation provided by your HSM vendor for information about installing, initializing, testing and troubleshooting the HSM.

Native PKCS#11

Native PKCS#11 mode will only work with an HSM capable of carrying out *every* cryptographic operation BIND 9 may need. The HSM's provider library must have a complete implementation of the PKCS#11 API, so that all these functions are accessible. As of this writing, only the Thales nShield HSM and SoftHSMv2 can be used in this fashion. For other HSMs, including the AEP Keyper, Sun SCA 6000 and older versions of SoftHSM, use OpenSSL-based PKCS#11. (Note: Eventually, when more HSMs become capable of supporting native PKCS#11, it is expected that OpenSSL-based PKCS#11 will be deprecated.)

To build BIND with native PKCS#11, configure as follows:

```
$ cd bind9
$ ./configure --enable-native-pkcs11 \
    --with-pkcs11=provider-library-path
```

This will cause all BIND tools, including **named** and the **dnssec-*** and **pkcs11-*** tools, to use the PKCS#11 provider library specified in `provider-library-path` for cryptography. (The provider library path can be overridden using the `-E` in **named** and the **dnssec-*** tools, or the `-m` in the **pkcs11-*** tools.)

Building SoftHSMv2

SoftHSMv2, the latest development version of SoftHSM, is available from https://github.com/-opendnssec/SoftHSMv2 . It is a software library developed by the OpenDNSSEC project (http://www.opendnss) which provides a PKCS#11 interface to a virtual HSM, implemented in the form of a SQLite3 database on the local filesystem. It provides less security than a true HSM, but it allows you to experiment with native PKCS#11 when an HSM is not available. SoftHSMv2 can be configured to use either OpenSSL or the Botan library to perform cryptographic functions, but when using it for native PKCS#11 in BIND, OpenSSL is required.

By default, the SoftHSMv2 configuration file is *prefix*/etc/softhsm2.conf (where *prefix* is configured at compile time). This location can be overridden by the SOFTHSM2_CONF environment variable. The SoftHSMv2 cryptographic store must be installed and initialized before using it with BIND.

```
$ cd SoftHSMv2
$ configure --with-crypto-backend=openssl --prefix=/opt/pkcs11/usr -- ↩
  enable-gost
$ make
$ make install
$ /opt/pkcs11/usr/bin/softhsm-util --init-token 0 --slot 0 --label  ↩
  softhsmv2
```

OpenSSL-based PKCS#11

OpenSSL-based PKCS#11 mode uses a modified version of the OpenSSL library; stock OpenSSL does not fully support PKCS#11. ISC provides a patch to OpenSSL to correct this. This patch is based on work originally done by the OpenSolaris project; it has been modified by ISC to provide new features such as PIN management and key-by-reference.

There are two "flavors" of PKCS#11 support provided by the patched OpenSSL, one of which must be chosen at configuration time. The correct choice depends on the HSM hardware:

- Use 'crypto-accelerator' with HSMs that have hardware cryptographic acceleration features, such as the SCA 6000 board. This causes OpenSSL to run all supported cryptographic operations in the HSM.
- Use 'sign-only' with HSMs that are designed to function primarily as secure key storage devices, but lack hardware acceleration. These devices are highly secure, but are not necessarily any faster at cryptography than the system CPU --- often, they are slower. It is therefore most efficient to use them only for those cryptographic functions that require access to the secured private key, such as zone signing, and to use the system CPU for all other computationally-intensive operations. The AEP Keyper is an example of such a device.

The modified OpenSSL code is included in the BIND 9 release, in the form of a context diff against the latest versions of OpenSSL. OpenSSL 0.9.8, 1.0.0, 1.0.1 and 1.0.2 are supported; there are separate diffs for each version. In the examples to follow, we use OpenSSL 0.9.8, but the same methods work with OpenSSL 1.0.0 through 1.0.2.

> NOTE
>
> The OpenSSL patches as of this writing (January 2016) support versions 0.9.8zh, 1.0.0t, 1.0.1q and 1.0.2f. ISC will provide updated patches as new versions of OpenSSL are released. The version number in the following examples is expected to change.

Before building BIND 9 with PKCS#11 support, it will be necessary to build OpenSSL with the patch in place, and configure it with the path to your HSM's PKCS#11 provider library.

Patching OpenSSL

```
$ wget http://www.openssl.org/source/openssl-0.9.8zc.tar.gz
```

Extract the tarball:

```
$ tar zxf openssl-0.9.8zc.tar.gz
```

Apply the patch from the BIND 9 release:

```
$ patch -p1 -d openssl-0.9.8zc \
        < bind9/bin/pkcs11/openssl-0.9.8zc-patch
```

> NOTE
>
> The patch file may not be compatible with the "patch" utility on all operating systems. You may need to install GNU patch.

When building OpenSSL, place it in a non-standard location so that it does not interfere with OpenSSL libraries elsewhere on the system. In the following examples, we choose to install into "/opt/pkcs11/usr". We will use this location when we configure BIND 9.

Later, when building BIND 9, the location of the custom-built OpenSSL library will need to be specified via configure.

Building OpenSSL for the AEP Keyper on Linux

The AEP Keyper is a highly secure key storage device, but does not provide hardware cryptographic acceleration. It can carry out cryptographic operations, but it is probably slower than your system's CPU. Therefore, we choose the 'sign-only' flavor when building OpenSSL.

The Keyper-specific PKCS#11 provider library is delivered with the Keyper software. In this example, we place it /opt/pkcs11/usr/lib:

```
$ cp pkcs11.GCC4.0.2.so.4.05 /opt/pkcs11/usr/lib/libpkcs11.so
```

The Keyper library requires threads, so we must specify -pthread.

```
$ cd openssl-0.9.8zc
$ ./Configure linux-x86_64 -pthread \
      --pk11-libname=/opt/pkcs11/usr/lib/libpkcs11.so \
      --pk11-flavor=sign-only \
      --prefix=/opt/pkcs11/usr
```

After configuring, run "**make**" and "**make test**". If "**make test**" fails with "pthread_atfork() not found", you forgot to add the -pthread above.

Building OpenSSL for the SCA 6000 on Solaris

The SCA-6000 PKCS#11 provider is installed as a system library, libpkcs11. It is a true crypto accelerator, up to 4 times faster than any CPU, so the flavor shall be 'crypto-accelerator'.

In this example, we are building on Solaris x86 on an AMD64 system.

```
$ cd openssl-0.9.8zc
$ ./Configure solaris64-x86_64-cc \
      --pk11-libname=/usr/lib/64/libpkcs11.so \
      --pk11-flavor=crypto-accelerator \
      --prefix=/opt/pkcs11/usr
```

(For a 32-bit build, use "solaris-x86-cc" and /usr/lib/libpkcs11.so.)

After configuring, run **make** and **make test**.

Building OpenSSL for SoftHSM

SoftHSM (version 1) is a software library developed by the OpenDNSSEC project (http://www.opendnssec.org) which provides a PKCS#11 interface to a virtual HSM, implemented in the form of a SQLite3 database on the local filesystem. SoftHSM uses the Botan library to perform cryptographic functions. Though less secure than a true HSM, it can allow you to experiment with PKCS#11 when an HSM is not available.

The SoftHSM cryptographic store must be installed and initialized before using it with OpenSSL, and the SOFTHSM_CONF environment variable must always point to the SoftHSM configuration file:

```
$  cd softhsm-1.3.7
$  configure --prefix=/opt/pkcs11/usr
$  make
$  make install
$  export SOFTHSM_CONF=/opt/pkcs11/softhsm.conf
$  echo "0:/opt/pkcs11/softhsm.db" > $SOFTHSM_CONF
$  /opt/pkcs11/usr/bin/softhsm --init-token 0 --slot 0 --label softhsm
```

SoftHSM can perform all cryptographic operations, but since it only uses your system CPU, there is no advantage to using it for anything but signing. Therefore, we choose the 'sign-only' flavor when building OpenSSL.

```
$ cd openssl-0.9.8zc
$ ./Configure linux-x86_64 -pthread \
      --pk11-libname=/opt/pkcs11/usr/lib/libsofthsm.so \
      --pk11-flavor=sign-only \
      --prefix=/opt/pkcs11/usr
```

After configuring, run "**make**" and "**make test**".

Once you have built OpenSSL, run "**apps/openssl engine pkcs11**" to confirm that PKCS#11 support was compiled in correctly. The output should be one of the following lines, depending on the flavor selected:

```
  (pkcs11) PKCS #11 engine support (sign only)
```

Or:

```
  (pkcs11) PKCS #11 engine support (crypto accelerator)
```

Next, run "**apps/openssl engine pkcs11 -t**". This will attempt to initialize the PKCS#11 engine. If it is able to do so successfully, it will report "`[ available ]`".

If the output is correct, run "**make install**" which will install the modified OpenSSL suite to `/opt/pkcs11/usr`.

Configuring BIND 9 for Linux with the AEP Keyper

To link with the PKCS#11 provider, threads must be enabled in the BIND 9 build.

```
$ cd ../bind9
$ ./configure --enable-threads \
     --with-openssl=/opt/pkcs11/usr \
     --with-pkcs11=/opt/pkcs11/usr/lib/libpkcs11.so
```

Configuring BIND 9 for Solaris with the SCA 6000

To link with the PKCS#11 provider, threads must be enabled in the BIND 9 build.

```
$ cd ../bind9
$ ./configure CC="cc -xarch=amd64" --enable-threads \
      --with-openssl=/opt/pkcs11/usr \
      --with-pkcs11=/usr/lib/64/libpkcs11.so
```

(For a 32-bit build, omit CC="cc -xarch=amd64".)

If configure complains about OpenSSL not working, you may have a 32/64-bit architecture mismatch. Or, you may have incorrectly specified the path to OpenSSL (it should be the same as the --prefix argument to the OpenSSL Configure).

Configuring BIND 9 for SoftHSM

```
$ cd ../bind9
$ ./configure --enable-threads \
     --with-openssl=/opt/pkcs11/usr \
     --with-pkcs11=/opt/pkcs11/usr/lib/libsofthsm.so
```

After configuring, run "**make**", "**make test**" and "**make install**".

(Note: If "make test" fails in the "pkcs11" system test, you may have forgotten to set the SOFTHSM_CONF environment variable.)

PKCS#11 Tools

BIND 9 includes a minimal set of tools to operate the HSM, including **pkcs11-keygen** to generate a new key pair within the HSM, **pkcs11-list** to list objects currently available, **pkcs11-destroy** to remove objects, and **pkcs11-tokens** to list available tokens.

In UNIX/Linux builds, these tools are built only if BIND 9 is configured with the --with-pkcs11 option. (Note: If --with-pkcs11 is set to "yes", rather than to the path of the PKCS#11 provider, then the tools will be built but the provider will be left undefined. Use the -m option or the PKCS11_PROVIDER environment variable to specify the path to the provider.)

Using the HSM

For OpenSSL-based PKCS#11, we must first set up the runtime environment so the OpenSSL and PKCS#11 libraries can be loaded:

```
$ export LD_LIBRARY_PATH=/opt/pkcs11/usr/lib:${LD_LIBRARY_PATH}
```

This causes **named** and other binaries to load the OpenSSL library from `/opt/pkcs11/usr/lib` rather than from the default location. This step is not necessary when using native PKCS#11.

Some HSMs require other environment variables to be set. For example, when operating an AEP Keyper, it is necessary to specify the location of the "machine" file, which stores information about the Keyper for use by the provider library. If the machine file is in `/opt/Keyper/PKCS11Provider/machine`, use:

```
$ export KEYPER_LIBRARY_PATH=/opt/Keyper/PKCS11Provider
```

Such environment variables must be set whenever running any tool that uses the HSM, including **pkcs11-keygen**, **pkcs11-list**, **pkcs11-destroy**, **dnssec-keyfromlabel**, **dnssec-signzone**, **dnssec-keygen**, and **named**.

We can now create and use keys in the HSM. In this case, we will create a 2048 bit key and give it the label "sample-ksk":

```
$ pkcs11-keygen -b 2048 -l sample-ksk
```

To confirm that the key exists:

```
$ pkcs11-list
Enter PIN:
object[0]: handle 2147483658 class 3 label[8] 'sample-ksk' id[0]
object[1]: handle 2147483657 class 2 label[8] 'sample-ksk' id[0]
```

Before using this key to sign a zone, we must create a pair of BIND 9 key files. The "dnssec-keyfromlabel" utility does this. In this case, we will be using the HSM key "sample-ksk" as the key-signing key for "example.net":

```
$ dnssec-keyfromlabel -l sample-ksk -f KSK example.net
```

The resulting K*.key and K*.private files can now be used to sign the zone. Unlike normal K* files, which contain both public and private key data, these files will contain only the public key data, plus an identifier for the private key which remains stored within the HSM. Signing with the private key takes place inside the HSM.

If you wish to generate a second key in the HSM for use as a zone-signing key, follow the same procedure above, using a different keylabel, a smaller key size, and omitting "-f KSK" from the dnssec-keyfromlabel arguments:

(Note: When using OpenSSL-based PKCS#11 the label is an arbitrary string which identifies the key. With native PKCS#11, the label is a PKCS#11 URI string which may include other details about the key and the HSM, including its PIN. See dnssec-keyfromlabel(8) for details.)

```
$ pkcs11-keygen -b 1024 -l sample-zsk
$ dnssec-keyfromlabel -l sample-zsk example.net
```

Alternatively, you may prefer to generate a conventional on-disk key, using dnssec-keygen:

```
$ dnssec-keygen example.net
```

This provides less security than an HSM key, but since HSMs can be slow or cumbersome to use for security reasons, it may be more efficient to reserve HSM keys for use in the less frequent key-signing operation. The zone-signing key can be rolled more frequently, if you wish, to compensate for a reduction in key security. (Note: When using native PKCS#11, there is no speed advantage to using on-disk keys, as cryptographic operations will be done by the HSM regardless.)

Now you can sign the zone. (Note: If not using the -S option to **dnssec-signzone**, it will be necessary to add the contents of both `K*.key` files to the zone master file before signing it.)

```
$ dnssec-signzone -S example.net
Enter PIN:
Verifying the zone using the following algorithms:
NSEC3RSASHA1.
Zone signing complete:
Algorithm: NSEC3RSASHA1: ZSKs: 1, KSKs: 1 active, 0 revoked, 0 stand-by
example.net.signed
```

Specifying the engine on the command line

When using OpenSSL-based PKCS#11, the "engine" to be used by OpenSSL can be specified in **named** and all of the BIND **dnssec-*** tools by using the "-E <engine>" command line option. If BIND 9 is built with the --with-pkcs11 option, this option defaults to "pkcs11". Specifying the engine will generally not be necessary unless for some reason you wish to use a different OpenSSL engine.

If you wish to disable use of the "pkcs11" engine --- for troubleshooting purposes, or because the HSM is unavailable --- set the engine to the empty string. For example:

```
$ dnssec-signzone -E '' -S example.net
```

This causes **dnssec-signzone** to run as if it were compiled without the --with-pkcs11 option.

When built with native PKCS#11 mode, the "engine" option has a different meaning: it specifies the path to the PKCS#11 provider library. This may be useful when testing a new provider library.

Running named with automatic zone re-signing

If you want **named** to dynamically re-sign zones using HSM keys, and/or to to sign new records inserted via nsupdate, then **named** must have access to the HSM PIN. In OpenSSL-based PKCS#11, this is accomplished by placing the PIN into the openssl.cnf file (in the above examples, `/opt/pkcs11/usr/ssl/openssl.cnf`).

The location of the openssl.cnf file can be overridden by setting the OPENSSL_CONF environment variable before running **named**.

Sample openssl.cnf:

```
openssl_conf = openssl_def
[ openssl_def ]
engines = engine_section
[ engine_section ]
pkcs11 = pkcs11_section
[ pkcs11_section ]
PIN = <PLACE PIN HERE>
```

This will also allow the dnssec-* tools to access the HSM without PIN entry. (The pkcs11-* tools access the HSM directly, not via OpenSSL, so a PIN will still be required to use them.)

In native PKCS#11 mode, the PIN can be provided in a file specified as an attribute of the key's label. For example, if a key had the label **`pkcs11:object=local-zsk;pin-source=/etc/hsmpin`**, then the PIN would be read from the file `/etc/hsmpin`.

WARNING

Placing the HSM's PIN in a text file in this manner may reduce the security advantage of using an HSM. Be sure this is what you want to do before configuring the system in this way.

4.12 DLZ (DYNAMICALLY LOADABLE ZONES)

Dynamically Loadable Zones (DLZ) are an extension to BIND 9 that allows zone data to be retrieved directly from an external database. There is no required format or schema. DLZ drivers exist for several different database backends, including PostgreSQL, MySQL, and LDAP, and can be written for any other.

Historically, DLZ drivers had to be statically linked with the **named** binary and were turned on via a configure option at compile time (for example, **`configure --with-dlz-ldap`**). The drivers provided in the BIND 9 tarball in `contrib/dlz/drivers` are still linked this way.

In BIND 9.8 and higher, it is possible to link some DLZ modules dynamically at runtime, via the DLZ "dlopen" driver, which acts as a generic wrapper around a shared object implementing the DLZ API. The "dlopen" driver is linked into **named** by default, so configure options are no longer necessary when using these dynamically linkable drivers; they are still needed for the older drivers in `contrib/dlz/drivers`.

The DLZ module provides data to **named** in text format, which is then converted to DNS wire format by **named**. This conversion, and the lack of any internal caching, places significant limits on the query performance of DLZ modules. Consequently, DLZ is not recommended for use on high-volume servers. However, it can be used in a hidden primary configuration, with secondaries retrieving zone updates via AXFR. Note, however, that DLZ has no built-in support for DNS notify; secondary servers are not automatically informed of changes to the zones in the database.

Configuring DLZ

A DLZ database is configured with a **dlz** statement in `named.conf`:

```
    dlz example {
  database "dlopen driver.so args";
  search yes;
    };
```

This specifies a DLZ module to search when answering queries; the module is implemented in `driver.so` and is loaded at runtime by the dlopen DLZ driver. Multiple **dlz** statements can be specified; when answering a query, all DLZ modules with `search` set to `yes` are queried to see whether they contain an answer for the query name. The best available answer is returned to the client.

The `search` option in the above example can be omitted, because `yes` is the default value.

If `search` is set to `no`, then this DLZ module is *not* searched for the best match when a query is received. Instead, zones in this DLZ must be separately specified in a zone statement. This allows users to configure a zone normally using standard zone-option semantics, but specify a different database backend for storage of the zone's data. For example, to implement NXDOMAIN redirection using a DLZ module for backend storage of redirection rules:

```
    dlz other {
  database "dlopen driver.so args";
  search no;
    };
```

```
  zone "." {
type redirect;
dlz other;
  };
```

Sample DLZ Driver

For guidance in the implementation of DLZ modules, the directory `contrib/dlz/example` contains a basic dynamically linkable DLZ module - i.e., one which can be loaded at runtime by the "dlopen" DLZ driver. The example sets up a single zone, whose name is passed to the module as an argument in the **dlz** statement:

```
  dlz other {
database "dlopen driver.so example.nil";
  };
```

In the above example, the module is configured to create a zone "example.nil", which can answer queries and AXFR requests and accept DDNS updates. At runtime, prior to any updates, the zone contains an SOA, NS, and a single A record at the apex:

```
example.nil.  3600    IN      SOA     example.nil. hostmaster.example.nil ↩
    . (
                123 900 600 86400 3600
              )
example.nil.  3600    IN      NS      example.nil.
example.nil.  1800    IN      A       10.53.0.1
```

The sample driver can retrieve information about the querying client and alter its response on the basis of this information. To demonstrate this feature, the example driver responds to queries for "source-addr.`zonename`>/TXT" with the source address of the query. Note, however, that this record will *not* be included in AXFR or ANY responses. Normally, this feature is used to alter responses in some other fashion, e.g., by providing different address records for a particular name depending on the network from which the query arrived.

Documentation of the DLZ module API can be found in `contrib/dlz/example/README`. This directory also contains the header file `dlz_minimal.h`, which defines the API and should be included by any dynamically linkable DLZ module.

4.13 DYNAMIC DATABASE (DYNDB)

Dynamic Database, or DynDB, is an extension to BIND 9 which, like DLZ (see Section 4.12), allows zone data to be retrieved from an external database. Unlike DLZ, a DynDB module provides a full-featured BIND zone database interface. Where DLZ translates DNS queries into real-time database lookups, resulting in relatively poor query performance, and is unable to handle DNSSEC-signed data due to its limited API, a DynDB module can pre-load an in-memory database from the external data source, providing the same performance and functionality as zones served natively by BIND.

A DynDB module supporting LDAP has been created by Red Hat and is available from https://pagure.io/-bind-dyndb-ldap.

A sample DynDB module for testing and developer guidance is included with the BIND source code, in the directory `bin/tests/system/dyndb/driver`.

Configuring DynDB

A DynDB database is configured with a **dyndb** statement in `named.conf`:

```
dyndb example "driver.so" {
    parameters
};
```

The file `driver.so` is a DynDB module which implements the full DNS database API. Multiple **dyndb** statements can be specified, to load different drivers or multiple instances of the same driver. Zones provided by a DynDB module are added to the view's zone table, and are treated as normal authoritative zones when BIND responds to queries. Zone configuration is handled internally by the DynDB module.

The *parameters* are passed as an opaque string to the DynDB module's initialization routine. Configuration syntax differs depending on the driver.

Sample DynDB Module

For guidance in the implementation of DynDB modules, the directory `bin/tests/system/dyndb/driver` contains a basic DynDB module. The example sets up two zones, whose names are passed to the module as arguments in the **dyndb** statement:

```
dyndb sample "sample.so" { example.nil. arpa. };
```

In the above example, the module is configured to create a zone, "example.nil", which can answer queries and AXFR requests, and accept DDNS updates. At runtime, prior to any updates, the zone contains an SOA, NS, and a single A record at the apex:

```
example.nil.   86400    IN      SOA     example.nil. example.nil. (
                                               0 28800 7200 604800 86400
                                       )
example.nil.   86400    IN      NS      example.nil.
example.nil.   86400    IN      A       127.0.0.1
```

When the zone is updated dynamically, the DynDB module determines whether the updated RR is an address (i.e., type A or AAAA); if so, it automatically updates the corresponding PTR record in a reverse zone. Note that updates are not stored permanently; all updates are lost when the server is restarted.

4.14 CATALOG ZONES

A "catalog zone" is a special DNS zone that contains a list of other zones to be served, along with their configuration parameters. Zones listed in a catalog zone are called "member zones." When

a catalog zone is loaded or transferred to a secondary server which supports this functionality, the secondary server creates the member zones automatically. When the catalog zone is updated (for example, to add or delete member zones, or change their configuration parameters), those changes are immediately put into effect. Because the catalog zone is a normal DNS zone, these configuration changes can be propagated using the standard AXFR/IXFR zone transfer mechanism.

Catalog zones' format and behavior are specified as an Internet draft for interoperability among DNS implementations. The latest revision of the DNS catalog zones draft can be found here: https://datatracker.ietf.org/doc/draft-toorop-dnsop-dns-catalog-zones/.

Principle of Operation

Normally, if a zone is to be served by a secondary server, the `named.conf` file on the server must list the zone, or the zone must be added using **rndc addzone**. In environments with a large number of secondary servers, and/or where the zones being served are changing frequently, the overhead involved in maintaining consistent zone configuration on all the secondary servers can be significant.

A catalog zone is a way to ease this administrative burden: it is a DNS zone that lists member zones that should be served by secondary servers. When a secondary server receives an update to the catalog zone, it adds, removes, or reconfigures member zones based on the data received.

To use a catalog zone, it must first be set up as a normal zone on both the primary and secondary servers that are configured to use it. It must also be added to a `catalog-zones` list in the `options` or `view` statement in `named.conf`. This is comparable to the way a policy zone is configured as a normal zone and also listed in a `response-policy` statement.

To use the catalog zone feature to serve a new member zone:

- Set up the the member zone to be served on the primary as normal. This can be done by editing `named.conf` or by running **rndc addzone**.
- Add an entry to the catalog zone for the new member zone. This can be done by editing the catalog zone's zone file and running **rndc reload**, or by updating the zone using **nsupdate**.

The change to the catalog zone is propagated from the primary to all secondaries using the normal AXFR/IXFR mechanism. When the secondary receives the update to the catalog zone, it detects the entry for the new member zone, creates an instance of that zone on the secondary server, and points that instance to the `masters` specified in the catalog zone data. The newly created member zone is a normal secondary zone, so BIND immediately initiates a transfer of zone contents from the primary. Once complete, the secondary starts serving the member zone.

Removing a member zone from a secondary server requires only deleting the member zone's entry in the catalog zone; the change to the catalog zone is propagated to the secondary server using the normal AXFR/IXFR transfer mechanism. The secondary server, on processing the update, notices that the member zone has been removed, stops serving the zone, and removes it from its list of configured zones. However, removing the member zone from the primary server must be done by editing the configuration file or running **rndc delzone**.)

Configuring Catalog Zones

Catalog zones are configured with a **catalog-zones** statement in the `options` or `view` section of `named.conf`. For example,

```
catalog-zones {
  zone "catalog.example"
       default-masters { 10.53.0.1; }
       in-memory no
       zone-directory "catzones"
       min-update-interval 10;
};
```

This statement specifies that the zone `catalog.example` is a catalog zone. This zone must be properly configured in the same view. In most configurations, it would be a secondary zone.

The options following the zone name are not required, and may be specified in any order:

The `default-masters` option defines the default primaries for member zones listed in a catalog zone, and can be overridden by options within a catalog zone. If no such options are included, then member zones transfer their contents from the servers listed in this option.

The `in-memory` option, if set to `yes`, causes member zones to be stored only in memory. This is functionally equivalent to configuring a secondary zone without a `file` option. The default is `no`; member zones' content is stored locally in a file whose name is automatically generated from the view name, catalog zone name, and member zone name.

The `zone-directory` option causes local copies of member zones' zone files to be stored in the specified directory, if `in-memory` is not set to `yes`. The default is to store zone files in the server's working directory. A non-absolute pathname in `zone-directory` is assumed to be relative to the working directory.

The `min-update-interval` option sets the minimum interval between processing of updates to catalog zones, in seconds. If an update to a catalog zone (for example, via IXFR) happens less than `min-update-interval` seconds after the most recent update, the changes are not carried out until this interval has elapsed. The default is `5` seconds.

Catalog zones are defined on a per-view basis. Configuring a non-empty `catalog-zones` statement in a view automatically turns on `allow-new-zones` for that view. This means that **rndc addzone** and **rndc delzone** also work in any view that supports catalog zones.

Catalog Zone Format

A catalog zone is a regular DNS zone; therefore, it must have a single `SOA` and at least one `NS` record.

A record stating the version of the catalog zone format is also required. If the version number listed is not supported by the server, then a catalog zone may not be used by that server.

```
catalog.example.    IN SOA . . 2016022901 900 600 86400 1
catalog.example.    IN NS nsexample.
version.catalog.example.    IN TXT "1"
```

Note that this record must have the domain name "version.*catalog-zone-name*". The data stored in a catalog zone is indicated by the the domain name label immediately before the catalog zone domain.

Catalog zone options can be set either globally for the whole catalog zone or for a single member zone. Global options override the settings in the configuration file, and member zone options override global options.

Global options are set at the apex of the catalog zone, e.g.:

```
masters.catalog.example.    IN AAAA 2001:db8::1
```

BIND currently supports the following options:

- A simple `masters` definition:

  ```
  masters.catalog.example.    IN A 192.0.2.1
  ```

  This option defines a primary server for the member zones, which can be either an A or AAAA record. If multiple primaries are set, the order in which they are used is random.

- A `masters` with a TSIG key defined:

  ```
  label.masters.catalog.example.     IN A 192.0.2.2
  label.masters.catalog.example.     IN TXT "tsig_key_name"
  ```

  This option defines a primary server for the member zone with a TSIG key set. The TSIG key must be configured in the configuration file. `label` can be any valid DNS label.

- `allow-query` and `allow-transfer` ACLs:

  ```
  allow-query.catalog.example. IN APL 1:10.0.0.1/24
  allow-transfer.catalog.example.  IN APL !1:10.0.0.1/32  ↩
      1:10.0.0.0/24
  ```

  These options are the equivalents of `allow-query` and `allow-transfer` in a zone declaration in the `named.conf` configuration file. The ACL is processed in order; if there is no match to any rule, the default policy is to deny access. For the syntax of the APL RR, see RFC 3123.

A member zone is added by including a `PTR` resource record in the `zones` sub-domain of the catalog zone. The record label is a `SHA-1` hash of the member zone name in wire format. The target of the PTR record is the member zone name. For example, to add the member zone `domain.example`:

```
5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN PTR  ↩
    domain.example.
```

The hash is necessary to identify options for a specific member zone. The member zone-specific options are defined the same way as global options, but in the member zone subdomain:

```
masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN ↩
    A 192.0.2.2
label.masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog. ↩
   example. IN AAAA 2001:db8::2
label.masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog. ↩
   example. IN TXT "tsig_key"
allow-query.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example ↩
   . IN APL 1:10.0.0.0/24
```

Options defined for a specific zone override the global options defined in the catalog zone. These in turn override the global options defined in the `catalog-zones` statement in the configuration file.

Note that none of the global records for an option are inherited if any records are defined for that option for the specific zone. For example, if the zone had a `masters` record of type A but not AAAA, it would *not* inherit the type AAAA record from the global option.

4.15 IPv6 Support in BIND 9

BIND 9 fully supports all currently defined forms of IPv6 name-to-address and address-to-name lookups. It also uses IPv6 addresses to make queries when running on an IPv6-capable system.

For forward lookups, BIND 9 supports only AAAA records. RFC 3363 deprecated the use of A6 records, and client-side support for A6 records was accordingly removed from BIND 9. However, authoritative BIND 9 name servers still load zone files containing A6 records correctly, answer queries for A6 records, and accept zone transfer for a zone containing A6 records.

For IPv6 reverse lookups, BIND 9 supports the traditional "nibble" format used in the *ip6.arpa* domain, as well as the older, deprecated *ip6.int* domain. Older versions of BIND 9 supported the "binary label" (also known as "bitstring") format, but support of binary labels has been completely removed per RFC 3363. Many applications in BIND 9 do not understand the binary label format at all anymore, and return an error if one is given. In particular, an authoritative BIND 9 name server will not load a zone file containing binary labels.

For an overview of the format and structure of IPv6 addresses, see Section C.1.

Address Lookups Using AAAA Records

The IPv6 AAAA record is a parallel to the IPv4 A record, and, unlike the deprecated A6 record, specifies the entire IPv6 address in a single record. For example:

```
$ORIGIN example.com.
host            3600    IN      AAAA    2001:db8::1
```

Use of IPv4-in-IPv6 mapped addresses is not recommended. If a host has an IPv4 address, use an A record, not a AAAA, with `::ffff:192.168.42.1` as the address.

Address-to-Name Lookups Using Nibble Format

When looking up an address in nibble format, the address components are simply reversed, just as in IPv4, and `ip6.arpa.` is appended to the resulting name. For example, the following would provide reverse name lookup for a host with address `2001:db8::1`:

```
$ORIGIN 0.0.0.0.0.0.0.0.8.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  14400   IN    PTR    (
                                    host.example.com. )
```

5 The BIND 9 Lightweight Resolver

5.1 The Lightweight Resolver Library

Traditionally, applications have been linked with a stub resolver library that sends recursive DNS queries to a local caching name server.

At first, IPv6 introduced new complexity into the resolution process, such as following A6 chains and DNAME records, and simultaneous lookup of IPv4 and IPv6 addresses. Though most of the complexity was then removed, these are hard or impossible to implement in a traditional stub resolver.

BIND 9 therefore can also provide resolution services to local clients using a combination of a lightweight resolver library and a resolver daemon process running on the local host. These communicate using a simple UDP-based protocol, the "lightweight resolver protocol," that is distinct from and simpler than the full DNS protocol.

5.2 Running a Resolver Daemon

To use the lightweight resolver interface, the system must run the resolver daemon **lwresd** or a local name server configured with a **lwres** statement.

By default, applications using the lightweight resolver library make UDP requests to the IPv4 loopback address (127.0.0.1) on port 921. The address can be overridden by **lwserver** lines in `/etc/resolv.conf`.

The **lwresd** daemon is essentially a caching-only name server that responds to requests using the lightweight resolver protocol rather than the DNS protocol. Because it needs to run on each host, it is designed to require no or minimal configuration. Unless otherwise instructed, it uses the name servers listed on **nameserver** lines in `/etc/resolv.conf` as forwarders, but is also capable of doing the resolution autonomously if none are specified.

The **lwresd** daemon may also be configured with a `named.conf`-style configuration file, in `/etc/lwresd.conf` by default. A name server may also be configured to act as a lightweight resolver daemon using the **lwres** statement in `named.conf`.

The number of client queries that the **lwresd** daemon serves can be set using the `lwres-tasks` and `lwres-clients` statements in the configuration.

6 BIND 9 Configuration Reference

6.1 Configuration File Elements

Following is a list of elements used throughout the BIND configuration file documentation:

`acl_name`	The name of an `address_match_list` as defined by the **acl** statement.
`address_match_list`	A list of one or more `ip_addr`, `ip_prefix`, `key_id`, or `acl_name` elements; see Section 6.1.
`masters_list`	A named list of one or more `ip_addr` with optional `key_id` and/or `ip_port`. A `masters_list` may include other `masters_list`s.
`domain_name`	A quoted string which is used as a DNS name; for example, `my.test.domain`.
`namelist`	A list of one or more `domain_name` elements.
`dotted_decimal`	One to four integers valued 0 through 255 separated by dots ("."), such as **123.45.67** or **89.123.45.67**.
`ip4_addr`	An IPv4 address with exactly four elements in `dotted_decimal` notation.

ip6_addr	An IPv6 address, such as **2001:db8::1234**. IPv6-scoped addresses that have ambiguity on their scope zones must be disambiguated by an appropriate zone ID with the percent character ("%") as a delimiter. It is strongly recommended to use string zone names rather than numeric identifiers, to be robust against system configuration changes. However, since there is no standard mapping for such names and identifier values, only interface names as link identifiers are supported, assuming one-to-one mapping between interfaces and links. For example, a link-local address **fe80::1** on the link attached to the interface **ne0** can be specified as **fe80::1%ne0**. Note that on most systems link-local addresses always have ambiguity and need to be disambiguated.
ip_addr	An ip4_addr or ip6_addr.
ip_dscp	A number between 0 and 63, used to select a differentiated services code point (DSCP) value for use with outgoing traffic on operating systems that support DSCP.
ip_port	An IP port number. The number is limited to 0 through 65535, with values below 1024 typically restricted to use by processes running as root. In some cases, an asterisk ("*") character can be used as a placeholder to select a random high-numbered port.
ip_prefix	An IP network specified as an ip_addr, followed by a slash ("/") and then the number of bits in the netmask. Trailing zeros in an ip_addr may be omitted. For example, **127/8** is the network **127.0.0.0** with netmask **255.0.0.0** and **1.2.3.0/28** is network **1.2.3.0** with netmask **255.255.255.240**. When specifying a prefix involving a IPv6-scoped address, the scope may be omitted. In that case, the prefix matches packets from any scope.
key_id	A domain_name representing the name of a shared key, to be used for transaction security.
key_list	A list of one or more key_ids, separated by semicolons and ending with a semicolon.
number	A non-negative 32-bit integer (i.e., a number between 0 and 4294967295, inclusive). Its acceptable value might be further limited by the context in which it is used.

fixedpoint	A non-negative real number that can be specified to the nearest one-hundredth. Up to five digits can be specified before a decimal point, and up to two digits after, so the maximum value is 99999.99. Acceptable values might be further limited by the contexts in which they are used.
path_name	A quoted string which is used as a pathname, such as `zones/master/my.test.domain`.
port_list	A list of an `ip_port` or a port range. A port range is specified in the form of **`range`** followed by two `ip_port`s, `port_low` and `port_high`, which represents port numbers from `port_low` through `port_high`, inclusive. `port_low` must not be larger than `port_high`. For example, **`range 1024 65535`** represents ports from 1024 through 65535. In either case an asterisk ("*") character is not allowed as a valid `ip_port`.
size_spec	A 64-bit unsigned integer, or the keywords **`unlimited`** or **`default`**. Integers may take values 0 <= value <= 18446744073709551615, though certain parameters (such as **max-journal-size**) may use a more limited range within these extremes. In most cases, setting a value to 0 does not literally mean zero; it means "undefined" or "as big as possible," depending on the context. See the explanations of particular parameters that use `size_spec` for details on how they interpret its use. Numeric values can optionally be followed by a scaling factor: **`K`** or **`k`** for kilobytes, **`M`** or **`m`** for megabytes, and **`G`** or **`g`** for gigabytes, which scale by 1024, 1024*1024, and 1024*1024*1024 respectively. `unlimited` generally means "as big as possible," and is usually the best way to safely set a very large number. `default` uses the limit that was in force when the server was started.
size_or_percent	A `size_spec` or integer value followed by "%" to represent percent. The behavior is exactly the same as `size_spec`, but `size_or_percent` also allows specifying a positive integer value followed by the "%"" sign to represent percent.
yes_or_no	Either **`yes`** or **`no`**. The words **`true`** and **`false`** are also accepted, as are the numbers **`1`** and **`0`**.

dialup_option	One of **yes**, **no**, **notify**, **notify-passive**, **refresh**, or **passive**. When used in a zone, **notify-passive**, **refresh**, and **passive** are restricted to secondary and stub zones.

Address Match Lists

Syntax

```
address_match_list = address_match_list_element ; ...

address_match_list_element = [ ! ] ( ip_address | ip_prefix |
     key key_id | acl_name | { address_match_list } )
```

Definition and Usage

Address match lists are primarily used to determine access control for various server operations. They are also used in the **listen-on** and **sortlist** statements. The elements which constitute an address match list can be any of the following:

- an IP address (IPv4 or IPv6)
- an IP prefix (in "/" notation)
- a key ID, as defined by the **key** statement
- the name of an address match list defined with the **acl** statement
- a nested address match list enclosed in braces

Elements can be negated with a leading exclamation mark ("!"), and the match list names "any", "none", "localhost", and "localnets" are predefined. More information on those names can be found in the description of the **acl** statement.

The addition of the key clause made the name of this syntactic element something of a misnomer, since security keys can be used to validate access without regard to a host or network address. Nonetheless, the term "address match list" is still used throughout the documentation.

When a given IP address or prefix is compared to an address match list, the comparison takes place in approximately O(1) time. However, key comparisons require that the list of keys be traversed until a matching key is found, and therefore may be somewhat slower.

The interpretation of a match depends on whether the list is being used for access control, defining **listen-on** ports, or in a **sortlist**, and whether the element was negated.

When used as an access control list, a non-negated match allows access and a negated match denies access. If there is no match, access is denied. The clauses **allow-notify**, **allow-recursion**, **allow-recursion-on**, **allow-query**, **allow-query-on**, **allow-query-cache**, **allow-query-cache-on**, **allow-transfer**, **allow-update**, **allow-update-forwarding**, **blackhole**, and **keep-response-order**

all use address match lists. Similarly, the **listen-on** option causes the server to refuse queries on any of the machine's addresses which do not match the list.

Order of insertion is significant. If more than one element in an ACL is found to match a given IP address or prefix, preference is given to the one that came *first* in the ACL definition. Because of this first-match behavior, an element that defines a subset of another element in the list should come before the broader element, regardless of whether either is negated. For example, in **1.2.3/24; ! 1.2.3.13;** the 1.2.3.13 element is completely useless because the algorithm matches any lookup for 1.2.3.13 to the 1.2.3/24 element. Using **! 1.2.3.13; 1.2.3/24** fixes that problem by blocking 1.2.3.13 via the negation, but all other 1.2.3.* hosts pass through.

Comment Syntax

The BIND 9 comment syntax allows comments to appear anywhere that whitespace may appear in a BIND configuration file. To appeal to programmers of all kinds, they can be written in the C, C++, or shell/perl style.

Syntax

```
/* This is a BIND comment as in C */
```

```
// This is a BIND comment as in C++
```

```
# This is a BIND comment as in common Unix shells
# and perl
```

Definition and Usage

Comments may appear anywhere that whitespace may appear in a BIND configuration file.

C-style comments start with the two characters /* (slash, star) and end with */ (star, slash). Because they are completely delimited with these characters, they can be used to comment only a portion of a line or to span multiple lines.

C-style comments cannot be nested. For example, the following is not valid because the entire comment ends with the first */:

```
/* This is the start of a comment.
   This is still part of the comment.
/* This is an incorrect attempt at nesting a comment. */
   This is no longer in any comment. */
```

C++-style comments start with the two characters // (slash, slash) and continue to the end of the physical line. They cannot be continued across multiple physical lines; to have one logical comment span multiple lines, each line must use the // pair. For example:

```
// This is the start of a comment.  The next line
// is a new comment, even though it is logically
// part of the previous comment.
```

Shell-style (or perl-style) comments start with the character # (number sign) and continue to the end of the physical line, as in C++ comments. For example:

```
# This is the start of a comment.  The next line
# is a new comment, even though it is logically
# part of the previous comment.
```

WARNING

The semicolon (";") character cannot start a comment, unlike in a zone file. The semicolon indicates the end of a configuration statement.

6.2 CONFIGURATION FILE GRAMMAR

A BIND 9 configuration consists of statements and comments. Statements end with a semicolon; statements and comments are the only elements that can appear without enclosing braces. Many statements contain a block of sub-statements, which are also terminated with a semicolon.

The following statements are supported:

acl	Defines a named IP address matching list, for access control and other uses.
controls	Declares control channels to be used by the **rndc** utility.
include	Includes a file.
key	Specifies key information for use in authentication and authorization using TSIG.
logging	Specifies what information the server logs and where the log messages are sent.
lwres	Configures **named** to also act as a lightweight resolver daemon (**lwresd**).
masters	Defines a named list of primary servers for inclusion in stub and secondary zones' **masters** or **also-notify** lists.
options	Controls global server configuration options and sets defaults for other statements.
server	Sets certain configuration options on a per-server basis.

statistics-channels	Declares communication channels to get access to **named** statistics.
trusted-keys	Defines trusted DNSSEC keys.
managed-keys	Lists DNSSEC keys to be kept up-to-date using RFC 5011 trust anchor maintenance.
view	Defines a view.
zone	Defines a zone.

The **logging** and **options** statements may only occur once per configuration.

acl Statement Grammar

```
acl string { address_match_element; ... };
```

acl Statement Definition and Usage

The **acl** statement assigns a symbolic name to an address match list. It gets its name from one of the primary uses of address match lists: Access Control Lists (ACLs).

The following ACLs are built-in:

any	Matches all hosts.
none	Matches no hosts.
localhost	Matches the IPv4 and IPv6 addresses of all network interfaces on the system. When addresses are added or removed, the **localhost** ACL element is updated to reflect the changes.
localnets	Matches any host on an IPv4 or IPv6 network for which the system has an interface. When addresses are added or removed, the **localnets** ACL element is updated to reflect the changes. Some systems do not provide a way to determine the prefix lengths of local IPv6 addresses; in such cases, **localnets** only matches the local IPv6 addresses, just like **localhost**.

controls Statement Grammar

```
controls {
  inet ( ipv4_address | ipv6_address |
```

```
      * ) [ port ( integer | * ) ] allow
      { address_match_element; ... } [
      keys { string; ... } ] [ read-only
      boolean ];
  unix quoted_string perm integer
      owner integer group integer [
      keys { string; ... } ] [ read-only
      boolean ];
};
```

controls Statement Definition and Usage

The **controls** statement declares control channels to be used by system administrators to manage the operation of the name server. These control channels are used by the **rndc** utility to send commands to and retrieve non-DNS results from a name server.

An **inet** control channel is a TCP socket listening at the specified **ip_port** on the specified **ip_addr**, which can be an IPv4 or IPv6 address. An **ip_addr** of * (asterisk) is interpreted as the IPv4 wildcard address; connections are accepted on any of the system's IPv4 addresses. To listen on the IPv6 wildcard address, use an **ip_addr** of **::**. If **rndc** is only used on the local host, using the loopback address (`127.0.0.1` or **::1**) is recommended for maximum security.

If no port is specified, port 953 is used. The asterisk "*" cannot be used for **ip_port**.

The ability to issue commands over the control channel is restricted by the **allow** and **keys** clauses. Connections to the control channel are permitted based on the **address_match_list**. This is for simple IP address-based filtering only; any **key_id** elements of the **address_match_list** are ignored.

A **unix** control channel is a Unix domain socket listening at the specified path in the file system. Access to the socket is specified by the **perm**, **owner**, and **group** clauses. Note on some platforms (SunOS and Solaris), the permissions (**perm**) are applied to the parent directory as the permissions on the socket itself are ignored.

The primary authorization mechanism of the command channel is the **key_list**, which contains a list of **key_id**s. Each **key_id** in the **key_list** is authorized to execute commands over the control channel. See Remote Name Daemon Control application in Section 3.3) for information about configuring keys in **rndc**.

If the **read-only** clause is enabled, the control channel is limited to the following set of read-only commands: **nta -dump**, **null**, **status**, **showzone**, **testgen**, and **zonestatus**. By default, **read-only** is not enabled and the control channel allows read-write access.

If no **controls** statement is present, **named** sets up a default control channel listening on the loopback address 127.0.0.1 and its IPv6 counterpart ::1. In this case, and also when the **controls** statement is present but does not have a **keys** clause, **named** attempts to load the command channel key from the file `rndc.key` in `/etc` (or whatever `sysconfdir` was specified when BIND was built). To create an `rndc.key` file, run **rndc-confgen -a**.

The key name and the size of the secret cannot be easily changed; if it is desirable to change those things, make a `rndc.conf` with a custom key. The `rndc.key` file also has its permissions set such that only the owner of the file (the user that **named** is running as) can access it. For greater

flexibility in allowing other users to access **rndc** commands, create a `rndc.conf` file and make it group-readable by a group that contains the users who should have access.

To disable the command channel, use an empty **controls** statement: **controls** { };.

include Statement Grammar

```
include filename;
```

include Statement Definition and Usage

The **include** statement inserts the specified file at the point where the **include** statement is encountered. The **include** statement facilitates the administration of configuration files by permitting the reading or writing of some things but not others. For example, the statement could include private keys that are readable only by the name server.

key Statement Grammar

```
key string {
  algorithm string;
  secret string;
};
```

key Statement Definition and Usage

The **key** statement defines a shared secret key for use with TSIG (see Section 4.5) or the command channel (see Section 6.2).

The **key** statement can occur at the top level of the configuration file or inside a **view** statement. Keys defined in top-level **key** statements can be used in all views. Keys intended for use in a **controls** statement (see Section 6.2) must be defined at the top level.

The *key_id*, also known as the key name, is a domain name that uniquely identifies the key. It can be used in a **server** statement to cause requests sent to that server to be signed with this key, or in address match lists to verify that incoming requests have been signed with a key matching this name, algorithm, and secret.

The *algorithm_id* is a string that specifies a security/authentication algorithm. The **named** server supports `hmac-md5`, `hmac-sha1`, `hmac-sha224`, `hmac-sha256`, `hmac-sha384`, and `hmac-sha512` TSIG authentication. Truncated hashes are supported by appending the minimum number of required bits preceded by a dash, e.g., `hmac-sha1-80`. The *secret_string* is the secret to be used by the algorithm, and is treated as a Base64-encoded string.

logging Statement Grammar

```
logging {
  category string { string; ... };
  channel string {
    buffered boolean;
    file quoted_string [ versions ( "unlimited" | integer )
        ] [ size size ];
    null;
    print-category boolean;
    print-severity boolean;
    print-time boolean;
    severity log_severity;
    stderr;
    syslog [ syslog_facility ];
  };
};
```

logging Statement Definition and Usage

The **logging** statement configures a wide variety of logging options for the name server. Its **channel** phrase associates output methods, format options, and severity levels with a name that can then be used with the **category** phrase to select how various classes of messages are logged.

Only one **logging** statement is used to define as many channels and categories as desired. If there is no **logging** statement, the logging configuration is:

```
logging {
     category default { default_syslog; default_debug; };
     category unmatched { null; };
};
```

If **named** is started with the `-L` option, it logs to the specified file at startup, instead of using syslog. In this case the logging configuration is:

```
logging {
     category default { default_logfile; default_debug; };
     category unmatched { null; };
};
```

The logging configuration is only established when the entire configuration file has been parsed. When the server starts up, all logging messages regarding syntax errors in the configuration file go to the default channels, or to standard error if the `-g` option was specified.

The channel Phrase

All log output goes to one or more *channels*; there is no limit to the number of channels that can be created.

Every channel definition must include a destination clause that says whether messages selected for the channel go to a file, go to a particular syslog facility, go to the standard error stream, or are discarded. The definition can optionally also limit the message severity level that is accepted by the channel (the default is **info**), and whether to include a **named**-generated time stamp, the category name, and/or the severity level (the default is not to include any).

The **null** destination clause causes all messages sent to the channel to be discarded; in that case, other options for the channel are meaningless.

The **file** destination clause directs the channel to a disk file. It can include limitations both on how large the file is allowed to become, and on how many versions of the file are saved each time the file is opened.

If the **versions** log file option is used, then **named** retains that many backup versions of the file by renaming them when opening. For example, to keep three old versions of the file `lamers.log`, just before it is opened `lamers.log.1` is renamed to `lamers.log.2`, `lamers.log.0` is renamed to `lamers.log.1`, and `lamers.log` is renamed to `lamers.log.0`. The **versions unlimited** option can be set to not limit the number of versions. If a **size** option is associated with the log file, then renaming is only done when the file being opened exceeds the indicated size. No backup versions are kept by default; any existing log file is simply appended.

The **size** option for files is used to limit log growth. If the file ever exceeds the size, then **named** stops writing to the file unless it also has a **versions** option associated with it. If backup versions are kept, the files are rolled as described above and a new one begun. If there is no **versions** option, no more data is written to the log until some out-of-band mechanism removes or truncates the log to less than the maximum size. The default behavior is not to limit the size of the file.

Here is an example using the **size** and **versions** options:

```
channel an_example_channel {
    file "example.log" versions 3 size 20m;
    print-time yes;
    print-category yes;
};
```

The **syslog** destination clause directs the channel to the system log. Its argument is a syslog facility as described in the **syslog** man page. Known facilities are **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **cron**, **authpriv**, **ftp**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, and **local7**; however, not all facilities are supported on all operating systems. How **syslog** handles messages sent to this facility is described in the **syslog.conf** man page. On a system which uses a very old version of **syslog**, which only uses two arguments to the **openlog()** function, then this clause is silently ignored.

On Windows machines, syslog messages are directed to the EventViewer.

The **severity** clause works like **syslog**'s "priorities," except that they can also be used when writing straight to a file rather than using **syslog**. Messages which are not at least of the severity level given are not selected for the channel; messages of higher severity levels are accepted.

When using **syslog**, the **syslog.conf** priorities also determine what eventually passes through. For example, defining a channel facility and severity as **daemon** and **debug**, but only logging **daemon.warning** via **syslog.conf**, causes messages of severity **info** and **notice** to be dropped. If the situation were reversed, with **named** writing messages of only **warning** or higher, then **syslogd** would print all messages it received from the channel.

The **stderr** destination clause directs the channel to the server's standard error stream. This is intended for use when the server is running as a foreground process, as when debugging a configuration, for example.

The server can supply extensive debugging information when it is in debugging mode. If the server's global debug level is greater than zero, debugging mode is active. The global debug level is set either by starting the **named** server with the `-d` flag followed by a positive integer, or by running **rndc trace**. The global debug level can be set to zero, and debugging mode turned off, by running **rndc notrace**. All debugging messages in the server have a debug level; higher debug levels give more detailed output. Channels that specify a specific debug severity, for example:

```
channel specific_debug_level {
    file "foo";
    severity debug 3;
};
```

get debugging output of level 3 or less any time the server is in debugging mode, regardless of the global debugging level. Channels with **dynamic** severity use the server's global debug level to determine what messages to print.

If **print-time** is set to **yes**, then the date and time are logged. **print-time** may be specified for a **syslog** channel, but is usually unnecessary since **syslog** also logs the date and time. If **print-category** is set to **yes**, then the category of the message is logged as well. Finally, if **print-severity** is set, then the severity level of the message is logged. The **print-** options may be used in any combination, and are always printed in the following order: time, category, severity. Here is an example where all three **print-** options are on:

```
28-Feb-2000 15:05:32.863 general:  notice:  running
```

If **buffered** has been turned on, the output to files is not flushed after each log entry. By default all log messages are flushed.

There are four predefined channels that are used for **named**'s default logging, as follows. If **named** is started with the `-L`, then a fifth channel, **default_logfile**, is added. How they are used is described in Section 6.2.

```
channel default_syslog {
    // send to syslog's daemon facility
    syslog daemon;
    // only send priority info and higher
    severity info;
};

channel default_debug {
    // write to named.run in the working directory
    // Note: stderr is used instead of "named.run" if
    // the server is started with the '-g' option.
    file "named.run";
    // log at the server's current debug level
    severity dynamic;
};

channel default_stderr {
```

```
    // writes to stderr
    stderr;
    // only send priority info and higher
    severity info;
};

channel null {
   // toss anything sent to this channel
   null;
};

channel default_logfile {
    // this channel is only present if named is
    // started with the -L option, whose argument
    // provides the file name
    file "...";
    // log at the server's current debug level
    severity dynamic;
};
```

The **default_debug** channel has the special property that it only produces output when the server's debug level is non-zero. It normally writes to a file called `named.run` in the server's working directory.

For security reasons, when the `-u` command-line option is used, the `named.run` file is created only after **named** has changed to the new UID, and any debug output generated while **named** is starting - and still running as root - is discarded. To capture this output, run the server with the `-L` option to specify a default logfile, or the `-g` option to log to standard error which can be redirected to a file.

Once a channel is defined, it cannot be redefined. The built-in channels cannot be altered directly, but the default logging can be modified by pointing categories at defined channels.

The category Phrase

There are many categories, so desired logs can be sent anywhere while unwanted logs are ignored. If a list of channels is not specified for a category, log messages in that category are sent to the **default** category instead. If no default category is specified, the following "default default" is used:

```
category default { default_syslog; default_debug; };
```

If **named** is started with the `-L` option, the default category is:

```
category default { default_logfile; default_debug; };
```

As an example, let's say a user wants to log security events to a file, but also wants to keep the default logging behavior. They would specify the following:

```
channel my_security_channel {
    file "my_security_file";
    severity info;
```

```
};
category security {
    my_security_channel;
    default_syslog;
    default_debug;
};
```

To discard all messages in a category, specify the **null** channel:

```
category xfer-out { null; };
category notify { null; };
```

The following are the available categories and brief descriptions of the types of log information they contain. More categories may be added in future BIND releases.

client	Processing of client requests.
cname	Name servers that are skipped for being a CNAME rather than A/AAAA records.
config	Configuration file parsing and processing.
database	Messages relating to the databases used internally by the name server to store zone and cache data.
default	Logging options for those categories where no specific configuration has been defined.
delegation-only	Queries that have been forced to NXDOMAIN as the result of a delegation-only zone or a **delegation-only** in a forward, hint, or stub zone declaration.
dispatch	Dispatching of incoming packets to the server modules where they are to be processed.
dnssec	DNSSEC and TSIG protocol processing.
dnstap	The "dnstap" DNS traffic capture system.

edns-disabled	Log queries that have been forced to use plain DNS due to timeouts. This is often due to the remote servers not being RFC 1034-compliant (not always returning FORMERR or similar to EDNS queries and other extensions to the DNS when they are not understood). In other words, this is targeted at servers that fail to respond to DNS queries that they don't understand. Note: the log message can also be due to packet loss. Before reporting servers for non-RFC 1034 compliance they should be re-tested to determine the nature of the non-compliance. This testing should prevent or reduce the number of false-positive reports. Note: eventually **named** will have to stop treating such timeouts as due to RFC 1034 non-compliance and start treating it as plain packet loss. Falsely classifying packet loss as due to RFC 1034 non-compliance impacts DNSSEC validation, which requires EDNS for the DNSSEC records to be returned.
general	Catch-all for many things that still are not classified into categories.
lame-servers	Misconfigurations in remote servers, discovered by BIND 9 when trying to query those servers during resolution.
network	Network operations.
notify	The NOTIFY protocol.

queries	Location where queries should be logged.
	At startup, specifying the category **queries** also enables query logging unless **querylog** option has been specified.
	The query log entry first reports a client object identifier in @0x<hexadecimal-number> format. Next, it reports the client's IP address and port number, and the query name, class, and type. Next, it reports whether the Recursion Desired flag was set (+ if set, - if not set), whether the query was signed (S), whether EDNS was in use along with the EDNS version number (E(#)), whether TCP was used (T), whether DO (DNSSEC Ok) was set (D), whether CD (Checking Disabled) was set (C), whether a valid DNS Server COOKIE was received (V), and whether a DNS COOKIE option without a valid Server COOKIE was present (K). After this, the destination address the query was sent to is reported.
	`client 127.0.0.1#62536 (www.example.com): query: www.example.com IN AAAA +SE`
	`client ::1#62537 (www.example.net): query: www.example.net IN AAAA -SE`
	The first part of this log message, showing the client address/port number and query name, is repeated in all subsequent log messages related to the same query.

query-errors	Information about queries that resulted in some failure.
rate-limit	The start, periodic, and final notices of the rate limiting of a stream of responses are logged at **info** severity in this category. These messages include a hash value of the domain name of the response and the name itself, except when there is insufficient memory to record the name for the final notice. The final notice is normally delayed until about one minute after rate limiting stops. A lack of memory can hurry the final notice, which is indicated by an initial asterisk (*). Various internal events are logged at debug level 1 and higher. Rate limiting of individual requests is logged in the **query-errors** category.
resolver	DNS resolution, such as the recursive lookups performed on behalf of clients by a caching name server.

rpz	Information about errors in response policy zone files, rewritten responses, and, at the highest **debug** levels, mere rewriting attempts.
security	Approval and denial of requests.
spill	Queries that have been terminated, either by dropping or responding with SERVFAIL, as a result of a fetchlimit quota being exceeded.
trust-anchor-telemetry	Trust-anchor-telemetry requests received by **named**.
unmatched	Messages that **named** was unable to determine the class of, or for which there was no matching **view**. A one-line summary is also logged to the **client** category. This category is best sent to a file or stderr; by default it is sent to the **null** channel.
update	Dynamic updates.
update-security	Approval and denial of update requests.
xfer-in	Zone transfers the server is receiving.
xfer-out	Zone transfers the server is sending.

The query-errors Category

The **query-errors** category is used to indicate why and how specific queries resulted in responses which indicate an error. Normally, these messages will be logged at **debug** logging levels; note, however, that if query logging is active, some are logged at **info**. The logging levels are described below:

At **debug** level 1 or higher - or at **info**, when query logging is active - each response with response code SERVFAIL is logged as follows:

```
client 127.0.0.1#61502:  query failed (SERVFAIL) for www.example.com/IN/AAAA
at query.c:3880
```

This means an error resulting in SERVFAIL was detected at line 3880 of source file `query.c`. Log messages of this level are particularly helpful in identifying the cause of SERVFAIL for an authoritative server.

At **debug** level 2 or higher, detailed context information about recursive resolutions that resulted in SERVFAIL is logged. The log message looks like this:

```
fetch completed at resolver.c:2970 for www.example.com/A
in 10.000183: timed out/success [domain:example.com,
referral:2,restart:7,qrysent:8,timeout:5,lame:0,quota:0,neterr:0,
badresp:1,adberr:0,findfail:0,valfail:0]
```

The first part before the colon shows that a recursive resolution for AAAA records of www.example.com completed in 10.000183 seconds, and the final result that led to the SERVFAIL was determined at line 2970 of source file `resolver.c`.

The next part shows the detected final result and the latest result of DNSSEC validation. The latter is always "success" when no validation attempt was made. In this example, this query probably resulted in SERVFAIL because all name servers are down or unreachable, leading to a timeout in 10 seconds. DNSSEC validation was probably not attempted.

The last part, enclosed in square brackets, shows statistics collected for this particular resolution attempt. The `domain` field shows the deepest zone that the resolver reached; it is the zone where the error was finally detected. The meaning of the other fields is summarized in the following table.

`referral`	The number of referrals the resolver received throughout the resolution process. In the above example there are two, which are most likely com and example.com.
`restart`	The number of cycles that the resolver tried remote servers at the `domain` zone. In each cycle, the resolver sends one query (possibly resending it, depending on the response) to each known name server of the `domain` zone.
`qrysent`	The number of queries the resolver sent at the `domain` zone.
`timeout`	The number of timeouts since the resolver received the last response.
`lame`	The number of lame servers the resolver detected at the `domain` zone. A server is detected to be lame either by an invalid response or as a result of lookup in BIND 9's address database (ADB), where lame servers are cached.
`quota`	The number of times the resolver was unable to send a query because it had exceeded the permissible fetch quota for a server.
`neterr`	The number of erroneous results that the resolver encountered in sending queries at the `domain` zone. One common case is when the remote server is unreachable and the resolver receives an "ICMP unreachable" error message.
`badresp`	The number of unexpected responses (other than `lame`) to queries sent by the resolver at the `domain` zone.

adberr	Failures in finding remote server addresses of the domain zone in the ADB. One common case of this is that the remote server's name does not have any address records.
findfail	Failures to resolve remote server addresses. This is a total number of failures throughout the resolution process.
valfail	Failures of DNSSEC validation. Validation failures are counted throughout the resolution process (not limited to the domain zone), but should only happen in domain.

At **debug** level 3 or higher, the same messages as those at **debug** level 1 are logged for errors other than SERVFAIL. Note that negative responses such as NXDOMAIN are not errors, and are not logged at this debug level.

At **debug** level 4 or higher, the detailed context information logged at **debug** level 2 is logged for errors other than SERVFAIL and for negative responses such as NXDOMAIN.

lwres Statement Grammar

This is the grammar of the **lwres** statement in the named.conf file:

```
lwres {
  [ listen-on {
    ( ip_addr [ port ip_port ] [ dscp ip_dscp ] ; )
      ...
    }; ]
  [ view view_name; ]
  [ search { domain_name ; ... }; ]
  [ ndots number; ]
  [ lwres-tasks number; ]
  [ lwres-clients number; ]
};
```

lwres Statement Definition and Usage

The **lwres** statement configures the name server to also act as a lightweight resolver server. (See Section 5.2.) There may be multiple **lwres** statements configuring lightweight resolver servers with different properties.

The **listen-on** statement specifies a list of IPv4 addresses (and ports) that this instance of a lightweight resolver daemon should accept requests on. If no port is specified, port 921 is used. If this statement is omitted, requests are accepted on 127.0.0.1, port 921.

The **view** statement binds this instance of a lightweight resolver daemon to a view in the DNS namespace, so that the response is constructed in the same manner as a normal DNS query

matching this view. If this statement is omitted, the default view is used; if there is no default view, an error is triggered.

The **search** statement is equivalent to the **search** statement in `/etc/resolv.conf`. It provides a list of domains which are appended to relative names in queries.

The **ndots** statement is equivalent to the **ndots** statement in `/etc/resolv.conf`. It indicates the minimum number of dots in a relative domain name that should result in an exact-match lookup before search path elements are appended.

The `lwres-tasks` statement specifies the number of worker threads the lightweight resolver dedicates to serving clients. By default, the number is the same as the number of CPUs on the system; this can be overridden using the `-n` command-line option when starting the server.

The `lwres-clients` statement specifies the number of client objects per thread the lightweight resolver should create to serve client queries. By default, if the lightweight resolver runs as a part of **named**, 256 client objects are created for each task; if it runs as **lwresd**, 1024 client objects are created for each thread. The maximum value is 32768; higher values are silently ignored and the maximum is used instead. Note that setting too high a value may overconsume system resources.

The maximum number of client queries that the lightweight resolver can handle at any one time equals `lwres-tasks` times `lwres-clients`.

masters Statement Grammar

```
masters string [ port integer ] [ dscp
    integer ] { ( masters | ipv4_address [
    port integer ] | ipv6_address [ port
    integer ] ) [ key string ]; ... };
```

masters Statement Definition and Usage

masters lists allow for a common set of primaries to be easily used by multiple stub and secondary zones in their **masters** or **also-notify** lists.

options Statement Grammar

This is the grammar of the **options** statement in the `named.conf` file:

```
options {
  acache-cleaning-interval integer;
  acache-enable boolean;
  additional-from-auth boolean;
  additional-from-cache boolean;
  allow-new-zones boolean;
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
  allow-query-cache { address_match_element; ... };
  allow-query-cache-on { address_match_element; ... };
```

```
allow-query-on { address_match_element; ... };
allow-recursion { address_match_element; ... };
allow-recursion-on { address_match_element; ... };
allow-transfer { address_match_element; ... };
allow-update { address_match_element; ... };
allow-update-forwarding { address_match_element; ... };
also-notify [ port integer ] [ dscp integer ] { ( masters |
    ipv4_address [ port integer ] | ipv6_address [ port
    integer ] ) [ key string ]; ... };
alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
    ] [ dscp integer ];
alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
    * ) ] [ dscp integer ];
answer-cookie boolean;
attach-cache string;
auth-nxdomain boolean; // default changed
auto-dnssec ( allow | maintain | off );
automatic-interface-scan boolean;
avoid-v4-udp-ports { portrange; ... };
avoid-v6-udp-ports { portrange; ... };
bindkeys-file quoted_string;
blackhole { address_match_element; ... };
cache-file quoted_string;
catalog-zones { zone string [ default-masters [ port integer ]
    [ dscp integer ] { ( masters | ipv4_address [ port
    integer ] | ipv6_address [ port integer ] ) [ key
    string ]; ... } ] [ zone-directory quoted_string ] [
    in-memory boolean ] [ min-update-interval integer ]; ... };
check-dup-records ( fail | warn | ignore );
check-integrity boolean;
check-mx ( fail | warn | ignore );
check-mx-cname ( fail | warn | ignore );
check-names ( master | slave | response
    ) ( fail | warn | ignore );
check-sibling boolean;
check-spf ( warn | ignore );
check-srv-cname ( fail | warn | ignore );
check-wildcard boolean;
cleaning-interval integer;
clients-per-query integer;
cookie-algorithm ( aes | sha1 | sha256 | siphash24 );
cookie-secret string;
coresize ( default | unlimited | sizeval );
datasize ( default | unlimited | sizeval );
deny-answer-addresses { address_match_element; ... } [
    except-from { quoted_string; ... } ];
deny-answer-aliases { quoted_string; ... } [ except-from {
    quoted_string; ... } ];
dialup ( notify | notify-passive | passive | refresh | boolean );
directory quoted_string;
disable-algorithms string { string;
    ... };
disable-ds-digests string { string;
```

```
    ... };
disable-empty-zone string;
dns64 netprefix {
  break-dnssec boolean;
  clients { address_match_element; ... };
  exclude { address_match_element; ... };
  mapped { address_match_element; ... };
  recursive-only boolean;
  suffix ipv6_address;
};
dns64-contact string;
dns64-server string;
dnssec-accept-expired boolean;
dnssec-dnskey-kskonly boolean;
dnssec-enable boolean;
dnssec-loadkeys-interval integer;
dnssec-lookaside ( string trust-anchor
    string | auto | no );
dnssec-must-be-secure string boolean;
dnssec-secure-to-insecure boolean;
dnssec-update-mode ( maintain | no-resign );
dnssec-validation ( yes | no | auto );
dnstap { ( all | auth | client | forwarder |
    resolver ) [ ( query | response ) ]; ... };
dnstap-identity ( quoted_string | none |
    hostname );
dnstap-output ( file | unix ) quoted_string;
dnstap-version ( quoted_string | none );
dscp integer;
dual-stack-servers [ port integer ] { ( quoted_string [ port
    integer ] [ dscp integer ] | ipv4_address [ port
    integer ] [ dscp integer ] | ipv6_address [ port
    integer ] [ dscp integer ] ); ... };
dump-file quoted_string;
edns-udp-size integer;
empty-contact string;
empty-server string;
empty-zones-enable boolean;
fetch-quota-params integer fixedpoint fixedpoint fixedpoint;
fetches-per-server integer [ ( drop | fail ) ];
fetches-per-zone integer [ ( drop | fail ) ];
files ( default | unlimited | sizeval );
filter-aaaa { address_match_element; ... };
filter-aaaa-on-v4 ( break-dnssec | boolean );
filter-aaaa-on-v6 ( break-dnssec | boolean );
flush-zones-on-shutdown boolean;
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
    | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
fstrm-set-buffer-hint integer;
fstrm-set-flush-timeout integer;
fstrm-set-input-queue-size integer;
fstrm-set-output-notify-threshold integer;
```

```
fstrm-set-output-queue-model ( mpsc | spsc );
fstrm-set-output-queue-size integer;
fstrm-set-reopen-interval integer;
geoip-directory ( quoted_string | none );
geoip-use-ecs boolean;
heartbeat-interval integer;
hostname ( quoted_string | none );
inline-signing boolean;
interface-interval integer;
ixfr-from-differences ( master | slave | boolean );
keep-response-order { address_match_element; ... };
key-directory quoted_string;
lame-ttl ttlval;
listen-on [ port integer ] [ dscp
    integer ] {
    address_match_element; ... };
listen-on-v6 [ port integer ] [ dscp
    integer ] {
    address_match_element; ... };
lmdb-mapsize sizeval;
lock-file ( quoted_string | none );
managed-keys-directory quoted_string;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
match-mapped-addresses boolean;
max-acache-size ( unlimited | sizeval );
max-cache-size ( default | unlimited | sizeval | percentage );
max-cache-ttl integer;
max-clients-per-query integer;
max-journal-size ( unlimited | sizeval );
max-ncache-ttl integer;
max-records integer;
max-recursion-depth integer;
max-recursion-queries integer;
max-refresh-time integer;
max-retry-time integer;
max-rsa-exponent-size integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
max-udp-size integer;
max-zone-ttl ( unlimited | ttlval );
memstatistics boolean;
memstatistics-file quoted_string;
message-compression boolean;
min-refresh-time integer;
min-retry-time integer;
minimal-any boolean;
minimal-responses ( no-auth | no-auth-recursive | boolean );
multi-master boolean;
no-case-compress { address_match_element; ... };
nocookie-udp-size integer;
```

```
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-rate integer;
notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
    [ dscp integer ];
notify-to-soa boolean;
nta-lifetime ttlval;
nta-recheck ttlval;
nxdomain-redirect string;
pid-file ( quoted_string | none );
port integer;
preferred-glue string;
prefetch integer [ integer ];
provide-ixfr boolean;
query-source ( ( [ address ] ( ipv4_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
querylog boolean;
random-device quoted_string;
rate-limit {
  all-per-second integer;
  errors-per-second integer;
  exempt-clients { address_match_element; ... };
  ipv4-prefix-length integer;
  ipv6-prefix-length integer;
  log-only boolean;
  max-table-size integer;
  min-table-size integer;
  nodata-per-second integer;
  nxdomains-per-second integer;
  qps-scale integer;
  referrals-per-second integer;
  responses-per-second integer;
  slip integer;
  window integer;
};
recursing-file quoted_string;
recursion boolean;
recursive-clients integer;
request-expire boolean;
request-ixfr boolean;
request-nsid boolean;
require-server-cookie boolean;
reserved-sockets integer;
resolver-query-timeout integer;
response-policy { zone string [ log boolean ] [ max-policy-ttl
    integer ] [ policy ( cname | disabled | drop | given | no-op
    | nodata | nxdomain | passthru | tcp-only quoted_string ) ] [
```

```
      recursive-only boolean ]; ... } [ break-dnssec boolean ] [
      max-policy-ttl integer ] [ min-ns-dots integer ] [
      nsip-wait-recurse boolean ] [ qname-wait-recurse boolean ]
      [ recursive-only boolean ];
  root-delegation-only [ exclude { quoted_string; ... } ];
  root-key-sentinel boolean;
  rrset-order { [ class string ] [ type string ] [ name
      quoted_string ] string string; ... };
  secroots-file quoted_string;
  send-cookie boolean;
  serial-query-rate integer;
  serial-update-method ( date | increment | unixtime );
  server-id ( quoted_string | none | hostname );
  servfail-ttl ttlval;
  session-keyalg string;
  session-keyfile ( quoted_string | none );
  session-keyname string;
  sig-signing-nodes integer;
  sig-signing-signatures integer;
  sig-signing-type integer;
  sig-validity-interval integer [ integer ];
  sortlist { address_match_element; ... };
  stacksize ( default | unlimited | sizeval );
  startup-notify-rate integer;
  statistics-file quoted_string;
  tcp-clients integer;
  tcp-listen-queue integer;
  tkey-dhkey quoted_string integer;
  tkey-domain quoted_string;
  tkey-gssapi-credential quoted_string;
  tkey-gssapi-keytab quoted_string;
  transfer-format ( many-answers | one-answer );
  transfer-message-size integer;
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  transfers-in integer;
  transfers-out integer;
  transfers-per-ns integer;
  trust-anchor-telemetry boolean; // experimental
  try-tcp-refresh boolean;
  update-check-ksk boolean;
  use-alt-transfer-source boolean;
  use-v4-udp-ports { portrange; ... };
  use-v6-udp-ports { portrange; ... };
  v6-bias integer;
  version ( quoted_string | none );
  zero-no-soa-ttl boolean;
  zero-no-soa-ttl-cache boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

options Statement Definition and Usage

The **options** statement sets up global options to be used by BIND. This statement may appear only once in a configuration file. If there is no **options** statement, an options block with each option set to its default is used.

attach-cache

This option allows multiple views to share a single cache database. Each view has its own cache database by default, but if multiple views have the same operational policy for name resolution and caching, those views can share a single cache to save memory, and possibly improve resolution efficiency, by using this option.

The **attach-cache** option may also be specified in **view** statements, in which case it overrides the global **attach-cache** option.

The *cache_name* specifies the cache to be shared. When the **named** server configures views which are supposed to share a cache, it creates a cache with the specified name for the first view of these sharing views. The rest of the views simply refer to the already-created cache.

One common configuration to share a cache is to allow all views to share a single cache. This can be done by specifying **attach-cache** as a global option with an arbitrary name.

Another possible operation is to allow a subset of all views to share a cache while the others retain their own caches. For example, if there are three views A, B, and C, and only A and B should share a cache, specify the **attach-cache** option as a view of A (or B)'s option, referring to the other view name:

```
view "A" {
  // this view has its own cache
  ...
};
view "B" {
  // this view refers to A's cache
  attach-cache "A";
};
view "C" {
  // this view has its own cache
  ...
};
```

Views that share a cache must have the same policy on configurable parameters that may affect caching. The current implementation requires the following configurable options be consistent among these views: **check-names**, **cleaning-interval**, **dnssec-accept-expired**, **dnssec-validation**, **max-cache-ttl**, **max-ncache-ttl**, **max-cache-size**, and **zero-no-soa-ttl**.

Note that there may be other parameters that may cause confusion if they are inconsistent for different views that share a single cache. For example, if these views define different sets of forwarders that can return different answers for the same question, sharing the answer does not make sense or could even be harmful. It is administrator's responsibility to ensure that configuration differences in different views do not cause disruption with a shared cache.

directory
This sets the working directory of the server. Any non-absolute pathnames in the configuration file are taken as relative to this directory. The default location for most server output files (e.g., `named.run`) is this directory. If a directory is not specified, the working directory defaults to `"."`, the directory from which the server was started. The directory specified should be an absolute path. It is *strongly recommended* that the directory be writable by the effective user ID of the **named** process.

dnstap
dnstap is a fast, flexible method for capturing and logging DNS traffic. Developed by Robert Edmonds at Farsight Security, Inc., and supported by multiple DNS implementations, **dnstap** uses **libfstrm** (a lightweight high-speed framing library, see https://github.com/-farsightsec/fstrm) to send event payloads which are encoded using Protocol Buffers (**libprotobuf-c**, a mechanism for serializing structured data developed by Google, Inc.; see https://developers.google.com/-protocol-buffers).

To enable **dnstap** at compile time, the **fstrm** and **protobuf-c** libraries must be available, and BIND must be configured with `--enable-dnstap`.

The **dnstap** option is a bracketed list of message types to be logged. These may be set differently for each view. Supported types are `client`, `auth`, `resolver`, and `forwarder`. Specifying type `all` causes all **dnstap** messages to be logged, regardless of type.

Each type may take an additional argument to indicate whether to log `query` messages or `response` messages; if not specified, both queries and responses are logged.

Example: To log all authoritative queries and responses, recursive client responses, and upstream queries sent by the resolver, use:

```
dnstap {
  auth;
  client response;
  resolver query;
};
```

Logged **dnstap** messages can be parsed using the **dnstap-read** utility (see dnstap-read(1) for details).

For more information on **dnstap**, see http://dnstap.info.

The fstrm library has a number of tunables that are exposed in `named.conf`, and can be modified if necessary to improve performance or prevent loss of data. These are:

- **fstrm-set-buffer-hint**: The threshold number of bytes to accumulate in the output buffer before forcing a buffer flush. The minimum is 1024, the maximum is 65536, and the default is 8192.
- **fstrm-set-flush-timeout**: The number of seconds to allow unflushed data to remain in the output buffer. The minimum is 1 second, the maximum is 600 seconds (10 minutes), and the default is 1 second.
- **fstrm-set-output-notify-threshold**: The number of outstanding queue entries to allow on an input queue before waking the I/O thread. The minimum is 1 and the default is 32.
- **fstrm-set-output-queue-model**: The queuing semantics to use for queue objects. The default is `mpsc` (multiple producer, single consumer); the other option is `spsc` (single producer, single consumer).

- **fstrm-set-input-queue-size**: The number of queue entries to allocate for each input queue. This value must be a power of 2. The minimum is 2, the maximum is 16384, and the default is 512.
- **fstrm-set-output-queue-size**: The number of queue entries to allocate for each output queue. The minimum is 2, the maximum is system-dependent and based on `IOV_MAX`, and the default is 64.
- **fstrm-set-reopen-interval**: The number of seconds to wait between attempts to reopen a closed output stream. The minimum is 1 second, the maximum is 600 seconds (10 minutes), and the default is 5 seconds.

Note that all of the above minimum, maximum, and default values are set by the **libfstrm** library, and may be subject to change in future versions of the library. See the **libfstrm** documentation for more information.

dnstap-output
This configures the path to which the **dnstap** frame stream is sent if **dnstap** is enabled at compile time and active.

The first argument is either `file` or `unix`, indicating whether the destination is a file or a Unix domain socket. The second argument is the path of the file or socket. (Note: when using a socket, **dnstap** messages are only sent if another process such as **fstrm_capture** (provided with **libfstrm**) is listening on the socket.)

dnstap-output can only be set globally in **options**. Currently, it can only be set once while **named** is running; once set, it cannot be changed by **rndc reload** or **rndc reconfig**.

dnstap-identity
This specifies an **identity** string to send in **dnstap** messages. If set to `hostname`, which is the default, the server's hostname is sent. If set to `none`, no identity string is sent.

dnstap-version
This specifies a **version** string to send in **dnstap** messages. The default is the version number of the BIND release. If set to `none`, no version string is sent.

geoip-directory
When **named** is compiled using the MaxMind GeoIP2 geolocation API, or the legacy GeoIP API, this specifies the directory containing GeoIP database files. By default, the option is set based on the prefix used to build the **libmaxminddb** module; for example, if the library is installed in `/usr/local/lib`, then the default **geoip-directory** is `/usr/local/share/GeoIP`. On Windows, the default is the **named** working directory. See Section 6.2 for details about **geoip** ACLs.

key-directory
This is the directory where the public and private DNSSEC key files should be found when performing a dynamic update of secure zones, if different than the current working directory. (Note that this option has no effect on the paths for files containing non-DNSSEC keys such as `bind.keys`, `rndc.key`, or `session.key`.)

lmdb-mapsize
When **named** is built with liblmdb, this option sets a maximum size for the memory map of the new-zone database (NZD) in LMDB database format. This database is used to store configuration information for zones added using **rndc addzone**. Note that this is not the NZD database file size, but the largest size that the database may grow to.

Because the database file is memory mapped, its size is limited by the address space of the **named** process. The default of 32 megabytes was chosen to be usable with 32-bit **named** builds. The largest permitted value is 1 terabyte. Given typical zone configurations without elaborate ACLs, a 32 MB NZD file ought to be able to hold configurations of about 100,000 zones.

managed-keys-directory
This specifies the directory in which to store the files that track managed DNSSEC keys. By default, this is the working directory. The directory *must* be writable by the effective user ID of the **named** process.

If **named** is not configured to use views, managed keys for the server are tracked in a single file called `managed-keys.bind`. Otherwise, managed keys are tracked in separate files, one file per view; each file name is the view name (or, if it contains characters that are incompatible with use as a file name, the SHA256 hash of the view name), followed by the extension `.mkeys`.

(Note: in earlier releases, file names for views always used the SHA256 hash of the view name. To ensure compatibility after upgrading, if a file using the old name format is found to exist, it is used instead of the new format.)

named-xfer
This option is obsolete. In BIND 9, no separate **named-xfer** program is needed; its functionality is built into the name server.

tkey-gssapi-keytab
This is the KRB5 keytab file to use for GSS-TSIG updates. If this option is set and tkey-gssapi-credential is not set, updates are allowed with any key matching a principal in the specified keytab.

tkey-gssapi-credential
This is the security credential with which the server should authenticate keys requested by the GSS-TSIG protocol. Currently only Kerberos 5 authentication is available; the credential is a Kerberos principal which the server can acquire through the default system key file, normally `/etc/krb5.keytab`. The location of the keytab file can be overridden using the **tkey-gssapi-keytab** option. Normally this principal is of the form "**DNS/**`server.domain`". To use GSS-TSIG, **tkey-domain** must also be set if a specific keytab is not set with **tkey-gssapi-keytab**.

tkey-domain
This domain is appended to the names of all shared keys generated with **TKEY**. When a client requests a **TKEY** exchange, it may or may not specify the desired name for the key. If present, the name of the shared key is `client-specified part` + `tkey-domain`. Otherwise, the name of the shared key is `random hex digits` + `tkey-domain`. In most cases, the **domainname** should be the server's domain name, or an otherwise nonexistent subdomain like "_tkey.`domainname`". If using GSS-TSIG, this variable must be defined, unless a specific keytab is specified using **tkey-gssapi-keytab**.

tkey-dhkey
This is the Diffie-Hellman key used by the server to generate shared keys with clients using the Diffie-Hellman mode of **TKEY**. The server must be able to load the public and private keys from files in the working directory. In most cases, the `key_name` should be the server's host name.

cache-file
This is for testing only. Do not use.

dump-file
This is the pathname of the file the server dumps the database to, when instructed to do so with **rndc dumpdb**. If not specified, the default is `named_dump.db`.

memstatistics-file
This is the pathname of the file the server writes memory usage statistics to on exit. If not specified, the default is `named.memstats`.

lock-file
This is the pathname of a file on which **named** attempts to acquire a file lock when starting for the first time; if unsuccessful, the server terminates, under the assumption that another server is already running. If not specified, the default is `none`.

Specifying **lock-file none** disables the use of a lock file. **lock-file** is ignored if **named** was run using the `-X` option, which overrides it. Changes to **lock-file** are ignored if **named** is being reloaded or reconfigured; it is only effective when the server is first started.

pid-file
This is the pathname of the file the server writes its process ID in. If not specified, the default is `/var/run/named/named.pid`. The PID file is used by programs that send signals to the running name server. Specifying **pid-file none** disables the use of a PID file; no file is written and any existing one is removed. Note that **none** is a keyword, not a filename, and therefore is not enclosed in double quotes.

recursing-file
This is the pathname of the file where the server dumps the queries that are currently recursing, when instructed to do so with **rndc recursing**. If not specified, the default is `named.recursing`.

statistics-file
This is the pathname of the file the server appends statistics to, when instructed to do so using **rndc stats**. If not specified, the default is `named.stats` in the server's current directory. The format of the file is described in Section 6.4.

bindkeys-file
This is the pathname of a file to override the built-in trusted keys provided by **named**. See the discussion of **dnssec-validation** for details. If not specified, the default is `/etc/bind.keys`.

secroots-file
This is the pathname of the file the server dumps security roots to, when instructed to do so with **rndc secroots**. If not specified, the default is `named.secroots`.

session-keyfile
This is the pathname of the file into which to write a TSIG session key generated by **named** for use by **nsupdate -l**. If not specified, the default is `/var/run/named/session.key`. (See Section 6.2, and in particular the discussion of the **update-policy** statement's **`local`** option for more information about this feature.)

session-keyname
This is the key name to use for the TSIG session key. If not specified, the default is `local-ddns`.

session-keyalg
This is the algorithm to use for the TSIG session key. Valid values are hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, hmac-sha512, and hmac-md5. If not specified, the default is hmac-sha256.

port
This is the UDP/TCP port number the server uses to receive and send DNS protocol traffic. The default is 53. This option is mainly intended for server testing; a server using a port other than 53 is not able to communicate with the global DNS.

dscp
This is the global Differentiated Services Code Point (DSCP) value to classify outgoing DNS traffic, on operating systems that support DSCP. Valid values are 0 through 63. It is not configured by default.

random-device
This specifies a source of entropy to be used by the server. Entropy is primarily needed for DNSSEC operations, such as TKEY transactions and dynamic update of signed zones. This option specifies the device (or file) from which to read entropy. If it is a file, operations requiring entropy will fail when the file has been exhausted. If **random-device** is not specified, the default value is `/dev/random` (or equivalent) when present, and none otherwise. The **random-device** option takes effect during the initial configuration load at server startup time and is ignored on subsequent reloads.

preferred-glue
If specified, the listed type (A or AAAA) is emitted before other glue in the additional section of a query response. The default is to prefer A records when responding to queries that arrived via IPv4 and AAAA when responding to queries that arrived via IPv6.

root-delegation-only
This turns on enforcement of delegation-only in TLDs (top-level domains) and root zones with an optional exclude list.

DS queries are expected to be made to and be answered by delegation-only zones. Such queries and responses are treated as an exception to delegation-only processing and are not converted to NXDOMAIN responses, provided a CNAME is not discovered at the query name.

If a delegation-only zone server also serves a child zone, it is not always possible to determine whether an answer comes from the delegation-only zone or the child zone. SOA NS and DNSKEY records are apex-only records and a matching response that contains these records or DS is treated as coming from a child zone. RRSIG records are also examined to see if they are signed by a child zone, and the authority section is examined to see if there is evidence that the answer is from the child zone. Answers that are determined to be from a child zone are not converted to NXDOMAIN responses. Despite all these checks, there is still a possibility of false negatives when a child zone is being served.

Similarly, false positives can arise from empty nodes (no records at the name) in the delegation-only zone when the query type is not `ANY`.

Note that some TLDs are not delegation-only; e.g., "DE", "LV", "US", and "MUSEUM". This list is not exhaustive.

```
options {
        root-delegation-only exclude { "de"; "lv"; "us"; "museum"; };
};
```

disable-algorithms
This disables the specified DNSSEC algorithms at and below the specified name. Multiple **disable-algorithms** statements are allowed. Only the best-match **disable-algorithms** clause is used to determine the algorithms.

If all supported algorithms are disabled, the zones covered by the **disable-algorithms** setting are treated as insecure.

Configured trust anchors in **trusted-keys** or **managed-keys** that match a disabled algorithm are ignored and treated as if they were not configured.

disable-ds-digests
This disables the specified DS digest types at and below the specified name. Multiple **disable-ds-digests** statements are allowed. Only the best-match **disable-ds-digests** clause is used to determine the digest types.

If all supported digest types are disabled, the zones covered by **disable-ds-digests** are treated as insecure.

dnssec-lookaside
When set, **dnssec-lookaside** provides the validator with an alternate method to validate DNSKEY records at the top of a zone. When a DNSKEY is at or below a domain specified by the deepest **dnssec-lookaside**, and the normal DNSSEC validation has left the key untrusted, the trust-anchor is appended to the key name and a DLV record is looked up to see if it can validate the key. If the DLV record validates a DNSKEY (similarly to the way a DS record does), the DNSKEY RRset is deemed to be trusted.

If **dnssec-lookaside** is set to **no**, then **dnssec-lookaside** is not used.

Note: the ISC-provided DLV service at `dlv.isc.org` has been shut down. The **dnssec-lookaside auto;** configuration option, which set **named** to use ISC DLV with minimal configuration, has accordingly been removed.

dnssec-must-be-secure
This specifies hierarchies which must be or may not be secure (signed and validated). If **yes**, then **named** only accepts answers if they are secure. If **no**, then normal DNSSEC validation applies, allowing insecure answers to be accepted. The specified domain must be under a **trusted-keys** or **managed-keys** statement, or **dnssec-validation auto** must be active.

dns64
This directive instructs **named** to return mapped IPv4 addresses to AAAA queries when there are no AAAA records. It is intended to be used in conjunction with a NAT64. Each **dns64** defines one DNS64 prefix. Multiple DNS64 prefixes can be defined.

Compatible IPv6 prefixes have lengths of 32, 40, 48, 56, 64, and 96, per RFC 6052. Bits 64..71 inclusive must be zero, with the most significant bit of the prefix in position 0.

In addition, a reverse IP6.ARPA zone is created for the prefix to provide a mapping from the IP6.ARPA names to the corresponding IN-ADDR.ARPA names using synthesized CNAMEs. **dns64-server** and **dns64-contact** can be used to specify the name of the server and contact for the zones. These can be set at the view/options level but not on a per-prefix basis.

Each **dns64** supports an optional **clients** ACL that determines which clients are affected by this directive. If not defined, it defaults to **any;**.

Each **dns64** supports an optional **mapped** ACL that selects which IPv4 addresses are to be mapped in the corresponding A RRset. If not defined, it defaults to **any;**.

Normally, DNS64 does not apply to a domain name that owns one or more AAAA records; these records are simply returned. The optional **exclude** ACL allows specification of a list of IPv6 addresses that are ignored if they appear in a domain name's AAAA records; DNS64 is applied to any A records the domain name owns. If not defined, **exclude** defaults to ::ffff:0.0.0.0/96.

A optional **suffix** can also be defined to set the bits trailing the mapped IPv4 address bits. By default these bits are set to `::`. The bits matching the prefix and mapped IPv4 address must be zero.

If **recursive-only** is set to **yes**, the DNS64 synthesis only happens for recursive queries. The default is **no**.

If **break-dnssec** is set to **yes**, the DNS64 synthesis happens even if the result, if validated, would cause a DNSSEC validation failure. If this option is set to **no** (the default), the DO is set on the incoming query, and there are RRSIGs on the applicable records, then synthesis does not happen.

```
acl rfc1918 { 10/8; 192.168/16; 172.16/12; };

dns64 64:FF9B::/96 {
        clients { any; };
        mapped { !rfc1918; any; };
        exclude { 64:FF9B::/96; ::ffff:0000:0000/96; };
        suffix ::;
};
```

dnssec-loadkeys-interval

When a zone is configured with **auto-dnssec maintain;**, its key repository must be checked periodically to see if any new keys have been added or any existing keys' timing metadata has been updated (see dnssec-keygen(8) and dnssec-settime(8)). The **dnssec-loadkeys-interval** option sets the frequency of automatic repository checks, in minutes. The default is `60` (1 hour), the minimum is `1` (1 minute), and the maximum is `1440` (24 hours); any higher value is silently reduced.

dnssec-update-mode

If this option is set to its default value of `maintain` in a zone of type `master` which is DNSSEC-signed and configured to allow dynamic updates (see Section 6.2), and if **named** has access to the private signing key(s) for the zone, then **named** automatically signs all new or changed records and maintains signatures for the zone by regenerating RRSIG records whenever they approach their expiration date.

If the option is changed to `no-resign`, then **named** signs all new or changed records, but scheduled maintenance of signatures is disabled.

With either of these settings, **named** rejects updates to a DNSSEC-signed zone when the signing keys are inactive or unavailable to **named**. (A planned third option, `external`, will disable all automatic signing and allow DNSSEC data to be submitted into a zone via dynamic update; this is not yet implemented.)

nta-lifetime
This specifies the default lifetime, in seconds, for negative trust anchors added via **rndc nta**.

A negative trust anchor selectively disables DNSSEC validation for zones that are known to be failing because of misconfiguration, rather than an attack. When data to be validated is at or below an active NTA (and above any other configured trust anchors), **named** aborts the DNSSEC validation process and treats the data as insecure rather than bogus. This continues until the NTA's lifetime is elapsed. NTAs persist across **named** restarts.

For convenience, TTL-style time-unit suffixes can be used to specify the NTA lifetime in seconds, minutes, or hours. `nta-lifetime` defaults to one hour; it cannot exceed one week.

nta-recheck
This specifies how often to check whether negative trust anchors added via **rndc nta** are still necessary.

A negative trust anchor is normally used when a domain has stopped validating due to operator error; it temporarily disables DNSSEC validation for that domain. In the interest of ensuring that DNSSEC validation is turned back on as soon as possible, **named** periodically sends a query to the domain, ignoring negative trust anchors, to find out whether it can now be validated. If so, the negative trust anchor is allowed to expire early.

Validity checks can be disabled for an individual NTA by using **rndc nta -f**, or for all NTAs by setting `nta-recheck` to zero.

For convenience, TTL-style time-unit suffixes can be used to specify the NTA recheck interval in seconds, minutes, or hours. The default is five minutes. It cannot be longer than `nta-lifetime`, which cannot be longer than a week.

max-zone-ttl
This specifies a maximum permissible TTL value in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the maximum value. When loading a zone file using a `masterfile-format` of `text` or `raw`, any record encountered with a TTL higher than `max-zone-ttl` causes the zone to be rejected.

This is useful in DNSSEC-signed zones because when rolling to a new DNSKEY, the old key needs to remain available until RRSIG records have expired from caches. The `max-zone-ttl` option guarantees that the largest TTL in the zone is no higher than the set value.

(Note: because `map`-format files load directly into memory, this option cannot be used with them.)

The default value is `unlimited`. A `max-zone-ttl` of zero is treated as `unlimited`.

serial-update-method
Zones configured for dynamic DNS may use this option to set the update method to be used for the zone serial number in the SOA record.

With the default setting of **serial-update-method increment;**, the SOA serial number is incremented by one each time the zone is updated.

When set to **serial-update-method unixtime;**, the SOA serial number is set to the number of seconds since the Unix epoch, unless the serial number is already greater than or equal to that value, in which case it is simply incremented by one.

When set to **serial-update-method date;**, the new SOA serial number is the current date in the form "YYYYMMDD", followed by two zeroes, unless the existing serial number is already greater than or equal to that value, in which case it is incremented by one.

zone-statistics
If **full**, the server collects statistical data on all zones, unless specifically turned off on a per-zone basis by specifying **zone-statistics terse** or **zone-statistics none** in the **zone** statement. The default is **terse**, providing minimal statistics on zones (including name and current serial number, but not query type counters).

These statistics may be accessed via the **statistics-channel** or using **rndc stats**, which dumps them to the file listed in the **statistics-file**. See also Section 6.4.

For backward compatibility with earlier versions of BIND 9, the **zone-statistics** option can also accept **yes** or **no**; **yes** has the same meaning as **full**. As of BIND 9.10, **no** has the same meaning as **none**; previously, it was the same as **terse**.

Boolean Options

automatic-interface-scan
If **yes** and supported by the operating system, this automatically rescans network interfaces when the interface addresses are added or removed. The default is **yes**. This configuration option does not affect the time-based **interface-interval** option; it is recommended to set the time-based **interface-interval** to 0 when the operator confirms that automatic interface scanning is supported by the operating system.

The **automatic-interface-scan** implementation uses routing sockets for the network interface discovery; therefore, the operating system must support the routing sockets for this feature to work.

allow-new-zones
If **yes**, then zones can be added at runtime via **rndc addzone**. The default is **no**.

Newly added zones' configuration parameters are stored so that they can persist after the server is restarted. The configuration information is saved in a file called `viewname.nzf` (or, if **named** is compiled with liblmdb, in an LMDB database file called `viewname.nzd`). `viewname` is the name of the view, unless the view name contains characters that are incompatible with use as a file name, in which case a cryptographic hash of the view name is used instead.

Configurations for zones added at runtime are stored either in a new-zone file (NZF) or a new-zone database (NZD), depending on whether **named** was linked with liblmdb at compile time. See rndc(8) for further details about **rndc addzone**.

auth-nxdomain
If **yes**, then the **AA** bit is always set on NXDOMAIN responses, even if the server is not actually authoritative. The default is **no**.

deallocate-on-exit
This option was used in BIND 8 to enable checking for memory leaks on exit. BIND 9 ignores the option and always performs the checks.

memstatistics
This writes memory statistics to the file specified by **memstatistics-file** at exit. The default is **no** unless **-m record** is specified on the command line, in which case it is **yes**.

dialup
If **yes**, then the server treats all zones as if they are doing zone transfers across a dial-on-demand dialup link, which can be brought up by traffic originating from this server. Although this setting has different effects according to zone type, it concentrates the zone maintenance so that everything happens quickly, once every **heartbeat-interval**, ideally during a single call. It also suppresses some normal zone maintenance traffic. The default is **no**.

If specified in the **view** and **zone** statements, the **dialup** option overrides the global **dialup** option.

If the zone is a primary zone, the server sends out a NOTIFY request to all the secondaries (default). This should trigger the zone serial number check in the secondary (providing it supports NOTIFY), allowing the secondary to verify the zone while the connection is active. The set of servers to which NOTIFY is sent can be controlled by **notify** and **also-notify**.

If the zone is a secondary or stub zone, the server suppresses the regular "zone up to date" (refresh) queries and only performs them when the **heartbeat-interval** expires, in addition to sending NOTIFY requests.

Finer control can be achieved by using **notify**, which only sends NOTIFY messages; **notify-passive**, which sends NOTIFY messages and suppresses the normal refresh queries; **refresh**, which suppresses normal refresh processing and sends refresh queries when the **heartbeat-interval** expires; and **passive**, which disables normal refresh processing.

dialup mode	normal refresh	heart-beat refresh	heart-beat notify
no (default)	yes	no	no
yes	no	yes	yes
notify	yes	no	yes
refresh	no	yes	no
passive	no	no	no
notify-passive	no	no	yes

Note that normal NOTIFY processing is not affected by **dialup**.

fake-iquery

In BIND 8, this option enabled simulating the obsolete DNS query type IQUERY. BIND 9 never does IQUERY simulation.

fetch-glue
This option is obsolete. In BIND 8, **fetch-glue yes** caused the server to attempt to fetch glue resource records it did not have when constructing the additional data section of a response. This is now considered a bad idea and BIND 9 never does it.

flush-zones-on-shutdown
When the nameserver exits upon receiving SIGTERM, flush or do not flush any pending zone writes. The default is **flush-zones-on-shutdown no**.

geoip-use-ecs
When BIND is compiled with GeoIP support and configured with "geoip" ACL elements, this option indicates whether the EDNS Client Subnet option, if present in a request, should be used for matching against the GeoIP database. The default is **geoip-use-ecs yes**.

has-old-clients
This option was incorrectly implemented in BIND 8, and is ignored by BIND 9. To achieve the intended effect of **has-old-clients yes**, specify the two separate options **auth-nxdomain yes** and **rfc2308-type1 no** instead.

host-statistics
In BIND 8, this enabled keeping of statistics for every host that the name server interacts with. It is not implemented in BIND 9.

root-key-sentinel
If **yes**, respond to root key sentinel probes as described in draft-ietf-dnsop-kskroll-sentinel-08. The default is **yes**.

maintain-ixfr-base
This option is obsolete. It was used in BIND 8 to determine whether a transaction log was kept for Incremental Zone Transfer. BIND 9 maintains a transaction log whenever possible. To disable outgoing incremental zone transfers, use **provide-ixfr no**.

message-compression
If **yes**, DNS name compression is used in responses to regular queries (not including AXFR or IXFR, which always use compression). Setting this option to **no** reduces CPU usage on servers and may improve throughput. However, it increases response size, which may cause more queries to be processed using TCP; a server with compression disabled is out of compliance with RFC 1123 Section 6.1.3.2. The default is **yes**.

minimal-responses
If set to **yes**, then when generating responses the server only adds records to the authority and additional data sections when they are required (e.g. delegations, negative responses). This may improve the performance of the server.

When set to **no-auth**, the server omits records from the authority section unless they are required, but it may still add records to the additional section. When set to **no-auth-recursive**, this is only done if the query is recursive. These settings are useful when answering stub clients, which usually ignore the authority section. **no-auth-recursive** is designed for mixed-mode servers that handle both authoritative and recursive queries.

The default is **no**.

minimal-any
If set to **yes**, the server replies with only one of the RRsets for the query name, and its covering RRSIGs if any, when generating a positive response to a query of type ANY over UDP, instead of replying with all known RRsets for the name. Similarly, a query for type RRSIG is answered with the RRSIG records covering only one type. This can reduce the impact of some kinds of attack traffic, without harming legitimate clients. (Note, however, that the RRset returned is the first one found in the database; it is not necessarily the smallest available RRset.) Additionally, `minimal-responses` is turned on for these queries, so no unnecessary records are added to the authority or additional sections. The default is **no**.

multiple-cnames
This option was used in BIND 8 to allow a domain name to have multiple CNAME records, in violation of the DNS standards. BIND 9.2 onwards always strictly enforces the CNAME rules both in primary files and dynamic updates.

notify
If **yes** (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes; see Section 4.1. The messages are sent to the servers listed in the zone's NS records (except the primary server identified in the SOA MNAME field), and to any servers listed in the **also-notify** option.

If **master-only**, notifies are only sent for primary zones. If **explicit**, notifies are sent only to servers explicitly listed using **also-notify**. If **no**, no notifies are sent.

The **notify** option may also be specified in the **zone** statement, in which case it overrides the **options notify** statement. It would only be necessary to turn off this option if it caused secondary zones to crash.

notify-to-soa
If **yes**, do not check the name servers in the NS RRset against the SOA MNAME. Normally a NOTIFY message is not sent to the SOA MNAME (SOA ORIGIN), as it is supposed to contain the name of the ultimate primary server. Sometimes, however, a secondary server is listed as the SOA MNAME in hidden primary configurations; in that case, the ultimate primary should be set to still send NOTIFY messages to all the name servers listed in the NS RRset.

recursion
If **yes**, and a DNS query requests recursion, then the server attempts to do all the work required to answer the query. If recursion is off and the server does not already know the answer, it returns a referral response. The default is **yes**. Note that setting **recursion no** does not prevent clients from getting data from the server's cache; it only prevents new data from being cached as an effect of client queries. Caching may still occur as an effect the server's internal operation, such as NOTIFY address lookups.

request-nsid
If **yes**, then an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution. If the authoritative server returns an NSID option in its response, then its contents are logged in the **resolver** category at level **info**. The default is **no**.

request-sit
This experimental option is obsolete.

require-server-cookie
If **yes**, require a valid server cookie before sending a full response to a UDP request from a cookie-aware client. BADCOOKIE is sent if there is a bad or nonexistent server cookie. The default is **no**.

Users wishing to test that DNS COOKIE clients correctly handle BADCOOKIE, or who are getting a lot of forged DNS requests with DNS COOKIES present, should set this to **yes**. Setting this to **yes** results in a reduced amplification effect in a reflection attack, as the BADCOOKIE response is smaller than a full response, while also requiring a legitimate client to follow up with a second query with the new, valid, cookie.

answer-cookie
When set to the default value of **yes**, COOKIE EDNS options are sent when applicable in replies to client queries. If set to **no**, COOKIE EDNS options are not sent in replies. This can only be set at the global options level, not per-view.

answer-cookie no is only intended as a temporary measure, for use when **named** shares an IP address with other servers that do not yet support DNS COOKIE. A mismatch between servers on the same address is not expected to cause operational problems, but the option to disable COOKIE responses so that all servers have the same behavior is provided out of an abundance of caution. DNS COOKIE is an important security mechanism, and should not be disabled unless absolutely necessary.

send-cookie
If **yes**, then a COOKIE EDNS option is sent along with the query. If the resolver has previously communicated with the server, the COOKIE returned in the previous transaction is sent. This is used by the server to determine whether the resolver has talked to it before. A resolver sending the correct COOKIE is assumed not to be an off-path attacker sending a spoofed-source query; the query is therefore unlikely to be part of a reflection/amplification attack, so resolvers sending a correct COOKIE option are not subject to response rate limiting (RRL). Resolvers which do not send a correct COOKIE option may be limited to receiving smaller responses via the **nocookie-udp-size** option. The default is **yes**.

nocookie-udp-size
This sets the maximum size of UDP responses that are sent to queries without a valid server COOKIE. A value below 128 is silently raised to 128. The default value is 4096, but the **max-udp-size** option may further limit the response size as the default for **max-udp-size** is 1232.

sit-secret
This experimental option is obsolete.

cookie-algorithm
This sets the algorithm to be used when generating the server cookie; the options are "aes", "sha1", or "sha256". The default is "aes" if supported by the cryptographic library; otherwise, "sha256".

cookie-secret
If set, this is a shared secret used for generating and verifying EDNS COOKIE options within an anycast cluster. If not set, the system generates a random secret at startup. The shared secret is encoded as a hex string and needs to be 128 bits for AES128, 160 bits for SHA1, and 256 bits for SHA256.

If there are multiple secrets specified, the first one listed in `named.conf` is used to generate new server cookies. The others are only used to verify returned cookies.

rfc2308-type1
Setting this to **yes** causes the server to send NS records along with the SOA record for negative answers. The default is **no**.

> NOTE
>
> This is not yet implemented in BIND 9.

trust-anchor-telemetry
This causes **named** to send specially formed queries once per day to domains for which trust anchors have been configured via **trusted-keys**, **managed-keys**, or **dnssec-validation auto**.

The query name used for these queries has the form "_ta-xxxx(-xxxx)(...)".<domain>, where each "xxxx" is a group of four hexadecimal digits representing the key ID of a trusted DNSSEC key. The key IDs for each domain are sorted smallest to largest prior to encoding. The query type is NULL.

By monitoring these queries, zone operators are able to see which resolvers have been updated to trust a new key; this may help them decide when it is safe to remove an old one.

The default is **yes**.

use-id-pool
This option is obsolete. BIND 9 always allocates query IDs from a pool.

use-ixfr
This option is obsolete. To disable IXFR to a particular server or servers, see the information on the **provide-ixfr** option in Section 6.2. See also Section 4.3.

provide-ixfr
See the description of **provide-ixfr** in Section 6.2.

request-ixfr
See the description of **request-ixfr** in Section 6.2.

request-expire
See the description of **request-expire** in Section 6.2.

treat-cr-as-space
This option was used in BIND 8 to make the server treat carriage return ("**r**") characters the same way as a space or tab character, to facilitate loading of zone files on a Unix system that were generated on an NT or DOS machine. In BIND 9, both UNIX "**n**" and NT/DOS "**r****n**" newlines are always accepted, and the option is ignored.

additional-from-auth, additional-from-cache

These options control the behavior of an authoritative server when answering queries which have additional data, or when following CNAME and DNAME chains.

When both of these options are set to **yes** (the default) and a query is being answered from authoritative data (a zone configured into the server), the additional data section of the reply is filled in using data from other authoritative zones and from the cache. In some situations this is undesirable, such as when there is concern over the correctness of the cache, or in servers where secondary zones may be added and modified by untrusted third parties. Also, avoiding the search for this additional data speeds up server operations at the possible expense of additional queries to resolve what would otherwise be provided in the additional section.

For example, if a query asks for an MX record for host `foo.example.com`, and the record found is "`MX 10 mail.example.net`", normally the address records (A and AAAA) for `mail.example.net` are provided as well, if known, even though they are not in the example.com zone. Setting these options to **no** disables this behavior and makes the server only search for additional data in the zone it answers from.

These options are intended for use in authoritative-only servers, or in authoritative-only views. Attempts to set them to **no** without also specifying **recursion no** will cause the server to ignore the options and log a warning message.

Specifying **additional-from-cache no** actually disables the use of the cache not only for additional data lookups but also when looking up the answer. This is usually the desired behavior in an authoritative-only server where the correctness of the cached data is an issue.

When a name server is non-recursively queried for a name that is not below the apex of any served zone, it normally answers with an "upwards referral" to the root servers or the servers of some other known parent of the query name. Since the data in an upwards referral comes from the cache, the server is not able to provide upwards referrals when **additional-from-cache no** has been specified. Instead, it responds to such queries with REFUSED. This should not cause any problems since upwards referrals are not required for the resolution process.

match-mapped-addresses

If **yes**, then an IPv4-mapped IPv6 address matches any address-match list entries that match the corresponding IPv4 address.

This option was introduced to work around a kernel quirk in some operating systems that causes IPv4 TCP connections, such as zone transfers, to be accepted on an IPv6 socket using mapped addresses. This caused address-match lists designed for IPv4 to fail to match. However, **named** now solves this problem internally. The use of this option is discouraged.

filter-aaaa-on-v4

This option is only available when BIND 9 is compiled with the **--enable-filter-aaaa** option on the "configure" command line. It is intended to help the transition from IPv4 to IPv6 by not giving IPv6 addresses to DNS clients unless they have connections to the IPv6 Internet. This is not recommended unless absolutely necessary. The default is **no**. The **filter-aaaa-on-v4** option may also be specified in **view** statements to override the global **filter-aaaa-on-v4** option.

If **yes**, the DNS client is at an IPv4 address, in **filter-aaaa**, and if the response does not include DNSSEC signatures, then all AAAA records are deleted from the response. This filtering applies to all responses and not only authoritative responses.

If **break-dnssec**, then AAAA records are deleted even when DNSSEC is enabled. As suggested by the name, this causes the response to not verify, because the DNSSEC protocol is designed to detect deletions.

This mechanism can erroneously cause other servers to not give AAAA records to their clients. A recursing server with both IPv6 and IPv4 network connections, that queries an authoritative server using this mechanism via IPv4, is denied AAAA records even if its client is using IPv6.

This mechanism is applied to authoritative as well as non-authoritative records. A client using IPv4 that is not allowed recursion can erroneously be given AAAA records because the server is not allowed to check for A records.

Some AAAA records are given to IPv4 clients in glue records. IPv4 clients that are servers can then erroneously answer requests for AAAA records received via IPv4.

filter-aaaa-on-v6
This is identical to **filter-aaaa-on-v4**, except it filters AAAA responses to queries from IPv6 clients instead of IPv4 clients. To filter all responses, set both options to **yes**.

ixfr-from-differences
When **yes** and the server loads a new version of a primary zone from its zone file or receives a new version of a secondary file via zone transfer, it compares the new version to the previous one and calculates a set of differences. The differences are then logged in the zone's journal file so that the changes can be transmitted to downstream secondaries as an incremental zone transfer.

By allowing incremental zone transfers to be used for non-dynamic zones, this option saves bandwidth at the expense of increased CPU and memory consumption at the primary server. In particular, if the new version of a zone is completely different from the previous one, the set of differences is of a size comparable to the combined size of the old and new zone versions, and the server needs to temporarily allocate memory to hold this complete difference set.

ixfr-from-differences also accepts **master** and **slave** at the view and options levels, which causes **ixfr-from-differences** to be enabled for all primary or secondary zones, respectively. It is off by default.

Note: if inline signing is enabled for a zone, the user-provided **ixfr-from-differences** setting is ignored for that zone.

multi-master
This should be set when there are multiple primary servers for a zone and the addresses refer to different machines. If **yes**, **named** does not log when the serial number on the primary is less than what **named** currently has. The default is **no**.

auto-dnssec
Zones configured for dynamic DNS may use this option to allow varying levels of automatic DNSSEC key management. There are three possible settings:

auto-dnssec allow; permits keys to be updated and the zone fully re-signed whenever the user issues the command **rndc sign *zonename***.

auto-dnssec maintain; includes the above, but also automatically adjusts the zone's DNSSEC keys on a schedule, according to the keys' timing metadata (see dnssec-keygen(8) and dnssec-settime(8)). The command **rndc sign** ***zonename*** causes **named** to load keys from the key repository and sign the zone with all keys that are active. **rndc loadkeys** ***zonename*** causes **named** to load keys from the key repository and schedule key maintenance events to occur in the future, but it does not sign the full zone immediately. Note: once keys have been loaded for a zone the first time, the repository is searched for changes periodically, regardless of whether **rndc loadkeys** is used. The recheck interval is defined by **dnssec-loadkeys-interval**.)

The default setting is **auto-dnssec off**.

dnssec-enable
This indicates whether DNSSEC-related resource records are to be returned by **named**. If set to **no**, **named** does not return DNSSEC-related resource records unless specifically queried for. The default is **yes**.

dnssec-validation
This option enables DNSSEC validation in **named**. Note that **dnssec-enable** also needs to be set to **yes** to be effective. If set to **no**, DNSSEC validation is disabled.

If set to **auto**, DNSSEC validation is enabled and a default trust anchor for the DNS root zone is used. If set to **yes**, DNSSEC validation is enabled, but a trust anchor must be manually configured using a **trusted-keys** or **managed-keys** statement. The default is **yes**.

The default root trust anchor is stored in the file `bind.keys`. **named** loads that key at startup if **dnssec-validation** is set to `auto`. A copy of the file is installed along with BIND 9, and is current as of the release date. If the root key expires, a new copy of `bind.keys` can be downloaded from https://www.isc.org/bind-keys.

(To prevent problems if `bind.keys` is not found, the current trust anchor is also compiled in to **named**. Relying on this is not recommended, however, as it requires **named** to be recompiled with a new key when the root key expires.)

NOTE

named loads *only* the root key from `bind.keys`. The file cannot be used to store keys for other zones. The root key in `bind.keys` is ignored if **dnssec-validation auto** is not in use.

Whenever the resolver sends out queries to an EDNS-compliant server, it always sets the DO bit indicating it can support DNSSEC responses, even if **dnssec-validation** is off.

dnssec-accept-expired
This accepts expired signatures when verifying DNSSEC signatures. The default is **no**. Setting this option to **yes** leaves **named** vulnerable to replay attacks.

querylog
Query logging provides a complete log of all incoming queries and all query errors. This provides more insight into the server's activity, but with a cost to performance which may be significant on heavily loaded servers.

The **querylog** option specifies whether query logging should be active when **named** first starts. If **querylog** is not specified, then query logging is determined by the presence of the logging category **queries**. Query logging can also be activated at runtime using the command **rndc querylog on**, or deactivated with **rndc querylog off**.

check-names
This option is used to restrict the character set and syntax of certain domain names in zone files and/or DNS responses received from the network. The default varies according to usage area. For primary zones (i.e., **type master**), the default is **fail**. For secondary zones (**type slave**), the default is **warn**. For answers received from the network (**response**), the default is **ignore**.

The rules for legal hostnames and mail domains are derived from RFC 952 and RFC 821 as modified by RFC 1123.

check-names applies to the owner names of A, AAAA, and MX records. It also applies to the domain names in the RDATA of NS, SOA, MX, and SRV records. It further applies to the RDATA of PTR records where the owner name indicates that it is a reverse lookup of a hostname (the owner name ends in IN-ADDR.ARPA, IP6.ARPA, or IP6.INT).

check-dup-records
This checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS. The default is to **warn**. Other possible values are **fail** and **ignore**.

check-mx
This checks whether the MX record appears to refer to a IP address. The default is to **warn**. Other possible values are **fail** and **ignore**.

check-wildcard
This option is used to check for non-terminal wildcards. The use of non-terminal wildcards is almost always as a result of a failure to understand the wildcard matching algorithm (RFC 1034). This option affects primary zones. The default (**yes**) is to check for non-terminal wildcards and issue a warning.

check-integrity
This performs post-load zone integrity checks on primary zones. It checks that MX and SRV records refer to address (A or AAAA) records and that glue address records exist for delegated zones. For MX and SRV records, only in-zone hostnames are checked (for out-of-zone hostnames, use **named-checkzone**). For NS records, only names below top-of-zone are checked (for out-of-zone names and glue consistency checks, use **named-checkzone**). The default is **yes**.

The use of the SPF record to publish Sender Policy Framework is deprecated, as the migration from using TXT records to SPF records was abandoned. Enabling this option also checks that a TXT Sender Policy Framework record exists (starts with "v=spf1") if there is an SPF record. Warnings are emitted if the TXT record does not exist; they can be suppressed with **check-spf**.

check-mx-cname
If **check-integrity** is set, then fail, warn, or ignore MX records that refer to CNAMES. The default is to **warn**.

check-srv-cname
If **check-integrity** is set, then fail, warn, or ignore SRV records that refer to CNAMES. The default is to **warn**.

check-sibling
When performing integrity checks, also check that sibling glue exists. The default is **yes**.

check-spf
If **check-integrity** is set, check that there is a TXT Sender Policy Framework record present (starts with "v=spf1") if there is an SPF record present. The default is **warn**.

zero-no-soa-ttl
If **yes**, when returning authoritative negative responses to SOA queries, set the TTL of the SOA record returned in the authority section to zero. The default is **yes**.

zero-no-soa-ttl-cache
If **yes**, when caching a negative response to an SOA query set the TTL to zero. The default is **no**.

update-check-ksk
When set to the default value of `yes`, check the KSK bit in each key to determine how the key should be used when generating RRSIGs for a secure zone.

Ordinarily, zone-signing keys (that is, keys without the KSK bit set) are used to sign the entire zone, while key-signing keys (keys with the KSK bit set) are only used to sign the DNSKEY RRset at the zone apex. However, if this option is set to `no`, then the KSK bit is ignored; KSKs are treated as if they were ZSKs and are used to sign the entire zone. This is similar to the **dnssec-signzone -z** command-line option.

When this option is set to `yes`, there must be at least two active keys for every algorithm represented in the DNSKEY RRset: at least one KSK and one ZSK per algorithm. If there is any algorithm for which this requirement is not met, this option is ignored for that algorithm.

dnssec-dnskey-kskonly
When this option and **update-check-ksk** are both set to `yes`, only key-signing keys (that is, keys with the KSK bit set) are used to sign the DNSKEY RRset at the zone apex. Zone-signing keys (keys without the KSK bit set) are used to sign the remainder of the zone, but not the DNSKEY RRset. This is similar to the **dnssec-signzone -x** command-line option.

The default is **no**. If **update-check-ksk** is set to `no`, this option is ignored.

try-tcp-refresh
If **yes**, try to refresh the zone using TCP if UDP queries fail. The default is **yes**.

dnssec-secure-to-insecure
This allows a dynamic zone to transition from secure to insecure (i.e., signed to unsigned) by deleting all of the DNSKEY records. The default is **no**. If set to **yes**, and if the DNSKEY RRset at the zone apex is deleted, all RRSIG and NSEC records are removed from the zone as well.

If the zone uses NSEC3, it is also necessary to delete the NSEC3PARAM RRset from the zone apex; this causes the removal of all corresponding NSEC3 records. (It is expected that this requirement will be eliminated in a future release.)

Note that if a zone has been configured with **auto-dnssec maintain** and the private keys remain accessible in the key repository, then the zone will be automatically signed again the next time **named** is started.

Forwarding

The forwarding facility can be used to create a large site-wide cache on a few servers, reducing traffic over links to external name servers. It can also be used to allow queries by servers that do not have direct access to the Internet, but wish to look up exterior names anyway. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.

forward
This option is only meaningful if the forwarders list is not empty. A value of `first` is the default and causes the server to query the forwarders first; if that does not answer the question, the server then looks for the answer itself. If `only` is specified, the server only queries the forwarders.

forwarders
This specifies a list of IP addresses to which queries are forwarded. The default is the empty list (no forwarding). Each address in the list can be associated with an optional port number and/or DSCP value, and a default port number and DSCP value can be set for the entire list.

Forwarding can also be configured on a per-domain basis, allowing for the global forwarding options to be overridden in a variety of ways. Particular domains can be set to use different forwarders, or have a different **forward only/first** behavior, or not forward at all; see Section 6.2.

Dual-stack Servers

Dual-stack servers are used as servers of last resort, to work around problems in reachability due the lack of support for either IPv4 or IPv6 on the host machine.

dual-stack-servers
This specifies host names or addresses of machines with access to both IPv4 and IPv6 transports. If a hostname is used, the server must be able to resolve the name using only the transport it has. If the machine is dual-stacked, the **dual-stack-servers** parameter has no effect unless access to a transport has been disabled on the command line (e.g., **named -4**).

Access Control

Access to the server can be restricted based on the IP address of the requesting system. See Section 6.1 for details on how to specify IP address lists.

allow-notify
This ACL specifies which hosts are allowed to notify this secondary server of zone changes in addition to the zone primaries. **allow-notify** may also be specified in the **zone** statement, in which case it overrides the **options allow-notify** statement. It is only meaningful for a secondary zone. If not specified, the default is to process notify messages only from a zone's primary.

allow-query
This specifies which hosts are allowed to ask ordinary DNS questions. **allow-query** may also be specified in the **zone** statement, in which case it overrides the **options allow-query** statement. If not specified, the default is to allow queries from all hosts.

NOTE

allow-query-cache is used to specify access to the cache.

allow-query-on
This specifies which local addresses can accept ordinary DNS questions. This makes it possible, for instance, to allow queries on internal-facing interfaces but disallow them on external-facing ones, without necessarily knowing the internal network's addresses.

Note that **allow-query-on** is only checked for queries that are permitted by **allow-query**. A query must be allowed by both ACLs, or it is refused.

allow-query-on may also be specified in the **zone** statement, in which case it overrides the **options allow-query-on** statement.

If not specified, the default is to allow queries on all addresses.

NOTE

allow-query-cache is used to specify access to the cache.

allow-query-cache
This specifies which hosts are allowed to get answers from the cache. If **allow-query-cache** is not set, BIND checks to see if the following parameters are set, in order: **allow-recursion** and **allow-query** (unless **recursion no;** is set, in which case **none;** is used). If neither of those parameters is set, the default (**localnets; localhost;**) is used.

allow-query-cache-on
This specifies which local addresses can send answers from the cache. If not specified, the default is to allow cache queries on any address, **localnets**, and **localhost**.

allow-recursion
This specifies which hosts are allowed to make recursive queries through this server. BIND checks to see if the following parameters are set, in order: **allow-recursion**, **allow-query-cache**, and **allow-query**. If none of those parameters are set, the default (**localnets; localhost;**) is used.

allow-recursion-on
This specifies which local addresses can accept recursive queries. If not specified, the default is to allow recursive queries on all addresses.

allow-update
This specifies which hosts are allowed to submit Dynamic DNS updates for primary zones. The default is to deny updates from all hosts. Note that allowing updates based on the requestor's IP address is insecure; see Section 7.3 for details.

allow-update-forwarding
This specifies which hosts are allowed to submit Dynamic DNS updates to secondary zones to be forwarded to the primary. The default is **`{ none; }`**, which means that no update forwarding is performed. To enable update forwarding, specify **`allow-update-forwarding { any; };`**. Specifying values other than **`{ none; }`** or **`{ any; }`** is usually counterproductive; the responsibility for update access control should rest with the primary server, not the secondaries.

Note that enabling the update forwarding feature on a secondary server may expose primary servers to attacks if they rely on insecure IP-address-based access control; see Section 7.3 for more details.

allow-v6-synthesis
This option was introduced for the smooth transition from AAAA to A6 and from "nibble labels" to binary labels. However, since both A6 and binary labels were then deprecated, this option was also deprecated. It is now ignored with some warning messages.

allow-transfer
This specifies which hosts are allowed to receive zone transfers from the server. **allow-transfer** may also be specified in the **zone** statement, in which case it overrides the **options allow-transfer** statement. If not specified, the default is to allow transfers to all hosts.

blackhole
This specifies a list of addresses which the server does accept queries from or use to resolve a query. Queries from these addresses are not responded to. The default is **`none`**.

filter-aaaa
This specifies a list of addresses to which **filter-aaaa-on-v4** and **filter-aaaa-on-v6** apply. The default is **`any`**.

keep-response-order
This specifies a list of addresses to which the server sends responses to TCP queries, in the same order in which they were received. This disables the processing of TCP queries in parallel. The default is **`none`**.

no-case-compress
This specifies a list of addresses which require responses to use case-insensitive compression. This ACL can be used when **named** needs to work with clients that do not comply

with the requirement in RFC 1034 to use case-insensitive name comparisons when checking for matching domain names.

If left undefined, the ACL defaults to **none**: case-insensitive compression is used for all clients. If the ACL is defined and matches a client, case is ignored when compressing domain names in DNS responses sent to that client.

This can result in slightly smaller responses; if a response contains the names "example.com" and "example.COM", case-insensitive compression treats the second one as a duplicate. It also ensures that the case of the query name exactly matches the case of the owner names of returned records, rather than matches the case of the records entered in the zone file. This allows responses to exactly match the query, which is required by some clients due to incorrect use of case-sensitive comparisons.

Case-insensitive compression is *always* used in AXFR and IXFR responses, regardless of whether the client matches this ACL.

There are circumstances in which **named** does not preserve the case of owner names of records: if a zone file defines records of different types with the same name, but the capitalization of the name is different (e.g., "www.example.com/A" and "WWW.EXAMPLE.COM/AAAA"), then all responses for that name use the *first* version of the name that was used in the zone file. This limitation may be addressed in a future release. However, domain names specified in the rdata of resource records (i.e., records of type NS, MX, CNAME, etc.) always have their case preserved unless the client matches this ACL.

resolver-query-timeout
This is the amount of time in seconds that the resolver spends attempting to resolve a recursive query before failing. The default and minimum is `10` and the maximum is `30`. Setting it to `0` results in the default being used.

Interfaces

The interfaces and ports that the server answers queries from may be specified using the **listen-on** option. **listen-on** takes an optional port and an `address_match_list` of IPv4 addresses. (IPv6 addresses are ignored, with a logged warning.) The server listens on all interfaces allowed by the address match list. If a port is not specified, port 53 is used.

Multiple **listen-on** statements are allowed. For example:

```
listen-on { 5.6.7.8; };
listen-on port 1234 { !1.2.3.4; 1.2/16; };
```

enables the name server on port 53 for the IP address 5.6.7.8, and on port 1234 of an address on the machine in net 1.2 that is not 1.2.3.4.

If no **listen-on** is specified, the server listens on port 53 on all IPv4 interfaces.

The **listen-on-v6** option is used to specify the interfaces and the ports on which the server listens for incoming queries sent using IPv6. If not specified, the server listens on port 53 on all IPv6 interfaces.

When

```
{ any; }
```

is specified as the `address_match_list` for the **listen-on-v6** option, the server does not bind a separate socket to each IPv6 interface address as it does for IPv4, if the operating system has enough API support for IPv6 (specifically, if it conforms to RFC 3493 and RFC 3542). Instead, it listens on the IPv6 wildcard address. If the system only has incomplete API support for IPv6, however, the behavior is the same as that for IPv4.

A list of particular IPv6 addresses can also be specified, in which case the server listens on a separate socket for each specified address, regardless of whether the desired API is supported by the system. IPv4 addresses specified in **listen-on-v6** are ignored, with a logged warning.

Multiple **listen-on-v6** options can be used. For example:

```
listen-on-v6 { any; };
listen-on-v6 port 1234 { !2001:db8::/32; any; };
```

enables the name server on port 53 for any IPv6 addresses (with a single wildcard socket), and on port 1234 of IPv6 addresses that are not in the prefix 2001:db8::/32 (with separate sockets for each matched address).

To instruct the server not to listen on any IPv6 address, use:

```
listen-on-v6 { none; };
```

Query Address

If the server does not know the answer to a question, it queries other name servers. **query-source** specifies the address and port used for such queries. For queries sent over IPv6, there is a separate **query-source-v6** option. If **address** is * (asterisk) or is omitted, a wildcard IP address (**INADDR_ANY**) is used.

If **port** is * or is omitted, a random port number from a pre-configured range is picked up and used for each query. The port range(s) is specified in the **use-v4-udp-ports** (for IPv4) and **use-v6-udp-ports** (for IPv6) options, excluding the ranges specified in the **avoid-v4-udp-ports** and **avoid-v6-udp-ports** options, respectively.

The defaults of the **query-source** and **query-source-v6** options are:

```
query-source address * port *;
query-source-v6 address * port *;
```

If **use-v4-udp-ports** or **use-v6-udp-ports** is unspecified, **named** checks whether the operating system provides a programming interface to retrieve the system's default range for ephemeral ports. If such an interface is available, **named** uses the corresponding system default range; otherwise, it uses its own defaults:

```
use-v4-udp-ports { range 1024 65535; };
use-v6-udp-ports { range 1024 65535; };
```

Note: make sure the ranges are sufficiently large for security. A desirable size depends on several parameters, but we generally recommend it contain at least 16384 ports (14 bits of entropy). Note also that the system's default range when used may be too small for this purpose, and that the range may even be changed while **named** is running; the new range is automatically

applied when **named** is reloaded. Explicit configuration of **use-v4-udp-ports** and **use-v6-udp-ports** is encouraged, so that the ranges are sufficiently large and are reasonably independent from the ranges used by other applications.

Note: the operational configuration where **named** runs may prohibit the use of some ports. For example, Unix systems do not allow **named**, if run without root privilege, to use ports less than 1024. If such ports are included in the specified (or detected) set of query ports, the corresponding query attempts will fail, resulting in resolution failures or delay. It is therefore important to configure the set of ports that can be safely used in the expected operational environment.

The defaults of the **avoid-v4-udp-ports** and **avoid-v6-udp-ports** options are:

```
avoid-v4-udp-ports {};
avoid-v6-udp-ports {};
```

Note: BIND 9.5.0 introduced the **use-queryport-pool** option to support a pool of such random ports, but this option is now obsolete because reusing the same ports in the pool may not be sufficiently secure. For the same reason, it is generally strongly discouraged to specify a particular port for the **query-source** or **query-source-v6** options; it implicitly disables the use of randomized port numbers.

use-queryport-pool
This option is obsolete.

queryport-pool-ports
This option is obsolete.

queryport-pool-updateinterval
This option is obsolete.

NOTE

The address specified in the **query-source** option is used for both UDP and TCP queries, but the port applies only to UDP queries. TCP queries always use a random unprivileged port.

NOTE

Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

NOTE

See also **transfer-source** and **notify-source**.

Zone Transfers

BIND has mechanisms in place to facilitate zone transfers and set limits on the amount of load that transfers place on the system. The following options apply to zone transfers.

also-notify
This option defines a global list of IP addresses of name servers that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded, in addition to the servers listed in the zone's NS records. This helps to ensure that copies of the zones quickly converge on stealth servers. Optionally, a port may be specified with each **also-notify** address to send the notify messages to a port other than the default of 53. An optional TSIG key can also be specified with each address to cause the notify messages to be signed; this can be useful when sending notifies to multiple views. In place of explicit addresses, one or more named **masters** lists can be used.

If an **also-notify** list is given in a **zone** statement, it overrides the **options also-notify** statement. When a **zone notify** statement is set to **no**, the IP addresses in the global **also-notify** list are not sent NOTIFY messages for that zone. The default is the empty list (no global notification list).

max-transfer-time-in
Inbound zone transfers running longer than this many minutes are terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

max-transfer-idle-in
Inbound zone transfers making no progress in this many minutes are terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

max-transfer-time-out
Outbound zone transfers running longer than this many minutes are terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

max-transfer-idle-out
Outbound zone transfers making no progress in this many minutes are terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

notify-rate
This specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations. (NOTIFY requests due to initial zone loading are subject to a separate rate limit; see below.) The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

startup-notify-rate
This is the rate at which NOTIFY requests are sent when the name server is first starting up, or when zones have been newly added to the name server. The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

serial-query-rate
Secondary servers periodically query primary servers to find out if zone serial numbers have changed. Each such query uses a minute amount of the secondary server's network bandwidth. To limit the amount of bandwidth used, BIND 9 limits the rate at which queries are sent. The value of the **serial-query-rate** option, an integer, is the maximum number of queries sent per second. The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

serial-queries
BIND 9 does not limit the number of outstanding serial queries and ignores the **serial-queries** option. Instead, it limits the rate at which the queries are sent as defined using the **serial-query-rate** option.

transfer-format
Zone transfers can be sent using two different formats, **one-answer** and **many-answers**. The **transfer-format** option is used on the primary server to determine which format it sends. **one-answer** uses one DNS message per resource record transferred. **many-answers** packs as many resource records as possible into one message. **many-answers** is more efficient; the default is **many-answers**. The **many-answers** format is also supported by recent Microsoft Windows name servers. **transfer-format** may be overridden on a per-server basis by using the **server** statement.

transfer-message-size
This is an upper bound on the uncompressed size of DNS messages used in zone transfers over TCP. If a message grows larger than this size, additional messages are used to complete the zone transfer. (Note, however, that this is a hint, not a hard limit; if a message contains a single resource record whose RDATA does not fit within the size limit, a larger message will be permitted so the record can be transferred.)

Valid values are between 512 and 65535 octets; any values outside that range are adjusted to the nearest value within it. The default is `20480`, which was selected to improve message compression; most DNS messages of this size will compress to less than 16536 bytes. Larger messages cannot be compressed as effectively, because 16536 is the largest permissible compression offset pointer in a DNS message.

This option is mainly intended for server testing; there is rarely any benefit in setting a value other than the default.

transfers-in
This is the maximum number of inbound zone transfers that can run concurrently. The default value is `10`. Increasing **transfers-in** may speed up the convergence of secondary zones, but it also may increase the load on the local system.

transfers-out
This is the maximum number of outbound zone transfers that can run concurrently. Zone transfer requests in excess of the limit are refused. The default value is `10`.

transfers-per-ns
This is the maximum number of inbound zone transfers that can concurrently transfer from a given remote name server. The default value is 2. Increasing **transfers-per-ns** may speed up the convergence of secondary zones, but it also may increase the load on the remote name server. **transfers-per-ns** may be overridden on a per-server basis by using the **transfers** phrase of the **server** statement.

transfer-source
transfer-source determines which local address is bound to IPv4 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv4 address, and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. If not set, it defaults to a system-controlled value which is usually the address of the interface "closest to" the remote end. This address must appear in the remote end's **allow-transfer** option for the zone being transferred, if one is specified. This statement sets the **transfer-source** for all zones, but can be overridden on a per-view or per-zone basis by including a **transfer-source** statement within the **view** or **zone** block in the configuration file.

> NOTE
>
> Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

transfer-source-v6
This option is the same as **transfer-source**, except zone transfers are performed using IPv6.

alt-transfer-source
This indicates an alternate transfer source if the one listed in **transfer-source** fails and **use-alt-transfer-source** is set.

> NOTE
>
> To avoid using the alternate transfer source, set **use-alt-transfer-source** appropriately and do not depend upon getting an answer back to the first refresh query.

alt-transfer-source-v6
This indicates an alternate transfer source if the one listed in **transfer-source-v6** fails and **use-alt-transfer-source** is set.

use-alt-transfer-source
This indicates whether the alternate transfer sources should be used. If views are specified, this defaults to **no**; otherwise, it defaults to **yes**.

notify-source
notify-source determines which local source address, and optionally UDP port, is used to send NOTIFY messages. This address must appear in the secondary server's **masters** zone clause or in an **allow-notify** clause. This statement sets the **notify-source** for all zones, but can be overridden on a per-zone or per-view basis by including a **notify-source** statement within the **zone** or **view** block in the configuration file.

> NOTE
>
> Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

notify-source-v6
This option acts like **notify-source**, but applies to notify messages sent to IPv6 addresses.

UDP Port Lists

use-v4-udp-ports, **avoid-v4-udp-ports**, **use-v6-udp-ports**, and **avoid-v6-udp-ports** specify a list of IPv4 and IPv6 UDP ports that are or are not used as source ports for UDP messages. See Section 6.2 about how the available ports are determined. For example, with the following configuration:

```
use-v6-udp-ports { range 32768 65535; };
avoid-v6-udp-ports { 40000; range 50000 60000; };
```

UDP ports of IPv6 messages sent from **named** are in one of the following ranges: 32768 to 39999, 40001 to 49999, and 60001 to 65535.

avoid-v4-udp-ports and **avoid-v6-udp-ports** can be used to prevent **named** from choosing as its random source port a port that is blocked by a firewall or a port that is used by other applications; if a query went out with a source port blocked by a firewall, the answer would not pass through the firewall and the name server would have to query again. Note: the desired range can also be represented only with **use-v4-udp-ports** and **use-v6-udp-ports**, and the **avoid-** options are redundant in that sense; they are provided for backward compatibility and to possibly simplify the port specification.

Operating System Resource Limits

The server's usage of many system resources can be limited. Scaled values are allowed when specifying resource limits. For example, **1G** can be used instead of **1073741824** to specify a limit of one gigabyte. **unlimited** requests unlimited use, or the maximum available amount. **default** uses the limit that was in force when the server was started. See the description of **size_spec** in Section 6.1.

The following options set operating system resource limits for the name server process. Some operating systems do not support some or any of the limits; on such systems, a warning is issued if an unsupported limit is used.

coresize
This sets the maximum size of a core dump. The default is `default`.

datasize
This sets the maximum amount of data memory the server may use. The default is `default`. This is a hard limit on server memory usage; if the server attempts to allocate memory in excess of this limit, the allocation will fail, which may in turn leave the server unable to perform DNS service. Therefore, this option is rarely useful as a way to limit the amount of memory used by the server, but it can be used to raise an operating system data size limit that is too small by default. To limit the amount of memory used by the server, use the **max-cache-size** and **recursive-clients** options instead.

files
This sets the maximum number of files the server may have open concurrently. The default is `unlimited`.

stacksize
This sets the maximum amount of stack memory the server may use. The default is `default`.

Server Resource Limits

The following options set limits on the server's resource consumption that are enforced internally by the server rather than by the operating system.

max-ixfr-log-size
This option is obsolete; it is accepted and ignored for BIND 8 compatibility. The option **max-journal-size** performs a similar function in BIND 9.

max-journal-size
This sets a maximum size for each journal file (see Section 4.2). When the journal file approaches the specified size, some of the oldest transactions in the journal are automatically removed. The largest permitted value is 2 gigabytes. The default is `unlimited`, which also means 2 gigabytes. This option may also be set on a per-zone basis.

max-records
This sets the maximum number of records permitted in a zone. The default is zero, which means the maximum is unlimited.

host-statistics-max
In BIND 8, this specified the maximum number of host statistics entries to be kept. It is not implemented in BIND 9.

recursive-clients
This sets the maximum number (a "hard quota") of simultaneous recursive lookups the server performs on behalf of clients. The default is `1000`. Because each recursing client uses a fair bit of memory (on the order of 20 kilobytes), the value of the **recursive-clients** option may have to be decreased on hosts with limited memory.

`recursive-clients` defines a "hard quota" limit for pending recursive clients; when more clients than this are pending, new incoming requests are not accepted, and for each incoming request a previous pending request is dropped.

A "soft quota" is also set. When this lower quota is exceeded, incoming requests are accepted, but for each one, a pending request is dropped. If `recursive-clients` is greater than 1000, the soft quota is set to `recursive-clients` minus 100; otherwise it is set to 90% of `recursive-clients`.

tcp-clients
This is the maximum number of simultaneous client TCP connections that the server accepts. The default is `150`.

clients-per-query, max-clients-per-query
These set the initial value (minimum) and maximum number of recursive simultaneous clients for any given query (<qname,qtype,qclass>) that the server accepts before dropping additional clients. **named** attempts to self-tune this value and changes are logged. The default values are 10 and 100.

This value should reflect how many queries come in for a given name in the time it takes to resolve that name. If the number of queries exceeds this value, **named** assumes that it is dealing with a non-responsive zone and drops additional queries. If it gets a response after dropping queries, it raises the estimate. The estimate is then lowered in 20 minutes if it has remained unchanged.

If **clients-per-query** is set to zero, there is no limit on the number of clients per query and no queries are dropped.

If **max-clients-per-query** is set to zero, there is no upper bound other than imposed by **recursive-clients**.

fetches-per-zone
This sets the maximum number of simultaneous iterative queries to any one domain that the server permits before blocking new queries for data in or beneath that zone. This value should reflect how many fetches would normally be sent to any one zone in the time it would take to resolve them. It should be smaller than `recursive-clients`.

When many clients simultaneously query for the same name and type, the clients are all attached to the same fetch, up to the `max-clients-per-query` limit, and only one iterative query is sent. However, when clients are simultaneously querying for *different* names or types, multiple queries are sent and `max-clients-per-query` is not effective as a limit.

Optionally, this value may be followed by the keyword `drop` or `fail`, indicating whether queries which exceed the fetch quota for a zone are dropped with no response, or answered with SERVFAIL. The default is `drop`.

If **fetches-per-zone** is set to zero, there is no limit on the number of fetches per query and no queries are dropped. The default is zero.

The current list of active fetches can be dumped by running **rndc recursing**. The list includes the number of active fetches for each domain and the number of queries that have been passed or dropped as a result of the `fetches-per-zone` limit. (Note: these counters are not cumulative over time; whenever the number of active fetches for a domain drops to zero, the counter for that domain is deleted, and the next time a fetch is sent to that domain, it is recreated with the counters set to zero.)

fetches-per-server
This sets the maximum number of simultaneous iterative queries that the server allows to

be sent to a single upstream name server before blocking additional queries. This value should reflect how many fetches would normally be sent to any one server in the time it would take to resolve them. It should be smaller than `recursive-clients`.

Optionally, this value may be followed by the keyword `drop` or `fail`, indicating whether queries are dropped with no response or answered with SERVFAIL, when all of the servers authoritative for a zone are found to have exceeded the per-server quota. The default is `fail`.

If **fetches-per-server** is set to zero, there is no limit on the number of fetches per query and no queries are dropped. The default is zero.

The **fetches-per-server** quota is dynamically adjusted in response to detected congestion. As queries are sent to a server and are either answered or time out, an exponentially weighted moving average is calculated of the ratio of timeouts to responses. If the current average timeout ratio rises above a "high" threshold, then **fetches-per-server** is reduced for that server. If the timeout ratio drops below a "low" threshold, then **fetches-per-server** is increased. The **fetch-quota-params** options can be used to adjust the parameters for this calculation.

fetch-quota-params

This sets the parameters to use for dynamic resizing of the `fetches-per-server` quota in response to detected congestion.

The first argument is an integer value indicating how frequently to recalculate the moving average of the ratio of timeouts to responses for each server. The default is 100, meaning that BIND recalculates the average ratio after every 100 queries have either been answered or timed out.

The remaining three arguments represent the "low" threshold (defaulting to a timeout ratio of 0.1), the "high" threshold (defaulting to a timeout ratio of 0.3), and the discount rate for the moving average (defaulting to 0.7). A higher discount rate causes recent events to weigh more heavily when calculating the moving average; a lower discount rate causes past events to weigh more heavily, smoothing out short-term blips in the timeout ratio. These arguments are all fixed-point numbers with precision of 1/100; at most two places after the decimal point are significant.

reserved-sockets

This sets the number of file descriptors reserved for TCP, stdio, etc. This needs to be big enough to cover the number of interfaces **named** listens on plus **tcp-clients**, as well as to provide room for outgoing TCP queries and incoming zone transfers. The default is `512`. The minimum value is `128` and the maximum value is `128` fewer than maxsockets (-S). This option may be removed in the future.

This option has little effect on Windows.

max-cache-size

This sets the maximum amount of memory to use for the server's cache, in bytes or percentage of total physical memory. When the amount of data in the cache reaches this limit, the server causes records to expire prematurely, following an LRU-based strategy, so that the limit is not exceeded. The keyword **`unlimited`**, or the value 0, places no limit on the cache size; records are purged from the cache only when their TTLs expire. Any positive values less than 2MB are ignored and reset to 2MB. In a server with multiple views, the limit applies separately to the cache of each view. The default is **`90%`**. On systems where

detection of the amount of physical memory is not supported, values represented as a percentage fall back to unlimited. Note that the detection of physical memory is done only once at startup, so **named** does not adjust the cache size if the amount of physical memory is changed during runtime.

tcp-listen-queue
This sets the listen-queue depth. The default and minimum is 10. If the kernel supports the accept filter "dataready", this also controls how many TCP connections are queued in kernel space waiting for some data before being passed to accept. Non-zero values less than 10 are silently raised. A value of 0 may also be used; on most platforms this sets the listen-queue length to a system-defined default value.

Periodic Task Intervals

cleaning-interval
This interval is effectively obsolete. Previously, the server removed expired resource records from the cache every **cleaning-interval** minutes. BIND 9 now manages cache memory in a more sophisticated manner and does not rely on periodic cleaning anymore. Specifying this option therefore has no effect on the server's behavior.

heartbeat-interval
The server performs zone maintenance tasks for all zones marked as **dialup** whenever this interval expires. The default is 60 minutes. Reasonable values are up to 1 day (1440 minutes). The maximum value is 28 days (40320 minutes). If set to 0, no zone maintenance for these zones occurs.

interface-interval
The server scans the network interface list every **interface-interval** minutes. The default is 60 minutes; the maximum value is 28 days (40320 minutes). If set to 0, interface scanning only occurs when the configuration file is loaded, or when **automatic-interface-scan** is enabled and supported by the operating system. After the scan, the server begins listening for queries on any newly discovered interfaces (provided they are allowed by the **listen-on** configuration), and stops listening on interfaces that have gone away.

statistics-interval
Name server statistics are logged every **statistics-interval** minutes. The default is 60, and the maximum value is 28 days (40320 minutes). If set to 0, no statistics are logged.

> NOTE
>
> This option is not implemented in BIND 9.

topology
In BIND 8, this option indicated network topology so that preferential treatment could be given to the topologically closest name servers when sending queries. It is not implemented in BIND 9.

The sortlist Statement

The response to a DNS query may consist of multiple resource records (RRs) forming a resource record set (RRset). The name server normally returns the RRs within the RRset in an indeterminate order (but see the **rrset-order** statement in Section 6.2). The client resolver code should rearrange the RRs as appropriate: that is, using any addresses on the local net in preference to other addresses. However, not all resolvers can do this or are correctly configured. When a client is using a local server, the sorting can be performed in the server, based on the client's address. This only requires configuring the name servers, not all the clients.

The **sortlist** statement (see below) takes an **address_match_list** and interprets it in a special way. Each top-level statement in the **sortlist** must itself be an explicit **address_match_list** with one or two elements. The first element (which may be an IP address, an IP prefix, an ACL name, or a nested **address_match_list**) of each top-level list is checked against the source address of the query until a match is found. When the addresses in the first element overlap, the first rule to match is selected.

Once the source address of the query has been matched, if the top-level statement contains only one element, the actual primitive element that matched the source address is used to select the address in the response to move to the beginning of the response. If the statement is a list of two elements, then the second element is interpreted as a topology preference list. Each top-level element is assigned a distance, and the address in the response with the minimum distance is moved to the beginning of the response.

In the following example, any queries received from any of the addresses of the host itself get responses preferring addresses on any of the locally connected networks. Next most preferred are addresses on the 192.168.1/24 network, and after that either the 192.168.2/24 or 192.168.3/24 network, with no preference shown between these two networks. Queries received from a host on the 192.168.1/24 network prefer other addresses on that network to the 192.168.2/24 and 192.168.3/24 networks. Queries received from a host on the 192.168.4/24 or the 192.168.5/24 network only prefer other addresses on their directly connected networks.

```
sortlist {
    // IF the local host
    // THEN first fit on the following nets
    { localhost;
        { localnets;
            192.168.1/24;
            { 192.168.2/24; 192.168.3/24; }; }; };
    // IF on class C 192.168.1 THEN use .1, or .2 or .3
    { 192.168.1/24;
        { 192.168.1/24;
            { 192.168.2/24; 192.168.3/24; }; }; };
    // IF on class C 192.168.2 THEN use .2, or .1 or .3
    { 192.168.2/24;
        { 192.168.2/24;
            { 192.168.1/24; 192.168.3/24; }; }; };
    // IF on class C 192.168.3 THEN use .3, or .1 or .2
    { 192.168.3/24;
        { 192.168.3/24;
            { 192.168.1/24; 192.168.2/24; }; }; };
    // IF .4 or .5 THEN prefer that net
    { { 192.168.4/24; 192.168.5/24; };
```

```
    };
};
```

The following example illustrates reasonable behavior for the local host and hosts on directly connected networks. Responses sent to queries from the local host favor any of the directly connected networks. Responses sent to queries from any other hosts on a directly connected network prefer addresses on that same network. Responses to other queries are not sorted.

```
sortlist {
           { localhost; localnets; };
           { localnets; };
};
```

RRset Ordering

> NOTE
>
> While alternating the order of records in a DNS response between subsequent queries is a known load distribution technique, certain caveats apply (mostly stemming from caching) which usually make it a suboptimal choice for load balancing purposes when used on its own.

The **rrset-order** statement permits configuration of the ordering of the records in a multiple-record response. See also: Section 6.2.

Each rule in an **rrset-order** statement is defined as follows:

[class *<class_name>*] [type *<type_name>*] [name "*<domain_name>*"] order *<ordering>*

The default qualifiers for each rule are:

- If no **class** is specified, the default is **ANY**.
- If no **type** is specified, the default is **ANY**.
- If no **name** is specified, the default is * (asterisk).

<domain_name> only matches the name itself, not any of its subdomains. To make a rule match all subdomains of a given name, a wildcard name (**.<domain_name>*) must be used. Note that **.<domain_name>* does *not* match *<domain_name>* itself; to specify RRset ordering for a name and all of its subdomains, two separate rules must be defined: one for *<domain_name>* and one for **.<domain_name>*.

The legal values for *<ordering>* are:

fixed
Records are returned in the order they are defined in the zone file.

> NOTE
>
> The **fixed** option is only available if BIND is configured with **--enable-fixed-rrset** at compile time.

random
Records are returned in a random order.

cyclic
Records are returned in a cyclic round-robin order, rotating by one record per query.

By default, records are returned in random order.

Note that if multiple **rrset-order** statements are present in the configuration file (at both the **options** and **view** levels), they are *not* combined; instead, the more-specific one (**view**) replaces the less-specific one (**options**).

If multiple rules within a single **rrset-order** statement match a given RRset, the first matching rule is applied.

Example:

```
rrset-order {
    type A name "foo.isc.org" order random;
    type AAAA name "foo.isc.org" order cyclic;
    name "bar.isc.org" order fixed;
    name "*.bar.isc.org" order random;
    name "*.baz.isc.org" order cyclic;
};
```

With the above configuration, the following RRset ordering is used:

QNAME	QTYPE	RRset Order
foo.isc.org	**A**	**random**
foo.isc.org	**AAAA**	**cyclic**
foo.isc.org	**TXT**	**random**
sub.foo.isc.org	all	**random**
bar.isc.org	all	**fixed**
sub.bar.isc.org	all	**random**
baz.isc.org	all	**random**
sub.baz.isc.org	all	**cyclic**

Tuning

lame-ttl
This is always set to 0. More information is available in the security advisory for CVE-2021-25219.

servfail-ttl
This sets the number of seconds to cache a SERVFAIL response due to DNSSEC validation failure or other general server failure. If set to `0`, SERVFAIL caching is disabled. The SERVFAIL cache is not consulted if a query has the CD (Checking Disabled) bit set; this allows a query that failed due to DNSSEC validation to be retried without waiting for the SERVFAIL TTL to expire.

The maximum value is `30` seconds; any higher value is silently reduced. The default is `1` second.

max-ncache-ttl
To reduce network traffic and increase performance, the server stores negative answers. **max-ncache-ttl** is used to set a maximum retention time for these answers in the server, in seconds. The default **max-ncache-ttl** is `10800` seconds (3 hours). **max-ncache-ttl** cannot exceed 7 days and is silently truncated to 7 days if set to a greater value.

max-cache-ttl
This sets the maximum time for which the server caches ordinary (positive) answers, in seconds. The default is 604800 (one week). A value of zero may cause all queries to return SERVFAIL, because of lost caches of intermediate RRsets (such as NS and glue AAAA/A records) in the resolution process.

min-roots
This sets the minimum number of root servers that is required for a request for the root servers to be accepted. The default is **2**.

> NOTE
>
> This is not implemented in BIND 9.

sig-validity-interval
This specifies the number of days into the future that DNSSEC signatures that are automatically generated as a result of dynamic updates (Section 4.2) will expire. There is an optional second field which specifies how long before expiry that the signatures are regenerated. If not specified, the signatures are regenerated at 1/4 of base interval. The second field is specified in days if the base interval is greater than 7 days; otherwise it is specified in hours. The default base interval is `30` days, giving a re-signing interval of 7 1/2 days. The maximum value is 10 years (3660 days).

The signature inception time is unconditionally set to one hour before the current time, to allow for a limited amount of clock skew.

The **sig-validity-interval** should be at least several multiples of the SOA expire interval, to allow for reasonable interaction between the various timer and expiry dates.

sig-signing-nodes
This specifies the maximum number of nodes to be examined in each quantum, when signing a zone with a new DNSKEY. The default is `100`.

sig-signing-signatures
This specifies a threshold number of signatures that terminates processing a quantum, when signing a zone with a new DNSKEY. The default is `10`.

sig-signing-type
This specifies a private RDATA type to be used when generating signing-state records. The default is `65534`.

This parameter may be removed in a future version, once there is a standard type.

Signing-state records are used internally by **named** to track the current state of a zone-signing process, i.e., whether it is still active or has been completed. The records can be inspected using the command **rndc signing -list *zone***. Once **named** has finished signing a zone with a particular key, the signing-state record associated with that key can be removed from the zone by running **rndc signing -clear *keyid/algorithm zone***. To clear all of the completed signing-state records for a zone, use **rndc signing -clear all *zone***.

min-refresh-time, max-refresh-time, min-retry-time, max-retry-time
These options control the server's behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, up to a hard-coded maximum expiry of 24 weeks. However, these values are set by the primary, giving secondary server administrators little control over their contents.

These options allow the administrator to set a minimum and maximum refresh and retry time in seconds per-zone, per-view, or globally. These options are valid for secondary and stub zones, and clamp the SOA refresh and retry times to the specified values.

The following defaults apply: **min-refresh-time** 300 seconds, **max-refresh-time** 2419200 seconds (4 weeks), **min-retry-time** 500 seconds, and **max-retry-time** 1209600 seconds (2 weeks).

edns-udp-size
This sets the maximum advertised EDNS UDP buffer size, in bytes, to control the size of packets received from authoritative servers in response to recursive queries. Valid values are 512 to 4096; values outside this range are silently adjusted to the nearest value within it. The default value is 1232.

The usual reason for setting **edns-udp-size** to a non-default value is to get UDP answers to pass through broken firewalls that block fragmented packets and/or block UDP DNS packets that are greater than 512 bytes.

When **named** first queries a remote server, it advertises a UDP buffer size of 512, as this has the greatest chance of success on the first try.

If the initial response times out, **named** tries again with plain DNS; if that is successful, it is taken as evidence that the server does not support EDNS. After enough failures using EDNS and successes using plain DNS, **named** defaults to plain DNS for future communications with that server. If that happens, **named** periodically sends an EDNS query to see if the situation has improved.

However, if the initial query is successful with EDNS advertising a buffer size of 512, then **named** advertises progressively larger buffer sizes on successive queries, until responses begin timing out or **edns-udp-size** is reached.

The default buffer sizes used by **named** are 512, 1232, 1432, and 4096, but never exceed **edns-udp-size**. (The values 1232 and 1432 are chosen to allow for an IPv4-/IPv6-encapsulated UDP message to be sent without fragmentation at the minimum MTU sizes for Ethernet and IPv6 networks.)

max-udp-size
This sets the maximum EDNS UDP message size that **named** sends, in bytes. Valid values are 512 to 4096; values outside this range are silently adjusted to the nearest value within it. The default value is 1232.

This value applies to responses sent by a server; to set the advertised buffer size in queries, see **edns-udp-size**.

The usual reason for setting **max-udp-size** to a non-default value is to allow UDP answers to pass through broken firewalls that block fragmented packets and/or block UDP packets that are greater than 512 bytes. This is independent of the advertised receive buffer (**edns-udp-size**).

Setting this to a low value encourages additional TCP traffic to the name server.

masterfile-format
This specifies the file format of zone files (see Section 6.3). The default value is `text`, which is the standard textual representation, except for secondary zones, in which the default value is `raw`. Files in formats other than `text` are typically expected to be generated by the **named-compilezone** tool, or dumped by **named**.

Note that when a zone file in a format other than `text` is loaded, **named** may omit some of the checks which would be performed for a file in `text` format. In particular, **check-names** checks do not apply for the `raw` format. This means a zone file in the `raw` format must be generated with the same check level as that specified in the **named** configuration file. Also, `map` format files are loaded directly into memory via memory mapping, with only minimal checking.

This statement sets the **masterfile-format** for all zones, but can be overridden on a per-zone or per-view basis by including a **masterfile-format** statement within the **zone** or **view** block in the configuration file.

masterfile-style
This specifies the formatting of zone files during dump, when the `masterfile-format` is `text`. This option is ignored with any other `masterfile-format`.

When set to `relative`, records are printed in a multi-line format, with owner names expressed relative to a shared origin. When set to `full`, records are printed in a single-line format with absolute owner names. The `full` format is most suitable when a zone file needs to be processed automatically by a script. The `relative` format is more human-readable, and is thus suitable when a zone is to be edited by hand. The default is `relative`.

max-recursion-depth
This sets the maximum number of levels of recursion that are permitted at any one time while servicing a recursive query. Resolving a name may require looking up a name server address, which in turn requires resolving another name, etc.; if the number of recursions

exceeds this value, the recursive query is terminated and returns SERVFAIL. The default is 7.

max-recursion-queries
This sets the maximum number of iterative queries that may be sent while servicing a recursive query. If more queries are sent, the recursive query is terminated and returns SERVFAIL. The default is 100.

notify-delay
This sets the delay, in seconds, between sending sets of NOTIFY messages for a zone. The default is 5 seconds.

The overall rate at which NOTIFY messages are sent for all zones is controlled by **serial-query-rate**.

max-rsa-exponent-size
This sets the maximum RSA exponent size, in bits, that is accepted when validating. Valid values are 35 to 4096 bits. The default, zero, is also accepted and is equivalent to 4096.

prefetch
When a query is received for cached data which is to expire shortly, **named** can refresh the data from the authoritative server immediately, ensuring that the cache always has an answer available.

`prefetch` specifies the "trigger" TTL value at which prefetch of the current query takes place; when a cache record with a lower TTL value is encountered during query processing, it is refreshed. Valid trigger TTL values are 1 to 10 seconds. Values larger than 10 seconds are silently reduced to 10. Setting a trigger TTL to zero causes prefetch to be disabled. The default trigger TTL is `2`.

An optional second argument specifies the "eligibility" TTL: the smallest *original* TTL value that is accepted for a record to be eligible for prefetching. The eligibility TTL must be at least six seconds longer than the trigger TTL; if not, **named** silently adjusts it upward. The default eligibility TTL is `9`.

v6-bias
When determining the next name server to try, this indicates by how many milliseconds to prefer IPv6 name servers. The default is `50` milliseconds.

Built-in Server Information Zones

The server provides some helpful diagnostic information through a number of built-in zones under the pseudo-top-level-domain `bind` in the **CHAOS** class. These zones are part of a built-in view (see Section 6.2) of class **CHAOS**, which is separate from the default view of class **IN**. Most global configuration options (**allow-query**, etc.) apply to this view, but some are locally overridden: **notify**, **recursion**, and **allow-new-zones** are always set to **no**, and **rate-limit** is set to allow three responses per second.

To disable these zones, use the options below or hide the built-in **CHAOS** view by defining an explicit view of class **CHAOS** that matches all clients.

version
This is the version the server should report via a query of the name `version.bind` with

type **TXT** and class **CHAOS**. The default is the real version number of this server. Specifying **version none** disables processing of the queries.

Setting **version** to any value (including `none`) also disables queries for `authors.bind TXT CH`.

hostname
This is the hostname the server should report via a query of the name `hostname.bind` with type **TXT** and class **CHAOS**. This defaults to the hostname of the machine hosting the name server, as found by the **gethostname()** function. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering the queries. Specifying **hostname none;** disables processing of the queries.

server-id
This is the ID the server should report when receiving a Name Server Identifier (NSID) query, or a query of the name `ID.SERVER` with type **TXT** and class **CHAOS**. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering the queries. Specifying **server-id none;** disables processing of the queries. Specifying **server-id hostname;** causes **named** to use the hostname as found by the **gethostname()** function. The default **server-id** is **none**.

Built-in Empty Zones

The **named** server has some built-in empty zones, for SOA and NS records only. These are for zones that should normally be answered locally and which queries should not be sent to the Internet's root servers. The official servers which cover these namespaces return NXDOMAIN responses to these queries. In particular, these cover the reverse namespaces for addresses from RFC 1918, RFC 4193, RFC 5737, and RFC 6598. They also include the reverse namespace for the IPv6 local address (locally assigned), IPv6 link local addresses, the IPv6 loopback address, and the IPv6 unknown address.

The server attempts to determine if a built-in zone already exists or is active (covered by a forward-only forwarding declaration) and does not create an empty zone if either is true.

The current list of empty zones is:

- 10.IN-ADDR.ARPA
- 16.172.IN-ADDR.ARPA
- 17.172.IN-ADDR.ARPA
- 18.172.IN-ADDR.ARPA
- 19.172.IN-ADDR.ARPA
- 20.172.IN-ADDR.ARPA
- 21.172.IN-ADDR.ARPA
- 22.172.IN-ADDR.ARPA
- 23.172.IN-ADDR.ARPA

- 24.172.IN-ADDR.ARPA
- 25.172.IN-ADDR.ARPA
- 26.172.IN-ADDR.ARPA
- 27.172.IN-ADDR.ARPA
- 28.172.IN-ADDR.ARPA
- 29.172.IN-ADDR.ARPA
- 30.172.IN-ADDR.ARPA
- 31.172.IN-ADDR.ARPA
- 168.192.IN-ADDR.ARPA
- 64.100.IN-ADDR.ARPA
- 65.100.IN-ADDR.ARPA
- 66.100.IN-ADDR.ARPA
- 67.100.IN-ADDR.ARPA
- 68.100.IN-ADDR.ARPA
- 69.100.IN-ADDR.ARPA
- 70.100.IN-ADDR.ARPA
- 71.100.IN-ADDR.ARPA
- 72.100.IN-ADDR.ARPA
- 73.100.IN-ADDR.ARPA
- 74.100.IN-ADDR.ARPA
- 75.100.IN-ADDR.ARPA
- 76.100.IN-ADDR.ARPA
- 77.100.IN-ADDR.ARPA
- 78.100.IN-ADDR.ARPA
- 79.100.IN-ADDR.ARPA
- 80.100.IN-ADDR.ARPA
- 81.100.IN-ADDR.ARPA
- 82.100.IN-ADDR.ARPA
- 83.100.IN-ADDR.ARPA
- 84.100.IN-ADDR.ARPA

- 85.100.IN-ADDR.ARPA
- 86.100.IN-ADDR.ARPA
- 87.100.IN-ADDR.ARPA
- 88.100.IN-ADDR.ARPA
- 89.100.IN-ADDR.ARPA
- 90.100.IN-ADDR.ARPA
- 91.100.IN-ADDR.ARPA
- 92.100.IN-ADDR.ARPA
- 93.100.IN-ADDR.ARPA
- 94.100.IN-ADDR.ARPA
- 95.100.IN-ADDR.ARPA
- 96.100.IN-ADDR.ARPA
- 97.100.IN-ADDR.ARPA
- 98.100.IN-ADDR.ARPA
- 99.100.IN-ADDR.ARPA
- 100.100.IN-ADDR.ARPA
- 101.100.IN-ADDR.ARPA
- 102.100.IN-ADDR.ARPA
- 103.100.IN-ADDR.ARPA
- 104.100.IN-ADDR.ARPA
- 105.100.IN-ADDR.ARPA
- 106.100.IN-ADDR.ARPA
- 107.100.IN-ADDR.ARPA
- 108.100.IN-ADDR.ARPA
- 109.100.IN-ADDR.ARPA
- 110.100.IN-ADDR.ARPA
- 111.100.IN-ADDR.ARPA
- 112.100.IN-ADDR.ARPA
- 113.100.IN-ADDR.ARPA
- 114.100.IN-ADDR.ARPA

- 115.100.IN-ADDR.ARPA
- 116.100.IN-ADDR.ARPA
- 117.100.IN-ADDR.ARPA
- 118.100.IN-ADDR.ARPA
- 119.100.IN-ADDR.ARPA
- 120.100.IN-ADDR.ARPA
- 121.100.IN-ADDR.ARPA
- 122.100.IN-ADDR.ARPA
- 123.100.IN-ADDR.ARPA
- 124.100.IN-ADDR.ARPA
- 125.100.IN-ADDR.ARPA
- 126.100.IN-ADDR.ARPA
- 127.100.IN-ADDR.ARPA
- 0.IN-ADDR.ARPA
- 127.IN-ADDR.ARPA
- 254.169.IN-ADDR.ARPA
- 2.0.192.IN-ADDR.ARPA
- 100.51.198.IN-ADDR.ARPA
- 113.0.203.IN-ADDR.ARPA
- 255.255.255.255.IN-ADDR.ARPA
- 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA
- 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
- 8.B.D.0.1.0.0.2.IP6.ARPA
- D.F.IP6.ARPA
- 8.E.F.IP6.ARPA
- 9.E.F.IP6.ARPA
- A.E.F.IP6.ARPA
- B.E.F.IP6.ARPA
- EMPTY.AS112.ARPA
- HOME.ARPA

Empty zones can be set at the view level and only apply to views of class IN. Disabled empty zones are only inherited from options if there are no disabled empty zones specified at the view level. To override the options list of disabled zones, disable the root zone at the view level. For example:

```
disable-empty-zone ".";
```

If using the address ranges covered here, reverse zones covering the addresses should already be in place. In practice this appears to not be the case, with many queries being made to the infrastructure servers for names in these spaces. So many, in fact, that sacrificial servers had to be deployed to channel the query load away from the infrastructure servers.

NOTE

The real parent servers for these zones should disable all empty zones under the parent zone they serve. For the real root servers, this is all built-in empty zones. This enables them to return referrals to deeper in the tree.

empty-server
This specifies the server name that appears in the returned SOA record for empty zones. If none is specified, the zone's name is used.

empty-contact
This specifies the contact name that appears in the returned SOA record for empty zones. If none is specified, "." is used.

empty-zones-enable
This enables or disables all empty zones. By default, they are enabled.

disable-empty-zone
This disables individual empty zones. By default, none are disabled. This option can be specified multiple times.

Additional Section Caching

The additional section cache, also called **acache**, is an internal cache to improve the response performance of BIND 9. When additional section caching is enabled, BIND 9 caches an internal shortcut to the additional section content for each answer RR. Note that **acache** is an internal caching mechanism of BIND 9, and is not related to the DNS caching server function.

Additional section caching does not change the response content (except the RRsets ordering of the additional section; see below), but can improve the response performance significantly. It is particularly effective when BIND 9 acts as an authoritative server for a zone that has many delegations with many glue RRs.

To obtain the maximum performance improvement from additional section caching, setting **additional-from-cache** to **no** is recommended, since the current implementation of **acache** does not shortcut additional section information from the DNS cache data.

One obvious disadvantage of **acache** is that it requires much more memory for the internal cached data. Thus, if the response performance does not matter and memory consumption is more critical, the **acache** mechanism can be disabled by setting **acache-enable** to **no**. It is also possible to specify the upper limit of memory consumption for **acache** by using **max-acache-size**.

Additional section caching also has a minor effect on the RRset ordering in the additional section. Without **acache**, **cyclic** order is effective for the additional section as well as for the answer and authority sections. However, additional section caching fixes the ordering when it first caches an RRset for the additional section, and the same ordering is kept in succeeding responses, regardless of the setting of **rrset-order**. The effect of this should be minor, however, since an RRset in the additional section typically only contains a small number of RRs (and in many cases only a single RR), so the ordering is not significant.

The following is a summary of options related to **acache**.

acache-enable
If **yes**, additional section caching is enabled. The default value is **no**.

acache-cleaning-interval
The server removes stale cache entries, based on an LRU-based algorithm, every **acache-cleaning-interval** minutes. The default is 60 minutes. If set to 0, no periodic cleaning occurs.

max-acache-size
This is the maximum amount of memory, in bytes, to use for the server's acache. When the amount of data in the acache reaches this limit, the server cleans more aggressively so that the limit is not exceeded. In a server with multiple views, the limit applies separately to the **acache** of each view. The default is `16M`.

Content Filtering

BIND 9 provides the ability to filter out responses from external DNS servers containing certain types of data in the answer section. Specifically, it can reject address (A or AAAA) records if the corresponding IPv4 or IPv6 addresses match the given `address_match_list` of the **deny-answer-addresses** option. It can also reject CNAME or DNAME records if the "alias" name (i.e., the CNAME alias or the substituted query name due to DNAME) matches the given `namelist` of the **deny-answer-aliases** option, where "match" means the alias name is a subdomain of one of the `name_list` elements. If the optional `namelist` is specified with **except-from**, records whose query name matches the list are accepted regardless of the filter setting. Likewise, if the alias name is a subdomain of the corresponding zone, the **deny-answer-aliases** filter does not apply; for example, even if "example.com" is specified for **deny-answer-aliases**,

```
www.example.com. CNAME xxx.example.com.
```

returned by an "example.com" server is accepted.

In the `address_match_list` of the **deny-answer-addresses** option, only `ip_addr` and `ip_prefix` are meaningful; any `key_id` is silently ignored.

If a response message is rejected due to the filtering, the entire message is discarded without being cached, and a SERVFAIL error is returned to the client.

This filtering is intended to prevent "DNS rebinding attacks," in which an attacker, in response to a query for a domain name the attacker controls, returns an IP address within the user's own network or an alias name within the user's own domain. A naive web browser or script could then serve as an unintended proxy, allowing the attacker to get access to an internal node of the local network that could not be externally accessed otherwise. See the paper available at https://dl.acm.org/doi/10.1145/1315245.1315298 for more details about these attacks.

For example, with a domain named "example.net" and an internal network using an IPv4 prefix 192.0.2.0/24, an administrator might specify the following rules:

```
deny-answer-addresses { 192.0.2.0/24; } except-from { "example.net"; };
deny-answer-aliases { "example.net"; };
```

If an external attacker let a web browser in the local network look up an IPv4 address of "attacker.example.com", the attacker's DNS server would return a response like this:

```
attacker.example.com. A 192.0.2.1
```

in the answer section. Since the rdata of this record (the IPv4 address) matches the specified prefix 192.0.2.0/24, this response would be ignored.

On the other hand, if the browser looked up a legitimate internal web server "www.example.net" and the following response were returned to the BIND 9 server:

```
www.example.net. A 192.0.2.2
```

it would be accepted, since the owner name "www.example.net" matches the **except-from** element, "example.net".

Note that this is not really an attack on the DNS per se. In fact, there is nothing wrong with having an "external" name mapped to an "internal" IP address or domain name from the DNS point of view; it might actually be provided for a legitimate purpose, such as for debugging. As long as the mapping is provided by the correct owner, it either is not possible or does not make sense to detect whether the intent of the mapping is legitimate within the DNS. The "rebinding" attack must primarily be protected at the application that uses the DNS. For a large site, however, it may be difficult to protect all possible applications at once. This filtering feature is provided only to help such an operational environment; turning it on is generally discouraged unless there is no other choice and the attack is a real threat to applications.

Care should be particularly taken if using this option for addresses within 127.0.0.0/8. These addresses are obviously "internal," but many applications conventionally rely on a DNS mapping from some name to such an address. Filtering out DNS records containing this address spuriously can break such applications.

Response Policy Zone (RPZ) Rewriting

BIND 9 includes a limited mechanism to modify DNS responses for requests analogous to email anti-spam DNS rejection lists. Responses can be changed to deny the existence of domains

(NXDOMAIN), deny the existence of IP addresses for domains (NODATA), or contain other IP addresses or data.

Response policy zones are named in the **response-policy** option for the view or among the global options if there is no **response-policy** option for the view. Response policy zones are ordinary DNS zones containing RRsets that can be queried normally if allowed. It is usually best to restrict those queries with something like **allow-query { localhost; };**. Note that zones using **masterfile-format map** cannot be used as policy zones.

A **response-policy** option can support multiple policy zones. To maximize performance, a radix tree is used to quickly identify response policy zones containing triggers that match the current query. This imposes an upper limit of 32 on the number of policy zones in a single **response-policy** option; more than that is a configuration error.

Rules encoded in response policy zones are processed after those defined in Access Control Lists (ACLs). All queries from clients which are not permitted access to the resolver are answered with a status code of REFUSED, regardless of configured RPZ rules.

Five policy triggers can be encoded in RPZ records.

RPZ-CLIENT-IP
IP records are triggered by the IP address of the DNS client. Client IP address triggers are encoded in records that have owner names that are subdomains of **rpz-client-ip**, relativized to the policy zone origin name, and encode an address or address block. IPv4 addresses are represented as **`prefixlength.B4.B3.B2.B1.rpz-client-ip`**. The IPv4 prefix length must be between 1 and 32. All four bytes - B4, B3, B2, and B1 - must be present. B4 is the decimal value of the least significant byte of the IPv4 address as in IN-ADDR.ARPA.

IPv6 addresses are encoded in a format similar to the standard IPv6 text representation, **`prefixlength.W8.W7.W6.W5.W4.W3.W2.W1.rpz-client-ip`**. Each of W8,...,W1 is a one- to four-digit hexadecimal number representing 16 bits of the IPv6 address as in the standard text representation of IPv6 addresses, but reversed as in IP6.ARPA. (Note that this representation of IPv6 address is different from IP6.ARPA where each hex digit occupies a label.) All 8 words must be present except when one set of consecutive zero words is replaced with **`.zz.`**, analogous to double colons (::) in standard IPv6 text encodings. The IPv6 prefix length must be between 1 and 128.

QNAME
QNAME policy records are triggered by query names of requests and targets of CNAME records resolved to generate the response. The owner name of a QNAME policy record is the query name relativized to the policy zone.

RPZ-IP
IP triggers are IP addresses in an A or AAAA record in the ANSWER section of a response. They are encoded like client-IP triggers, except as subdomains of **rpz-ip**.

RPZ-NSDNAME
NSDNAME triggers match names of authoritative servers for the query name, a parent of the query name, a CNAME for the query name, or a parent of a CNAME. They are encoded as subdomains of **rpz-nsdname**, relativized to the RPZ origin name. NSIP triggers match IP addresses in A and AAAA RRsets for domains that can be checked against

NSDNAME policy records. The **nsdname-enable** phrase turns NSDNAME triggers off or on for a single policy zone or for all zones.

If authoritative nameservers for the query name are not yet known, **named** recursively looks up the authoritative servers for the query name before applying an RPZ-NSDNAME rule, which can cause a processing delay. To speed up processing at the cost of precision, the **nsdname-wait-recurse** option can be used; when set to **no**, RPZ-NSDNAME rules are only applied when authoritative servers for the query name have already been looked up and cached. If authoritative servers for the query name are not in the cache, the RPZ-NSDNAME rule is ignored, but the authoritative servers for the query name are looked up in the background and the rule is applied to subsequent queries. The default is **yes**, meaning RPZ-NSDNAME rules are always applied, even if authoritative servers for the query name need to be looked up first.

RPZ-NSIP

NSIP triggers match the IP addresses of authoritative servers. They are enncoded like IP triggers, except as subdomains of **rpz-nsip**. NSDNAME and NSIP triggers are checked only for names with at least **min-ns-dots** dots. The default value of **min-ns-dots** is 1, to exclude top-level domains.

If a name server's IP address is not yet known, **named** recursively looks up the IP address before applying an RPZ-NSIP rule, which can cause a processing delay. To speed up processing at the cost of precision, the **nsip-wait-recurse** option can be used: when set to **no**, RPZ-NSIP rules are only applied when a name server's IP address has already been looked up and cached. If a server's IP address is not in the cache, the RPZ-NSIP rule is ignored, but the address is looked up in the background and the rule is applied to subsequent queries. The default is **yes**, meaning RPZ-NSIP rules are always applied, even if an address needs to be looked up first.

The query response is checked against all response policy zones, so two or more policy records can be triggered by a response. Because DNS responses are rewritten according to at most one policy record, a single record encoding an action (other than **DISABLED** actions) must be chosen. Triggers, or the records that encode them, are chosen for rewriting in the following order:

1. Choose the triggered record in the zone that appears first in the **response-policy** option.
2. Prefer CLIENT-IP to QNAME to IP to NSDNAME to NSIP triggers in a single zone.
3. Among NSDNAME triggers, prefer the trigger that matches the smallest name under the DNSSEC ordering.
4. Among IP or NSIP triggers, prefer the trigger with the longest prefix.
5. Among triggers with the same prefix length, prefer the IP or NSIP trigger that matches the smallest IP address.

When the processing of a response is restarted to resolve DNAME or CNAME records and a policy record set has not been triggered, all response policy zones are again consulted for the DNAME or CNAME names and addresses.

RPZ record sets are any types of DNS record, except DNAME or DNSSEC, that encode actions or responses to individual queries. Any of the policies can be used with any of the triggers. For example, while the **TCP-only** policy is commonly used with **client-IP** triggers, it can be used with any type of trigger to force the use of TCP for responses with owner names in a zone.

PASSTHRU
The auto-acceptance policy is specified by a CNAME whose target is **rpz-passthru**. It causes the response to not be rewritten and is most often used to "poke holes" in policies for CIDR blocks.

DROP
The auto-rejection policy is specified by a CNAME whose target is **rpz-drop**. It causes the response to be discarded. Nothing is sent to the DNS client.

TCP-Only
The "slip" policy is specified by a CNAME whose target is **rpz-tcp-only**. It changes UDP responses to short, truncated DNS responses that require the DNS client to try again with TCP. It is used to mitigate distributed DNS reflection attacks.

NXDOMAIN
The "domain undefined" response is encoded by a CNAME whose target is the root domain (.)

NODATA
The empty set of resource records is specified by a CNAME whose target is the wildcard top-level domain (`*.`). It rewrites the response to NODATA or ANCOUNT=0.

Local Data
A set of ordinary DNS records can be used to answer queries. Queries for record types not the set are answered with NODATA.

A special form of local data is a CNAME whose target is a wildcard such as *.example.com. It is used as if an ordinary CNAME after the asterisk (*) has been replaced with the query name. This special form is useful for query logging in the walled garden's authoritative DNS server.

All of the actions specified in all of the individual records in a policy zone can be overridden with a **policy** clause in the **response-policy** option. An organization using a policy zone provided by another organization might use this mechanism to redirect domains to its own walled garden.

GIVEN
The placeholder policy says "do not override but perform the action specified in the zone."

DISABLED
The testing override policy causes policy zone records to do nothing but log what they would have done if the policy zone were not disabled. The response to the DNS query is written (or not) according to any triggered policy records that are not disabled. Disabled policy zones should appear first, because they are often not logged if a higher-precedence trigger is found first.

PASSTHRU, DROP, TCP-Only, NXDOMAIN, NODATA
each override the corresponding per-record policy.

CNAME domain
causes all RPZ policy records to act as if they were "cname domain" records.

By default, the actions encoded in a response policy zone are applied only to queries that ask for recursion (RD=1). That default can be changed for a single policy zone, or for all response policy zones in a view, with a **recursive-only no** clause. This feature is useful for serving the same zone files both inside and outside an RFC 1918 cloud and using RPZ to delete answers that would otherwise contain RFC 1918 values on the externally visible name server or view.

Also by default, RPZ actions are applied only to DNS requests that either do not request DNSSEC metadata (DO=0) or when no DNSSEC records are available for the requested name in the original zone (not the response policy zone). This default can be changed for all response policy zones in a view with a **break-dnssec yes** clause. In that case, RPZ actions are applied regardless of DNSSEC. The name of the clause option reflects the fact that results rewritten by RPZ actions cannot verify.

No DNS records are needed for a QNAME or Client-IP trigger; the name or IP address itself is sufficient, so in principle the query name need not be recursively resolved. However, not resolving the requested name can leak the fact that response policy rewriting is in use, and that the name is listed in a policy zone, to operators of servers for listed names. To prevent that information leak, by default any recursion needed for a request is done before any policy triggers are considered. Because listed domains often have slow authoritative servers, this behavior can cost significant time. The **qname-wait-recurse no** option overrides that default behavior when recursion cannot change a non-error response. The option does not affect QNAME or client-IP triggers in policy zones listed after other zones containing IP, NSIP, and NSDNAME triggers, because those may depend on the A, AAAA, and NS records that would be found during recursive resolution. It also does not affect DNSSEC requests (DO=1) unless **break-dnssec yes** is in use, because the response would depend on whether RRSIG records were found during resolution. Using this option can cause error responses such as SERVFAIL to appear to be rewritten, since no recursion is being done to discover problems at the authoritative server.

The TTL of a record modified by RPZ policies is set from the TTL of the relevant record in the policy zone. It is then limited to a maximum value. The **max-policy-ttl** clause changes the maximum number of seconds from its default of 5.

For example, an administrator might use this option statement:

```
    response-policy { zone "badlist"; };
```

and this zone statement:

```
    zone "badlist" {type master; file "master/badlist"; allow-query {none ↩
        ;}; };
```

with this zone file:

```
$TTL 1H
@                       SOA LOCALHOST. named-mgr.example.com (1 1h 15m 30d ↩
     2h)
                        NS  LOCALHOST.

; QNAME policy records.  There are no periods (.) after the owner names.
nxdomain.domain.com     CNAME   .               ; NXDOMAIN policy
```

```
*.nxdomain.domain.com   CNAME   .               ; NXDOMAIN policy
nodata.domain.com       CNAME   *.              ; NODATA policy
*.nodata.domain.com     CNAME   *.              ; NODATA policy
bad.domain.com          A       10.0.0.1        ; redirect to a walled  ↩
    garden
                        AAAA    2001:2::1
bzone.domain.com        CNAME   garden.example.com.

; do not rewrite (PASSTHRU) OK.DOMAIN.COM
ok.domain.com           CNAME   rpz-passthru.

; redirect x.bzone.domain.com to x.bzone.domain.com.garden.example.com
*.bzone.domain.com      CNAME   *.garden.example.com.


; IP policy records that rewrite all responses containing A records in  ↩
    127/8
;       except 127.0.0.1
8.0.0.0.127.rpz-ip      CNAME   .
32.1.0.0.127.rpz-ip     CNAME   rpz-passthru.

; NSDNAME and NSIP policy records
ns.domain.com.rpz-nsdname   CNAME   .
48.zz.2.2001.rpz-nsip       CNAME   .

; auto-reject and auto-accept some DNS clients
112.zz.2001.rpz-client-ip    CNAME   rpz-drop.
8.0.0.0.127.rpz-client-ip    CNAME   rpz-drop.

; force some DNS clients and responses in the example.com zone to TCP
16.0.0.1.10.rpz-client-ip   CNAME   rpz-tcp-only.
example.com                 CNAME   rpz-tcp-only.
*.example.com               CNAME   rpz-tcp-only.
```

RPZ can affect server performance. Each configured response policy zone requires the server to perform one to four additional database lookups before a query can be answered. For example, a DNS server with four policy zones, each with all four kinds of response triggers (QNAME, IP, NSIP, and NSDNAME), requires a total of 17 times as many database lookups as a similar DNS server with no response policy zones. A BIND 9 server with adequate memory and one response policy zone with QNAME and IP triggers might achieve a maximum queries-per-second (QPS) rate about 20% lower. A server with four response policy zones with QNAME and IP triggers might have a maximum QPS rate about 50% lower.

Responses rewritten by RPZ are counted in the **RPZRewrites** statistics.

The **log** clause can be used to optionally turn off rewrite logging for a particular response policy zone. By default, all rewrites are logged.

Response Rate Limiting

Excessive, almost identical UDP *responses* can be controlled by configuring a **rate-limit** clause in an **options** or **view** statement. This mechanism keeps authoritative BIND 9 from being used

to amplify reflection denial of service (DoS) attacks. Short, truncated (TC=1) responses can be sent to provide rate-limited responses to legitimate clients within a range of forged, attacked IP addresses. Legitimate clients react to dropped or truncated responses by retrying with UDP or with TCP, respectively.

This mechanism is intended for authoritative DNS servers. It can be used on recursive servers, but can slow applications such as SMTP servers (mail receivers) and HTTP clients (web browsers) that repeatedly request the same domains. When possible, closing "open" recursive servers is better.

Response rate limiting uses a "credit" or "token bucket" scheme. Each combination of identical response and client has a conceptual "account" that earns a specified number of credits every second. A prospective response debits its account by one. Responses are dropped or truncated while the account is negative. Responses are tracked within a rolling window of time which defaults to 15 seconds, but which can be configured with the **window** option to any value from 1 to 3600 seconds (1 hour). The account cannot become more positive than the per-second limit or more negative than **window** times the per-second limit. When the specified number of credits for a class of responses is set to 0, those responses are not rate-limited.

The notions of "identical response" and "DNS client" for rate limiting are not simplistic. All responses to an address block are counted as if to a single client. The prefix lengths of address blocks are specified with **ipv4-prefix-length** (default 24) and **ipv6-prefix-length** (default 56).

All non-empty responses for a valid domain name (qname) and record type (qtype) are identical and have a limit specified with **responses-per-second** (default 0 or no limit). All empty (NODATA) responses for a valid domain, regardless of query type, are identical. Responses in the NODATA class are limited by **nodata-per-second** (default **responses-per-second**). Requests for any and all undefined subdomains of a given valid domain result in NXDOMAIN errors, and are identical regardless of query type. They are limited by **nxdomains-per-second** (default **responses-per-second**). This controls some attacks using random names, but can be relaxed or turned off (set to 0) on servers that expect many legitimate NXDOMAIN responses, such as from anti-spam rejection lists. Referrals or delegations to the server of a given domain are identical and are limited by **referrals-per-second** (default **responses-per-second**).

Responses generated from local wildcards are counted and limited as if they were for the parent domain name. This controls flooding using random.wild.example.com.

All requests that result in DNS errors other than NXDOMAIN, such as SERVFAIL and FORMERR, are identical regardless of requested name (qname) or record type (qtype). This controls attacks using invalid requests or distant, broken authoritative servers. By default the limit on errors is the same as the **responses-per-second** value, but it can be set separately with **errors-per-second**.

Many attacks using DNS involve UDP requests with forged source addresses. Rate limiting prevents the use of BIND 9 to flood a network with responses to requests with forged source addresses, but could let a third party block responses to legitimate requests. There is a mechanism that can answer some legitimate requests from a client whose address is being forged in a flood. Setting **slip** to 2 (its default) causes every other UDP request to be answered with a small truncated (TC=1) response. The small size and reduced frequency, and resulting lack of amplification, of "slipped" responses make them unattractive for reflection DoS attacks. **slip** must be between 0 and 10. A value of 0 does not "slip"; no truncated responses are sent due to rate limiting. Rather, all responses are dropped. A value of 1 causes every response to slip; values between 2 and 10 cause every nth response to slip. Some error responses, including REFUSED

and SERVFAIL, cannot be replaced with truncated responses and are instead leaked at the **slip** rate.

(Note: dropped responses from an authoritative server may reduce the difficulty of a third party successfully forging a response to a recursive resolver. The best security against forged responses is for authoritative operators to sign their zones using DNSSEC and for resolver operators to validate the responses. When this is not an option, operators who are more concerned with response integrity than with flood mitigation may consider setting **slip** to 1, causing all rate-limited responses to be truncated rather than dropped. This reduces the effectiveness of rate-limiting against reflection attacks.)

When the approximate query-per-second rate exceeds the **qps-scale** value, the **responses-per-second**, **errors-per-second**, **nxdomains-per-second**, and **all-per-second** values are reduced by the ratio of the current rate to the **qps-scale** value. This feature can tighten defenses during attacks. For example, with **qps-scale 250; responses-per-second 20;** and a total query rate of 1000 queries/second for all queries from all DNS clients including via TCP, then the effective responses/second limit changes to (250/1000)*20, or 5. Responses sent via TCP are not limited but are counted to compute the query-per-second rate.

Communities of DNS clients can be given their own parameters or no rate limiting by putting **rate-limit** statements in **view** statements instead of in the global **option** statement. A **rate-limit** statement in a view replaces, rather than supplements, a **rate-limit** statement among the main options. DNS clients within a view can be exempted from rate limits with the **exempt-clients** clause.

UDP responses of all kinds can be limited with the **all-per-second** phrase. This rate limiting is unlike the rate limiting provided by **responses-per-second**, **errors-per-second**, and **nxdomains-per-second** on a DNS server, which are often invisible to the victim of a DNS reflection attack. Unless the forged requests of the attack are the same as the legitimate requests of the victim, the victim's requests are not affected. Responses affected by an **all-per-second** limit are always dropped; the **slip** value has no effect. An **all-per-second** limit should be at least 4 times as large as the other limits, because single DNS clients often send bursts of legitimate requests. For example, the receipt of a single mail message can prompt requests from an SMTP server for NS, PTR, A, and AAAA records as the incoming SMTP/TCP/IP connection is considered. The SMTP server can need additional NS, A, AAAA, MX, TXT, and SPF records as it considers the SMTP **Mail From** command. Web browsers often repeatedly resolve the same names that are duplicated in HTML <IMG> tags in a page. **all-per-second** is similar to the rate limiting offered by firewalls but is often inferior. Attacks that justify ignoring the contents of DNS responses are likely to be attacks on the DNS server itself. They usually should be discarded before the DNS server spends resources make TCP connections or parsing DNS requests, but that rate limiting must be done before the DNS server sees the requests.

The maximum size of the table used to track requests and rate-limit responses is set with **max-table-size**. Each entry in the table is between 40 and 80 bytes. The table needs approximately as many entries as the number of requests received per second. The default is 20,000. To reduce the cold start of growing the table, **min-table-size** (default 500) can set the minimum table size. Enable **rate-limit** category logging to monitor expansions of the table and inform choices for the initial and maximum table size.

Use **log-only yes** to test rate-limiting parameters without actually dropping any requests.

Responses dropped by rate limits are included in the **RateDropped** and **QryDropped** statistics. Responses that truncated by rate limits are included in **RateSlipped** and **RespTruncated**.

NXDOMAIN Redirection

named supports NXDOMAIN redirection via two methods:

- Redirect zone Section 6.2
- Redirect namespace

With either method, when **named** gets an NXDOMAIN response it examines a separate namespace to see if the NXDOMAIN response should be replaced with an alternative response.

With a redirect zone (**zone "." { type redirect; };**), the data used to replace the NXDOMAIN is held in a single zone which is not part of the normal namespace. All the redirect information is contained in the zone; there are no delegations.

With a redirect namespace (**option { nxdomain-redirect <suffix> };**), the data used to replace the NXDOMAIN is part of the normal namespace and is looked up by appending the specified suffix to the original query name. This roughly doubles the cache required to process NXDOMAIN responses, as both the original NXDOMAIN response and the replacement data (or a NXDOMAIN indicating that there is no replacement) must be stored.

If both a redirect zone and a redirect namespace are configured, the redirect zone is tried first.

server Statement Grammar

```
server netprefix {
  bogus boolean;
  edns boolean;
  edns-udp-size integer;
  edns-version integer;
  keys server_key;
  max-udp-size integer;
  notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
      [ dscp integer ];
  provide-ixfr boolean;
  query-source ( ( [ address ] ( ipv4_address | * ) [ port (
      integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
      port ( integer | * ) ) ) [ dscp integer ];
  query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
      integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
      port ( integer | * ) ) ) [ dscp integer ];
  request-expire boolean;
  request-ixfr boolean;
  request-nsid boolean;
  send-cookie boolean;
  tcp-only boolean;
  transfer-format ( many-answers | one-answer );
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
```

```
      ] [ dscp integer ];
  transfers integer;
};
```

server Statement Definition and Usage

The **server** statement defines characteristics to be associated with a remote name server. If a prefix length is specified, then a range of servers is covered. Only the most specific server clause applies, regardless of the order in `named.conf`.

The **server** statement can occur at the top level of the configuration file or inside a **view** statement. If a **view** statement contains one or more **server** statements, only those apply to the view and any top-level ones are ignored. If a view contains no **server** statements, any top-level **server** statements are used as defaults.

If a remote server is giving out bad data, marking it as bogus prevents further queries to it. The default value of **bogus** is **no**.

The **provide-ixfr** clause determines whether the local server, acting as primary, responds with an incremental zone transfer when the given remote server, a secondary, requests it. If set to **yes**, incremental transfer is provided whenever possible. If set to **no**, all transfers to the remote server are non-incremental. If not set, the value of the **provide-ixfr** option in the view or global options block is used as a default.

The **request-ixfr** clause determines whether the local server, acting as a secondary, requests incremental zone transfers from the given remote server, a primary. If not set, the value of the **request-ixfr** option in the view or global options block is used as a default. It may also be set in the zone block; if set there, it overrides the global or view setting for that zone.

IXFR requests to servers that do not support IXFR automatically fall back to AXFR. Therefore, there is no need to manually list which servers support IXFR and which ones do not; the global default of **yes** should always work. The purpose of the **provide-ixfr** and **request-ixfr** clauses is to make it possible to disable the use of IXFR even when both primary and secondary claim to support it: for example, if one of the servers is buggy and crashes or corrupts data when IXFR is used.

The **request-expire** clause determines whether the local server, when acting as a secondary, requests the EDNS EXPIRE value. The EDNS EXPIRE value indicates the remaining time before the zone data expires and needs to be refreshed. This is used when a secondary server transfers a zone from another secondary server; when transferring from the primary, the expiration timer is set from the EXPIRE field of the SOA record instead. The default is **yes**.

The **edns** clause determines whether the local server attempts to use EDNS when communicating with the remote server. The default is **yes**.

The **edns-udp-size** option sets the EDNS UDP size that is advertised by **named** when querying the remote server. Valid values are 512 to 4096 bytes; values outside this range are silently adjusted to the nearest value within it. This option is useful when advertising a different value to this server than the value advertised globally: for example, when there is a firewall at the remote site that is blocking large replies. Note: currently, this sets a single UDP size for all packets sent to the server; **named** does not deviate from this value. This differs from the behavior of **edns-udp-size** in **options** or **view** statements, where it specifies a maximum value. The **server**

statement behavior may be brought into conformance with the **options/view** behavior in future releases.

The **edns-version** option sets the maximum EDNS VERSION that is sent to the server(s) by the resolver. The actual EDNS version sent is still subject to normal EDNS version-negotiation rules (see RFC 6891), the maximum EDNS version supported by the server, and any other heuristics that indicate that a lower version should be sent. This option is intended to be used when a remote server reacts badly to a given EDNS version or higher; it should be set to the highest version the remote server is known to support. Valid values are 0 to 255; higher values are silently adjusted. This option is not needed until higher EDNS versions than 0 are in use.

The **max-udp-size** option sets the maximum EDNS UDP message size **named** sends. Valid values are 512 to 4096 bytes; values outside this range are silently adjusted. This option is useful when there is a firewall that is blocking large replies from **named**.

The **tcp-only** option sets the transport protocol to TCP. The default is to use the UDP transport and to fallback on TCP only when a truncated response is received.

The server supports two zone transfer methods. The first, **one-answer**, uses one DNS message per resource record transferred. **many-answers** packs as many resource records as possible into a single message, which is more efficient. It is possible to specify which method to use for a server via the **transfer-format** option; If not set there, the **transfer-format** specified by the **options** statement is used.

transfers is used to limit the number of concurrent inbound zone transfers from the specified server. If no **transfers** clause is specified, the limit is set according to the **transfers-per-ns** option.

The **keys** clause identifies a **key_id** defined by the **key** statement, to be used for transaction security (TSIG, Section 4.5) when talking to the remote server. When a request is sent to the remote server, a request signature is generated using the key specified here and appended to the message. A request originating from the remote server is not required to be signed by this key.

Only a single key per server is currently supported.

The **transfer-source** and **transfer-source-v6** clauses specify the IPv4 and IPv6 source address, respectively, to be used for zone transfer with the remote server. For an IPv4 remote server, only **transfer-source** can be specified. Similarly, for an IPv6 remote server, only **transfer-source-v6** can be specified. For more details, see the description of **transfer-source** and **transfer-source-v6** in Section 6.2.

The **notify-source** and **notify-source-v6** clauses specify the IPv4 and IPv6 source address, respectively, to be used for notify messages sent to remote servers. For an IPv4 remote server, only **notify-source** can be specified. Similarly, for an IPv6 remote server, only **notify-source-v6** can be specified.

The **query-source** and **query-source-v6** clauses specify the IPv4 and IPv6 source address, respectively, to be used for queries sent to remote servers. For an IPv4 remote server, only **query-source** can be specified. Similarly, for an IPv6 remote server, only **query-source-v6** can be specified.

The **request-nsid** clause determines whether the local server adds an NSID EDNS option to requests sent to the server. This overrides **request-nsid** set at the view or option level.

The **send-cookie** clause determines whether the local server adds a COOKIE EDNS option to requests sent to the server. This overrides **send-cookie** set at the view or option level. The

named server may determine that COOKIE is not supported by the remote server and not add a COOKIE EDNS option to requests.

statistics-channels Statement Grammar

```
statistics-channels {
  inet ( ipv4_address | ipv6_address |
       * ) [ port ( integer | * ) ] [
       allow { address_match_element; ...
       } ];
};
```

statistics-channels Statement Definition and Usage

The **statistics-channels** statement declares communication channels to be used by system administrators to get access to statistics information on the name server.

This statement is intended to be flexible to support multiple communication protocols in the future, but currently only HTTP access is supported. It requires that BIND 9 be compiled with libxml2 and/or json-c (also known as libjson0); the **statistics-channels** statement is still accepted even if it is built without the library, but any HTTP access fails with an error.

An **inet** control channel is a TCP socket listening at the specified **ip_port** on the specified **ip_addr**, which can be an IPv4 or IPv6 address. An **ip_addr** of * (asterisk) is interpreted as the IPv4 wildcard address; connections are accepted on any of the system's IPv4 addresses. To listen on the IPv6 wildcard address, use an **ip_addr** of **::**.

If no port is specified, port 80 is used for HTTP channels. The asterisk (*) cannot be used for **ip_port**.

Attempts to open a statistics channel are restricted by the optional **allow** clause. Connections to the statistics channel are permitted based on the **address_match_list**. If no **allow** clause is present, **named** accepts connection attempts from any address; since the statistics may contain sensitive internal information, it is highly recommended to restrict the source of connection requests appropriately.

If no **statistics-channels** statement is present, **named** does not open any communication channels.

The statistics are available in various formats and views, depending on the URI used to access them. For example, if the statistics channel is configured to listen on 127.0.0.1 port 8888, then the statistics are accessible in XML format at http://127.0.0.1:8888/ or http://127.0.0.1:8888/xml. A CSS file is included, which can format the XML statistics into tables when viewed with a stylesheet-capable browser, and into charts and graphs using the Google Charts API when using a JavaScript-capable browser.

Broken-out subsets of the statistics can be viewed at http://127.0.0.1:8888/xml/v3/status (server uptime and last reconfiguration time), http://127.0.0.1:8888/xml/v3/server (server and resolver statistics), http://127.0.0.1:8888/xml/v3/zones (zone statistics), http://127.0.0.1:8888/xml/v3/net (network status and socket statistics), http://127.0.0.1:8888/xml/v3/mem (memory manager statistics), http://127.0.0.1:8888/xml/v3/tasks (task manager statistics), and http://127.0.0.1:8888/xml/v3/traffic (traffic sizes).

The full set of statistics can also be read in JSON format at http://127.0.0.1:8888/json, with the broken-out subsets at http://127.0.0.1:8888/json/v1/status (server uptime and last reconfiguration time), http://127.0.0.1:8888/json/v1/server (server and resolver statistics), http://127.0.0.1:8888/-json/v1/zones (zone statistics), http://127.0.0.1:8888/json/v1/net (network status and socket statistics), http://127.0.0.1:8888/json/v1/mem (memory manager statistics), http://127.0.0.1:8888/-json/v1/tasks (task manager statistics), and http://127.0.0.1:8888/json/v1/traffic (traffic sizes).

trusted-keys Statement Grammar

```
trusted-keys { string integer integer
    integer quoted_string; ... };
```

trusted-keys Statement Definition and Usage

The **trusted-keys** statement defines DNSSEC security roots. DNSSEC is described in Section 4.8. A security root is defined when the public key for a non-authoritative zone is known, but cannot be securely obtained through DNS, either because it is the DNS root zone or because its parent zone is unsigned. Once a key has been configured as a trusted key, it is treated as if it has been validated and proven secure. The resolver attempts DNSSEC validation on all DNS data in subdomains of a security root.

All keys (and corresponding zones) listed in **trusted-keys** are deemed to exist regardless of what parent zones say. Similarly, for all keys listed in **trusted-keys**, only those keys are used to validate the DNSKEY RRset. The parent's DS RRset is not used.

The **trusted-keys** statement can contain multiple key entries, each consisting of the key's domain name, flags, protocol, and algorithm, and the Base64 representation of the key data. Spaces, tabs, newlines, and carriage returns are ignored in the key data, so the configuration may be split into multiple lines.

trusted-keys may be set at the top level of `named.conf` or within a view. If it is set in both places, they are additive; keys defined at the top level are inherited by all views, but keys defined in a view are only used within that view.

Validation below specified names can be temporarily disabled by using **rndc nta**.

managed-keys Statement Grammar

```
managed-keys { string string integer
    integer integer quoted_string; ... };
```

managed-keys Statement Definition and Usage

The **managed-keys** statement, like **trusted-keys**, defines DNSSEC security roots. The difference is that **managed-keys** can be kept up-to-date automatically, without intervention from the resolver operator.

Suppose, for example, that a zone's key-signing key was compromised, and the zone owner had to revoke and replace the key. A resolver which had the old key in a **trusted-keys** statement would be unable to validate this zone; it would reply with a SERVFAIL response code. This would continue until the resolver operator updated the **trusted-keys** statement with the new key.

If, however, the zone were listed in a **managed-keys** statement instead, the zone owner could add a "stand-by" key to the zone in advance. **named** would store the stand-by key, and when the original key was revoked, **named** would be able to transition smoothly to the new key. It would also recognize that the old key had been revoked and cease using that key to validate answers, minimizing the damage that the compromised key could do.

A **managed-keys** statement contains a list of the keys to be managed, along with information about how the keys are to be initialized for the first time. The only initialization method currently supported is `initial-key`. This means the **managed-keys** statement must contain a copy of the initializing key. (Future releases may allow keys to be initialized by other methods, eliminating this requirement.)

Consequently, a **managed-keys** statement appears similar to a **trusted-keys** statement, differing by the presence of the second field, which contains the keyword `initial-key`. The difference is, whereas the keys listed in a **trusted-keys** continue to be trusted until they are removed from `named.conf`, an initializing key listed in a **managed-keys** statement is only trusted *once*: for as long as it takes to load the managed-key database and start the RFC 5011 key-maintenance process.

The first time **named** runs with a managed key configured in `named.conf`, it fetches the DNSKEY RRset directly from the zone apex, and validates it using the key specified in the **managed-keys** statement. If the DNSKEY RRset is validly signed, then it is used as the basis for a new managed-keys database.

From that point on, whenever **named** runs, it sees the **managed-keys** statement, checks to make sure RFC 5011 key maintenance has already been initialized for the specified domain, and if so, simply moves on. The key specified in the **managed-keys** statement is not used to validate answers; it is superseded by the key or keys stored in the managed-keys database.

The next time **named** runs after a name has been *removed* from the **managed-keys** statement, the corresponding zone is removed from the managed-keys database, and RFC 5011 key maintenance is no longer used for that domain.

In the current implementation, the managed-keys database is stored as a master-format zone file.

On servers which do not use views, this file is named `managed-keys.bind`. When views are in use, there is a separate managed-keys database for each view; the filename is the view name (or, if a view name contains characters which would make it illegal as a filename, a hash of the view name), followed by the suffix `.mkeys`.

When the key database is changed, the zone is updated. As with any other dynamic zone, changes are written into a journal file, e.g., `managed-keys.bind.jnl` or `internal.mkeys.jnl`. Changes are committed to the zone file as soon as possible afterward, usually within 30 seconds. Whenever **named** is using automatic key maintenance, the zone file and journal file can be expected to exist in the working directory. (For this reason, among others, the working directory should be always be writable by **named**.)

If the **dnssec-validation** option is set to **auto**, **named** automatically initializes a managed key for the root zone. The key that is used to initialize the key-maintenance process is stored in `bind.keys`; the location of this file can be overridden with the **bindkeys-file** option. As a fallback in the event no `bind.keys` can be found, the initializing key is also compiled directly into **named**.

view Statement Grammar

```
view view_name [ class ] {
    match-clients { address_match_list } ;
    match-destinations { address_match_list } ;
    match-recursive-only yes_or_no ;
  [ view_option ; ... ]
  [ zone_statement ; ... ]
} ;
```

view Statement Definition and Usage

The **view** statement is a powerful feature of BIND 9 that lets a name server answer a DNS query differently depending on who is asking. It is particularly useful for implementing split DNS setups without having to run multiple servers.

Each **view** statement defines a view of the DNS namespace that is seen by a subset of clients. A client matches a view if its source IP address matches the `address_match_list` of the view's **match-clients** clause and its destination IP address matches the `address_match_list` of the view's **match-destinations** clause. If not specified, both **match-clients** and **match-destinations** default to matching all addresses. In addition to checking IP addresses, **match-clients** and **match-destinations** can also take **keys** which provide an mechanism for the client to select the view. A view can also be specified as **match-recursive-only**, which means that only recursive requests from matching clients match that view. The order of the **view** statements is significant; a client request is resolved in the context of the first **view** that it matches.

Zones defined within a **view** statement are only accessible to clients that match the **view**. By defining a zone of the same name in multiple views, different zone data can be given to different clients: for example, "internal" and "external" clients in a split DNS setup.

Many of the options given in the **options** statement can also be used within a **view** statement, and then apply only when resolving queries with that view. When no view-specific value is given, the value in the **options** statement is used as a default. Also, zone options can have default values specified in the **view** statement; these view-specific defaults take precedence over those in the **options** statement.

Views are class-specific. If no class is given, class IN is assumed. Note that all non-IN views must contain a hint zone, since only the IN class has compiled-in default hints.

If there are no **view** statements in the config file, a default view that matches any client is automatically created in class IN. Any **zone** statements specified on the top level of the configuration file are considered to be part of this default view, and the **options** statement applies to the default view. If any explicit **view** statements are present, all **zone** statements must occur inside **view** statements.

Here is an example of a typical split DNS setup implemented using **view** statements:

```
view "internal" {
      // This should match our internal networks.
      match-clients { 10.0.0.0/8; };

      // Provide recursive service to internal
      // clients only.
      recursion yes;

      // Provide a complete view of the example.com
      // zone including addresses of internal hosts.
      zone "example.com" {
            type master;
            file "example-internal.db";
      };
};

view "external" {
      // Match all clients not matched by the
      // previous view.
      match-clients { any; };

      // Refuse recursive service to external clients.
      recursion no;

      // Provide a restricted view of the example.com
      // zone containing only publicly accessible hosts.
      zone "example.com" {
           type master;
           file "example-external.db";
      };
};
```

zone Statement Grammar

```
zone string [ class ] {
  type ( master | primary );
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  allow-transfer { address_match_element; ... };
  allow-update { address_match_element; ... };
  also-notify [ port integer ] [ dscp integer ] { ( masters | ipv4_address ↩
       [ port integer ] | ipv6_address [ port integer ] ) [ key string ]; ↩
      ... };
  alt-transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp ↩
       integer ];
  alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ ↩
      dscp integer ];
  auto-dnssec ( allow | maintain | off );
  check-dup-records ( fail | warn | ignore );
  check-integrity boolean;
  check-mx ( fail | warn | ignore );
```

```
  check-mx-cname ( fail | warn | ignore );
  check-names ( fail | warn | ignore );
  check-sibling boolean;
  check-spf ( warn | ignore );
  check-srv-cname ( fail | warn | ignore );
  check-wildcard boolean;
  database string;
  dialup ( notify | notify-passive | passive | refresh | boolean );
  dlz string;
  dnssec-dnskey-kskonly boolean;
  dnssec-loadkeys-interval integer;
  dnssec-secure-to-insecure boolean;
  dnssec-update-mode ( maintain | no-resign );
  file quoted_string;
  forward ( first | only );
  forwarders [ port integer ] [ dscp integer ] { ( ipv4_address |
      ipv6_address ) [ port integer ] [ dscp integer ]; ... };
  inline-signing boolean;
  ixfr-from-differences boolean;
  journal quoted_string;
  key-directory quoted_string;
  masterfile-format ( map | raw | text );
  masterfile-style ( full | relative );
  max-journal-size ( unlimited | sizeval );
  max-records integer;
  max-transfer-idle-out integer;
  max-transfer-time-out integer;
  max-zone-ttl ( unlimited | ttlval );
  notify ( explicit | master-only | boolean );
  notify-delay integer;
  notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp
      integer ];
  notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ dscp
      integer ];
  notify-to-soa boolean;
  serial-update-method ( date | increment | unixtime );
  sig-signing-nodes integer;
  sig-signing-signatures integer;
  sig-signing-type integer;
  sig-validity-interval integer [ integer ];
  update-check-ksk boolean;
  update-policy ( local | { ( deny | grant ) string ( 6to4-self | external
       | krb5-self | krb5-selfsub | krb5-subdomain | ms-self | ms-selfsub
      | ms-subdomain | name | self | selfsub | selfwild | subdomain | tcp-
      self | wildcard | zonesub ) [ string ] rrtypelist; ... };
  zero-no-soa-ttl boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

```
zone string [ class ] {
  type ( slave | secondary );
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
```

```
allow-query-on { address_match_element; ... };
allow-transfer { address_match_element; ... };
allow-update-forwarding { address_match_element; ... };
also-notify [ port integer ] [ dscp integer ] { ( masters | ipv4_address ↩
    [ port integer ] | ipv6_address [ port integer ] ) [ key string ]; ↩
    ... };
alt-transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp ↩
    integer ];
alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ ↩
    dscp integer ];
auto-dnssec ( allow | maintain | off );
check-names ( fail | warn | ignore );
database string;
dialup ( notify | notify-passive | passive | refresh | boolean );
dlz string;
dnssec-dnskey-kskonly boolean;
dnssec-loadkeys-interval integer;
dnssec-update-mode ( maintain | no-resign );
file quoted_string;
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { ( ipv4_address | ↩
    ipv6_address ) [ port integer ] [ dscp integer ]; ... };
inline-signing boolean;
ixfr-from-differences boolean;
journal quoted_string;
key-directory quoted_string;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
masters [ port integer ] [ dscp integer ] { ( masters | ipv4_address [ ↩
    port integer ] | ipv6_address [ port integer ] ) [ key string ]; ... ↩
    };
max-journal-size ( unlimited | sizeval );
max-records integer;
max-refresh-time integer;
max-retry-time integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
min-refresh-time integer;
min-retry-time integer;
multi-master boolean;
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp ↩
    integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ dscp ↩
    integer ];
notify-to-soa boolean;
request-expire boolean;
request-ixfr boolean;
sig-signing-nodes integer;
sig-signing-signatures integer;
```

```
  sig-signing-type integer;
  sig-validity-interval integer [ integer ];
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp ↩
      integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ dscp ↩
      integer ];
  try-tcp-refresh boolean;
  update-check-ksk boolean;
  use-alt-transfer-source boolean;
  zero-no-soa-ttl boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

```
zone string [ class ] {
  type hint;
  check-names ( fail | warn | ignore );
  delegation-only boolean;
  file quoted_string;
};
```

```
zone string [ class ] {
  type stub;
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  check-names ( fail | warn | ignore );
  database string;
  delegation-only boolean;
  dialup ( notify | notify-passive | passive | refresh | boolean );
  file quoted_string;
  forward ( first | only );
  forwarders [ port integer ] [ dscp integer ] { ( ipv4_address | ↩
      ipv6_address ) [ port integer ] [ dscp integer ]; ... };
  masterfile-format ( map | raw | text );
  masterfile-style ( full | relative );
  masters [ port integer ] [ dscp integer ] { ( masters | ipv4_address [ ↩
      port integer ] | ipv6_address [ port integer ] ) [ key string ]; ... ↩
       };
  max-records integer;
  max-refresh-time integer;
  max-retry-time integer;
  max-transfer-idle-in integer;
  max-transfer-time-in integer;
  min-refresh-time integer;
  min-retry-time integer;
  multi-master boolean;
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [ dscp ↩
      integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ] [ dscp ↩
      integer ];
  use-alt-transfer-source boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

```
zone string [ class ] {
  type static-stub;
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  forward ( first | only );
  forwarders [ port integer ] [ dscp integer ] { ( ipv4_address |
      ipv6_address ) [ port integer ] [ dscp integer ]; ... };
  max-records integer;
  server-addresses { ( ipv4_address | ipv6_address ); ... };
  server-names { quoted_string; ... };
  zone-statistics ( full | terse | none | boolean );
};
```

```
zone string [ class ] {
  type forward;
  delegation-only boolean;
  forward ( first | only );
  forwarders [ port integer ] [ dscp integer ] { ( ipv4_address |
      ipv6_address ) [ port integer ] [ dscp integer ]; ... };
};
```

```
zone string [ class ] {
  type redirect;
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  dlz string;
  file quoted_string;
  masterfile-format ( map | raw | text );
  masterfile-style ( full | relative );
  masters [ port integer ] [ dscp integer ] { ( masters | ipv4_address [
      port integer ] | ipv6_address [ port integer ] ) [ key string ]; ...
       };
  max-records integer;
  max-zone-ttl ( unlimited | ttlval );
  zone-statistics ( full | terse | none | boolean );
};
```

```
zone string [ class ] {
  type delegation-only;
};
```

```
zone string [ class ] {
  in-view string;
};
```

zone Statement Definition and Usage

Zone Types

The **type** keyword is required for the **zone** configuration unless it is an **in-view** configuration. Its acceptable values are: `master`, `slave`, `hint`, `stub`, `static-stub`, `forward`, `redirect`, or `delegation-only`.

> NOTE
>
> Later versions of BIND added **type primary** and **type secondary** as synonyms for **type master** and **type slave**, as those terms are in more common use now. BIND 9.11's configuration syntax predates this change.

`master`	The server has a master copy of the data for the zone and is able to provide authoritative answers for it.
`slave`	A secondary zone, replicating a primary zone provided by another authoritative server. The **masters** list specifies one or more IP addresses of primary servers that the secondary contacts to update its copy of the zone. Masters list elements can also be names of other masters lists. By default, transfers are made from port 53 on the servers; this can be changed for all servers by specifying a port number before the list of IP addresses, or on a per-server basis after the IP address. Authentication to the primary can also be done with per-server TSIG keys. If a file is specified, then the replica is written to this file whenever the zone is changed, and reloaded from this file on a server restart. Use of a file is recommended, since it often speeds server startup and eliminates a needless waste of bandwidth. Note that for large numbers (in the tens or hundreds of thousands) of zones per server, it is best to use a two-level naming scheme for zone filenames. For example, a secondary server for the zone `example.com` might place the zone contents into a file called `ex/example.com` where `ex/` is just the first two letters of the zone name. (Most operating systems behave very slowly if there are 100000 files in a single directory.)
`hint`	The initial set of root name servers is specified using a hint zone. When the server starts, it uses the root hints 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 hints. Classes other than IN have no built-in default hints.

`stub`	A stub zone is similar to a secondary zone, except that it replicates only the NS records of a primary 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. Stub zones can be used to eliminate the need for a glue NS record in a parent zone, at the expense of maintaining a stub zone entry and a set of name server addresses in `named.conf`. This usage is not recommended for new configurations, and BIND 9 supports it only in a limited way. If a BIND 9 primary, serving a parent zone, has child stub zones configured, all the secondary servers for the parent zone also need to have the same child stub zones configured. Stub zones can also be used as a way to force the resolution of a given domain to use a particular set of authoritative servers. For example, the caching name servers on a private network using RFC 1918 addressing may be configured with stub zones for `10.in-addr.arpa` to use a set of internal name servers as the authoritative servers for that domain.
`static-stub`	A static-stub zone is similar to a stub zone with the following exceptions: the zone data is statically configured, rather than transferred from a primary server; and when recursion is necessary for a query that matches a static-stub zone, the locally configured data (name server names and glue addresses) is always used, even if different authoritative information is cached. Zone data is configured via the **server-addresses** and **server-names** zone options. The zone data is maintained in the form of NS and (if necessary) glue A or AAAA RRs internally, which can be seen by dumping zone databases by **rndc dumpdb -all**. The configured RRs are considered local configuration parameters rather than public data. Non-recursive queries (i.e., those with the RD bit off) to a static-stub zone are therefore prohibited and are responded to with REFUSED. Since the data is statically configured, no zone maintenance action takes place for a static-stub zone. For example, there is no periodic refresh attempt, and an incoming notify message will be rejected with an rcode of NOTAUTH. Each static-stub zone is configured with internally generated NS and (if necessary) glue A or AAAA RRs.

`forward` A forward zone is a way to configure forwarding on a per-domain basis. A **zone** statement of type **forward** can contain a **forward** and/or **forwarders** statement, which applies to queries within the domain given by the zone name. If no **forwarders** statement is present, or an empty list for **forwarders** is given, then no forwarding is done for the domain, canceling the effects of any forwarders in the **options** statement. Thus, to use this type of zone to change the behavior of the global **forward** option (that is, "forward first" to, then "forward only", or vice versa), but use the same servers as set globally, re-specify the global forwarders.

`redirect` Redirect zones are used to provide answers to queries when normal resolution would result in NXDOMAIN being returned. Only one redirect zone is supported per view. **allow-query** can be used to restrict which clients see these answers.
If the client has requested DNSSEC records (DO=1) and the NXDOMAIN response is signed, no substitution occurs.
To redirect all NXDOMAIN responses to 100.100.100.2 and 2001:ffff:ffff::100.100.100.2, configure a type `redirect` zone named ".", with the zone file containing wildcard records that point to the desired addresses: `*. IN A 100.100.100.2` and `*. IN AAAA 2001:ffff:ffff::100.100.100.2`.
As another example, to redirect all Spanish names (under .ES), use similar entries but with the names "*.ES." instead of "*.". To redirect all commercial Spanish names (under COM.ES), use wildcard entries called "*.COM.ES.".
Note that the redirect zone supports all possible types; it is not limited to A and AAAA records.
Because redirect zones are not referenced directly by name, they are not kept in the zone lookup table with normal primary and secondary zones. Consequently, it is not currently possible to use **rndc reload *zonename*** to reload a redirect zone. However, when using **rndc reload** without specifying a zone name, redirect zones are reloaded along with other zones.

`delegation-only` This zone type is used to enforce the delegation-only status of infrastructure zones (e.g., COM, NET, ORG). Any answer that is received without an explicit or implicit delegation in the authority section is treated as NXDOMAIN. This does not apply to the zone apex, and should not be applied to leaf zones.
`delegation-only` has no effect on answers received from forwarders.
See caveats in root-delegation-only.

`in-view`	When using multiple views, a primary or secondary zone configured in one view can be referenced in a subsequent view. This allows both views to serve the same zone without the overhead of loading it more than once. This is configured using a `zone` statement, with an `in-view` option specifying the view in which the zone is defined. A `zone` statement containing `in-view` does not need to specify a type, since that is part of the zone definition in the other view. See Section 6.2 for more information.

Class

The zone's name may optionally be followed by a class. If a class is not specified, class `IN` (for `Internet`), is assumed. This is correct for the vast majority of cases.

The `hesiod` class is named for an information service from MIT's Project Athena. It was used to share information about various systems databases, such as users, groups, printers, and so on. The keyword `HS` is a synonym for hesiod.

Another MIT development is Chaosnet, a LAN protocol created in the mid-1970s. Zone data for it can be specified with the `CHAOS` class.

Zone Options

allow-notify
See the description of **allow-notify** in Section 6.2.

allow-query
See the description of **allow-query** in Section 6.2.

allow-query-on
See the description of **allow-query-on** in Section 6.2.

allow-transfer
See the description of **allow-transfer** in Section 6.2.

allow-update
See the description of **allow-update** in Section 6.2.

update-policy
This specifies a "Simple Secure Update" policy. See Section 6.2.

allow-update-forwarding
See the description of **allow-update-forwarding** in Section 6.2.

also-notify
This option is only meaningful if **notify** is active for this zone. The set of machines that receive a `DNS NOTIFY` message for this zone is made up of all the listed name servers (other than the primary) for the zone, plus any IP addresses specified with **also-notify**. A port may be specified with each **also-notify** address to send the notify messages to a port other than the default of 53. A TSIG key may also be specified to cause the `NOTIFY` to be

signed by the given key. **also-notify** is not meaningful for stub zones. The default is the empty list.

check-names
This option is used to restrict the character set and syntax of certain domain names in zone files and/or DNS responses received from the network. The default varies according to zone type. For primary zones the default is **fail**; for secondary zones the default is **warn**. It is not implemented for **hint** zones.

check-mx
See the description of **check-mx** in Section 6.2.

check-spf
See the description of **check-spf** in Section 6.2.

check-wildcard
See the description of **check-wildcard** in Section 6.2.

check-integrity
See the description of **check-integrity** in Section 6.2.

check-sibling
See the description of **check-sibling** in Section 6.2.

zero-no-soa-ttl
See the description of **zero-no-soa-ttl** in Section 6.2.

update-check-ksk
See the description of **update-check-ksk** in Section 6.2.

dnssec-loadkeys-interval
See the description of **dnssec-loadkeys-interval** in Section 6.2.

dnssec-update-mode
See the description of **dnssec-update-mode** in Section 6.2.

dnssec-dnskey-kskonly
See the description of **dnssec-dnskey-kskonly** in Section 6.2.

try-tcp-refresh
See the description of **try-tcp-refresh** in Section 6.2.

database
This specifies the type of database to be used to store the zone data. The string following the **database** keyword is interpreted as a list of whitespace-delimited words. The first word identifies the database type, and any subsequent words are passed as arguments to the database to be interpreted in a way specific to the database type.

The default is **"rbt"**, BIND 9's native in-memory red-black tree database. This database does not take arguments.

Other values are possible if additional database drivers have been linked into the server. Some sample drivers are included with the distribution but none are linked in by default.

dialup
See the description of **dialup** in Section 6.2.

delegation-only
This flag only applies to forward, hint, and stub zones. If set to **yes**, then the zone is treated as if it is also a delegation-only type zone.

See caveats in root-delegation-only.

file
This sets the zone's filename. In **master**, **hint**, and **redirect** zones which do not have **masters** defined, zone data is loaded from this file. In **slave**, **stub**, and **redirect** zones which do have **masters** defined, zone data is retrieved from another server and saved in this file. This option is not applicable to other zone types.

forward
This option is only meaningful if the zone has a forwarders list. The **only** value causes the lookup to fail after trying the forwarders and getting no answer, while **first** allows a normal lookup to be tried.

forwarders
This is used to override the list of global forwarders. If it is not specified in a zone of type **forward**, no forwarding is done for the zone and the global options are not used.

ixfr-base
This was used in BIND 8 to specify the name of the transaction log (journal) file for dynamic update and IXFR. BIND 9 ignores the option and constructs the name of the journal file by appending "`.jnl`" to the name of the zone file.

ixfr-tmp-file
This was an undocumented option in BIND 8. It is ignored in BIND 9.

journal
This allows the default journal's filename to be overridden. The default is the zone's filename with "`.jnl`" appended. This is applicable to primary (**master**) and secondary (**slave**) zones.

max-journal-size
See the description of **max-journal-size** in Section 6.2.

max-records
See the description of **max-records** in Section 6.2.

max-transfer-time-in
See the description of **max-transfer-time-in** in Section 6.2.

max-transfer-idle-in
See the description of **max-transfer-idle-in** in Section 6.2.

max-transfer-time-out
See the description of **max-transfer-time-out** in Section 6.2.

max-transfer-idle-out
See the description of **max-transfer-idle-out** in Section 6.2.

notify
See the description of **notify** in Section 6.2.

notify-delay
See the description of **notify-delay** in Section 6.2.

notify-to-soa
See the description of **notify-to-soa** in Section 6.2.

pubkey
In BIND 8, this option was intended to specify a public zone key for verification of signatures in DNSSEC-signed zones when they were loaded from disk. BIND 9 does not verify signatures on load and ignores the option.

zone-statistics
See the description of **zone-statistics** in Section 6.2.

server-addresses
This option is only meaningful for static-stub zones. This is a list of IP addresses to which queries should be sent in recursive resolution for the zone. A non-empty list for this option internally configures the apex NS RR with associated glue A or AAAA RRs.

For example, if "example.com" is configured as a static-stub zone with 192.0.2.1 and 2001:db8::1234 in a **server-addresses** option, the following RRs are internally configured:

```
example.com. NS example.com.
example.com. A 192.0.2.1
example.com. AAAA 2001:db8::1234
```

These records are used internally to resolve names under the static-stub zone. For instance, if the server receives a query for "www.example.com" with the RD bit on, the server initiates recursive resolution and sends queries to 192.0.2.1 and/or 2001:db8::1234.

server-names
This option is only meaningful for static-stub zones. This is a list of domain names of name servers that act as authoritative servers of the static-stub zone. These names are resolved to IP addresses when **named** needs to send queries to these servers. For this supplemental resolution to be successful, these names must not be a subdomain of the origin name of the static-stub zone. That is, when "example.net" is the origin of a static-stub zone, "ns.example" and "master.example.com" can be specified in the **server-names** option, but "ns.example.net" cannot; it is rejected by the configuration parser.

A non-empty list for this option internally configures the apex NS RR with the specified names. For example, if "example.com" is configured as a static-stub zone with "ns1.example.net" and "ns2.example.net" in a **server-names** option, the following RRs are internally configured:

```
example.com. NS ns1.example.net.
example.com. NS ns2.example.net.
```

These records are used internally to resolve names under the static-stub zone. For instance, if the server receives a query for "www.example.com" with the RD bit on, the server initiates recursive resolution, resolves "ns1.example.net" and/or "ns2.example.net" to IP addresses, and then sends queries to one or more of these addresses.

sig-validity-interval
See the description of **sig-validity-interval** in Section 6.2.

sig-signing-nodes
See the description of **sig-signing-nodes** in Section 6.2.

sig-signing-signatures
See the description of **sig-signing-signatures** in Section 6.2.

sig-signing-type
See the description of **sig-signing-type** in Section 6.2.

transfer-source
See the description of **transfer-source** in Section 6.2.

transfer-source-v6
See the description of **transfer-source-v6** in Section 6.2.

alt-transfer-source
See the description of **alt-transfer-source** in Section 6.2.

alt-transfer-source-v6
See the description of **alt-transfer-source-v6** in Section 6.2.

use-alt-transfer-source
See the description of **use-alt-transfer-source** in Section 6.2.

notify-source
See the description of **notify-source** in Section 6.2.

notify-source-v6
See the description of **notify-source-v6** in Section 6.2.

min-refresh-time, max-refresh-time, min-retry-time, max-retry-time
See the descriptions in Section 6.2.

ixfr-from-differences
See the description of **ixfr-from-differences** in Section 6.2. (Note that the **ixfr-from-differences** choices of **`master`** and **`slave`** are not available at the zone level.)

key-directory
See the description of **key-directory** in Section 6.2.

auto-dnssec
See the description of **auto-dnssec** in Section 6.2.

serial-update-method
See the description of **serial-update-method** in Section 6.2.

inline-signing
If `yes`, this enables "bump in the wire" signing of a zone, where an unsigned zone is transferred in or loaded from disk and a signed version of the zone is served, with, possibly, a different serial number. This behavior is disabled by default.

multi-master
See the description of **multi-master** in Section 6.2.

masterfile-format
See the description of **masterfile-format** in Section 6.2.

max-zone-ttl
See the description of **max-zone-ttl** in Section 6.2.

dnssec-secure-to-insecure
See the description of **dnssec-secure-to-insecure** in Section 6.2.

Dynamic Update Policies

BIND 9 supports two methods of granting clients the right to perform dynamic updates to a zone, configured by the **allow-update** and **update-policy** options.

The **allow-update** clause is a simple access control list. Any client that matches the ACL is granted permission to update any record in the zone.

The **update-policy** clause allows more fine-grained control over which updates are allowed. It specifies a set of rules, in which each rule either grants or denies permission for one or more names in the zone to be updated by one or more identities. Identity is determined by the key that signed the update request, using either TSIG or SIG(0). In most cases, **update-policy** rules only apply to key-based identities. There is no way to specify update permissions based on client source address.

update-policy rules are only meaningful for primary zones (type **master**), and are not allowed in any other zone type. It is a configuration error to specify both **allow-update** and **update-policy** at the same time.

A pre-defined **update-policy** rule can be switched on with the command **update-policy local;**. **named** automatically generates a TSIG session key when starting and stores it in a file; this key can then be used by local clients to update the zone while **named** is running. By default, the session key is stored in the file `/var/run/named/session.key`, the key name is "local-ddns", and the key algorithm is HMAC-SHA256. These values are configurable with the **session-keyfile**, **session-keyname**, and **session-keyalg** options, respectively. A client running on the local system, if run with appropriate permissions, may read the session key from the key file and use it to sign update requests. The zone's update policy is set to allow that key to change any record within the zone. Assuming the key name is "local-ddns", this policy is equivalent to:

```
update-policy { grant local-ddns zonesub any; };
```

with the additional restriction that only clients connecting from the local system are permitted to send updates.

Note that only one session key is generated by **named**; all zones configured to use **update-policy local** accept the same key.

The command **nsupdate -l** implements this feature, sending requests to localhost and signing them using the key retrieved from the session key file.

Other rule definitions look like this:

```
( grant | deny ) identity ruletype  name   types
```

Each rule grants or denies privileges. Rules are checked in the order in which they are specified in the **update-policy** statement. Once a message has successfully matched a rule, the operation is immediately granted or denied, and no further rules are examined. There are 13 types of

rules; the rule type is specified by the **ruletype** field, and the interpretation of other fields varies depending on the rule type.

In general, a rule is matched when the key that signed an update request matches the **identity** field, the name of the record to be updated matches the **name** field (in the manner specified by the **ruletype** field), and the type of the record to be updated matches the **types** field. Details for each rule type are described below.

The **identity** field must be set to a fully qualified domain name. In most cases, this represents the name of the TSIG or SIG(0) key that must be used to sign the update request. If the specified name is a wildcard, it is subject to DNS wildcard expansion, and the rule may apply to multiple identities. When a TKEY exchange has been used to create a shared secret, the identity of the key used to authenticate the TKEY exchange is used as the identity of the shared secret. Some rule types use identities matching the client's Kerberos principal (e.g, **"host/machine@REALM"**) or Windows realm (**machine$@REALM**).

The *name* field also specifies a fully qualified domain name. This often represents the name of the record to be updated. Interpretation of this field is dependent on rule type.

If no **types** are explicitly specified, then a rule matches all types except RRSIG, NS, SOA, NSEC, and NSEC3. Types may be specified by name, including "ANY"; ANY matches all types except NSEC and NSEC3, which can never be updated. Note that when an attempt is made to delete all records associated with a name, the rules are checked for each existing record type.

The *ruletype* field has 16 values: `name`, `subdomain`, `zonesub`, `wildcard`, `self`, `selfsub`, `selfwild`, `ms-self`, `ms-selfsub`, `ms-subdomain`, `krb5-self`, `krb5-selfsub`, `krb5-subdomain`, `tcp-self`, `6to4-self`, and `external`.

`name`	With exact-match semantics, this rule matches when the name being updated is identical to the contents of the *name* field.
`subdomain`	This rule matches when the name being updated is a subdomain of, or identical to, the contents of the *name* field.
`zonesub`	This rule is similar to subdomain, except that it matches when the name being updated is a subdomain of the zone in which the **update-policy** statement appears. This obviates the need to type the zone name twice, and enables the use of a standard **update-policy** statement in multiple zones without modification. When this rule is used, the *name* field is omitted.
`wildcard`	The *name* field is subject to DNS wildcard expansion, and this rule matches when the name being updated is a valid expansion of the wildcard.

`self`	This rule matches when the name of the record being updated matches the contents of the *`identity`* field. The *`name`* field is ignored. To avoid confusion, it is recommended that this field be set to the same value as the *`identity`* field or to "." The `self` rule type is most useful when allowing one key per name to update, where the key has the same name as the record to be updated. In this case, the *`identity`* field can be specified as `*` (asterisk).
`selfsub`	This rule is similar to `self`, except that subdomains of `self` can also be updated.
`selfwild`	This rule is similar to `self`, except that only subdomains of `self` can be updated.
`ms-self`	When a client sends an UPDATE using a Windows machine principal (for example, "machine$@REALM"), this rule allows records with the absolute name of "machine.REALM" to be updated. The realm to be matched is specified in the *`identity`* field. The *`name`* field has no effect on this rule; it should be set to "." as a placeholder. For example, **`grant EXAMPLE.COM ms-self . A AAAA`** allows any machine with a valid principal in the realm **`EXAMPLE.COM`** to update its own address records.
`ms-selfsub`	This is similar to **ms-self**, except it also allows updates to any subdomain of the name specified in the Windows machine principal, not just to the name itself.
`ms-subdomain`	When a client sends an UPDATE using a Windows machine principal (for example, "machine$@REALM"), this rule allows any machine in the specified realm to update any record in the zone or in a specified subdomain of the zone. The realm to be matched is specified in the *`identity`* field. The *`name`* field specifies the subdomain that may be updated. If set to "." or any other name at or above the zone apex, any name in the zone can be updated. For example, if **update-policy** for the zone "example.com" includes **`grant EXAMPLE.COM ms-subdomain hosts.example.com. A AAAA`**, any machine with a valid principal in the realm **`EXAMPLE.COM`** is able to update address records at or below "hosts.example.com".

`krb5-self`

When a client sends an UPDATE using a Kerberos machine principal (for example, "host/machine@REALM"), this rule allows records with the absolute name of "machine" to be updated, provided it has been authenticated by REALM. This is similar but not identical to **ms-self**, due to the "machine" part of the Kerberos principal being an absolute name instead of an unqualified name.
The realm to be matched is specified in the *identity* field. The *name* field has no effect on this rule; it should be set to "." as a placeholder.
For example, **grant EXAMPLE.COM krb5-self . A AAAA** allows any machine with a valid principal in the realm **EXAMPLE.COM** to update its own address records.

`krb5-selfsub`

This is similar to **krb5-self**, except it also allows updates to any subdomain of the name specified in the "machine" part of the Kerberos principal, not just to the name itself.

`krb5-subdomain`

This rule is identical to **ms-subdomain**, except that it works with Kerberos machine principals (i.e., "host/machine@REALM") rather than Windows machine principals.

`tcp-self`

This rule allows updates that have been sent via TCP and for which the standard mapping from the client's IP address into the `in-addr.arpa` and `ip6.arpa` namespaces matches the name to be updated. The **identity** field must match that name. The **name** field should be set to ".". Note that, since identity is based on the client's IP address, it is not necessary for update request messages to be signed.

> NOTE
>
> It is theoretically possible to spoof these TCP sessions.

`6to4-self`

This allows the name matching a 6to4 IPv6 prefix, as specified in RFC 3056, to be updated by any TCP connection from either the 6to4 network or from the corresponding IPv4 address. This is intended to allow NS or DNAME RRsets to be added to the `ip6.arpa` reverse tree.
The **identity** field must match the 6to4 prefix in `ip6.arpa`. The **name** field should be set to ".". Note that, since identity is based on the client's IP address, it is not necessary for update request messages to be signed.
In addition, if specified for an `ip6.arpa` name outside of the `2.0.0.2.ip6.arpa` namespace, the corresponding /48 reverse name can be updated. For example, TCP/IPv6 connections from 2001:DB8:ED0C::/48 can update records at `C.0.D.E.8.B.D.0.1.0.0.2.ip6.arpa`.

> NOTE
>
> It is theoretically possible to spoof these TCP sessions.

`external`

This rule allows **named** to defer the decision of whether to allow a given update to an external daemon.
The method of communicating with the daemon is specified in the *`identity`* field, the format of which is "`local:`*`path`*", where *`path`* is the location of a Unix-domain socket. (Currently, "local" is the only supported mechanism.)
Requests to the external daemon are sent over the Unix-domain socket as datagrams with the following format:

```
Protocol version number (4 bytes, network byte order, currently 1)
Request length (4 bytes, network byte order)
Signer (null-terminated string)
Name (null-terminated string)
TCP source address (null-terminated string)
Rdata type (null-terminated string)
Key (null-terminated string)
TKEY token length (4 bytes, network byte order)
TKEY token (remainder of packet)
```

The daemon replies with a four-byte value in network byte order, containing either 0 or 1; 0 indicates that the specified update is not permitted, and 1 indicates that it is.

Multiple Views

When multiple views are in use, a zone may be referenced by more than one of them. Often, the views contain different zones with the same name, allowing different clients to receive different answers for the same queries. At times, however, it is desirable for multiple views to contain identical zones. The **in-view** zone option provides an efficient way to do this; it allows a view to reference a zone that was defined in a previously configured view. Example:

```
view internal {
    match-clients { 10/8; };

    zone example.com {
        type master;
        file "example-external.db";
    };
};

view external {
    match-clients { any; };

    zone example.com {
        in-view internal;
    };
};
```

An **in-view** option cannot refer to a view that is configured later in the configuration file.

A **zone** statement which uses the **in-view** option may not use any other , with the exception of **forward** and **forwarders**. (These options control the behavior of the containing view, rather than change the zone object itself.)

Zone-level ACLs (e.g., allow-query, allow-transfer), and other configuration details of the zone, are all set in the view the referenced zone is defined in. Be careful to ensure that ACLs are wide enough for all views referencing the zone.

An **in-view** zone cannot be used as a response policy zone.

An **in-view** zone is not intended to reference a **forward** zone.

6.3 ZONE FILE

Types of Resource Records and When to Use Them

This section, largely borrowed from RFC 1034, describes the concept of a Resource Record (RR) and explains when each type is used. Since the publication of RFC 1034, several new RRs have been identified and implemented in the DNS. These are also included.

Resource Records

A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of

separate RRs. The order of RRs in a set is not significant and need not be preserved by name servers, resolvers, or other parts of the DNS. However, sorting of multiple RRs is permitted for optimization purposes: for example, to specify that a particular nearby server be tried first. See Section 6.2 and Section 6.2.

The components of a Resource Record are:

owner name	The domain name where the RR is found.
type	An encoded 16-bit value that specifies the type of the resource record.
TTL	The time-to-live of the RR. This field is a 32-bit integer in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.
class	An encoded 16-bit value that identifies a protocol family or an instance of a protocol.
RDATA	The resource data. The format of the data is type- and sometimes class-specific.

For a complete list of *types* of valid RRs, including those that have been obsoleted, please refer to https://en.wikipedia.org/wiki/List_of_DNS_record_types.

The following *classes* of resource records are currently valid in the DNS:

IN	The Internet.
CH	Chaosnet, a LAN protocol created at MIT in the mid-1970s. It was rarely used for its historical purpose, but was reused for BIND's built-in server information zones, e.g., `version.bind`.
HS	Hesiod, an information service developed by MIT's Project Athena. It was used to share information about various systems databases, such as users, groups, printers, etc.

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL), which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; that also times out, but follows the refreshing policies for the zone. The TTL is assigned by the administrator for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of

Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

Textual Expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In the examples provided in RFC 1034, a style similar to that used in zone files was employed in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner are list the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. To avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interest of clarity.

The resource data or RDATA section of the RR is given using knowledge of the typical representation for the data.

For example, the RRs carried in a message might be shown as:

```
ISI.EDU.            MX              10 VENERA.ISI.EDU.
                    MX              10 VAXA.ISI.EDU
VENERA.ISI.EDU      A               128.9.0.32
                    A               10.1.0.52
VAXA.ISI.EDU        A               10.2.0.27
                    A               128.9.0.33
```

The MX RRs have an RDATA section which consists of a 16-bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32-bit Internet address.

The above example shows six RRs, with two RRs at each of three domain names.

Here is another possible example:

```
XX.LCS.MIT.EDU.      IN A           10.0.0.44
                     CH A           MIT.EDU. 2420
```

This example shows two addresses for `XX.LCS.MIT.EDU`, each of a different class.

Discussion of MX Records

As described above, domain servers store information as a series of resource records, each of which contains a particular piece of information about a given domain name (which is usually, but not always, a host). The simplest way to think of a RR is as a typed pair of data, a domain name matched with a relevant datum and stored with some additional type information, to help systems determine when the RR is relevant.

MX records are used to control delivery of email. The data specified in the record is a priority and a domain name. The priority controls the order in which email delivery is attempted, with the lowest number first. If two priorities are the same, a server is chosen randomly. If no servers at a given priority are responding, the mail transport agent falls back to the next largest priority. Priority numbers do not have any absolute meaning; they are relevant only respective to other MX records for that domain name. The domain name given is the machine to which the mail is delivered. It *must* have an associated address record (A or AAAA); CNAME is not sufficient.

For a given domain, if there is both a CNAME record and an MX record, the MX record is in error, and is ignored. Instead, the mail is delivered to the server specified in the MX record pointed to by the CNAME. For example:

```
example.com.            IN      MX      10              mail.example.com.
                        IN      MX      10              mail2.example.com.
                        IN      MX      20              mail.backup.org.

mail.example.com.       IN      A       10.0.0.1

mail2.example.com.      IN      A       10.0.0.2
```

Mail delivery is attempted to `mail.example.com` and `mail2.example.com` (in any order); if neither of those succeeds, delivery to `mail.backup.org` is attempted.

Setting TTLs

The time-to- (TTL) of the RR field is a 32-bit integer represented in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long an RR can be cached before it should be discarded. The following three types of TTLs are currently used in a zone file.

SOA	The last field in the SOA is the negative caching TTL. This controls how long other servers cache no-such-domain (NXDOMAIN) responses from this server. The maximum time for negative caching is 3 hours (3h).
$TTL	The $TTL directive at the top of the zone file (before the SOA) gives a default TTL for every RR without a specific TTL set.

RR TTLs	Each RR can have a TTL as the second field in the RR, which controls how long other servers can cache it.

All of these TTLs default to units of seconds, though units can be explicitly specified: for example, `1h30m`.

Inverse Mapping in IPv4

Reverse name resolution (that is, translation from IP address to name) is achieved by means of the *in-addr.arpa* domain and PTR records. Entries in the in-addr.arpa domain are made in least-to-most significant order, read left to right. This is the opposite order to the way IP addresses are usually written. Thus, a machine with an IP address of 10.1.2.3 would have a corresponding in-addr.arpa name of 3.2.1.10.in-addr.arpa. This name should have a PTR resource record whose data field is the name of the machine or, optionally, multiple PTR records if the machine has more than one name. For example, in the [example.com] domain:

```
$ORIGIN         2.1.10.in-addr.arpa
3               IN PTR foo.example.com.
```

NOTE

The **$ORIGIN** line in this example is only to provide context; it does not necessarily appear in the actual usage. It is only used here to indicate that the example is relative to the listed origin.

Other Zone File Directives

The DNS "master file" format was initially defined in RFC 1035 and has subsequently been extended. While the format itself is class-independent, all records in a zone file must be of the same class.

Master file directives include **$ORIGIN**, **$INCLUDE**, and **$TTL.**

The @ (at-sign)

When used in the label (or name) field, the asperand or at-sign (@) symbol represents the current origin. At the start of the zone file, it is the <zone_name>, followed by a trailing dot (.).

The $ORIGIN Directive

Syntax: **$ORIGIN** *domain-name* [*comment*]

$ORIGIN sets the domain name that is appended to any unqualified records. When a zone is first read, there is an implicit **$ORIGIN** <zone_name>.; note the trailing dot. The current **$ORIGIN** is appended to the domain specified in the **$ORIGIN** argument if it is not absolute.

```
$ORIGIN example.com.
WWW     CNAME   MAIN-SERVER
```

is equivalent to

```
WWW.EXAMPLE.COM. CNAME MAIN-SERVER.EXAMPLE.COM.
```

The $INCLUDE Directive

Syntax: **$INCLUDE** *filename* [*origin*] [*comment*]

This reads and processes the file `filename` as if it were included in the file at this point. If **origin** is specified, the file is processed with **$ORIGIN** set to that value; otherwise, the current **$ORIGIN** is used.

The origin and the current domain name revert to the values they had prior to the **$INCLUDE** once the file has been read.

> NOTE
>
> RFC 1035 specifies that the current origin should be restored after an **$INCLUDE**, but it is silent on whether the current domain name should also be restored. BIND 9 restores both of them. This could be construed as a deviation from RFC 1035, a feature, or both.

The $TTL Directive

Syntax: **$TTL** *default-ttl* [*comment*]

This sets the default Time-To-Live (TTL) for subsequent records with undefined TTLs. Valid TTLs are of the range 0-2147483647 seconds.

$TTL is defined in RFC 2308.

BIND Primary File Extension: the $GENERATE Directive

Syntax: **$GENERATE** *range lhs* [*ttl*] [*class*] *type rhs* [*comment*]

$GENERATE is used to create a series of resource records that only differ from each other by an iterator. **$GENERATE** can be used to easily generate the sets of records required to support sub-/24 reverse delegations described in RFC 2317: Classless IN-ADDR.ARPA delegation.

```
$ORIGIN 0.0.192.IN-ADDR.ARPA.
$GENERATE 1-2 @ NS SERVER$.EXAMPLE.
$GENERATE 1-127 $ CNAME $.0
```

is equivalent to

```
0.0.0.192.IN-ADDR.ARPA. NS SERVER1.EXAMPLE.
0.0.0.192.IN-ADDR.ARPA. NS SERVER2.EXAMPLE.
1.0.0.192.IN-ADDR.ARPA. CNAME 1.0.0.0.192.IN-ADDR.ARPA.
2.0.0.192.IN-ADDR.ARPA. CNAME 2.0.0.0.192.IN-ADDR.ARPA.
...
127.0.0.192.IN-ADDR.ARPA. CNAME 127.0.0.0.192.IN-ADDR.ARPA.
```

Both generate a set of A and MX records. Note the MX's right-hand side is a quoted string. The quotes are stripped when the right-hand side is processed.

```
$ORIGIN EXAMPLE.
$GENERATE 1-127 HOST-$ A 1.2.3.$
$GENERATE 1-127 HOST-$ MX "0 ."
```

is equivalent to

```
HOST-1.EXAMPLE.   A  1.2.3.1
HOST-1.EXAMPLE.   MX 0 .
HOST-2.EXAMPLE.   A  1.2.3.2
HOST-2.EXAMPLE.   MX 0 .
HOST-3.EXAMPLE.   A  1.2.3.3
HOST-3.EXAMPLE.   MX 0 .
...
HOST-127.EXAMPLE. A  1.2.3.127
HOST-127.EXAMPLE. MX 0 .
```

range	This can be one of two forms: start-stop or start-stop/step. If the first form is used, then step is set to 1. "start", "stop", and "step" must be positive integers between 0 and (2^31)-1. "start" must not be larger than "stop".

lhs	This describes the owner name of the resource records to be created. Any single **$** (dollar sign) symbols within the **lhs** string are replaced by the iterator value. To get a $ in the output, escape the **$** using a backslash \, e.g., **\$**. The **$** may optionally be followed by modifiers which change the offset from the iterator, field width, and base. Modifiers are introduced by a { (left brace) immediately following the **$**, as in **${offset[,width[,base]]}**. For example, **${-20,3,d}** subtracts 20 from the current value and prints the result as a decimal in a zero-padded field of width 3. Available output forms are decimal (**d**), octal (**o**), hexadecimal (**x** or **X** for uppercase), and nibble (**n** or **N** for uppercase). The default modifier is **${0,0,d}**. If the **lhs** is not absolute, the current **$ORIGIN** is appended to the name. In nibble mode, the value is treated as if it were a reversed hexadecimal string, with each hexadecimal digit as a separate label. The width field includes the label separator. For compatibility with earlier versions, **$$** is still recognized as indicating a literal $ in the output.
ttl	This specifies the time-to-live of the generated records. If not specified, this is inherited using the normal TTL inheritance rules. **class** and **ttl** can be entered in either order.
class	This specifies the class of the generated records. This must match the zone class if it is specified. **class** and **ttl** can be entered in either order.
type	This can be any valid type.
rhs	**rhs** is an optionally quoted string.

The **$GENERATE** directive is a BIND extension and not part of the standard zone file format.

BIND 8 did not support the optional TTL and CLASS fields.

Additional File Formats

In addition to the standard text format, BIND 9 supports the ability to read or dump to zone files in other formats.

The `raw` format is a binary representation of zone data in a manner similar to that used in zone transfers. Since it does not require parsing text, load time is significantly reduced.

An even faster alternative is the `map` format, which is an image of a BIND 9 in-memory zone database; it can be loaded directly into memory via the **mmap()** function and the zone can begin serving queries almost immediately.

For a primary server, a zone file in `raw` or `map` format is expected to be generated from a textual zone file by the **named-compilezone** command. For a secondary server or for a dynamic zone,

the zone file is automatically generated when **named** dumps the zone contents after zone transfer or when applying prior updates, if one of these formats is specified by the **masterfile-format** option.

If a zone file in a binary format needs manual modification, it first must be converted to a textual form by the **named-compilezone** command. Make any necessary modifications to the text file, and then convert it to the binary form via the **named-compilezone** command again.

Note that **map** format is extremely architecture-specific. A `map` file *cannot* be used on a system with different pointer size, endianness, or data alignment than the system on which it was generated, and should in general be used only inside a single system. While `raw` format uses network byte order and avoids architecture-dependent data alignment so that it is as portable as possible, it is also primarily expected to be used inside the same single system. To export a zone file in either `raw` or `map` format, or make a portable backup of such a file, conversion to `text` format is recommended.

6.4 BIND 9 STATISTICS

BIND 9 maintains lots of statistics information and provides several interfaces for users to access those statistics. The available statistics include all statistics counters that are meaningful in BIND 9, and other information that is considered useful.

The statistics information is categorized into the following sections:

Incoming Requests	The number of incoming DNS requests for each OPCODE.
Incoming Queries	The number of incoming queries for each RR type.
Outgoing Queries	The number of outgoing queries for each RR type sent from the internal resolver, maintained per view.
Name Server Statistics	Statistics counters for incoming request processing.
Zone Maintenance Statistics	Statistics counters regarding zone maintenance operations, such as zone transfers.
Resolver Statistics	Statistics counters for name resolutions performed in the internal resolver, maintained per view.

Cache DB RRsets	Statistics counters related to cache contents, maintained per view. The "NXDOMAIN" counter is the number of names that have been cached as nonexistent. Counters named for RR types indicate the number of active RRsets for each type in the cache database. If an RR type name is preceded by an exclamation point (!), it represents the number of records in the cache which indicate that the type does not exist for a particular name; this is also known as "NXRRSET". If an RR type name is preceded by a hash mark (#), it represents the number of RRsets for this type that are present in the cache but whose TTLs have expired; these RRsets may only be used if stale answers are enabled. If an RR type name is preceded by a tilde (~), it represents the number of RRsets for this type that are present in the cache database but are marked for garbage collection; these RRsets cannot be used.
Socket I/O Statistics	Statistics counters for network-related events.

A subset of Name Server Statistics is collected and shown per zone for which the server has the authority, when **zone-statistics** is set to **`full`** (or **`yes`**), for backward compatibility. See the description of **zone-statistics** in Section 6.2 for further details.

These statistics counters are shown with their zone and view names. The view name is omitted when the server is not configured with explicit views.

There are currently two user interfaces to get access to the statistics. One is in plain-text format, dumped to the file specified by the **statistics-file** configuration option; the other is remotely accessible via a statistics channel when the **statistics-channels** statement is specified in the configuration file (see Section 6.2.)

The Statistics File

The text format statistics dump begins with a line, like:

+++ Statistics Dump +++ (973798949)

The number in parentheses is a standard Unix-style timestamp, measured in seconds since January 1, 1970. Following that line is a set of statistics information, which is categorized as described above. Each section begins with a line, like:

++ Name Server Statistics ++

Each section consists of lines, each containing the statistics counter value followed by its textual description; see below for available counters. For brevity, counters that have a value of 0 are not shown in the statistics file.

The statistics dump ends with the line where the number is identical to the number in the beginning line; for example:

--- Statistics Dump --- (973798949)

Statistics Counters

The following tables summarize the statistics counters that BIND 9 provides. For each row of the tables, the leftmost column is the abbreviated symbol name of that counter; these symbols are shown in the statistics information accessed via an HTTP statistics channel. The rightmost column gives the description of the counter, which is also shown in the statistics file, but, in this document, may be slightly modified for better readability. Additional notes may also be provided in this column. When a middle column exists between these two columns, it gives the corresponding counter name of the BIND 8 statistics, if applicable.

Name Server Statistics Counters

Symbol	*BIND 8 Symbol*	*Description*
Requestv4	**RQ**	This indicates the number of IPv4 requests received. Note: this also counts non-query requests.
Requestv6	**RQ**	This indicates the number of IPv6 requests received. Note: this also counts non-query requests.
ReqEdns0		This indicates the number of requests received with EDNS(0).
ReqBadEDNSVer		This indicates the number of requests received with an unsupported EDNS version.
ReqTSIG		This indicates the number of requests received with TSIG.
ReqSIG0		This indicates the number of requests received with SIG(0).
ReqBadSIG		This indicates the number of requests received with an invalid (TSIG or SIG(0)) signature.
ReqTCP	**RTCP**	This indicates the number of TCP requests received.
AuthQryRej	**RUQ**	This indicates the number of rejected authoritative (non-recursive) queries.

RecQryRej	**RURQ**	This indicates the number of rejected recursive queries.
XfrRej	**RUXFR**	This indicates the number of rejected zone transfer requests.
UpdateRej	**RUUpd**	This indicates the number of rejected dynamic update requests.
Response	**SAns**	This indicates the number of responses sent.
RespTruncated		This indicates the number of truncated responses sent.
RespEDNS0		This indicates the number of responses sent with EDNS(0).
RespTSIG		This indicates the number of responses sent with TSIG.
RespSIG0		This indicates the number of responses sent with SIG(0).
QrySuccess		This indicates the number of queries that resulted in a successful answer, meaning queries which return a NOERROR response with at least one answer RR. This corresponds to the **success** counter of previous versions of BIND 9.
QryAuthAns		This indicates the number of queries that resulted in an authoritative answer.
QryNoauthAns	**SNaAns**	This indicates the number of queries that resulted in a non-authoritative answer.
QryReferral		This indicates the number of queries that resulted in a referral answer. This corresponds to the **referral** counter of previous versions of BIND 9.
QryNxrrset		This indicates the number of queries that resulted in NOERROR responses with no data. This corresponds to the **nxrrset** counter of previous versions of BIND 9.
QrySERVFAIL	**SFail**	This indicates the number of queries that resulted in SERVFAIL.
QryFORMERR	**SFErr**	This indicates the number of queries that resulted in FORMERR.

QryNXDOMAIN	**SNXD**	This indicates the number of queries that resulted in NXDOMAIN. This corresponds to the **nxdomain** counter of previous versions of BIND 9.
QryRecursion	**RFwdQ**	This indicates the number of queries that caused the server to perform recursion in order to find the final answer. This corresponds to the **recursion** counter of previous versions of BIND 9.
QryDuplicate	**RDupQ**	This indicates the number of queries which the server attempted to recurse but for which it discovered an existing query with the same IP address, port, query ID, name, type, and class already being processed. This corresponds to the **duplicate** counter of previous versions of BIND 9.
QryDropped		This indicates the number of recursive queries for which the server discovered an excessive number of existing recursive queries for the same name, type, and class, and which were subsequently dropped. This is the number of dropped queries due to the reason explained with the **clients-per-query** and **max-clients-per-query** options (see the description about clients-per-query.) This corresponds to the **dropped** counter of previous versions of BIND 9.
QryFailure		This indicates the number of query failures. This corresponds to the **failure** counter of previous versions of BIND 9. Note: this counter is provided mainly for backward compatibility with the previous versions; normally, more fine-grained counters such as **AuthQryRej** and **RecQryRej** that would also fall into this counter are provided, so this counter is not of much interest in practice.
QryNXRedir		This indicates the number of queries that resulted in NXDOMAIN that were redirected.
QryNXRedirRLookup		This indicates the number of queries that resulted in NXDOMAIN that were redirected and resulted in a successful remote lookup.
XfrReqDone		This indicates the number of requested and completed zone transfers.
UpdateReqFwd		This indicates the number of forwarded update requests.

UpdateRespFwd	This indicates the number of forwarded update responses.
UpdateFwdFail	This indicates the number of forwarded dynamic updates that failed.
UpdateDone	This indicates the number of completed dynamic updates.
UpdateFail	This indicates the number of failed dynamic updates.
UpdateBadPrereq	This indicates the number of dynamic updates rejected due to a prerequisite failure.
RateDropped	This indicates the number of responses dropped due to rate limits.
RateSlipped	This indicates the number of responses truncated by rate limits.
RPZRewrites	This indicates the number of response policy zone rewrites.

Zone Maintenance Statistics Counters

Symbol	*Description*
NotifyOutv4	This indicates the number of IPv4 notifies sent.
NotifyOutv6	This indicates the number of IPv6 notifies sent.
NotifyInv4	This indicates the number of IPv4 notifies received.
NotifyInv6	This indicates the number of IPv6 notifies received.
NotifyRej	This indicates the number of incoming notifies rejected.
SOAOutv4	This indicates the number of IPv4 SOA queries sent.
SOAOutv6	This indicates the number of IPv6 SOA queries sent.
AXFRReqv4	This indicates the number of requested IPv4 AXFRs.
AXFRReqv6	This indicates the number of requested IPv6 AXFRs.
IXFRReqv4	This indicates the number of requested IPv4 IXFRs.

IXFRReqv6	This indicates the number of requested IPv6 IXFRs.
XfrSuccess	This indicates the number of successful zone transfer requests.
XfrFail	This indicates the number of failed zone transfer requests.

Resolver Statistics Counters

Symbol	*BIND 8 Symbol*	*Description*
Queryv4	**SFwdQ**	This indicates the number of IPv4 queries sent.
Queryv6	**SFwdQ**	This indicates the number of IPv6 queries sent.
Responsev4	**RR**	This indicates the number of IPv4 responses received.
Responsev6	**RR**	This indicates the number of IPv6 responses received.
NXDOMAIN	**RNXD**	This indicates the number of NXDOMAINs received.
SERVFAIL	**RFail**	This indicates the number of SERVFAILs received.
FORMERR	**RFErr**	This indicates the number of FORMERRs received.
OtherError	**RErr**	This indicates the number of other errors received.
EDNS0Fail		This indicates the number of EDNS(0) query failures.
Mismatch	**RDupR**	This indicates the number of mismatched responses received, meaning the DNS ID, response's source address, and/or the response's source port does not match what was expected. (The port must be 53 or as defined by the **port** option.) This may be an indication of a cache poisoning attempt.
Truncated		This indicates the number of truncated responses received.
Lame	**RLame**	This indicates the number of lame delegations received.
Retry	**SDupQ**	This indicates the number of query retries performed.
QueryAbort		This indicates the number of queries aborted due to quota control.

QuerySockFail		This indicates the number of failures in opening query sockets. One common reason for such failures is a due to a limitation on file descriptors.
QueryTimeout		This indicates the number of query timeouts.
GlueFetchv4	**SSysQ**	This indicates the number of IPv4 NS address fetches invoked.
GlueFetchv6	**SSysQ**	This indicates the number of IPv6 NS address fetches invoked.
GlueFetchv4Fail		This indicates the number of failed IPv4 NS address fetches.
GlueFetchv6Fail		This indicates the number of failed IPv6 NS address fetches.
ValAttempt		This indicates the number of attempted DNSSEC validations.
ValOk		This indicates the number of successful DNSSEC validations.
ValNegOk		This indicates the number of successful DNSSEC validations on negative information.
ValFail		This indicates the number of failed DNSSEC validations.
QryRTTnn		This provides a frequency table on query round-trip times (RTTs). Each **nn** specifies the corresponding frequency. In the sequence of **nn_1**, **nn_2**, ..., **nn_m**, the value of **nn_i** is the number of queries whose RTTs are between **nn_(i-1)** (inclusive) and **nn_i** (exclusive) milliseconds. For the sake of convenience, we define **nn_0** to be 0. The last entry should be represented as **nn_m+**, which means the number of queries whose RTTs are equal to or greater than **nn_m** milliseconds.

Socket I/O Statistics Counters

Socket I/O statistics counters are defined per socket type, which are **UDP4** (UDP/IPv4), **UDP6** (UDP/IPv6), **TCP4** (TCP/IPv4), **TCP6** (TCP/IPv6), **Unix** (Unix Domain), and **FDwatch** (sockets opened outside the socket module). In the following table, **<TYPE>** represents a socket type. Not all counters are available for all socket types; exceptions are noted in the description field.

Symbol	*Description*
<TYPE>Open	This indicates the number of sockets opened successfully. This counter does not apply to the **FDwatch** type.
<TYPE>OpenFail	This indicates the number of failures to open sockets. This counter does not apply to the **FDwatch** type.
<TYPE>Close	This indicates the number of closed sockets.
<TYPE>BindFail	This indicates the number of failures to bind sockets.
<TYPE>ConnFail	This indicates the number of failures to connect sockets.
<TYPE>Conn	This indicates the number of connections established successfully.
<TYPE>AcceptFail	This indicates the number of failures to accept incoming connection requests. This counter does not apply to the **UDP** and **FDwatch** types.
<TYPE>Accept	This indicates the number of incoming connections successfully accepted. This counter does not apply to the **UDP** and **FDwatch** types.
<TYPE>SendErr	This indicates the number of errors in socket send operations. This counter corresponds to the **SErr** counter of **BIND** 8.
<TYPE>RecvErr	This indicates the number of errors in socket receive operations, including errors of send operations on a connected UDP socket, notified by an ICMP error message.

Compatibility with *BIND* 8 Counters

Most statistics counters that were available in **BIND** 8 are also supported in **BIND** 9, as shown in the above tables. Here are notes about other counters that do not appear in these tables.

RFwdR,SFwdR
: These counters are not supported, because **BIND** 9 does not adopt the notion of *forwarding* as **BIND** 8 did.

RAXFR
: This counter is accessible in the Incoming Queries section.

RIQ
This counter is accessible in the Incoming Requests section.

ROpts
This counter is not supported, because **BIND** 9 does not care about IP options.

7 BIND 9 Security Considerations

7.1 Access Control Lists

Access Control Lists (ACLs) are address match lists that can be set up and nicknamed for future use in **allow-notify**, **allow-query**, **allow-query-on**, **allow-recursion**, **blackhole**, **allow-transfer**, **match-clients**, etc.

ACLs give users finer control over who can access the name server, without cluttering up configuration files with huge lists of IP addresses.

It is a *good idea* to use ACLs, and to control access. Limiting access to the server by outside parties can help prevent spoofing and denial of service (DoS) attacks against the server.

ACLs match clients on the basis of up to three characteristics: 1) The client's IP address; 2) the TSIG or SIG(0) key that was used to sign the request, if any; and 3) an address prefix encoded in an EDNS Client-Subnet option, if any.

Here is an example of ACLs based on client addresses:

```
// Set up an ACL named "bogusnets" that blocks
// RFC 1918 space and some reserved space, which is
// commonly used in spoofing attacks.
acl bogusnets {
        0.0.0.0/8;  192.0.2.0/24; 224.0.0.0/3;
        10.0.0.0/8; 172.16.0.0/12; 192.168.0.0/16;
};

// Set up an ACL called our-nets. Replace this with the
// real IP numbers.
acl our-nets { x.x.x.x/24; x.x.x.x/21; };
options {
  ...
  ...
  allow-query { our-nets; };
  allow-recursion { our-nets; };
  ...
  blackhole { bogusnets; };
  ...
};

zone "example.com" {
```

```
  type master;
  file "m/example.com";
  allow-query { any; };
};
```

This allows authoritative queries for "example.com" from any address, but recursive queries only from the networks specified in "our-nets", and no queries at all from the networks specified in "bogusnets".

In addition to network addresses and prefixes, which are matched against the source address of the DNS request, ACLs may include `key` elements, which specify the name of a TSIG or SIG(0) key, or `ecs` elements, which specify a network prefix but are only matched if that prefix matches an EDNS client-subnet option included in the request.

The EDNS Client-Subnet (ECS) option is used by a recursive resolver to inform an authoritative name server of the network address block from which the original query was received, enabling authoritative servers to give different answers to the same resolver for different resolver clients. An ACL containing an element of the form **ecs *prefix*** will match if a request arrives in containing an ECS option encoding an address within that prefix. If the request has no ECS option, then "ecs" elements are simply ignored. Addresses in ACLs that are not prefixed with "ecs" are matched only against the source address.

> NOTE
>
> (Note: the authoritative ECS implementation in **named** is based on an early version of the specification, and is known to have incompatibilities with other implementations. It is also inefficient, requiring a separate view for each client subnet to be sent different answers, and it is unable to correct for overlapping subnets in the configuration. It can be used for testing purposes, but is not recommended for production use.)

When BIND 9 is built with GeoIP support, ACLs can also be used for geographic access restrictions. This is done by specifying an ACL element of the form: **geoip [db *database*] *field value***

The *field* parameter indicates which field to search for a match. Available fields are "country", "region", "city", "continent", "postal" (postal code), "metro" (metro code), "area" (area code), "tz" (timezone), "isp", "asnum", and "domain".

value is the value to search for within the database. A string may be quoted if it contains spaces or other special characters. An "asnum" search for autonomous system number can be specified using the string "ASNNNN" or the integer NNNN. When "country" search is specified with a string that is two characters long, it must be a standard ISO-3166-1 two-letter country code; otherwise, it is interpreted as the full name of the country. Similarly, if "region" is the search term and the string is two characters long, it is treated as a standard two-letter state or province abbreviation; otherwise, it is treated as the full name of the state or province.

The *database* field indicates which GeoIP database to search for a match. In most cases this is unnecessary, because most search fields can only be found in a single database. However,

searches for "continent" or "country" can be answered from either the "city" or "country" databases, so for these search types, specifying a *database* forces the query to be answered from that database and no other. If *database* is not specified, these queries are first answered from the "city" database if it is installed, and then from the "country" database if it is installed. Valid database names are "country", "city", "asnum", "isp", and "domain". (If using the legacy GeoIP API, "netspeed" and "org" databases are also available.)

By default, if a DNS query includes an EDNS Client-Subnet (ECS) option which encodes a non-zero address prefix, then GeoIP ACL elements are matched against that address prefix. Otherwise, they are matched against the source address of the query. To prevent GeoIP ACLs from matching against ECS options, set the **geoip-use-ecs** to `no`.

Some example GeoIP ACLs:

```
geoip country US;
geoip country JP;
geoip db country country Canada;
geoip region WA;
geoip city "San Francisco";
geoip region Oklahoma;
geoip postal 95062;
geoip tz "America/Los_Angeles";
geoip org "Internet Systems Consortium";
```

ACLs use a "first-match" logic rather than "best-match"; if an address prefix matches an ACL element, then that ACL is considered to have matched even if a later element would have matched more specifically. For example, the ACL **{ 10/8; !10.0.0.1; }** would actually match a query from 10.0.0.1, because the first element indicates that the query should be accepted, and the second element is ignored.

When using "nested" ACLs (that is, ACLs included or referenced within other ACLs), a negative match of a nested ACL tells the containing ACL to continue looking for matches. This enables complex ACLs to be constructed, in which multiple client characteristics can be checked at the same time. For example, to construct an ACL which allows a query only when it originates from a particular network *and* only when it is signed with a particular key, use:

```
allow-query { !{ !10/8; any; }; key example; };
```

Within the nested ACL, any address that is *not* in the 10/8 network prefix is rejected, which terminates processing of the ACL. Any address that *is* in the 10/8 network prefix is accepted, but this causes a negative match of the nested ACL, so the containing ACL continues processing. The query is accepted if it is signed by the key "example", and rejected otherwise. The ACL, then, only matches when *both* conditions are true.

7.2 CHROOT AND SETUID

On Unix servers, it is possible to run BIND in a *chrooted* environment (using the **chroot()** function) by specifying the `-t` option for **named**. This can help improve system security by placing BIND in a "sandbox," which limits the damage done if a server is compromised.

Another useful feature in the Unix version of BIND is the ability to run the daemon as an unprivileged user (`-u user`). We suggest running as an unprivileged user when using the **chroot** feature.

Here is an example command line to load BIND in a **chroot** sandbox, **/var/named**, and to run **named setuid** to user 202:

```
/usr/local/sbin/named -u 202 -t /var/named
```

The chroot Environment

For a **chroot** environment to work properly in a particular directory (for example, `/var/named`), the environment must include everything BIND needs to run. From BIND's point of view, `/var/named` is the root of the filesystem; the values of options like **directory** and **pid-file** must be adjusted to account for this.

Unlike with earlier versions of BIND, **named** does *not* typically need to be compiled statically, nor do shared libraries need to be installed under the new root. However, depending on the operating system, it may be necessary to set up locations such as `/dev/zero`, `/dev/random`, `/dev/log`, and `/etc/localtime`.

Using the setuid Function

Prior to running the **named** daemon, use the **touch** utility (to change file access and modification times) or the **chown** utility (to set the user id and/or group id) on files where BIND should write.

> NOTE
>
> If the **named** daemon is running as an unprivileged user, it cannot bind to new restricted ports if the server is reloaded.

7.3 DYNAMIC UPDATE SECURITY

Access to the dynamic update facility should be strictly limited. In earlier versions of BIND, the only way to do this was based on the IP address of the host requesting the update, by listing an IP address or network prefix in the **allow-update** zone option. This method is insecure, since the source address of the update UDP packet is easily forged. Also note that if the IP addresses allowed by the **allow-update** option include the address of a secondary server which performs forwarding of dynamic updates, the primary can be trivially attacked by sending the update to the secondary, which forwards it to the primary with its own source IP address - causing the primary to approve it without question.

For these reasons, we strongly recommend that updates be cryptographically authenticated by means of transaction signatures (TSIG). That is, the **allow-update** option should list only TSIG key names, not IP addresses or network prefixes. Alternatively, the **update-policy** option can be used.

Some sites choose to keep all dynamically updated DNS data in a subdomain and delegate that subdomain to a separate zone. This way, the top-level zone containing critical data, such as the IP addresses of public web and mail servers, need not allow dynamic update at all.

8 Troubleshooting

8.1 Common Problems

It's Not Working; How Can I Figure Out What's Wrong?

The best solution to installation and configuration issues is to take preventive measures by setting up logging files beforehand. The log files provide a source of hints and information that can be used to identify what went wrong and fix the problem.

8.2 Incrementing and Changing the Serial Number

Zone serial numbers are just numbers --- they are not date-related. However, many people set them to a number that represents a date, usually of the form YYYYMMDDRR. Occasionally they make a mistake and set the serial number to a date in the future, then try to correct it by setting it to the current date. This causes problems because serial numbers are used to indicate that a zone has been updated. If the serial number on the secondary server is lower than the serial number on the primary, the secondary server attempts to update its copy of the zone.

Setting the serial number to a lower number on the primary server than the one on the secondary server means that the secondary will not perform updates to its copy of the zone.

The solution to this is to add 2147483647 (2^31-1) to the number, reload the zone and make sure all secondaries have updated to the new zone serial number, then reset it to the desired number and reload the zone again.

8.3 Where Can I Get Help?

The BIND-users mailing list, at https://lists.isc.org/mailman/listinfo/bind-users, is an excellent resource for peer user support. In addition, ISC maintains a Knowledgebase of helpful articles at https://kb.isc.org.

Internet Systems Consortium (ISC) offers annual support agreements for BIND9, ISC DHCP, and Kea DHCP. All paid support contracts include advance security notifications; some levels include service level agreements (SLAs), premium software features, and increased priority on bug fixes and feature requests.

Please contact info@isc.org or visit https://www.isc.org/contact/ for more information.

9 Manual pages

9.1 ARPANAME

arpaname — translate IP addresses to the corresponding ARPA names

Synopsis

`arpaname` *`ipaddress`* ...

DESCRIPTION

arpaname translates IP addresses (IPv4 and IPv6) to the corresponding IN-ADDR.ARPA or IP6.ARPA names.

SEE ALSO

BIND 9 Administrator Reference Manual.

9.2 DDNS-CONFGEN

ddns-confgen — ddns key generation tool

Synopsis

`tsig-keygen` [`-a` *`algorithm`*] [`-h`] [`-r` *`randomfile`*] [name]

`ddns-confgen` [`-a` *`algorithm`*] [`-h`] [`-k` *`keyname`*] [`-q`] [`-r` *`randomfile`*] [**-s** *`name`* | **-z** *`zone`*]

DESCRIPTION

tsig-keygen and **ddns-confgen** are invocation methods for a utility that generates keys for use in TSIG signing. The resulting keys can be used, for example, to secure dynamic DNS updates to a zone or for the **rndc** command channel.

When run as **tsig-keygen**, a domain name can be specified on the command line which will be used as the name of the generated key. If no name is specified, the default is `tsig-key`.

When run as **ddns-confgen**, the generated key is accompanied by configuration text and instructions that can be used with **nsupdate** and **named** when setting up dynamic DNS, including an example **update-policy** statement. (This usage similar to the **rndc-confgen** command for setting up command channel security.)

Note that **named** itself can configure a local DDNS key for use with **nsupdate -l**: it does this when a zone is configured with **update-policy local;**. **ddns-confgen** is only needed when a more elaborate configuration is required: for instance, if **nsupdate** is to be used from a remote system.

OPTIONS

-a *algorithm*
: Specifies the algorithm to use for the TSIG key. Available choices are: hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384 and hmac-sha512. The default is hmac-sha256. Options are case-insensitive, and the "hmac-" prefix may be omitted.

-h
: Prints a short summary of options and arguments.

-k *keyname*
: Specifies the key name of the DDNS authentication key. The default is `ddns-key` when neither the `-s` nor `-z` option is specified; otherwise, the default is `ddns-key` as a separate label followed by the argument of the option, e.g., `ddns-key.example.com`. The key name must have the format of a valid domain name, consisting of letters, digits, hyphens and periods.

-q
: (**ddns-confgen** only.) Quiet mode: Print only the key, with no explanatory text or usage examples; This is essentially identical to **tsig-keygen**.

-r *randomfile*
: Specifies a source of random data for generating the authorization. If the operating system does not provide a `/dev/random` or equivalent device, the default source of randomness is keyboard input. `randomdev` specifies the name of a character device or file containing random data to be used instead of the default. The special value `keyboard` indicates that keyboard input should be used.

-s *name*
: (**ddns-confgen** only.) Generate configuration example to allow dynamic updates of a single hostname. The example **named.conf** text shows how to set an update policy for the specified *name* using the "name" nametype. The default key name is ddns-key.*name*. Note

that the "self" nametype cannot be used, since the name to be updated may differ from the key name. This option cannot be used with the `-z` option.

-z zone

(**ddns-confgen** only.) Generate configuration example to allow dynamic updates of a zone: The example **named.conf** text shows how to set an update policy for the specified *zone* using the "zonesub" nametype, allowing updates to all subdomain names within that *zone*. This option cannot be used with the `-s` option.

SEE ALSO

nsupdate(1), named.conf(5), named(8), *BIND 9 Administrator Reference Manual*.

9.3 DELV

delv — DNS lookup and validation utility

Synopsis

`delv` [@server] [`-4` | `-6`] [`-a` *anchor-file*] [`-b` *address*] [`-c` *class*] [`-d` *level*] [`-i`] [`-m`] [`-p` *port#*] [`-q` *name*] [`-t` *type*] [`-x` *addr*] [name] [type] [class] [queryopt...]

`delv` [`-h`]

`delv` [`-v`]

`delv` [queryopt...] [query...]

DESCRIPTION

delv is a tool for sending DNS queries and validating the results, using the same internal resolver and validator logic as **named**.

delv will send to a specified name server all queries needed to fetch and validate the requested data; this includes the original requested query, subsequent queries to follow CNAME or DNAME chains, and queries for DNSKEY, DS and DLV records to establish a chain of trust for DNSSEC validation. It does not perform iterative resolution, but simulates the behavior of a name server configured for DNSSEC validating and forwarding.

By default, responses are validated using built-in DNSSEC trust anchor for the root zone ("."). Records returned by **delv** are either fully validated or were not signed. If validation fails, an explanation of the failure is included in the output; the validation process can be traced in detail. Because **delv** does not rely on an external server to carry out validation, it can be used to check the validity of DNS responses in environments where local name servers may not be trustworthy.

Unless it is told to query a specific name server, **delv** will try each of the servers listed in `/etc/resolv.conf`. If no usable server addresses are found, **delv** will send queries to the localhost addresses (127.0.0.1 for IPv4, ::1 for IPv6).

When no command line arguments or options are given, **delv** will perform an NS query for "." (the root zone).

SIMPLE USAGE

A typical invocation of **delv** looks like:

```
delv @server name type
```

where:

server
is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied *server* argument is a hostname, **delv** resolves that name before querying that name server (note, however, that this initial lookup is *not* validated by DNSSEC).

If no *server* argument is provided, **delv** consults `/etc/resolv.conf`; if an address is found there, it queries the name server at that address. If either of the `-4` or `-6` options are in use, then only addresses for the corresponding transport will be tried. If no usable addresses are found, **delv** will send queries to the localhost addresses (127.0.0.1 for IPv4, ::1 for IPv6).

name
is the domain name to be looked up.

type
indicates what type of query is required --- ANY, A, MX, etc. *type* can be any valid query type. If no *type* argument is supplied, **delv** will perform a lookup for an A record.

OPTIONS

-a *anchor-file*
Specifies a file from which to read DNSSEC trust anchors. The default is `/etc/bind.keys`, which is included with BIND 9 and contains one or more trust anchors for the root zone (".").

Keys that do not match the root zone name are ignored. An alternate key name can be specified using the `+root=NAME` options. DNSSEC Lookaside Validation can also be turned on by using the `+dlv=NAME` to specify the name of a zone containing DLV records.

Note: When reading the trust anchor file, **delv** treats `managed-keys` statements and `trusted-keys` statements identically. That is, for a managed key, it is the *initial* key that is trusted; RFC 5011 key management is not supported. **delv** will not consult the managed-keys database maintained by **named**. This means that if either of the keys in `/etc/bind.keys` is revoked and rolled over, it will be necessary to update `/etc/bind.keys` to use DNSSEC validation in **delv**.

-b *address*
Sets the source IP address of the query to *address*. This must be a valid address on one of the host's network interfaces or "0.0.0.0" or "::". An optional source port may be specified by appending "#<port>"

-c *class*
Sets the query class for the requested data. Currently, only class "IN" is supported in **delv** and any other value is ignored.

-d *level*
Set the systemwide debug level to `level`. The allowed range is from 0 to 99. The default is 0 (no debugging). Debugging traces from **delv** become more verbose as the debug level increases. See the `+mtrace`, `+rtrace`, and `+vtrace` options below for additional debugging details.

-h
Display the **delv** help usage output and exit.

-i
Insecure mode. This disables internal DNSSEC validation. (Note, however, this does not set the CD bit on upstream queries. If the server being queried is performing DNSSEC validation, then it will not return invalid data; this can cause **delv** to time out. When it is necessary to examine invalid data to debug a DNSSEC problem, use **dig +cd**.)

-m
Enables memory usage debugging.

-p *port#*
Specifies a destination port to use for queries instead of the standard DNS port number 53. This option would be used with a name server that has been configured to listen for queries on a non-standard port number.

-q *name*
Sets the query name to *name*. While the query name can be specified without using the `-q`, it is sometimes necessary to disambiguate names from types or classes (for example, when looking up the name "ns", which could be misinterpreted as the type NS, or "ch", which could be misinterpreted as class CH).

-t *type*
Sets the query type to *type*, which can be any valid query type supported in BIND 9 except for zone transfer types AXFR and IXFR. As with `-q`, this is useful to distinguish query name type or class when they are ambiguous. it is sometimes necessary to disambiguate names from types.

The default query type is "A", unless the `-x` option is supplied to indicate a reverse lookup, in which case it is "PTR".

-v
Print the **delv** version and exit.

-x *addr*
Performs a reverse lookup, mapping an addresses to a name. *addr* is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. When `-x` is used, there is no need to provide the *name* or *type* arguments. **delv** automatically performs a lookup for a name like `11.12.13.10.in-addr.arpa` and sets the query type to PTR. IPv6 addresses are looked up using nibble format under the IP6.ARPA domain.

-4
Forces **delv** to only use IPv4.

-6
Forces **delv** to only use IPv6.

QUERY OPTIONS

delv provides a number of query options which affect the way results are displayed, and in some cases the way lookups are performed.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option. These may be preceded by the string `no` to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form `+keyword=value`. The query options are:

+[no]cdflag
Controls whether to set the CD (checking disabled) bit in queries sent by **delv**. This may be useful when troubleshooting DNSSEC problems from behind a validating resolver. A validating resolver will block invalid responses, making it difficult to retrieve them for analysis. Setting the CD flag on queries will cause the resolver to return invalid responses, which **delv** can then validate internally and report the errors in detail.

+[no]class
Controls whether to display the CLASS when printing a record. The default is to display the CLASS.

+[no]ttl
Controls whether to display the TTL when printing a record. The default is to display the TTL.

+[no]rtrace
Toggle resolver fetch logging. This reports the name and type of each query sent by **delv** in the process of carrying out the resolution and validation process: this includes including the original query and all subsequent queries to follow CNAMEs and to establish a chain of trust for DNSSEC validation.

This is equivalent to setting the debug level to 1 in the "resolver" logging category. Setting the systemwide debug level to 1 using the `-d` option will product the same output (but will affect other logging categories as well).

+[no]mtrace
Toggle message logging. This produces a detailed dump of the responses received by **delv** in the process of carrying out the resolution and validation process.

This is equivalent to setting the debug level to 10 for the "packets" module of the "resolver" logging category. Setting the systemwide debug level to 10 using the `-d` option will produce the same output (but will affect other logging categories as well).

+[no]vtrace
Toggle validation logging. This shows the internal process of the validator as it determines whether an answer is validly signed, unsigned, or invalid.

This is equivalent to setting the debug level to 3 for the "validator" module of the "dnssec" logging category. Setting the systemwide debug level to 3 using the `-d` option will produce the same output (but will affect other logging categories as well).

+[no]short
Provide a terse answer. The default is to print the answer in a verbose form.

+[no]comments
Toggle the display of comment lines in the output. The default is to print comments.

+[no]rrcomments
Toggle the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is to print per-record comments.

+[no]crypto
Toggle the display of cryptographic fields in DNSSEC records. The contents of these field are unnecessary to debug most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted they are replaced by the string "[omitted]" or in the DNSKEY case the key id is displayed as the replacement, e.g. "[key id = value]".

+[no]trust
Controls whether to display the trust level when printing a record. The default is to display the trust level.

+[no]split[=W]
Split long hex- or base64-formatted fields in resource records into chunks of *W* characters (where *W* is rounded up to the nearest multiple of 4). *+nosplit* or *+split=0* causes fields not to be split at all. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]all
Set or clear the display options +[no]comments, +[no]rrcomments, and +[no]trust as a group.

+[no]multiline
Print long records (such as RRSIG, DNSKEY, and SOA records) in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the **delv** output.

+[no]dnssec
Indicates whether to display RRSIG records in the **delv** output. The default is to do so. Note that (unlike in **dig**) this does *not* control whether to request DNSSEC records or whether to validate them. DNSSEC records are always requested, and validation will always occur unless suppressed by the use of -i or +noroot and +nodlv.

+[no]root[=ROOT]
Indicates whether to perform conventional (non-lookaside) DNSSEC validation, and if so, specifies the name of a trust anchor. The default is to validate using a trust anchor of "." (the root zone), for which there is a built-in key. If specifying a different trust anchor, then -a must be used to specify a file containing the key.

+[no]dlv[=DLV]
Indicates whether to perform DNSSEC lookaside validation, and if so, specifies the name of the DLV trust anchor. The -a option must also be used to specify a file containing the DLV key.

+[no]tcp
Controls whether to use TCP when sending queries. The default is to use UDP unless a truncated response has been received.

+[no]unknownformat
Print all RDATA in unknown RR type presentation format (RFC 3597). The default is to print RDATA for known types in the type's presentation format.

FILES

`/etc/bind.keys`

`/etc/resolv.conf`

SEE ALSO

dig(1), named(8), *RFC4034*, *RFC4035*, *RFC4431*, *RFC5074*, *RFC5155*.

9.4 DIG

dig — DNS lookup utility

Synopsis

`dig` [@server] [`-b` *address*] [`-c` *class*] [`-f` *filename*] [`-k` *filename*] [`-m`] [`-p` *port#*] [`-q` *name*] [`-t` *type*] [`-v`] [`-x` *addr*] [`-y` *[hmac:]name:key*] [`-4` | `-6`] [name] [type] [class] [query-opt...]

`dig` [`-h`]

`dig` [global-queryopt...] [query...]

DESCRIPTION

dig is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried. Most DNS administrators use **dig** to troubleshoot DNS problems because of its flexibility, ease of use and clarity of output. Other lookup tools tend to have less functionality than **dig**.

Although **dig** is normally used with command-line arguments, it also has a batch mode of operation for reading lookup requests from a file. A brief summary of its command-line arguments and options is printed when the `-h` option is given. Unlike earlier versions, the BIND 9 implementation of **dig** allows multiple lookups to be issued from the command line.

Unless it is told to query a specific name server, **dig** will try each of the servers listed in `/etc/resolv.conf`. If no usable server addresses are found, **dig** will send the query to the local host.

When no command line arguments or options are given, **dig** will perform an NS query for "." (the root).

It is possible to set per-user defaults for **dig** via `${HOME}/.digrc`. This file is read and any options in it are applied before the command line arguments. The `-r` option disables this feature, for scripts that need predictable behaviour.

The IN and CH class names overlap with the IN and CH top level domain names. Either use the `-t` and `-c` options to specify the type and class, use the `-q` the specify the domain name, or use "IN." and "CH." when looking up these top level domains.

SIMPLE USAGE

A typical invocation of **dig** looks like:

```
dig @server name type
```

where:

server
: is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied *server* argument is a hostname, **dig** resolves that name before querying that name server.

  If no *server* argument is provided, **dig** consults `/etc/resolv.conf`; if an address is found there, it queries the name server at that address. If either of the `-4` or `-6` options are in use, then only addresses for the corresponding transport will be tried. If no usable addresses are found, **dig** will send the query to the local host. The reply from the name server that responds is displayed.

name
: is the name of the resource record that is to be looked up.

type
: indicates what type of query is required --- ANY, A, MX, SIG, etc. *type* can be any valid query type. If no *type* argument is supplied, **dig** will perform a lookup for an A record.

OPTIONS

-4
: Use IPv4 only.

-6
: Use IPv6 only.

-b *address[#port]*
: Set the source IP address of the query. The *address* must be a valid address on one of the host's network interfaces, or "0.0.0.0" or "::". An optional port may be specified by appending "#<port>"

-c *class*
Set the query class. The default *class* is IN; other classes are HS for Hesiod records or CH for Chaosnet records.

-f *file*
Batch mode: **dig** reads a list of lookup requests to process from the given *file*. Each line in the file should be organized in the same way they would be presented as queries to **dig** using the command-line interface.

-i
Do reverse IPv6 lookups using the obsolete RFC 1886 IP6.INT domain, which is no longer in use. Obsolete bit string label queries (RFC 2874) are not attempted.

-k *keyfile*
Sign queries using TSIG using a key read from the given file. Key files can be generated using tsig-keygen(8). When using TSIG authentication with **dig**, the name server that is queried needs to know the key and algorithm that is being used. In BIND, this is done by providing appropriate **key** and **server** statements in `named.conf`.

-m
Enable memory usage debugging.

-p *port*
Send the query to a non-standard port on the server, instead of the default port 53. This option would be used to test a name server that has been configured to listen for queries on a non-standard port number.

-q *name*
The domain name to query. This is useful to distinguish the *name* from other arguments.

-r
Do not read options from `${HOME}/.digrc`. This is useful for scripts that need predictable behaviour.

-t *type*
The resource record type to query. It can be any valid query type. If it is a resource record type supported in BIND 9, it can be given by the type mnemonic (such as "NS" or "AAAA"). The default query type is "A", unless the `-x` option is supplied to indicate a reverse lookup. A zone transfer can be requested by specifying a type of AXFR. When an incremental zone transfer (IXFR) is required, set the *type* to `ixfr=N`. The incremental zone transfer will contain the changes made to the zone since the serial number in the zone's SOA record was *N*.

All resource record types can be expressed as "TYPEnn", where "nn" is the number of the type. If the resource record type is not supported in BIND 9, the result will be displayed as described in RFC 3597.

-u
Print query times in microseconds instead of milliseconds.

-v
Print the version number and exit.

-x ***addr***

Simplified reverse lookups, for mapping addresses to names. The *addr* is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. When the `-x` is used, there is no need to provide the *name*, *class* and *type* arguments. **dig** automatically performs a lookup for a name like `94.2.0.192.in-addr.arpa` and sets the query type and class to PTR and IN respectively. IPv6 addresses are looked up using nibble format under the IP6.ARPA domain (but see also the `-i` option).

-y ***[hmac:]keyname:secret***

Sign queries using TSIG with the given authentication key. *keyname* is the name of the key, and *secret* is the base64 encoded shared secret. *hmac* is the name of the key algorithm; valid choices are `hmac-md5`, `hmac-sha1`, `hmac-sha224`, `hmac-sha256`, `hmac-sha384`, or `hmac-sha512`. If *hmac* is not specified, the default is `hmac-md5` or if MD5 was disabled `hmac-sha256`.

NOTE: You should use the `-k` option and avoid the `-y` option, because with `-y` the shared secret is supplied as a command line argument in clear text. This may be visible in the output from ps(1) or in a history file maintained by the user's shell.

QUERY OPTIONS

dig provides a number of query options which affect the way in which lookups are made and the results displayed. Some of these set or reset flag bits in the query header, some determine which sections of the answer get printed, and others determine the timeout and retry strategies.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option. These may be preceded by the string `no` to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form `+keyword=value`. Keywords may be abbreviated, provided the abbreviation is unambiguous; for example, `+cd` is equivalent to `+cdflag`. The query options are:

+[no]aaflag

A synonym for *+[no]aaonly*.

+[no]aaonly

Sets the "aa" flag in the query.

+[no]additional

Display [do not display] the additional section of a reply. The default is to display it.

+[no]adflag

Set [do not set] the AD (authentic data) bit in the query. This requests the server to return whether all of the answer and authority sections have all been validated as secure according to the security policy of the server. AD=1 indicates that all records have been validated as secure and the answer is not from a OPT-OUT range. AD=0 indicate that some part of the answer was insecure or not validated. This bit is set by default.

+[no]all

Set or clear all display flags.

+[no]answer

Display [do not display] the answer section of a reply. The default is to display it.

+[no]authority
Display [do not display] the authority section of a reply. The default is to display it.

+[no]badcookie
Retry lookup with the new server cookie if a BADCOOKIE response is received.

+[no]besteffort
Attempt to display the contents of messages which are malformed. The default is to not display malformed answers.

+bufsize=B
This option sets the UDP message buffer size advertised using EDNS0 to *B* bytes. The maximum and minimum sizes of this buffer are 65535 and 0, respectively. `+bufsize=0` disables EDNS (use `+bufsize=0 +edns` to send a EDNS messages with a advertised size of 0 bytes). `+bufsize` restores the default buffer size.

+[no]cdflag
Set [do not set] the CD (checking disabled) bit in the query. This requests the server to not perform DNSSEC validation of responses.

+[no]class
Display [do not display] the CLASS when printing the record.

+[no]cmd
Toggles the printing of the initial comment in the output, identifying the version of **dig** and the query options that have been applied. This option always has global effect; it cannot be set globally and then overridden on a per-lookup basis. The default is to print this comment.

+[no]comments
Toggles the display of some comment lines in the output, containing information about the packet header and OPT pseudosection, and the names of the response section. The default is to print these comments.

Other types of comments in the output are not affected by this option, but can be controlled using other command line switches. These include **+[no]cmd**, **+[no]question**, **+[no]stats**, and **+[no]rrcomments**.

+[no]cookie[=####]
Send a COOKIE EDNS option, with optional value. Replaying a COOKIE from a previous response will allow the server to identify a previous client. The default is `+cookie`.

+cookie is also set when +trace is set to better emulate the default queries from a nameserver.

+[no]crypto
Toggle the display of cryptographic fields in DNSSEC records. The contents of these field are unnecessary to debug most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted they are replaced by the string "[omitted]" or in the DNSKEY case the key id is displayed as the replacement, e.g. "[key id = value]".

+[no]defname
Deprecated, treated as a synonym for *+[no]search*

+[no]dnssec
Requests DNSSEC records be sent by setting the DNSSEC OK bit (DO) in the OPT record in the additional section of the query.

+domain=somename
Set the search list to contain the single domain *somename*, as if specified in a **domain** directive in `/etc/resolv.conf`, and enable search list processing as if the *+search* option were given.

+dscp=value
Set the DSCP code point to be used when sending the query. Valid DSCP code points are in the range [0..63]. By default no code point is explicitly set.

+[no]edns[=#]
Specify the EDNS version to query with. Valid values are 0 to 255. Setting the EDNS version will cause a EDNS query to be sent. `+noedns` clears the remembered EDNS version. EDNS is set to 0 by default.

+[no]ednsflags[=#]
Set the must-be-zero EDNS flags bits (Z bits) to the specified value. Decimal, hex and octal encodings are accepted. Setting a named flag (e.g. DO) will silently be ignored. By default, no Z bits are set.

+[no]ednsnegotiation
Enable / disable EDNS version negotiation. By default EDNS version negotiation is enabled.

+[no]ednsopt[=code[:value]]
Specify EDNS option with code point `code` and optionally payload of `value` as a hexadecimal string. `code` can be either an EDNS option name (for example, `NSID` or `ECS`), or an arbitrary numeric value. `+noednsopt` clears the EDNS options to be sent.

+[no]expire
Send an EDNS Expire option.

+[no]fail
Do not try the next server if you receive a SERVFAIL. The default is to not try the next server which is the reverse of normal stub resolver behavior.

+[no]header-only
Send a query with a DNS header without a question section. The default is to add a question section. The query type and query name are ignored when this is set.

+[no]identify
Show [or do not show] the IP address and port number that supplied the answer when the *+short* option is enabled. If short form answers are requested, the default is not to show the source address and port number of the server that provided the answer.

+[no]idnin
Process [do not process] IDN domain names on input. This requires IDN SUPPORT to have been enabled at compile time.

The default is to process IDN input when standard output is a tty. The IDN processing on input is disabled when dig output is redirected to files, pipes, and other non-tty file descriptors.

+[no]idnout
Convert [do not convert] puny code on output. This requires IDN SUPPORT to have been enabled at compile time.

The default is to process puny code on output when standard output is a tty. The puny code processing on output is disabled when dig output is redirected to files, pipes, and other non-tty file descriptors.

+[no]ignore
Ignore truncation in UDP responses instead of retrying with TCP. By default, TCP retries are performed.

+[no]keepopen
Keep the TCP socket open between queries and reuse it rather than creating a new TCP socket for each lookup. The default is `+nokeepopen`.

+[no]mapped
Allow mapped IPv4 over IPv6 addresses to be used. The default is `+mapped`.

+[no]multiline
Print records like the SOA records in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the **dig** output.

+ndots=D
Set the number of dots that have to appear in *`name`* to *`D`* for it to be considered absolute. The default value is that defined using the ndots statement in `/etc/resolv.conf`, or 1 if no ndots statement is present. Names with fewer dots are interpreted as relative names and will be searched for in the domains listed in the `search` or `domain` directive in `/etc/resolv.conf` if `+search` is set.

+[no]nsid
Include an EDNS name server ID request when sending a query.

+[no]nssearch
When this option is set, **dig** attempts to find the authoritative name servers for the zone containing the name being looked up and display the SOA record that each name server has for the zone.

+[no]onesoa
Print only one (starting) SOA record when performing an AXFR. The default is to print both the starting and ending SOA records.

+[no]opcode=value
Set [restore] the DNS message opcode to the specified value. The default value is QUERY (0).

+[no]qr
Toggles the display of the query message as it is sent. By default, the query is not printed.

+[no]question
Toggles the display of the question section of a query when an answer is returned. The default is to print the question section as a comment.

+[no]rdflag
A synonym for *+[no]recurse*.

+[no]recurse
Toggle the setting of the RD (recursion desired) bit in the query. This bit is set by default, which means **dig** normally sends recursive queries. Recursion is automatically disabled when using the *+nssearch* option, and when using *+trace* except for an initial recursive query to get the list of root servers.

+retry=T
Sets the number of times to retry UDP queries to server to *T* instead of the default, 2. Unlike *+tries*, this does not include the initial query.

+[no]rrcomments
Toggle the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is not to print record comments unless multiline mode is active.

+[no]search
Use [do not use] the search list defined by the searchlist or domain directive in `resolv.conf` (if any). The search list is not used by default.

'ndots' from `resolv.conf` (default 1) which may be overridden by *+ndots* determines if the name will be treated as relative or not and hence whether a search is eventually performed or not.

+[no]short
Provide a terse answer. The default is to print the answer in a verbose form. This option always has global effect; it cannot be set globally and then overridden on a per-lookup basis.

+[no]showsearch
Perform [do not perform] a search showing intermediate results.

+[no]sigchase
Chase DNSSEC signature chains. Requires dig be compiled with -DDIG_SIGCHASE. This feature is deprecated. Use **delv** instead.

+split=W
Split long hex- or base64-formatted fields in resource records into chunks of *W* characters (where *W* is rounded up to the nearest multiple of 4). *+nosplit* or *+split=0* causes fields not to be split at all. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]stats
Toggles the printing of statistics: when the query was made, the size of the reply and so on. The default behavior is to print the query statistics as a comment after each lookup.

+[no]subnet=addr[/prefix-length]
Send (don't send) an EDNS Client Subnet option with the specified IP address or network prefix.

dig +subnet=0.0.0.0/0, or simply **dig +subnet=0** for short, sends an EDNS CLIENT-SUBNET option with an empty address and a source prefix-length of zero, which signals a resolver that the client's address information must *not* be used when resolving this query.

+[no]tcp
Use [do not use] TCP when querying name servers. The default behavior is to use UDP unless a type `any` or `ixfr=N` query is requested, in which case the default is TCP. AXFR queries always use TCP.

+timeout=T
Sets the timeout for a query to *T* seconds. The default timeout is 5 seconds. An attempt to set *T* to less than 1 will result in a query timeout of 1 second being applied.

+[no]topdown
When chasing DNSSEC signature chains perform a top-down validation. Requires dig be compiled with -DDIG_SIGCHASE. This feature is deprecated. Use **delv** instead.

+[no]trace
Toggle tracing of the delegation path from the root name servers for the name being looked up. Tracing is disabled by default. When tracing is enabled, **dig** makes iterative queries to resolve the name being looked up. It will follow referrals from the root servers, showing the answer from each server that was used to resolve the lookup.

If @server is also specified, it affects only the initial query for the root zone name servers.

+dnssec is also set when +trace is set to better emulate the default queries from a nameserver.

+tries=T
Sets the number of times to try UDP queries to server to *T* instead of the default, 3. If *T* is less than or equal to zero, the number of tries is silently rounded up to 1.

+trusted-key=####
Specifies a file containing trusted keys to be used with `+sigchase`. Each DNSKEY record must be on its own line.

If not specified, **dig** will look for `/etc/trusted-key.key` then `trusted-key.key` in the current directory.

Requires dig be compiled with -DDIG_SIGCHASE. This feature is deprecated. Use **delv** instead.

+[no]ttlid
Display [do not display] the TTL when printing the record.

+[no]ttlunits
Display [do not display] the TTL in friendly human-readable time units of "s", "m", "h", "d", and "w", representing seconds, minutes, hours, days and weeks. Implies +ttlid.

+[no]unknownformat
Print all RDATA in unknown RR type presentation format (RFC 3597). The default is to print RDATA for known types in the type's presentation format.

+[no]vc
Use [do not use] TCP when querying name servers. This alternate syntax to *+[no]tcp* is provided for backwards compatibility. The "vc" stands for "virtual circuit".

+[no]zflag
Set [do not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

MULTIPLE QUERIES

The BIND 9 implementation of **dig** supports specifying multiple queries on the command line (in addition to supporting the `-f` batch file option). Each of those queries can be supplied with its own set of flags, options and query options.

In this case, each *query* argument represent an individual query in the command-line syntax described above. Each consists of any of the standard options and flags, the name to be looked up, an optional query type and class and any query options that should be applied to that query.

A global set of query options, which should be applied to all queries, can also be supplied. These global query options must precede the first tuple of name, class, type, options, flags, and query options supplied on the command line. Any global query options (except the `+[no]cmd` option) can be overridden by a query-specific set of query options. For example:

```
dig +qr www.isc.org any -x 127.0.0.1 isc.org ns +noqr
```

shows how **dig** could be used from the command line to make three lookups: an ANY query for `www.isc.org`, a reverse lookup of 127.0.0.1 and a query for the NS records of `isc.org`. A global query option of *+qr* is applied, so that **dig** shows the initial query it made for each lookup. The final query has a local query option of *+noqr* which means that **dig** will not print the initial query when it looks up the NS records for `isc.org`.

IDN SUPPORT

If **dig** has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. **dig** appropriately converts character encoding of domain name before sending a request to DNS server or displaying a reply from the server. If you'd like to turn off the IDN support for some reason, use parameters *+noidnin* and *+noidnout*.

FILES

`/etc/resolv.conf`

`${HOME}/.digrc`

SEE ALSO

delv(1), host(1), named(8), dnssec-keygen(8), *RFC 1035*.

BUGS

There are probably too many query options.

9.5 DNSSEC-CHECKDS

dnssec-checkds — DNSSEC delegation consistency checking tool

Synopsis

`dnssec-checkds` [`-l` *domain*] [`-f` *file*] [`-d` *dig path*] [`-D` *dsfromkey path*] zone

`dnssec-dsfromkey` [`-l` *domain*] [`-f` *file*] [`-d` *dig path*] [`-D` *dsfromkey path*] zone

DESCRIPTION

dnssec-checkds verifies the correctness of Delegation Signer (DS) or DNSSEC Lookaside Validation (DLV) resource records for keys in a specified zone.

OPTIONS

-f *file*
If a `file` is specified, then the zone is read from that file to find the DNSKEY records. If not, then the DNSKEY records for the zone are looked up in the DNS.

-l *domain*
Check for a DLV record in the specified lookaside domain, instead of checking for a DS record in the zone's parent.

-d *dig path*
Specifies a path to a **dig** binary. Used for testing.

-D *dsfromkey path*
Specifies a path to a **dnssec-dsfromkey** binary. Used for testing.

SEE ALSO

dnssec-dsfromkey(8), dnssec-keygen(8), dnssec-signzone(8),

9.6 DNSSEC-COVERAGE

dnssec-coverage — checks future DNSKEY coverage for a zone

Synopsis

`dnssec-coverage` [`-K` *directory*] [`-l` *length*] [`-f` *file*] [`-d` *DNSKEY TTL*] [`-m` *max TTL*] [`-r` *interval*] [`-c` *compilezone path*] [`-k`] [`-z`] [zone...]

DESCRIPTION

dnssec-coverage verifies that the DNSSEC keys for a given zone or a set of zones have timing metadata set properly to ensure no future lapses in DNSSEC coverage.

If `zone` is specified, then keys found in the key repository matching that zone are scanned, and an ordered list is generated of the events scheduled for that key (i.e., publication, activation, inactivation, deletion). The list of events is walked in order of occurrence. Warnings are generated if any event is scheduled which could cause the zone to enter a state in which validation failures might occur: for example, if the number of published or active keys for a given algorithm drops to zero, or if a key is deleted from the zone too soon after a new key is rolled, and cached data signed by the prior key has not had time to expire from resolver caches.

If `zone` is not specified, then all keys in the key repository will be scanned, and all zones for which there are keys will be analyzed. (Note: This method of reporting is only accurate if all the zones that have keys in a given repository share the same TTL parameters.)

OPTIONS

-K *directory*
Sets the directory in which keys can be found. Defaults to the current working directory.

-f *file*
If a `file` is specified, then the zone is read from that file; the largest TTL and the DNSKEY TTL are determined directly from the zone data, and the `-m` and `-d` options do not need to be specified on the command line.

-l *duration*
The length of time to check for DNSSEC coverage. Key events scheduled further into the future than `duration` will be ignored, and assumed to be correct.

The value of `duration` can be set in seconds, or in larger units of time by adding a suffix: 'mi' for minutes, 'h' for hours, 'd' for days, 'w' for weeks, 'mo' for months, 'y' for years.

-m *maximum TTL*
Sets the value to be used as the maximum TTL for the zone or zones being analyzed when determining whether there is a possibility of validation failure. When a zone-signing key is deactivated, there must be enough time for the record in the zone with the longest TTL to have expired from resolver caches before that key can be purged from the DNSKEY RRset. If that condition does not apply, a warning will be generated.

The length of the TTL can be set in seconds, or in larger units of time by adding a suffix: 'mi' for minutes, 'h' for hours, 'd' for days, 'w' for weeks, 'mo' for months, 'y' for years.

This option is not necessary if the `-f` has been used to specify a zone file. If `-f` has been specified, this option may still be used; it will override the value found in the file.

If this option is not used and the maximum TTL cannot be retrieved from a zone file, a warning is generated and a default value of 1 week is used.

-d *DNSKEY TTL*
Sets the value to be used as the DNSKEY TTL for the zone or zones being analyzed when determining whether there is a possibility of validation failure. When a key is rolled (that

is, replaced with a new key), there must be enough time for the old DNSKEY RRset to have expired from resolver caches before the new key is activated and begins generating signatures. If that condition does not apply, a warning will be generated.

The length of the TTL can be set in seconds, or in larger units of time by adding a suffix: 'mi' for minutes, 'h' for hours, 'd' for days, 'w' for weeks, 'mo' for months, 'y' for years.

This option is not necessary if `-f` has been used to specify a zone file from which the TTL of the DNSKEY RRset can be read, or if a default key TTL was set using ith the `-L` to **dnssec-keygen**. If either of those is true, this option may still be used; it will override the values found in the zone file or the key file.

If this option is not used and the key TTL cannot be retrieved from the zone file or the key file, then a warning is generated and a default value of 1 day is used.

-r ***resign interval***

Sets the value to be used as the resign interval for the zone or zones being analyzed when determining whether there is a possibility of validation failure. This value defaults to 22.5 days, which is also the default in **named**. However, if it has been changed by the `sig-validity-interval` option in `named.conf`, then it should also be changed here.

The length of the interval can be set in seconds, or in larger units of time by adding a suffix: 'mi' for minutes, 'h' for hours, 'd' for days, 'w' for weeks, 'mo' for months, 'y' for years.

-k

Only check KSK coverage; ignore ZSK events. Cannot be used with `-z`.

-z

Only check ZSK coverage; ignore KSK events. Cannot be used with `-k`.

-c ***compilezone path***

Specifies a path to a **named-compilezone** binary. Used for testing.

SEE ALSO

dnssec-checkds(8), dnssec-dsfromkey(8), dnssec-keygen(8), dnssec-signzone(8)

9.7 DNSSEC-DSFROMKEY

dnssec-dsfromkey — DNSSEC DS RR generation tool

Synopsis

`dnssec-dsfromkey` [`-1` | `-2` | `-a` *alg*] [`-C` | `-l` *domain*] [`-T` *TTL*] [`-v` *level*] [`-K` *directory*] keyfile

`dnssec-dsfromkey` [`-1` | `-2` | `-a` *alg*] [`-C` | `-l` *domain*] [`-T` *TTL*] [`-v` *level*] [`-c` *class*] [`-A`] `-f` *file* [dnsname]

`dnssec-dsfromkey` [`-1` | `-2` | `-a` *alg*] [`-C` | `-l` *domain*] [`-T` *TTL*] [`-v` *level*] [`-c` *class*] [`-K` *directory*] -s dnsname

`dnssec-dsfromkey` [`-h` | `-V`]

DESCRIPTION

The **dnssec-dsfromkey** command outputs DS (Delegation Signer) resource records (RRs) and other similarly-constructed RRs: with the `-l` option it outputs DLV (DNSSEC Lookaside Validation) RRs; or with the `-C` it outputs CDS (Child DS) RRs.

The input keys can be specified in a number of ways:

By default, **dnssec-dsfromkey** reads a key file named like `Knnnn.+aaa+iiiii.key`, as generated by **dnssec-keygen**.

With the `-f` *`file`* option, **dnssec-dsfromkey** reads keys from a zone file or partial zone file (which can contain just the DNSKEY records).

With the `-s` option, **dnssec-dsfromkey** reads a `keyset-` file, as generated by **dnssec-keygen** `-C`.

OPTIONS

-1
An abbreviation for `-a SHA1`

-2
An abbreviation for `-a SHA-256`

-a *algorithm*
Specify a digest algorithm to use when converting DNSKEY records to DS records. This option can be repeated, so that multiple DS records are created for each DNSKEY record.

The *`algorithm`* must be one of SHA-1, SHA-256, or SHA-384. These values are case insensitive, and the hyphen may be omitted. If no algorithm is specified, the default is to use both SHA-1 and SHA-256.

-A
Include ZSKs when generating DS records. Without this option, only keys which have the KSK flag set will be converted to DS records and printed. Useful only in `-f` zone file mode.

-c *class*
Specifies the DNS class (default is IN). Useful only in `-s` keyset or `-f` zone file mode.

-C
Generate CDS records rather than DS records. This is mutually exclusive with the `-l` option for generating DLV records.

-f *file*
Zone file mode: **dnssec-dsfromkey**'s final *`dnsname`* argument is the DNS domain name of a zone whose master file can be read from `file`. If the zone name is the same as `file`, then it may be omitted.

If *`file`* is `"-"`, then the zone data is read from the standard input. This makes it possible to use the output of the **dig** command as input, as in:

```
dig dnskey example.com | dnssec-dsfromkey -f - example.com
```

-h
Prints usage information.

-K ***directory***
Look for key files or `keyset-` files in `directory`.

-l ***domain***
Generate a DLV set instead of a DS set. The specified *domain* is appended to the name for each record in the set. This is mutually exclusive with the `-C` option for generating CDS records.

-s
Keyset mode: **dnssec-dsfromkey**'s final *dnsname* argument is the DNS domain name used to locate a `keyset-` file.

-T ***TTL***
Specifies the TTL of the DS records. By default the TTL is omitted.

-v ***level***
Sets the debugging level.

-V
Prints version information.

EXAMPLE

To build the SHA-256 DS RR from the **Kexample.com.+003+26160** keyfile name, you can issue the following command:

```
dnssec-dsfromkey -2 Kexample.com.+003+26160
```

The command would print something like:

```
example.com.  IN DS 26160 5 2 3A1EADA7A74B8D0BA86726B0C227AA85AB8BBD2B2004F41A868A
```

FILES

The keyfile can be designated by the key identification `Knnnn.+aaa+iiiii` or the full file name `Knnnn.+aaa+iiiii.key` as generated by dnssec-keygen(8).

The keyset file name is built from the `directory`, the string `keyset-` and the `dnsname`.

CAVEAT

A keyfile error can give a "file not found" even if the file exists.

SEE ALSO

dnssec-keygen(8), dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 3658* (DS RRs), *RFC 4431* (DLV RRs), *RFC 4509* (SHA-256 for DS RRs), *RFC 6605* (SHA-384 for DS RRs), *RFC 7344* (CDS and CDNSKEY RRs).

9.8 DNSSEC-IMPORTKEY

dnssec-importkey — import DNSKEY records from external systems so they can be managed

Synopsis

`dnssec-importkey` [`-K` *directory*] [`-L` *ttl*] [`-P` *date/offset*] [`-P sync` *date/offset*] [`-D` *date/offset*] [`-D sync` *date/offset*] [`-h`] [`-v` *level*] [`-V`] `keyfile`

`dnssec-importkey -f` *filename* [`-K` *directory*] [`-L` *ttl*] [`-P` *date/offset*] [`-P sync` *date/offset*] [`-D` *date/offset*] [`-D sync` *date/offset*] [`-h`] [`-v` *level*] [`-V`] [`dnsname`]

DESCRIPTION

dnssec-importkey reads a public DNSKEY record and generates a pair of .key/.private files. The DNSKEY record may be read from an existing .key file, in which case a corresponding .private file will be generated, or it may be read from any other file or from the standard input, in which case both .key and .private files will be generated.

The newly-created .private file does *not* contain private key data, and cannot be used for signing. However, having a .private file makes it possible to set publication (`-P`) and deletion (`-D`) times for the key, which means the public key can be added to and removed from the DNSKEY RRset on schedule even if the true private key is stored offline.

OPTIONS

-f *filename*
: Zone file mode: instead of a public keyfile name, the argument is the DNS domain name of a zone master file, which can be read from `file`. If the domain name is the same as `file`, then it may be omitted.

  If `file` is set to `"-"`, then the zone data is read from the standard input.

-K *directory*
: Sets the directory in which the key files are to reside.

-L *ttl*
: Sets the default TTL to use for this key when it is converted into a DNSKEY RR. If the key is imported into a zone, this is the TTL that will be used for it, unless there was already a DNSKEY RRset in place, in which case the existing TTL would take precedence. Setting the default TTL to `0` or `none` removes it.

-h
: Emit usage message and exit.

-v *level*
: Sets the debugging level.

-V
: Prints version information.

TIMING OPTIONS

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a '+' or '-', it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use 'none' or 'never'.

-P *date/offset*
Sets the date on which a key is to be published to the zone. After that date, the key will be included in the zone but will not be used to sign it.

-P sync *date/offset*
Sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

-D *date/offset*
Sets the date on which the key is to be deleted. After that date, the key will no longer be included in the zone. (It may remain in the key repository, however.)

-D sync *date/offset*
Sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

FILES

A keyfile can be designed by the key identification `Knnnn.+aaa+iiiii` or the full file name `Knnnn.+aaa+iiiii.key` as generated by dnssec-keygen(8).

SEE ALSO

dnssec-keygen(8), dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 5011*.

9.9 DNSSEC-KEYFROMLABEL

dnssec-keyfromlabel — DNSSEC key generation tool

Synopsis

`dnssec-keyfromlabel` -l *label* [-3] [-a *algorithm*] [-A *date/offset*] [-c *class*] [-D *date/offset*] [-D sync *date/offset*] [-E *engine*] [-f *flag*] [-G] [-I *date/offset*] [-i *interval*] [-k] [-K *directory*] [-L *ttl*] [-n *nametype*] [-P *date/offset*] [-P sync *date/offset*] [-p *protocol*] [-R *date/offset*] [-S *key*] [-t *type*] [-v *level*] [-V] [-y] name

DESCRIPTION

dnssec-keyfromlabel generates a key pair of files that referencing a key object stored in a cryptographic hardware service module (HSM). The private key file can be used for DNSSEC signing of zone data as if it were a conventional signing key created by **dnssec-keygen**, but the key material is stored within the HSM, and the actual signing takes place there.

The `name` of the key is specified on the command line. This must match the name of the zone for which the key is being generated.

OPTIONS

-a *algorithm*

Selects the cryptographic algorithm. The value of `algorithm` must be one of RSAMD5, RSASHA1, DSA, NSEC3RSASHA1, NSEC3DSA, RSASHA256, RSASHA512, ECCGOST, ECDSAP256SHA256, ECDSAP384SHA384, ED25519 or ED448. These values are case insensitive.

If no algorithm is specified, then RSASHA1 will be used by default, unless the `-3` option is specified, in which case NSEC3RSASHA1 will be used instead. (If `-3` is used and an algorithm is specified, that algorithm will be checked for compatibility with NSEC3.)

Note 1: that for DNSSEC, RSASHA1 is a mandatory to implement algorithm, and DSA is recommended.

Note 2: DH automatically sets the -k flag.

-3

Use an NSEC3-capable algorithm to generate a DNSSEC key. If this option is used and no algorithm is explicitly set on the command line, NSEC3RSASHA1 will be used by default.

-E *engine*

Specifies the cryptographic hardware to use.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-l *label*

Specifies the label for a key pair in the crypto hardware.

When BIND 9 is built with OpenSSL-based PKCS#11 support, the label is an arbitrary string that identifies a particular key.

When BIND 9 is built with native PKCS#11 support, the label is a PKCS#11 URI string in the format "pkcs11:`keyword=`*`value`*[;`keyword=`*`value`*;...]" Keywords include "token", which identifies the HSM; "object", which identifies the key; and "pin-source", which identifies a file from which the HSM's PIN code can be obtained. The label will be stored in the on-disk "private" file.

If the label contains a `pin-source` field, tools using the generated key files will be able to use the HSM for signing and other operations without any need for an operator to manually enter a PIN. Note: Making the HSM's PIN accessible in this manner may reduce the

security advantage of using an HSM; be sure this is what you want to do before making use of this feature.

-n ***nametype***
Specifies the owner type of the key. The value of `nametype` must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user(KEY)) or OTHER (DNSKEY). These values are case insensitive.

-C
Compatibility mode: generates an old-style key, without any metadata. By default, **dnssec-keyfromlabel** will include the key's creation date in the metadata stored with the private key, and other dates may be set there as well (publication date, activation date, etc). Keys that include this data may be incompatible with older versions of BIND; the `-C` option suppresses them.

-c ***class***
Indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-f ***flag***
Set the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flags are KSK (Key Signing Key) and REVOKE.

-G
Generate a key, but do not publish it or sign with it. This option is incompatible with -P and -A.

-h
Prints a short summary of the options and arguments to **dnssec-keyfromlabel**.

-K ***directory***
Sets the directory in which the key files are to be written.

-k
Generate KEY records rather than DNSKEY records.

-L ***ttl***
Sets the default TTL to use for this key when it is converted into a DNSKEY RR. If the key is imported into a zone, this is the TTL that will be used for it, unless there was already a DNSKEY RRset in place, in which case the existing TTL would take precedence. Setting the default TTL to `0` or `none` removes it.

-p ***protocol***
Sets the protocol value for the key. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in RFC 2535 and its successors.

-S ***key***
Generate a key as an explicit successor to an existing key. The name, algorithm, size, and type of the key will be set to match the predecessor. The activation date of the new key will be set to the inactivation date of the existing one. The publication date will be set to the activation date minus the prepublication interval, which defaults to 30 days.

-t *type*
Indicates the use of the key. `type` must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF the ability to encrypt data.

-v *level*
Sets the debugging level.

-V
Prints version information.

-y
Allows DNSSEC key files to be generated even if the key ID would collide with that of an existing key, in the event of either key being revoked. (This is only safe to use if you are sure you won't be using RFC 5011 trust anchor maintenance with either of the keys involved.)

TIMING OPTIONS

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a '+' or '-', it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use 'none' or 'never'.

-P *date/offset*
Sets the date on which a key is to be published to the zone. After that date, the key will be included in the zone but will not be used to sign it. If not set, and if the -G option has not been used, the default is "now".

-P sync *date/offset*
Sets the date on which the CDS and CDNSKEY records which match this key are to be published to the zone.

-A *date/offset*
Sets the date on which the key is to be activated. After that date, the key will be included in the zone and used to sign it. If not set, and if the -G option has not been used, the default is "now".

-R *date/offset*
Sets the date on which the key is to be revoked. After that date, the key will be flagged as revoked. It will be included in the zone and will be used to sign it.

-I *date/offset*
Sets the date on which the key is to be retired. After that date, the key will still be included in the zone, but it will not be used to sign it.

-D *date/offset*
Sets the date on which the key is to be deleted. After that date, the key will no longer be included in the zone. (It may remain in the key repository, however.)

-D sync ***date/offset***
Sets the date on which the CDS and CDNSKEY records which match this key are to be deleted.

-i ***interval***
Sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date isn't, then the publication date will default to this much time before the activation date; conversely, if the publication date is specified but activation date isn't, then activation will be set to this much time after publication.

If the key is being created as an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

As with date offsets, if the argument is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

GENERATED KEY FILES

When **dnssec-keyfromlabel** completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key files it has generated.

- `nnnn` is the key name.
- `aaa` is the numeric representation of the algorithm.
- `iiiii` is the key identifier (or footprint).

dnssec-keyfromlabel creates two files, with names based on the printed string. `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNS KEY record that can be inserted into a zone file (directly or with a $INCLUDE statement).

The `.private` file contains algorithm-specific fields. For obvious security reasons, this file does not have general read permission.

SEE ALSO

dnssec-keygen(8), dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 4034*, *The PKCS#11 URI Scheme (draft-pechanec-pkcs11uri-13)*.

9.10 DNSSEC-KEYGEN

dnssec-keygen — DNSSEC key generation tool

Synopsis

dnssec-keygen [-3] [-A *date/offset*] [-a *algorithm*] [-b *keysize*] [-C] [-c *class*] [-D *date/offset*] [-D sync *date/offset*] [-E *engine*] [-f *flag*] [-G] [-g *generator*] [-h] [-I *date/offset*] [-i *interval*] [-K *directory*] [-k] [-L *ttl*] [-n *nametype*] [-P *date/offset*] [-P sync *date/offset*] [-p *protocol*] [-q] [-R *date/offset*] [-r *randomdev*] [-S *key*] [-s *strength*] [-t *type*] [-V] [-v *level*] name

DESCRIPTION

dnssec-keygen generates keys for DNSSEC (Secure DNS), as defined in RFC 2535 and RFC 4034. It can also generate keys for use with TSIG (Transaction Signatures) as defined in RFC 2845, or TKEY (Transaction Key) as defined in RFC 2930.

The `name` of the key is specified on the command line. For DNSSEC keys, this must match the name of the zone for which the key is being generated.

OPTIONS

-3

Use an NSEC3-capable algorithm to generate a DNSSEC key. If this option is used with an algorithm that has both NSEC and NSEC3 versions, then the NSEC3 version will be used; for example, **dnssec-keygen -3a RSASHA1** specifies the NSEC3RSASHA1 algorithm.

-a *algorithm*

Selects the cryptographic algorithm. For DNSSEC keys, the value of `algorithm` must be one of RSAMD5, RSASHA1, DSA, NSEC3RSASHA1, NSEC3DSA, RSASHA256, RSASHA512, ECCGOST, ECDSAP256SHA256, ECDSAP384SHA384, ED25519 or ED448. For TSIG/TKEY, the value must be DH (Diffie Hellman), HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384, or HMAC-SHA512. These values are case insensitive.

If no algorithm is specified, then RSASHA1 will be used by default, unless the `-3` option is specified, in which case NSEC3RSASHA1 will be used instead. (If `-3` is used and an algorithm is specified, that algorithm will be checked for compatibility with NSEC3.)

Note 1: that for DNSSEC, RSASHA1 is a mandatory to implement algorithm, and DSA is recommended. For TSIG, HMAC-MD5 is mandatory.

Note 2: DH, HMAC-MD5, and HMAC-SHA1 through HMAC-SHA512 automatically set the -T KEY option.

-b *keysize*

Specifies the number of bits in the key. The choice of key size depends on the algorithm used. RSA keys must be between 512 and 2048 bits. Diffie Hellman keys must be between 128 and 4096 bits. DSA keys must be between 512 and 1024 bits and an exact multiple of 64. HMAC keys must be between 1 and 512 bits. Elliptic curve algorithms don't need this parameter.

The key size does not need to be specified if using a default algorithm. The default key size is 1024 bits for zone signing keys (ZSKs) and 2048 bits for key signing keys (KSKs, generated with `-f KSK`). However, if an algorithm is explicitly specified with the `-a`, then there is no default key size, and the `-b` must be used.

-C

Compatibility mode: generates an old-style key, without any timing metadata. By default, **dnssec-keygen** will include the key's creation date in the metadata stored with the private key, and other dates may be set there as well (publication date, activation date, etc). Keys that include this data may be incompatible with older versions of BIND; the `-C` option suppresses them.

-c *class*

Indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-E *engine*

Specifies the cryptographic hardware to use, when applicable.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-f *flag*

Set the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flags are KSK (Key Signing Key) and REVOKE.

-G

Generate a key, but do not publish it or sign with it. This option is incompatible with -P and -A.

-g *generator*

If generating a Diffie Hellman key, use this generator. Allowed values are 2 and 5. If no generator is specified, a known prime from RFC 2539 will be used if possible; otherwise the default is 2.

-h

Prints a short summary of the options and arguments to **dnssec-keygen**.

-K *directory*

Sets the directory in which the key files are to be written.

-k

Deprecated in favor of -T KEY.

-L *ttl*

Sets the default TTL to use for this key when it is converted into a DNSKEY RR. If the key is imported into a zone, this is the TTL that will be used for it, unless there was already a DNSKEY RRset in place, in which case the existing TTL would take precedence. If this value is not set and there is no existing DNSKEY RRset, the TTL will default to the SOA TTL. Setting the default TTL to `0` or `none` is the same as leaving it unset.

-n *nametype*

Specifies the owner type of the key. The value of `nametype` must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user(KEY)) or OTHER (DNSKEY). These values are case insensitive. Defaults to ZONE for DNSKEY generation.

-p *protocol*
Sets the protocol value for the generated key, for use with `-T KEY`. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in RFC 2535 and its successors.

-q
Quiet mode: Suppresses unnecessary output, including progress indication. Without this option, when **dnssec-keygen** is run interactively to generate an RSA or DSA key pair, it will print a string of symbols to `stderr` indicating the progress of the key generation. A '.' indicates that a random number has been found which passed an initial sieve test; '+' means a number has passed a single round of the Miller-Rabin primality test; a space means that the number has passed all the tests and is a satisfactory key.

-r *randomdev*
Specifies the source of randomness. If the operating system does not provide a `/dev/random` or equivalent device, the default source of randomness is keyboard input. `randomdev` specifies the name of a character device or file containing random data to be used instead of the default. The special value `keyboard` indicates that keyboard input should be used.

-S *key*
Create a new key which is an explicit successor to an existing key. The name, algorithm, size, and type of the key will be set to match the existing key. The activation date of the new key will be set to the inactivation date of the existing one. The publication date will be set to the activation date minus the prepublication interval, which defaults to 30 days.

-s *strength*
Specifies the strength value of the key. The strength is a number between 0 and 15, and currently has no defined purpose in DNSSEC.

-T *rrtype*
Specifies the resource record type to use for the key. `rrtype` must be either DNSKEY or KEY. The default is DNSKEY when using a DNSSEC algorithm, but it can be overridden to KEY for use with SIG(0).

Using any TSIG algorithm (HMAC-* or DH) forces this option to KEY.

-t *type*
Indicates the use of the key, for use with `-T KEY`. `type` must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF the ability to encrypt data.

-V
Prints version information.

-v *level*
Sets the debugging level.

TIMING OPTIONS

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a '+' or '-', it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the offset

is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use 'none' or 'never'.

-P ***date/offset***
Sets the date on which a key is to be published to the zone. After that date, the key will be included in the zone but will not be used to sign it. If not set, and if the -G option has not been used, the default is "now".

-P sync ***date/offset***
Sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

-A ***date/offset***
Sets the date on which the key is to be activated. After that date, the key will be included in the zone and used to sign it. If not set, and if the -G option has not been used, the default is "now". If set, if and -P is not set, then the publication date will be set to the activation date minus the prepublication interval.

-R ***date/offset***
Sets the date on which the key is to be revoked. After that date, the key will be flagged as revoked. It will be included in the zone and will be used to sign it.

-I ***date/offset***
Sets the date on which the key is to be retired. After that date, the key will still be included in the zone, but it will not be used to sign it.

-D ***date/offset***
Sets the date on which the key is to be deleted. After that date, the key will no longer be included in the zone. (It may remain in the key repository, however.)

-D sync ***date/offset***
Sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

-i ***interval***
Sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date isn't, then the publication date will default to this much time before the activation date; conversely, if the publication date is specified but activation date isn't, then activation will be set to this much time after publication.

If the key is being created as an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

As with date offsets, if the argument is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

GENERATED KEYS

When **dnssec-keygen** completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key it has generated.

- `nnnn` is the key name.
- `aaa` is the numeric representation of the algorithm.
- `iiiii` is the key identifier (or footprint).

dnssec-keygen creates two files, with names based on the printed string. `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNS KEY record that can be inserted into a zone file (directly or with a $INCLUDE statement).

The `.private` file contains algorithm-specific fields. For obvious security reasons, this file does not have general read permission.

Both `.key` and `.private` files are generated for symmetric cryptography algorithms such as HMAC-MD5, even though the public and private key are equivalent.

EXAMPLE

To generate a 768-bit DSA key for the domain **`example.com`**, the following command would be issued:

```
dnssec-keygen -a DSA -b 768 -n ZONE example.com
```

The command would print a string of the form:

```
Kexample.com.+003+26160
```

In this example, **dnssec-keygen** creates the files `Kexample.com.+003+26160.key` and `Kexample.com.+003+26160.private`.

To generate a matching key-signing key, issue the command:

```
dnssec-keygen -a DSA -b 768 -n ZONE -f KSK example.com
```

SEE ALSO

dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 2539*, *RFC 2845*, *RFC 4034*.

9.11 DNSSEC-KEYMGR

dnssec-keymgr — Ensures correct DNSKEY coverage for a zone based on a defined policy

Synopsis

`dnssec-keymgr` [`-K` *directory*] [`-c` *file*] [`-f`] [`-k`] [`-q`] [`-v`] [`-z`] [`-g` *path*] [`-r` *path*] [`-s` *path*] [zone...]

DESCRIPTION

dnssec-keymgr is a high level Python wrapper to facilitate the key rollover process for zones handled by BIND. It uses the BIND commands for manipulating DNSSEC key metadata: **dnssec-keygen** and **dnssec-settime**.

DNSSEC policy can be read from a configuration file (default `/etc/dnssec-policy.conf`), from which the key parameters, publication and rollover schedule, and desired coverage duration for any given zone can be determined. This file may be used to define individual DNSSEC policies on a per-zone basis, or to set a "`default`" policy used for all zones.

When **dnssec-keymgr** runs, it examines the DNSSEC keys for one or more zones, comparing their timing metadata against the policies for those zones. If key settings do not conform to the DNSSEC policy (for example, because the policy has been changed), they are automatically corrected.

A zone policy can specify a duration for which we want to ensure the key correctness (`coverage`). It can also specify a rollover period (`roll-period`). If policy indicates that a key should roll over before the coverage period ends, then a successor key will automatically be created and added to the end of the key series.

If zones are specified on the command line, **dnssec-keymgr** will examine only those zones. If a specified zone does not already have keys in place, then keys will be generated for it according to policy.

If zones are *not* specified on the command line, then **dnssec-keymgr** will search the key directory (either the current working directory or the directory set by the `-K` option), and check the keys for all the zones represented in the directory.

Key times that are in the past will not be updated unless the `-f` is used (see below). Key inactivation and deletion times that are less than five minutes in the future will be delayed by five minutes.

It is expected that this tool will be run automatically and unattended (for example, by **cron**).

OPTIONS

-c *file*
: If `-c` is specified, then the DNSSEC policy is read from `file`. (If not specified, then the policy is read from `/etc/dnssec-policy.conf`; if that file doesn't exist, a built-in global default policy is used.)

-f
: Force: allow updating of key events even if they are already in the past. This is not recommended for use with zones in which keys have already been published. However, if a set of keys has been generated all of which have publication and activation dates in the past, but the keys have not been published in a zone as yet, then this option can be used to clean them up and turn them into a proper series of keys with appropriate rollover intervals.

-g *keygen-path*
: Specifies a path to a **dnssec-keygen** binary. Used for testing. See also the `-s` option.

-h
Print the **dnssec-keymgr** help summary and exit.

-K *directory*
Sets the directory in which keys can be found. Defaults to the current working directory.

-k
Only apply policies to KSK keys. See also the `-z` option.

-q
Quiet: suppress printing of **dnssec-keygen** and **dnssec-settime**.

-r *randomdev*
Specifies a path to a file containing random data. This is passed to the **dnssec-keygen** binary using its `-r` option.

-s *settime-path*
Specifies a path to a **dnssec-settime** binary. Used for testing. See also the `-g` option.

-v
Print the **dnssec-keymgr** version and exit.

-z
Only apply policies to ZSK keys. See also the `-k` option.

POLICY CONFIGURATION

The `dnssec-policy.conf` file can specify three kinds of policies:

- *Policy classes* (`policy` *`name`* `{ ... };`) can be inherited by zone policies or other policy classes; these can be used to create sets of different security profiles. For example, a policy class **`normal`** might specify 1024-bit key sizes, but a class **`extra`** might specify 2048 bits instead; **`extra`** would be used for zones that had unusually high security needs.
- *Algorithm policies:* (`algorithm-policy` *`algorithm`* `{ ... };`) override default per-algorithm settings. For example, by default, RSASHA256 keys use 2048-bit key sizes for both KSK and ZSK. This can be modified using **algorithm-policy**, and the new key sizes would then be used for any key of type RSASHA256.
- *Zone policies:* (`zone` *`name`* `{ ... };`) set policy for a single zone by name. A zone policy can inherit a policy class by including a `policy` option. Zone names beginning with digits (i.e., 0-9) must be quoted. If a zone does not have its own policy then the "`default`" policy applies.

Options that can be specified in policies:

algorithm *`name`*;
The key algorithm. If no policy is defined, the default is RSASHA256.

coverage *duration*;
The length of time to ensure that keys will be correct; no action will be taken to create new keys to be activated after this time. This can be represented as a number of seconds, or as a duration using human-readable units (examples: "1y" or "6 months"). A default value for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is six months.

directory *path*;
Specifies the directory in which keys should be stored.

key-size *keytype size*;
Specifies the number of bits to use in creating keys. The keytype is either "zsk" or "ksk". A default value for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is 1024 bits for DSA keys and 2048 for RSA.

keyttl *duration*;
The key TTL. If no policy is defined, the default is one hour.

post-publish *keytype duration*;
How long after inactivation a key should be deleted from the zone. Note: If `roll-period` is not set, this value is ignored. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. The default is one month.

pre-publish *keytype duration*;
How long before activation a key should be published. Note: If `roll-period` is not set, this value is ignored. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. The default is one month.

roll-period *keytype duration*;
How frequently keys should be rolled over. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is one year for ZSKs. KSKs do not roll over by default.

standby *keytype number*;
Not yet implemented.

REMAINING WORK

- Enable scheduling of KSK rollovers using the `-P sync` and `-D sync` options to **dnssec-keygen** and **dnssec-settime**. Check the parent zone (as in **dnssec-checkds**) to determine when it's safe for the key to roll.
- Allow configuration of standby keys and use of the REVOKE bit, for keys that use RFC 5011 semantics.

SEE ALSO

dnssec-coverage(8), dnssec-keygen(8), dnssec-settime(8), dnssec-checkds(8)

9.12 DNSSEC-REVOKE

dnssec-revoke — set the REVOKED bit on a DNSSEC key

Synopsis

`dnssec-revoke` [`-hr`] [`-v` *`level`*] [`-V`] [`-K` *`directory`*] [`-E` *`engine`*] [`-f`] [`-R`] keyfile

DESCRIPTION

dnssec-revoke reads a DNSSEC key file, sets the REVOKED bit on the key as defined in RFC 5011, and creates a new pair of key files containing the now-revoked key.

OPTIONS

-h
Emit usage message and exit.

-K *`directory`*
Sets the directory in which the key files are to reside.

-r
After writing the new keyset files remove the original keyset files.

-v *`level`*
Sets the debugging level.

-V
Prints version information.

-E *`engine`*
Specifies the cryptographic hardware to use, when applicable.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-f
Force overwrite: Causes **dnssec-revoke** to write the new key pair even if a file already exists matching the algorithm and key ID of the revoked key.

-R
Print the key tag of the key with the REVOKE bit set but do not revoke the key.

SEE ALSO

dnssec-keygen(8), *BIND 9 Administrator Reference Manual*, *RFC 5011*.

9.13 DNSSEC-SETTIME

dnssec-settime — set the key timing metadata for a DNSSEC key

Synopsis

`dnssec-settime` [`-f`] [`-K` *directory*] [`-L` *ttl*] [`-P` *date/offset*] [`-P sync` *date/offset*] [`-A` *date/offset*] [`-R` *date/offset*] [`-I` *date/offset*] [`-D` *date/offset*] [`-D sync` *date/offset*] [`-S` *key*] [`-i` *interval*] [`-h`] [`-V`] [`-v` *level*] [`-E` *engine*] keyfile

DESCRIPTION

dnssec-settime reads a DNSSEC private key file and sets the key timing metadata as specified by the `-P`, `-A`, `-R`, `-I`, and `-D` options. The metadata can then be used by **dnssec-signzone** or other signing software to determine when a key is to be published, whether it should be used for signing a zone, etc.

If none of these options is set on the command line, then **dnssec-settime** simply prints the key timing metadata already stored in the key.

When key metadata fields are changed, both files of a key pair (`Knnnn.+aaa+iiiii.key` and `Knnnn.+aaa+iiiii.private`) are regenerated. Metadata fields are stored in the private file. A human-readable description of the metadata is also placed in comments in the key file. The private file's permissions are always set to be inaccessible to anyone other than the owner (mode 0600).

OPTIONS

-f
: Force an update of an old-format key with no metadata fields. Without this option, **dnssec-settime** will fail when attempting to update a legacy key. With this option, the key will be recreated in the new format, but with the original key data retained. The key's creation date will be set to the present time. If no other values are specified, then the key's publication and activation dates will also be set to the present time.

-K *directory*
: Sets the directory in which the key files are to reside.

-L *ttl*
: Sets the default TTL to use for this key when it is converted into a DNSKEY RR. If the key is imported into a zone, this is the TTL that will be used for it, unless there was already a DNSKEY RRset in place, in which case the existing TTL would take precedence. If this value is not set and there is no existing DNSKEY RRset, the TTL will default to the SOA TTL. Setting the default TTL to `0` or `none` removes it from the key.

-h
: Emit usage message and exit.

-V
Prints version information.

-v *level*
Sets the debugging level.

-E *engine*
Specifies the cryptographic hardware to use, when applicable.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

TIMING OPTIONS

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a '+' or '-', it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To unset a date, use 'none' or 'never'.

-P *date/offset*
Sets the date on which a key is to be published to the zone. After that date, the key will be included in the zone but will not be used to sign it.

-P sync *date/offset*
Sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

-A *date/offset*
Sets the date on which the key is to be activated. After that date, the key will be included in the zone and used to sign it.

-R *date/offset*
Sets the date on which the key is to be revoked. After that date, the key will be flagged as revoked. It will be included in the zone and will be used to sign it.

-I *date/offset*
Sets the date on which the key is to be retired. After that date, the key will still be included in the zone, but it will not be used to sign it.

-D *date/offset*
Sets the date on which the key is to be deleted. After that date, the key will no longer be included in the zone. (It may remain in the key repository, however.)

-D sync *date/offset*
Sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

-S ***predecessor key***
Select a key for which the key being modified will be an explicit successor. The name, algorithm, size, and type of the predecessor key must exactly match those of the key being modified. The activation date of the successor key will be set to the inactivation date of the predecessor. The publication date will be set to the activation date minus the prepublication interval, which defaults to 30 days.

-i ***interval***
Sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date isn't, then the publication date will default to this much time before the activation date; conversely, if the publication date is specified but activation date isn't, then activation will be set to this much time after publication.

If the key is being set to be an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

As with date offsets, if the argument is followed by one of the suffixes 'y', 'mo', 'w', 'd', 'h', or 'mi', then the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

PRINTING OPTIONS

dnssec-settime can also be used to print the timing metadata associated with a key.

-u
Print times in UNIX epoch format.

-p ***C/P/Psync/A/R/I/D/Dsync/all***
Print a specific metadata value or set of metadata values. The `-p` option may be followed by one or more of the following letters or strings to indicate which value or values to print: `C` for the creation date, `P` for the publication date, `Psync` for the CDS and CDNSKEY publication date, `A` for the activation date, `R` for the revocation date, `I` for the inactivation date, `D` for the deletion date, and `Dsync` for the CDS and CDNSKEY deletion date To print all of the metadata, use `-p all`.

SEE ALSO

dnssec-keygen(8), dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 5011*.

9.14 DNSSEC-SIGNZONE

dnssec-signzone — DNSSEC zone signing tool

Synopsis

dnssec-signzone [-a] [-c *class*] [-d *directory*] [-D] [-E *engine*] [-e *end-time*] [-f *output-file*] [-g] [-h] [-i *interval*] [-I *input-format*] [-j *jitter*] [-K *directory*] [-k *key*] [-L *serial*] [-l *domain*] [-M *maxttl*] [-N *soa-serial-format*] [-o *origin*] [-O *output-format*] [-P] [-p] [-Q] [-R] [-r *randomdev*] [-S] [-s *start-time*] [-T *ttl*] [-t] [-u] [-v *level*] [-V] [-X *extended end-time*] [-x] [-z] [-3 *salt*] [-H *iterations*] [-A] zonefile [key...]

DESCRIPTION

dnssec-signzone signs a zone. It generates NSEC and RRSIG records and produces a signed version of the zone. The security status of delegations from the signed zone (that is, whether the child zones are secure or not) is determined by the presence or absence of a `keyset` file for each child zone.

OPTIONS

-a
Verify all generated signatures.

-c *class*
Specifies the DNS class of the zone.

-C
Compatibility mode: Generate a `keyset-`*zonename* file in addition to `dsset-`*zonename* when signing a zone, for use by older versions of **dnssec-signzone**.

-d *directory*
Look for `dsset-` or `keyset-` files in `directory`.

-D
Output only those record types automatically managed by **dnssec-signzone**, i.e. RRSIG, NSEC, NSEC3 and NSEC3PARAM records. If smart signing (`-S`) is used, DNSKEY records are also included. The resulting file can be included in the original zone file with **$INCLUDE**. This option cannot be combined with `-O raw`, `-O map`, or serial number updating.

-E *engine*
When applicable, specifies the hardware to use for cryptographic operations, such as a secure key store used for signing.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-g
Generate DS records for child zones from `dsset-` or `keyset-` file. Existing DS records will be removed.

-K ***directory***
Key repository: Specify a directory to search for DNSSEC keys. If not specified, defaults to the current directory.

-k ***key***
Treat specified key as a key signing key ignoring any key flags. This option may be specified multiple times.

-l ***domain***
Generate a DLV set in addition to the key (DNSKEY) and DS sets. The domain is appended to the name of the records.

-M ***maxttl***
Sets the maximum TTL for the signed zone. Any TTL higher than *maxttl* in the input zone will be reduced to *maxttl* in the output. This provides certainty as to the largest possible TTL in the signed zone, which is useful to know when rolling keys because it is the longest possible time before signatures that have been retrieved by resolvers will expire from resolver caches. Zones that are signed with this option should be configured to use a matching `max-zone-ttl` in `named.conf`. (Note: This option is incompatible with `-D`, because it modifies non-DNSSEC data in the output zone.)

-s ***start-time***
Specify the date and time when the generated RRSIG records become valid. This can be either an absolute or relative time. An absolute start time is indicated by a number in YYYYMMDDHHMMSS notation; 20000530144500 denotes 14:45:00 UTC on May 30th, 2000. A relative start time is indicated by +N, which is N seconds from the current time. If no `start-time` is specified, the current time minus 1 hour (to allow for clock skew) is used.

-e ***end-time***
Specify the date and time when the generated RRSIG records expire. As with `start-time`, an absolute time is indicated in YYYYMMDDHHMMSS notation. A time relative to the start time is indicated with +N, which is N seconds from the start time. A time relative to the current time is indicated with now+N. If no `end-time` is specified, 30 days from the start time is used as a default. `end-time` must be later than `start-time`.

-X ***extended end-time***
Specify the date and time when the generated RRSIG records for the DNSKEY RRset will expire. This is to be used in cases when the DNSKEY signatures need to persist longer than signatures on other records; e.g., when the private component of the KSK is kept offline and the KSK signature is to be refreshed manually.

As with `start-time`, an absolute time is indicated in YYYYMMDDHHMMSS notation. A time relative to the start time is indicated with +N, which is N seconds from the start time. A time relative to the current time is indicated with now+N. If no `extended end-time` is specified, the value of `end-time` is used as the default. (`end-time`, in turn, defaults to 30 days from the start time.) `extended end-time` must be later than `start-time`.

-f ***output-file***
The name of the output file containing the signed zone. The default is to append `.signed` to the input filename. If `output-file` is set to `"-"`, then the signed zone is written to the standard output, with a default output format of "full".

-h
Prints a short summary of the options and arguments to **dnssec-signzone**.

-V
Prints version information.

-i *interval*
When a previously-signed zone is passed as input, records may be resigned. The `interval` option specifies the cycle interval as an offset from the current time (in seconds). If a RRSIG record expires after the cycle interval, it is retained. Otherwise, it is considered to be expiring soon, and it will be replaced.

The default cycle interval is one quarter of the difference between the signature end and start times. So if neither `end-time` or `start-time` are specified, **dnssec-signzone** generates signatures that are valid for 30 days, with a cycle interval of 7.5 days. Therefore, if any existing RRSIG records are due to expire in less than 7.5 days, they would be replaced.

-I *input-format*
The format of the input zone file. Possible formats are **"text"** (default), **"raw"**, and **"map"**. This option is primarily intended to be used for dynamic signed zones so that the dumped zone file in a non-text format containing updates can be signed directly. The use of this option does not make much sense for non-dynamic zones.

-j *jitter*
When signing a zone with a fixed signature lifetime, all RRSIG records issued at the time of signing expires simultaneously. If the zone is incrementally signed, i.e. a previously-signed zone is passed as input to the signer, all expired signatures have to be regenerated at about the same time. The `jitter` option specifies a jitter window that will be used to randomize the signature expire time, thus spreading incremental signature regeneration over time.

Signature lifetime jitter also to some extent benefits validators and servers by spreading out cache expiration, i.e. if large numbers of RRSIGs don't expire at the same time from all caches there will be less congestion than if all validators need to refetch at mostly the same time.

-L *serial*
When writing a signed zone to "raw" or "map" format, set the "source serial" value in the header to the specified serial number. (This is expected to be used primarily for testing purposes.)

-n *ncpus*
Specifies the number of threads to use. By default, one thread is started for each detected CPU.

-N *soa-serial-format*
The SOA serial number format of the signed zone. Possible formats are **"keep"** (default), **"increment"**, **"unixtime"**, and **"date"**.

"keep"
Do not modify the SOA serial number.

"increment"
Increment the SOA serial number using RFC 1982 arithmetics.

"unixtime"
Set the SOA serial number to the number of seconds since epoch.

"date"
Set the SOA serial number to today's date in YYYYMMDDNN format.

-o *origin*
The zone origin. If not specified, the name of the zone file is assumed to be the origin.

-O *output-format*
The format of the output file containing the signed zone. Possible formats are **"text"** (default), which is the standard textual representation of the zone; **"full"**, which is text output in a format suitable for processing by external scripts; and **"map"**, **"raw"**, and **"raw=N"**, which store the zone in binary formats for rapid loading by **named**. **"raw=N"** specifies the format version of the raw zone file: if N is 0, the raw file can be read by any version of **named**; if N is 1, the file can be read by release 9.9.0 or higher; the default is 1.

-p
Use pseudo-random data when signing the zone. This is faster, but less secure, than using real random data. This option may be useful when signing large zones or when the entropy source is limited.

-P
Disable post sign verification tests.

The post sign verification test ensures that for each algorithm in use there is at least one non revoked self signed KSK key, that all revoked KSK keys are self signed, and that all records in the zone are signed by the algorithm. This option skips these tests.

-Q
Remove signatures from keys that are no longer active.

Normally, when a previously-signed zone is passed as input to the signer, and a DNSKEY record has been removed and replaced with a new one, signatures from the old key that are still within their validity period are retained. This allows the zone to continue to validate with cached copies of the old DNSKEY RRset. The `-Q` forces **dnssec-signzone** to remove signatures from keys that are no longer active. This enables ZSK rollover using the procedure described in RFC 4641, section 4.2.1.1 ("Pre-Publish Key Rollover").

-R
Remove signatures from keys that are no longer published.

This option is similar to `-Q`, except it forces **dnssec-signzone** to signatures from keys that are no longer published. This enables ZSK rollover using the procedure described in RFC 4641, section 4.2.1.2 ("Double Signature Zone Signing Key Rollover").

-r *randomdev*
Specifies the source of randomness. If the operating system does not provide a `/dev/random` or equivalent device, the default source of randomness is keyboard input. `randomdev` specifies the name of a character device or file containing random data to be used instead of the default. The special value `keyboard` indicates that keyboard input should be used.

-S
Smart signing: Instructs **dnssec-signzone** to search the key repository for keys that match the zone being signed, and to include them in the zone if appropriate.

When a key is found, its timing metadata is examined to determine how it should be used, according to the following rules. Each successive rule takes priority over the prior ones:

If no timing metadata has been set for the key, the key is published in the zone and used to sign the zone.

If the key's publication date is set and is in the past, the key is published in the zone.

If the key's activation date is set and in the past, the key is published (regardless of publication date) and used to sign the zone.

If the key's revocation date is set and in the past, and the key is published, then the key is revoked, and the revoked key is used to sign the zone.

If either of the key's unpublication or deletion dates are set and in the past, the key is NOT published or used to sign the zone, regardless of any other metadata.

-T ***ttl***

Specifies a TTL to be used for new DNSKEY records imported into the zone from the key repository. If not specified, the default is the TTL value from the zone's SOA record. This option is ignored when signing without `-S`, since DNSKEY records are not imported from the key repository in that case. It is also ignored if there are any pre-existing DNSKEY records at the zone apex, in which case new records' TTL values will be set to match them, or if any of the imported DNSKEY records had a default TTL value. In the event of a a conflict between TTL values in imported keys, the shortest one is used.

-t

Print statistics at completion.

-u

Update NSEC/NSEC3 chain when re-signing a previously signed zone. With this option, a zone signed with NSEC can be switched to NSEC3, or a zone signed with NSEC3 can be switch to NSEC or to NSEC3 with different parameters. Without this option, **dnssec-signzone** will retain the existing chain when re-signing.

-v ***level***

Sets the debugging level.

-x

Only sign the DNSKEY RRset with key-signing keys, and omit signatures from zone-signing keys. (This is similar to the **dnssec-dnskey-kskonly yes;** zone option in **named**.)

-z

Ignore KSK flag on key when determining what to sign. This causes KSK-flagged keys to sign all records, not just the DNSKEY RRset. (This is similar to the **update-check-ksk no;** zone option in **named**.)

-3 ***salt***

Generate an NSEC3 chain with the given hex encoded salt. A dash (`salt`) can be used to indicate that no salt is to be used when generating the NSEC3 chain.

-H ***iterations***
When generating an NSEC3 chain, use this many iterations. The default is 10.

-A
When generating an NSEC3 chain set the OPTOUT flag on all NSEC3 records and do not generate NSEC3 records for insecure delegations.

Using this option twice (i.e., `-AA`) turns the OPTOUT flag off for all records. This is useful when using the `-u` option to modify an NSEC3 chain which previously had OPTOUT set.

zonefile
The file containing the zone to be signed.

key
Specify which keys should be used to sign the zone. If no keys are specified, then the zone will be examined for DNSKEY records at the zone apex. If these are found and there are matching private keys, in the current directory, then these will be used for signing.

EXAMPLE

The following command signs the **`example.com`** zone with the DSA key generated by **dnssec-keygen** (Kexample.com.+003+17247). Because the **-S** option is not being used, the zone's keys must be in the master file (`db.example.com`). This invocation looks for `dsset` files, in the current directory, so that DS records can be imported from them (**-g**).

```
% dnssec-signzone -g -o example.com db.example.com \
Kexample.com.+003+17247
db.example.com.signed
%
```

In the above example, **dnssec-signzone** creates the file `db.example.com.signed`. This file should be referenced in a zone statement in a `named.conf` file.

This example re-signs a previously signed zone with default parameters. The private keys are assumed to be in the current directory.

```
% cp db.example.com.signed db.example.com
% dnssec-signzone -o example.com db.example.com
db.example.com.signed
%
```

SEE ALSO

dnssec-keygen(8), *BIND 9 Administrator Reference Manual*, *RFC 4033*, *RFC 4641*.

9.15 DNSSEC-VERIFY

dnssec-verify — DNSSEC zone verification tool

Synopsis

`dnssec-verify` [`-c` *class*] [`-E` *engine*] [`-I` *input-format*] [`-o` *origin*] [`-v` *level*] [`-V`] [`-x`] [`-z`] zonefile

DESCRIPTION

dnssec-verify verifies that a zone is fully signed for each algorithm found in the DNSKEY RRset for the zone, and that the NSEC / NSEC3 chains are complete.

OPTIONS

-c *class*
Specifies the DNS class of the zone.

-E *engine*
Specifies the cryptographic hardware to use, when applicable.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-I *input-format*
The format of the input zone file. Possible formats are **"text"** (default) and **"raw"**. This option is primarily intended to be used for dynamic signed zones so that the dumped zone file in a non-text format containing updates can be verified independently. The use of this option does not make much sense for non-dynamic zones.

-o *origin*
The zone origin. If not specified, the name of the zone file is assumed to be the origin.

-v *level*
Sets the debugging level.

-V
Prints version information.

-x
Only verify that the DNSKEY RRset is signed with key-signing keys. Without this flag, it is assumed that the DNSKEY RRset will be signed by all active keys. When this flag is set, it will not be an error if the DNSKEY RRset is not signed by zone-signing keys. This corresponds to the `-x` option in **dnssec-signzone**.

-z
Ignore the KSK flag on the keys when determining whether the zone if correctly signed. Without this flag it is assumed that there will be a non-revoked, self-signed DNSKEY with the KSK flag set for each algorithm and that RRsets other than DNSKEY RRset will be signed with a different DNSKEY without the KSK flag set.

With this flag set, we only require that for each algorithm, there will be at least one non-revoked, self-signed DNSKEY, regardless of the KSK flag state, and that other RRsets will be signed by a non-revoked key for the same algorithm that includes the self-signed key; the same key may be used for both purposes. This corresponds to the `-z` option in **dnssec-signzone**.

zonefile
The file containing the zone to be signed.

SEE ALSO

dnssec-signzone(8), *BIND 9 Administrator Reference Manual*, *RFC 4033*.

9.16 DNSTAP-READ

dnstap-read — print dnstap data in human-readable form

Synopsis

`dnstap-read` [`-m`] [`-p`] [`-y`] *`file`*

DESCRIPTION

dnstap-read reads **dnstap** data from a specified file and prints it in a human-readable format. By default, **dnstap** data is printed in a short summary format, but if the `-y` option is specified, then a longer and more detailed YAML format is used instead.

OPTIONS

-m
Trace memory allocations; used for debugging memory leaks.

-p
After printing the **dnstap** data, print the text form of the DNS message that was encapsulated in the **dnstap** frame.

-y
Print **dnstap** data in a detailed YAML format.

SEE ALSO

named(8), rndc(8), *BIND 9 Administrator Reference Manual*.

9.17 GENRANDOM

genrandom — generate a file containing random data

Synopsis

`genrandom` [`-n` *number*] *size filename*

DESCRIPTION

genrandom generates a file or a set of files containing a specified quantity of pseudo-random data, which can be used as a source of entropy for other commands on systems with no random device.

ARGUMENTS

-n *number*
In place of generating one file, generates `number` (from 2 to 9) files, appending `number` to the name.

size
The size of the file, in kilobytes, to generate.

filename
The file name into which random data should be written.

SEE ALSO

rand(3), arc4random(3)

9.18 HOST

host — DNS lookup utility

Synopsis

`host` [`-aCdlnrsTUwv`] [`-c` *class*] [`-N` *ndots*] [`-p` *port*] [`-R` *number*] [`-t` *type*] [`-W` *wait*] [`-m` *flag*] [`-4` | `-6`] [`-v`] [`-V`] name [server]

DESCRIPTION

host is a simple utility for performing DNS lookups. It is normally used to convert names to IP addresses and vice versa. When no arguments or options are given, **host** prints a short summary of its command line arguments and options.

name is the domain name that is to be looked up. It can also be a dotted-decimal IPv4 address or a colon-delimited IPv6 address, in which case **host** will by default perform a reverse lookup for that address. *server* is an optional argument which is either the name or IP address of the name server that **host** should query instead of the server or servers listed in `/etc/resolv.conf`.

OPTIONS

-4
Use IPv4 only for query transport. See also the `-6` option.

-6
Use IPv6 only for query transport. See also the `-4` option.

-a
"All". The `-a` option is normally equivalent to `-v -t ANY`. It also affects the behaviour of the `-l` list zone option.

-c *class*
Query class: This can be used to lookup HS (Hesiod) or CH (Chaosnet) class resource records. The default class is IN (Internet).

-C
Check consistency: **host** will query the SOA records for zone *name* from all the listed authoritative name servers for that zone. The list of name servers is defined by the NS records that are found for the zone.

-d
Print debugging traces. Equivalent to the `-v` verbose option.

-i
Obsolete. Use the IP6.INT domain for reverse lookups of IPv6 addresses as defined in RFC1886 and deprecated in RFC4159. The default is to use IP6.ARPA as specified in RFC3596.

-l
List zone: The **host** command performs a zone transfer of zone *name* and prints out the NS, PTR and address records (A/AAAA).

Together, the `-l -a` options print all records in the zone.

-N *ndots*
The number of dots that have to be in *name* for it to be considered absolute. The default value is that defined using the ndots statement in `/etc/resolv.conf`, or 1 if no ndots statement is present. Names with fewer dots are interpreted as relative names and will be searched for in the domains listed in the search or domain directive in `/etc/resolv.conf`.

-p *port*
Specify the port on the server to query. The default is 53.

-r
Non-recursive query: Setting this option clears the RD (recursion desired) bit in the query. This should mean that the name server receiving the query will not attempt to resolve *name*. The `-r` option enables **host** to mimic the behavior of a name server by making non-recursive queries and expecting to receive answers to those queries that can be referrals to other name servers.

-R *number*
Number of retries for UDP queries: If *number* is negative or zero, the number of retries will default to 1. The default value is 1, or the value of the *attempts* option in `/etc/resolv.conf`, if set.

-s
Do *not* send the query to the next nameserver if any server responds with a SERVFAIL response, which is the reverse of normal stub resolver behavior.

-t *type*
Query type: The *type* argument can be any recognized query type: CNAME, NS, SOA, TXT, DNSKEY, AXFR, etc.

When no query type is specified, **host** automatically selects an appropriate query type. By default, it looks for A, AAAA, and MX records. If the `-C` option is given, queries will be made for SOA records. If *name* is a dotted-decimal IPv4 address or colon-delimited IPv6 address, **host** will query for PTR records.

If a query type of IXFR is chosen the starting serial number can be specified by appending an equal followed by the starting serial number (like `-t IXFR=12345678`).

-T, -U
TCP/UDP: By default, **host** uses UDP when making queries. The `-T` option makes it use a TCP connection when querying the name server. TCP will be automatically selected for queries that require it, such as zone transfer (AXFR) requests. Type ANY queries default to TCP but can be forced to UDP initially using `-U`.

-m *flag*
Memory usage debugging: the flag can be *record*, *usage*, or *trace*. You can specify the `-m` option more than once to set multiple flags.

-v
Verbose output. Equivalent to the `-d` debug option. Verbose output can also be enabled by setting the *debug* option in `/etc/resolv.conf`.

-V
Print the version number and exit.

-w
Wait forever: The query timeout is set to the maximum possible. See also the `-W` option.

-W *wait*
Timeout: Wait for up to *wait* seconds for a reply. If *wait* is less than one, the wait interval is set to one second.

By default, **host** will wait for 5 seconds for UDP responses and 10 seconds for TCP connections. These defaults can be overridden by the *timeout* option in `/etc/resolv.conf`.

See also the `-w` option.

IDN SUPPORT

If **host** has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. **host** appropriately converts character encoding of domain name before sending a request to DNS server or displaying a reply from the server. If you'd like to turn off the IDN support for some reason, defines the `IDN_DISABLE` environment variable. The IDN support is disabled if the variable is set when **host** runs.

FILES

`/etc/resolv.conf`

SEE ALSO

dig(1), named(8).

9.19 ISC-HMAC-FIXUP

isc-hmac-fixup — fixes HMAC keys generated by older versions of BIND

Synopsis

`isc-hmac-fixup` *algorithm secret*

DESCRIPTION

Versions of BIND 9 up to and including BIND 9.6 had a bug causing HMAC-SHA* TSIG keys which were longer than the digest length of the hash algorithm (i.e., SHA1 keys longer than 160 bits, SHA256 keys longer than 256 bits, etc) to be used incorrectly, generating a message authentication code that was incompatible with other DNS implementations.

This bug was fixed in BIND 9.7. However, the fix may cause incompatibility between older and newer versions of BIND, when using long keys. **isc-hmac-fixup** modifies those keys to restore compatibility.

To modify a key, run **isc-hmac-fixup** and specify the key's algorithm and secret on the command line. If the secret is longer than the digest length of the algorithm (64 bytes for SHA1 through SHA256, or 128 bytes for SHA384 and SHA512), then a new secret will be generated consisting of a hash digest of the old secret. (If the secret did not require conversion, then it will be printed without modification.)

SECURITY CONSIDERATIONS

Secrets that have been converted by **isc-hmac-fixup** are shortened, but as this is how the HMAC protocol works in operation anyway, it does not affect security. RFC 2104 notes, "Keys longer than [the digest length] are acceptable but the extra length would not significantly increase the function strength."

SEE ALSO

BIND 9 Administrator Reference Manual, *RFC 2104*.

9.20 LWRESD

lwresd — lightweight resolver daemon

Synopsis

`lwresd` [`-c` *config-file*] [`-C` *config-file*] [`-d` *debug-level*] [`-f`] [`-g`] [`-i` *pid-file*] [`-m` *flag*] [`-n` *#cpus*] [`-P` *port*] [`-p` *port*] [`-s`] [`-t` *directory*] [`-u` *user*] [`-v`] [`-4` | `-6`]

DESCRIPTION

lwresd is the daemon providing name lookup services to clients that use the BIND 9 lightweight resolver library. It is essentially a stripped-down, caching-only name server that answers queries using the BIND 9 lightweight resolver protocol rather than the DNS protocol.

lwresd listens for resolver queries on a UDP port on the IPv4 loopback interface, 127.0.0.1. This means that **lwresd** can only be used by processes running on the local machine. By default, UDP port number 921 is used for lightweight resolver requests and responses.

Incoming lightweight resolver requests are decoded by the server which then resolves them using the DNS protocol. When the DNS lookup completes, **lwresd** encodes the answers in the lightweight resolver format and returns them to the client that made the request.

If `/etc/resolv.conf` contains any `nameserver` entries, **lwresd** sends recursive DNS queries to those servers. This is similar to the use of forwarders in a caching name server. If no `nameserver` entries are present, or if forwarding fails, **lwresd** resolves the queries autonomously starting at the root name servers, using a built-in list of root server hints.

OPTIONS

-4

Use IPv4 only even if the host machine is capable of IPv6. `-4` and `-6` are mutually exclusive.

-6

Use IPv6 only even if the host machine is capable of IPv4. `-4` and `-6` are mutually exclusive.

-c *config-file*

Use *config-file* as the configuration file instead of the default, `/etc/lwresd.conf`. `-c` can not be used with `-C`.

-C *config-file*

Use *config-file* as the configuration file instead of the default, `/etc/resolv.conf`. `-C` can not be used with `-c`.

-d *debug-level*

Set the daemon's debug level to *debug-level*. Debugging traces from **lwresd** become more verbose as the debug level increases.

-f

Run the server in the foreground (i.e. do not daemonize).

-g

Run the server in the foreground and force all logging to `stderr`.

-i *pid-file*

Use *pid-file* as the PID file instead of the default, `/var/run/lwresd/lwresd.pid`.

-m *flag*

Turn on memory usage debugging flags. Possible flags are *usage*, *trace*, *record*, *size*, and *mctx*. These correspond to the ISC_MEM_DEBUGXXXX flags described in `<isc/mem.h>`.

-n *#cpus*

Create *#cpus* worker threads to take advantage of multiple CPUs. If not specified, **lwresd** 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 worker thread will be created.

-P *port*

Listen for lightweight resolver queries on port *port*. If not specified, the default is port 921.

-p *port*

Send DNS lookups to port *port*. If not specified, the default is port 53. This provides a way of testing the lightweight resolver daemon with a name server that listens for queries on a non-standard port number.

-s

Write memory usage statistics to `stdout` on exit.

> NOTE
>
> This option is mainly of interest to BIND 9 developers and may be removed or changed in a future release.

-t *directory*
Chroot to *directory* after processing the command line arguments, but before reading the configuration file.

> WARNING
>
> 
>
> This option should be used in conjunction with the `-u` option, as chrooting a process running as root doesn't enhance security on most systems; the way `chroot(2)` is defined allows a process with root privileges to escape a chroot jail.

-u *user*
Setuid to *user* after completing privileged operations, such as creating sockets that listen on privileged ports.

-v
Report the version number and exit.

FILES

/etc/resolv.conf
The default configuration file.

/var/run/lwresd.pid
The default process-id file.

SEE ALSO

named(8), lwres(3), resolver(5).

9.21 MDIG

mdig — DNS pipelined lookup utility

Synopsis

`mdig` @server [`-f` *filename*] [`-h`] [`-v`] [`-4 | -6`] [`-m`] [`-b` *address*] [`-p` *port#*] [`-c` *class*] [`-t` *type*] [`-i`] [`-x` *addr*] [plusopt...]

`mdig` -h

`mdig` [@server] global-opt... local-opt... query ...

DESCRIPTION

mdig is a multiple/pipelined query version of **dig**: instead of waiting for a response after sending each query, it begins by sending all queries. Responses are displayed in the order in which they are received, not in the order the corresponding queries were sent.

mdig options are a subset of the **dig** options, and are divided into "anywhere options" which can occur anywhere, "global options" which must occur before the query name (or they are ignored with a warning), and "local options" which apply to the next query on the command line.

The @server option is a mandatory global option. It is the name or IP address of the name server to query. (Unlike **dig**, this value is not retrieved from `/etc/resolv.conf`.) It can be an IPv4 address in dotted-decimal notation, an IPv6 address in colon-delimited notation, or a hostname. When the supplied *server* argument is a hostname, **mdig** resolves that name before querying the name server.

mdig provides a number of query options which affect the way in which lookups are made and the results displayed. Some of these set or reset flag bits in the query header, some determine which sections of the answer get printed, and others determine the timeout and retry strategies.

Each query option is identified by a keyword preceded by a plus sign (`+`). Some keywords set or reset an option. These may be preceded by the string `no` to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form `+keyword=value`.

ANYWHERE OPTIONS

The `-f` option makes **mdig** operate in batch mode by reading a list of lookup requests to process from the file *filename*. The file contains a number of queries, one per line. Each entry in the file should be organized in the same way they would be presented as queries to **mdig** using the command-line interface.

The `-h` causes **mdig** to print the detailed help with the full list of options and exit.

The `-v` causes **mdig** to print the version number and exit.

GLOBAL OPTIONS

The `-4` option forces **mdig** to only use IPv4 query transport.

The `-6` option forces **mdig** to only use IPv6 query transport.

The `-b` option sets the source IP address of the query to *`address`*. This must be a valid address on one of the host's network interfaces or "0.0.0.0" or "::". An optional port may be specified by appending "#<port>"

The `-m` option enables memory usage debugging.

The `-p` option is used when a non-standard port number is to be queried. *`port#`* is the port number that **mdig** will send its queries instead of the standard DNS port number 53. This option would be used to test a name server that has been configured to listen for queries on a non-standard port number.

The global query options are:

+[no]additional
Display [do not display] the additional section of a reply. The default is to display it.

+[no]all
Set or clear all display flags.

+[no]answer
Display [do not display] the answer section of a reply. The default is to display it.

+[no]authority
Display [do not display] the authority section of a reply. The default is to display it.

+[no]besteffort
Attempt to display the contents of messages which are malformed. The default is to not display malformed answers.

+burst
This option delays queries until the start of the next second.

+[no]cl
Display [do not display] the CLASS when printing the record.

+[no]comments
Toggle the display of comment lines in the output. The default is to print comments.

+[no]continue
Continue on errors (e.g. timeouts).

+[no]crypto
Toggle the display of cryptographic fields in DNSSEC records. The contents of these field are unnecessary to debug most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted they are replaced by the string "[omitted]" or in the DNSKEY case the key id is displayed as the replacement, e.g. "[key id = value]".

+dscp[=value]
Set the DSCP code point to be used when sending the query. Valid DSCP code points are in the range [0..63]. By default no code point is explicitly set.

+[no]multiline
Print records like the SOA records in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the **mdig** output.

+[no]question
Print [do not print] the question section of a query when an answer is returned. The default is to print the question section as a comment.

+[no]rrcomments
Toggle the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is not to print record comments unless multiline mode is active.

+[no]short
Provide a terse answer. The default is to print the answer in a verbose form.

+split=W
Split long hex- or base64-formatted fields in resource records into chunks of *W* characters (where *W* is rounded up to the nearest multiple of 4). *+nosplit* or *+split=0* causes fields not to be split at all. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]tcp
Use [do not use] TCP when querying name servers. The default behavior is to use UDP.

+[no]ttlid
Display [do not display] the TTL when printing the record.

+[no]ttlunits
Display [do not display] the TTL in friendly human-readable time units of "s", "m", "h", "d", and "w", representing seconds, minutes, hours, days and weeks. Implies +ttlid.

+[no]vc
Use [do not use] TCP when querying name servers. This alternate syntax to *+[no]tcp* is provided for backwards compatibility. The "vc" stands for "virtual circuit".

LOCAL OPTIONS

The `-c` option sets the query class to *class*. It can be any valid query class which is supported in BIND 9. The default query class is "IN".

The `-t` option sets the query type to *type*. It can be any valid query type which is supported in BIND 9. The default query type is "A", unless the `-x` option is supplied to indicate a reverse lookup with the "PTR" query type.

The `-i` option sets the reverse domain for IPv6 addresses to IP6.INT.

Reverse lookups --- mapping addresses to names --- are simplified by the `-x` option. *addr* is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. **mdig** automatically performs a lookup for a query name like `11.12.13.10.in-addr.arpa` and sets the query type and class to PTR and IN respectively. By default, IPv6 addresses are looked up using nibble format under the IP6.ARPA domain. To use the older RFC1886 method using the IP6.INT domain specify the `-i` option.

The local query options are:

+[no]aaflag
A synonym for *+[no]aaonly*.

+[no]aaonly
Sets the "aa" flag in the query.

+[no]adflag
Set [do not set] the AD (authentic data) bit in the query. This requests the server to return whether all of the answer and authority sections have all been validated as secure according to the security policy of the server. AD=1 indicates that all records have been validated as secure and the answer is not from a OPT-OUT range. AD=0 indicate that some part of the answer was insecure or not validated. This bit is set by default.

+bufsize=B
Set the UDP message buffer size advertised using EDNS0 to `B` bytes. The maximum and minimum sizes of this buffer are 65535 and 0 respectively. Values outside this range are rounded up or down appropriately. Values other than zero will cause a EDNS query to be sent.

+[no]cdflag
Set [do not set] the CD (checking disabled) bit in the query. This requests the server to not perform DNSSEC validation of responses.

+[no]cookie[=####]
Send a COOKIE EDNS option, with optional value. Replaying a COOKIE from a previous response will allow the server to identify a previous client. The default is `+nocookie`.

+[no]dnssec
Requests DNSSEC records be sent by setting the DNSSEC OK bit (DO) in the OPT record in the additional section of the query.

+[no]edns[=#]
Specify the EDNS version to query with. Valid values are 0 to 255. Setting the EDNS version will cause a EDNS query to be sent. `+noedns` clears the remembered EDNS version. EDNS is set to 0 by default.

+[no]ednsflags[=#]
Set the must-be-zero EDNS flags bits (Z bits) to the specified value. Decimal, hex and octal encodings are accepted. Setting a named flag (e.g. DO) will silently be ignored. By default, no Z bits are set.

+[no]ednsopt[=code[:value]]
Specify EDNS option with code point `code` and optionally payload of `value` as a hexadecimal string. `+noednsopt` clears the EDNS options to be sent.

+[no]expire
Send an EDNS Expire option.

+[no]nsid
Include an EDNS name server ID request when sending a query.

+[no]recurse
Toggle the setting of the RD (recursion desired) bit in the query. This bit is set by default, which means **mdig** normally sends recursive queries.

+retry=T
Sets the number of times to retry UDP queries to server to *T* instead of the default, 2. Unlike *+tries*, this does not include the initial query.

+[no]subnet=addr[/prefix-length]
Send (don't send) an EDNS Client Subnet option with the specified IP address or network prefix.

mdig +subnet=0.0.0.0/0, or simply **mdig +subnet=0** for short, sends an EDNS client-subnet option with an empty address and a source prefix-length of zero, which signals a resolver that the client's address information must *not* be used when resolving this query.

+timeout=T
Sets the timeout for a query to *T* seconds. The default timeout is 5 seconds for UDP transport and 10 for TCP. An attempt to set *T* to less than 1 will result in a query timeout of 1 second being applied.

+tries=T
Sets the number of times to try UDP queries to server to *T* instead of the default, 3. If *T* is less than or equal to zero, the number of tries is silently rounded up to 1.

+udptimeout=T
Sets the timeout between UDP query retries.

+[no]unknownformat
Print all RDATA in unknown RR type presentation format (RFC 3597). The default is to print RDATA for known types in the type's presentation format.

+[no]zflag
Set [do not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

SEE ALSO

dig(1), *RFC1035*.

9.22 NAMED-CHECKCONF

named-checkconf — named configuration file syntax checking tool

Synopsis

`named-checkconf` [`-hjvz`] [`-p` [`-x`]] [`-t` *directory*] filename

DESCRIPTION

named-checkconf checks the syntax, but not the semantics, of a **named** configuration file. The file is parsed and checked for syntax errors, along with all files included by it. If no file is specified, `/etc/named.conf` is read by default.

Note: files that **named** reads in separate parser contexts, such as `rndc.key` and `bind.keys`, are not automatically read by **named-checkconf**. Configuration errors in these files may cause **named** to fail to run, even if **named-checkconf** was successful. **named-checkconf** can be run on these files explicitly, however.

OPTIONS

-h
Print the usage summary and exit.

-j
When loading a zonefile read the journal if it exists.

-p
Print out the `named.conf` and included files in canonical form if no errors were detected. See also the `-x` option.

-t *directory*
Chroot to `directory` so that include directives in the configuration file are processed as if run by a similarly chrooted **named**.

-v
Print the version of the **named-checkconf** program and exit.

-x
When printing the configuration files in canonical form, obscure shared secrets by replacing them with strings of question marks ('?'). This allows the contents of `named.conf` and related files to be shared --- for example, when submitting bug reports --- without compromising private data. This option cannot be used without `-p`.

-z
Perform a test load of all master zones found in `named.conf`.

filename
The name of the configuration file to be checked. If not specified, it defaults to `/etc/named.conf`.

RETURN VALUES

named-checkconf returns an exit status of 1 if errors were detected and 0 otherwise.

SEE ALSO

named(8), named-checkzone(8), *BIND 9 Administrator Reference Manual*.

9.23 NAMED-CHECKZONE

named-checkzone, named-compilezone — zone file validity checking or converting tool

Synopsis

`named-checkzone` [`-d`] [`-h`] [`-j`] [`-q`] [`-v`] [`-c` *class*] [`-f` *format*] [`-F` *format*] [`-J` *filename*] [`-i` *mode*] [`-k` *mode*] [`-m` *mode*] [`-M` *mode*] [`-n` *mode*] [`-l` *ttl*] [`-L` *serial*] [`-o` *filename*] [`-r` *mode*] [`-s` *style*] [`-S` *mode*] [`-t` *directory*] [`-T` *mode*] [`-w` *directory*] [`-D`] [`-W` *mode*] zonename filename

`named-compilezone` [`-d`] [`-j`] [`-q`] [`-v`] [`-c` *class*] [`-C` *mode*] [`-f` *format*] [`-F` *format*] [`-J` *filename*] [`-i` *mode*] [`-k` *mode*] [`-m` *mode*] [`-n` *mode*] [`-l` *ttl*] [`-L` *serial*] [`-r` *mode*] [`-s` *style*] [`-t` *directory*] [`-T` *mode*] [`-w` *directory*] [`-D`] [`-W` *mode*] `-o` *filename* zonename filename

DESCRIPTION

named-checkzone checks the syntax and integrity of a zone file. It performs the same checks as **named** does when loading a zone. This makes **named-checkzone** useful for checking zone files before configuring them into a name server.

named-compilezone is similar to **named-checkzone**, but it always dumps the zone contents to a specified file in a specified format. Additionally, it applies stricter check levels by default, since the dump output will be used as an actual zone file loaded by **named**. When manually specified otherwise, the check levels must at least be as strict as those specified in the **named** configuration file.

OPTIONS

-d
Enable debugging.

-h
Print the usage summary and exit.

-q
Quiet mode - exit code only.

-v
Print the version of the **named-checkzone** program and exit.

-j
When loading a zone file, read the journal if it exists. The journal file name is assumed to be the zone file name appended with the string `.jnl`.

-J *filename*
When loading the zone file read the journal from the given file, if it exists. (Implies -j.)

-c ***class***
Specify the class of the zone. If not specified, "IN" is assumed.

-i ***mode***
Perform post-load zone integrity checks. Possible modes are **"full"** (default), **"full-sibling"**, **"local"**, **"local-sibling"** and **"none"**.

Mode **"full"** checks that MX records refer to A or AAAA record (both in-zone and out-of-zone hostnames). Mode **"local"** only checks MX records which refer to in-zone hostnames.

Mode **"full"** checks that SRV records refer to A or AAAA record (both in-zone and out-of-zone hostnames). Mode **"local"** only checks SRV records which refer to in-zone hostnames.

Mode **"full"** checks that delegation NS records refer to A or AAAA record (both in-zone and out-of-zone hostnames). It also checks that glue address records in the zone match those advertised by the child. Mode **"local"** only checks NS records which refer to in-zone hostnames or that some required glue exists, that is when the nameserver is in a child zone.

Mode **"full-sibling"** and **"local-sibling"** disable sibling glue checks but are otherwise the same as **"full"** and **"local"** respectively.

Mode **"none"** disables the checks.

-f ***format***
Specify the format of the zone file. Possible formats are **"text"** (default), **"raw"**, and **"map"**.

-F ***format***
Specify the format of the output file specified. For **named-checkzone**, this does not cause any effects unless it dumps the zone contents.

Possible formats are **"text"** (default), which is the standard textual representation of the zone, and **"map"**, **"raw"**, and **"raw=N"**, which store the zone in a binary format for rapid loading by **named**. **"raw=N"** specifies the format version of the raw zone file: if N is 0, the raw file can be read by any version of **named**; if N is 1, the file can be read by release 9.9.0 or higher; the default is 1.

-k ***mode***
Perform **"check-names"** checks with the specified failure mode. Possible modes are **"fail"** (default for **named-compilezone**), **"warn"** (default for **named-checkzone**) and **"ignore"**.

-l ***ttl***
Sets a maximum permissible TTL for the input file. Any record with a TTL higher than this value will cause the zone to be rejected. This is similar to using the **max-zone-ttl** option in `named.conf`.

-L ***serial***
When compiling a zone to "raw" or "map" format, set the "source serial" value in the header to the specified serial number. (This is expected to be used primarily for testing purposes.)

-m ***mode***
Specify whether MX records should be checked to see if they are addresses. Possible modes are **"fail"**, **"warn"** (default) and **"ignore"**.

-M ***mode***
Check if a MX record refers to a CNAME. Possible modes are **"fail"**, **"warn"** (default) and **"ignore"**.

-n ***mode***
Specify whether NS records should be checked to see if they are addresses. Possible modes are **"fail"** (default for **named-compilezone**), **"warn"** (default for **named-checkzone**) and **"ignore"**.

-o ***filename***
Write zone output to `filename`. If `filename` is `-` then write to standard out. This is mandatory for **named-compilezone**.

-r ***mode***
Check for records that are treated as different by DNSSEC but are semantically equal in plain DNS. Possible modes are **"fail"**, **"warn"** (default) and **"ignore"**.

-s ***style***
Specify the style of the dumped zone file. Possible styles are **"full"** (default) and **"relative"**. The full format is most suitable for processing automatically by a separate script. On the other hand, the relative format is more human-readable and is thus suitable for editing by hand. For **named-checkzone** this does not cause any effects unless it dumps the zone contents. It also does not have any meaning if the output format is not text.

-S ***mode***
Check if a SRV record refers to a CNAME. Possible modes are **"fail"**, **"warn"** (default) and **"ignore"**.

-t ***directory***
Chroot to `directory` so that include directives in the configuration file are processed as if run by a similarly chrooted **named**.

-T ***mode***
Check if Sender Policy Framework (SPF) records exist and issues a warning if an SPF-formatted TXT record is not also present. Possible modes are **"warn"** (default), **"ignore"**.

-w ***directory***
chdir to `directory` so that relative filenames in master file $INCLUDE directives work. This is similar to the directory clause in `named.conf`.

-D
Dump zone file in canonical format. This is always enabled for **named-compilezone**.

-W ***mode***
Specify whether to check for non-terminal wildcards. Non-terminal wildcards are almost always the result of a failure to understand the wildcard matching algorithm (RFC 1034). Possible modes are **"warn"** (default) and **"ignore"**.

zonename
The domain name of the zone being checked.

filename
The name of the zone file.

RETURN VALUES

named-checkzone returns an exit status of 1 if errors were detected and 0 otherwise.

SEE ALSO

named(8), named-checkconf(8), *RFC 1035*, *BIND 9 Administrator Reference Manual*.

9.24 NAMED-JOURNALPRINT

named-journalprint — print zone journal in human-readable form

Synopsis

`named-journalprint` *`journal`*

DESCRIPTION

named-journalprint prints the contents of a zone journal file in a human-readable form.

Journal files are automatically created by **named** when changes are made to dynamic zones (e.g., by **nsupdate**). They record each addition or deletion of a resource record, in binary format, allowing the changes to be re-applied to the zone when the server is restarted after a shutdown or crash. By default, the name of the journal file is formed by appending the extension `.jnl` to the name of the corresponding zone file.

named-journalprint converts the contents of a given journal file into a human-readable text format. Each line begins with "add" or "del", to indicate whether the record was added or deleted, and continues with the resource record in master-file format.

SEE ALSO

named(8), nsupdate(1), *BIND 9 Administrator Reference Manual*.

9.25 NAMED-NZD2NZF

named-nzd2nzf — Convert an NZD database to NZF text format

Synopsis

`named-nzd2nzf` filename

DESCRIPTION

named-nzd2nzf converts an NZD database to NZF format and prints it to standard output. This can be used to review the configuration of zones that were added to **named** via **rndc addzone**. It can also be used to restore the old file format when rolling back from a newer version of BIND to an older version.

ARGUMENTS

filename
The name of the `.nzd` file whose contents should be printed.

SEE ALSO

BIND 9 Administrator Reference Manual

AUTHOR

Internet Systems Consortium

9.26 NAMED-RRCHECKER

named-rrchecker — syntax checker for individual DNS resource records

Synopsis

`named-rrchecker` [`-h`] [`-o` *origin*] [`-p`] [`-u`] [`-C`] [`-T`] [`-P`]

DESCRIPTION

named-rrchecker read a individual DNS resource record from standard input and checks if it is syntactically correct.

The `-h` prints out the help menu.

The `-o` *origin* option specifies a origin to be used when interpreting the record.

The `-p` prints out the resulting record in canonical form. If there is no canonical form defined then the record will be printed in unknown record format.

The `-u` prints out the resulting record in unknown record form.

The `-C`, `-T` and `-P` print out the known class, standard type and private type mnemonics respectively.

SEE ALSO

RFC 1034, *RFC 1035*, named(8)

9.27 NAMED.CONF

`named.conf` — configuration file for **named**

Synopsis

```
named.conf
```

DESCRIPTION

`named.conf` is the configuration file for **named**. Statements are enclosed in braces and terminated with a semi-colon. Clauses in the statements are also semi-colon terminated. The usual comment styles are supported:

C style: /* */

C++ style: // to end of line

Unix style: # to end of line

ACL

```
acl string { address_match_element; ... };
```

CONTROLS

```
controls {
  inet ( ipv4_address | ipv6_address |
      * ) [ port ( integer | * ) ] allow
      { address_match_element; ... } [
      keys { string; ... } ] [ read-only
      boolean ];
  unix quoted_string perm integer
      owner integer group integer [
      keys { string; ... } ] [ read-only
      boolean ];
};
```

DLZ

```
dlz string {
  database string;
  search boolean;
};
```

DYNDB

```
dyndb string quoted_string {
    unspecified-text };
```

KEY

```
key string {
  algorithm string;
  secret string;
};
```

LOGGING

```
logging {
  category string { string; ... };
  channel string {
    buffered boolean;
    file quoted_string [ versions ( "unlimited" | integer )
        ] [ size size ];
    null;
    print-category boolean;
    print-severity boolean;
    print-time boolean;
    severity log_severity;
    stderr;
    syslog [ syslog_facility ];
  };
};
```

LWRES

```
lwres {
  listen-on [ port integer ] [ dscp integer ] { ( ipv4_address
      | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
```

```
  lwres-clients integer;
  lwres-tasks integer;
  ndots integer;
  search { string; ... };
  view string [ class ];
};
```

MANAGED-KEYS

```
managed-keys { string string integer
    integer integer quoted_string; ... };
```

MASTERS

```
masters string [ port integer ] [ dscp
    integer ] { ( masters | ipv4_address [
    port integer ] | ipv6_address [ port
    integer ] ) [ key string ]; ... };
```

OPTIONS

```
options {
  acache-cleaning-interval integer;
  acache-enable boolean;
  additional-from-auth boolean;
  additional-from-cache boolean;
  allow-new-zones boolean;
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
  allow-query-cache { address_match_element; ... };
  allow-query-cache-on { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  allow-recursion { address_match_element; ... };
  allow-recursion-on { address_match_element; ... };
  allow-transfer { address_match_element; ... };
  allow-update { address_match_element; ... };
  allow-update-forwarding { address_match_element; ... };
  also-notify [ port integer ] [ dscp integer ] { ( masters |
      ipv4_address [ port integer ] | ipv6_address [ port
      integer ] ) [ key string ]; ... };
  alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
      * ) ] [ dscp integer ];
```

```
answer-cookie boolean;
attach-cache string;
auth-nxdomain boolean; // default changed
auto-dnssec ( allow | maintain | off );
automatic-interface-scan boolean;
avoid-v4-udp-ports { portrange; ... };
avoid-v6-udp-ports { portrange; ... };
bindkeys-file quoted_string;
blackhole { address_match_element; ... };
cache-file quoted_string;
catalog-zones { zone string [ default-masters [ port integer ]
    [ dscp integer ] { ( masters | ipv4_address [ port
    integer ] | ipv6_address [ port integer ] ) [ key
    string ]; ... } ] [ zone-directory quoted_string ] [
    in-memory boolean ] [ min-update-interval integer ]; ... };
check-dup-records ( fail | warn | ignore );
check-integrity boolean;
check-mx ( fail | warn | ignore );
check-mx-cname ( fail | warn | ignore );
check-names ( master | slave | response
    ) ( fail | warn | ignore );
check-sibling boolean;
check-spf ( warn | ignore );
check-srv-cname ( fail | warn | ignore );
check-wildcard boolean;
cleaning-interval integer;
clients-per-query integer;
cookie-algorithm ( aes | sha1 | sha256 | siphash24 );
cookie-secret string;
coresize ( default | unlimited | sizeval );
datasize ( default | unlimited | sizeval );
deny-answer-addresses { address_match_element; ... } [
    except-from { quoted_string; ... } ];
deny-answer-aliases { quoted_string; ... } [ except-from {
    quoted_string; ... } ];
dialup ( notify | notify-passive | passive | refresh | boolean ) ↩
   ;
directory quoted_string;
disable-algorithms string { string;
    ... };
disable-ds-digests string { string;
    ... };
disable-empty-zone string;
dns64 netprefix {
  break-dnssec boolean;
  clients { address_match_element; ... };
  exclude { address_match_element; ... };
  mapped { address_match_element; ... };
  recursive-only boolean;
```

```
  suffix ipv6_address;
};
dns64-contact string;
dns64-server string;
dnssec-accept-expired boolean;
dnssec-dnskey-kskonly boolean;
dnssec-enable boolean;
dnssec-loadkeys-interval integer;
dnssec-lookaside ( string trust-anchor
    string | auto | no );
dnssec-must-be-secure string boolean;
dnssec-secure-to-insecure boolean;
dnssec-update-mode ( maintain | no-resign );
dnssec-validation ( yes | no | auto );
dnstap { ( all | auth | client | forwarder |
    resolver ) [ ( query | response ) ]; ... };
dnstap-identity ( quoted_string | none |
    hostname );
dnstap-output ( file | unix ) quoted_string;
dnstap-version ( quoted_string | none );
dscp integer;
dual-stack-servers [ port integer ] { ( quoted_string [ port
    integer ] [ dscp integer ] | ipv4_address [ port
    integer ] [ dscp integer ] | ipv6_address [ port
    integer ] [ dscp integer ] ); ... };
dump-file quoted_string;
edns-udp-size integer;
empty-contact string;
empty-server string;
empty-zones-enable boolean;
fetch-quota-params integer fixedpoint fixedpoint fixedpoint;
fetches-per-server integer [ ( drop | fail ) ];
fetches-per-zone integer [ ( drop | fail ) ];
files ( default | unlimited | sizeval );
filter-aaaa { address_match_element; ... };
filter-aaaa-on-v4 ( break-dnssec | boolean );
filter-aaaa-on-v6 ( break-dnssec | boolean );
flush-zones-on-shutdown boolean;
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
    | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
fstrm-set-buffer-hint integer;
fstrm-set-flush-timeout integer;
fstrm-set-input-queue-size integer;
fstrm-set-output-notify-threshold integer;
fstrm-set-output-queue-model ( mpsc | spsc );
fstrm-set-output-queue-size integer;
fstrm-set-reopen-interval integer;
geoip-directory ( quoted_string | none );
```

```
geoip-use-ecs boolean;
heartbeat-interval integer;
hostname ( quoted_string | none );
inline-signing boolean;
interface-interval integer;
ixfr-from-differences ( master | slave | boolean );
keep-response-order { address_match_element; ... };
key-directory quoted_string;
lame-ttl ttlval;
listen-on [ port integer ] [ dscp
    integer ] {
    address_match_element; ... };
listen-on-v6 [ port integer ] [ dscp
    integer ] {
    address_match_element; ... };
lmdb-mapsize sizeval;
lock-file ( quoted_string | none );
managed-keys-directory quoted_string;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
match-mapped-addresses boolean;
max-acache-size ( unlimited | sizeval );
max-cache-size ( default | unlimited | sizeval | percentage );
max-cache-ttl integer;
max-clients-per-query integer;
max-journal-size ( unlimited | sizeval );
max-ncache-ttl integer;
max-records integer;
max-recursion-depth integer;
max-recursion-queries integer;
max-refresh-time integer;
max-retry-time integer;
max-rsa-exponent-size integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
max-udp-size integer;
max-zone-ttl ( unlimited | ttlval );
memstatistics boolean;
memstatistics-file quoted_string;
message-compression boolean;
min-refresh-time integer;
min-retry-time integer;
minimal-any boolean;
minimal-responses ( no-auth | no-auth-recursive | boolean );
multi-master boolean;
no-case-compress { address_match_element; ... };
nocookie-udp-size integer;
```

```
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-rate integer;
notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
    [ dscp integer ];
notify-to-soa boolean;
nta-lifetime ttlval;
nta-recheck ttlval;
nxdomain-redirect string;
pid-file ( quoted_string | none );
port integer;
preferred-glue string;
prefetch integer [ integer ];
provide-ixfr boolean;
query-source ( ( [ address ] ( ipv4_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
querylog boolean;
random-device quoted_string;
rate-limit {
  all-per-second integer;
  errors-per-second integer;
  exempt-clients { address_match_element; ... };
  ipv4-prefix-length integer;
  ipv6-prefix-length integer;
  log-only boolean;
  max-table-size integer;
  min-table-size integer;
  nodata-per-second integer;
  nxdomains-per-second integer;
  qps-scale integer;
  referrals-per-second integer;
  responses-per-second integer;
  slip integer;
  window integer;
};
recursing-file quoted_string;
recursion boolean;
recursive-clients integer;
request-expire boolean;
request-ixfr boolean;
request-nsid boolean;
require-server-cookie boolean;
reserved-sockets integer;
```

```
resolver-query-timeout integer;
response-policy { zone string [ log boolean ] [ max-policy-ttl
    integer ] [ policy ( cname | disabled | drop | given | no-op
    | nodata | nxdomain | passthru | tcp-only quoted_string ) ]  ↩
       [
    recursive-only boolean ]; ... } [ break-dnssec boolean ] [
    max-policy-ttl integer ] [ min-ns-dots integer ] [
    nsip-wait-recurse boolean ] [ qname-wait-recurse boolean ]
    [ recursive-only boolean ];
root-delegation-only [ exclude { quoted_string; ... } ];
root-key-sentinel boolean;
rrset-order { [ class string ] [ type string ] [ name
    quoted_string ] string string; ... };
secroots-file quoted_string;
send-cookie boolean;
serial-query-rate integer;
serial-update-method ( date | increment | unixtime );
server-id ( quoted_string | none | hostname );
servfail-ttl ttlval;
session-keyalg string;
session-keyfile ( quoted_string | none );
session-keyname string;
sig-signing-nodes integer;
sig-signing-signatures integer;
sig-signing-type integer;
sig-validity-interval integer [ integer ];
sortlist { address_match_element; ... };
stacksize ( default | unlimited | sizeval );
startup-notify-rate integer;
statistics-file quoted_string;
tcp-clients integer;
tcp-listen-queue integer;
tkey-dhkey quoted_string integer;
tkey-domain quoted_string;
tkey-gssapi-credential quoted_string;
tkey-gssapi-keytab quoted_string;
transfer-format ( many-answers | one-answer );
transfer-message-size integer;
transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
    ] [ dscp integer ];
transfers-in integer;
transfers-out integer;
transfers-per-ns integer;
trust-anchor-telemetry boolean; // experimental
try-tcp-refresh boolean;
update-check-ksk boolean;
use-alt-transfer-source boolean;
```

```
  use-v4-udp-ports { portrange; ... };
  use-v6-udp-ports { portrange; ... };
  v6-bias integer;
  version ( quoted_string | none );
  zero-no-soa-ttl boolean;
  zero-no-soa-ttl-cache boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

SERVER

```
server netprefix {
  bogus boolean;
  edns boolean;
  edns-udp-size integer;
  edns-version integer;
  keys server_key;
  max-udp-size integer;
  notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
      [ dscp integer ];
  provide-ixfr boolean;
  query-source ( ( [ address ] ( ipv4_address | * ) [ port (
      integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
      port ( integer | * ) ) ) [ dscp integer ];
  query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
      integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
      port ( integer | * ) ) ) [ dscp integer ];
  request-expire boolean;
  request-ixfr boolean;
  request-nsid boolean;
  send-cookie boolean;
  tcp-only boolean;
  transfer-format ( many-answers | one-answer );
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  transfers integer;
};
```

STATISTICS-CHANNELS

```
statistics-channels {
  inet ( ipv4_address | ipv6_address |
```

```
      * ) [ port ( integer | * ) ] [
      allow { address_match_element; ...
      } ];
};
```

TRUSTED-KEYS

```
trusted-keys { string integer integer
    integer quoted_string; ... };
```

VIEW

```
view string [ class ] {
  acache-cleaning-interval integer;
  acache-enable boolean;
  additional-from-auth boolean;
  additional-from-cache boolean;
  allow-new-zones boolean;
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
  allow-query-cache { address_match_element; ... };
  allow-query-cache-on { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  allow-recursion { address_match_element; ... };
  allow-recursion-on { address_match_element; ... };
  allow-transfer { address_match_element; ... };
  allow-update { address_match_element; ... };
  allow-update-forwarding { address_match_element; ... };
  also-notify [ port integer ] [ dscp integer ] { ( masters |
      ipv4_address [ port integer ] | ipv6_address [ port
      integer ] ) [ key string ]; ... };
  alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
      * ) ] [ dscp integer ];
  attach-cache string;
  auth-nxdomain boolean; // default changed
  auto-dnssec ( allow | maintain | off );
  cache-file quoted_string;
  catalog-zones { zone string [ default-masters [ port integer ]
      [ dscp integer ] { ( masters | ipv4_address [ port
      integer ] | ipv6_address [ port integer ] ) [ key
      string ]; ... } ] [ zone-directory quoted_string ] [
      in-memory boolean ] [ min-update-interval integer ]; ... };
  check-dup-records ( fail | warn | ignore );
  check-integrity boolean;
```

```
check-mx ( fail | warn | ignore );
check-mx-cname ( fail | warn | ignore );
check-names ( master | slave | response
    ) ( fail | warn | ignore );
check-sibling boolean;
check-spf ( warn | ignore );
check-srv-cname ( fail | warn | ignore );
check-wildcard boolean;
cleaning-interval integer;
clients-per-query integer;
deny-answer-addresses { address_match_element; ... } [
    except-from { quoted_string; ... } ];
deny-answer-aliases { quoted_string; ... } [ except-from {
    quoted_string; ... } ];
dialup ( notify | notify-passive | passive | refresh | boolean ) ↩
   ;
disable-algorithms string { string;
    ... };
disable-ds-digests string { string;
    ... };
disable-empty-zone string;
dlz string {
  database string;
  search boolean;
};
dns64 netprefix {
  break-dnssec boolean;
  clients { address_match_element; ... };
  exclude { address_match_element; ... };
  mapped { address_match_element; ... };
  recursive-only boolean;
  suffix ipv6_address;
};
dns64-contact string;
dns64-server string;
dnssec-accept-expired boolean;
dnssec-dnskey-kskonly boolean;
dnssec-enable boolean;
dnssec-loadkeys-interval integer;
dnssec-lookaside ( string trust-anchor
    string | auto | no );
dnssec-must-be-secure string boolean;
dnssec-secure-to-insecure boolean;
dnssec-update-mode ( maintain | no-resign );
dnssec-validation ( yes | no | auto );
dnstap { ( all | auth | client | forwarder |
    resolver ) [ ( query | response ) ]; ... };
dual-stack-servers [ port integer ] { ( quoted_string [ port
    integer ] [ dscp integer ] | ipv4_address [ port
```

```
    integer ] [ dscp integer ] | ipv6_address [ port
    integer ] [ dscp integer ] ); ... };
dyndb string quoted_string {
    unspecified-text };
edns-udp-size integer;
empty-contact string;
empty-server string;
empty-zones-enable boolean;
fetch-quota-params integer fixedpoint fixedpoint fixedpoint;
fetches-per-server integer [ ( drop | fail ) ];
fetches-per-zone integer [ ( drop | fail ) ];
filter-aaaa { address_match_element; ... };
filter-aaaa-on-v4 ( break-dnssec | boolean );
filter-aaaa-on-v6 ( break-dnssec | boolean );
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
    | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
inline-signing boolean;
ixfr-from-differences ( master | slave | boolean );
key string {
  algorithm string;
  secret string;
};
key-directory quoted_string;
lame-ttl ttlval;
lmdb-mapsize sizeval;
managed-keys { string string
    integer integer integer
    quoted_string; ... };
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
match-clients { address_match_element; ... };
match-destinations { address_match_element; ... };
match-recursive-only boolean;
max-acache-size ( unlimited | sizeval );
max-cache-size ( default | unlimited | sizeval | percentage );
max-cache-ttl integer;
max-clients-per-query integer;
max-journal-size ( unlimited | sizeval );
max-ncache-ttl integer;
max-records integer;
max-recursion-depth integer;
max-recursion-queries integer;
max-refresh-time integer;
max-retry-time integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
```

```
max-udp-size integer;
max-zone-ttl ( unlimited | ttlval );
message-compression boolean;
min-refresh-time integer;
min-retry-time integer;
minimal-any boolean;
minimal-responses ( no-auth | no-auth-recursive | boolean );
multi-master boolean;
no-case-compress { address_match_element; ... };
nocookie-udp-size integer;
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
    [ dscp integer ];
notify-to-soa boolean;
nta-lifetime ttlval;
nta-recheck ttlval;
nxdomain-redirect string;
preferred-glue string;
prefetch integer [ integer ];
provide-ixfr boolean;
query-source ( ( [ address ] ( ipv4_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
    integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
    port ( integer | * ) ) ) [ dscp integer ];
rate-limit {
  all-per-second integer;
  errors-per-second integer;
  exempt-clients { address_match_element; ... };
  ipv4-prefix-length integer;
  ipv6-prefix-length integer;
  log-only boolean;
  max-table-size integer;
  min-table-size integer;
  nodata-per-second integer;
  nxdomains-per-second integer;
  qps-scale integer;
  referrals-per-second integer;
  responses-per-second integer;
  slip integer;
  window integer;
};
recursion boolean;
request-expire boolean;
request-ixfr boolean;
```

```
request-nsid boolean;
require-server-cookie boolean;
resolver-query-timeout integer;
response-policy { zone string [ log boolean ] [ max-policy-ttl
    integer ] [ policy ( cname | disabled | drop | given | no-op
    | nodata | nxdomain | passthru | tcp-only quoted_string ) ]  ↩
       [
    recursive-only boolean ]; ... } [ break-dnssec boolean ] [
    max-policy-ttl integer ] [ min-ns-dots integer ] [
    nsip-wait-recurse boolean ] [ qname-wait-recurse boolean ]
    [ recursive-only boolean ];
root-delegation-only [ exclude { quoted_string; ... } ];
root-key-sentinel boolean;
rrset-order { [ class string ] [ type string ] [ name
    quoted_string ] string string; ... };
send-cookie boolean;
serial-update-method ( date | increment | unixtime );
server netprefix {
  bogus boolean;
  edns boolean;
  edns-udp-size integer;
  edns-version integer;
  keys server_key;
  max-udp-size integer;
  notify-source ( ipv4_address | * ) [ port ( integer | *
      ) ] [ dscp integer ];
  notify-source-v6 ( ipv6_address | * ) [ port ( integer
      | * ) ] [ dscp integer ];
  provide-ixfr boolean;
  query-source ( ( [ address ] ( ipv4_address | * ) [ port
      ( integer | * ) ] ) | ( [ [ address ] (
      ipv4_address | * ) ] port ( integer | * ) ) ) [
      dscp integer ];
  query-source-v6 ( ( [ address ] ( ipv6_address | * ) [
      port ( integer | * ) ] ) | ( [ [ address ] (
      ipv6_address | * ) ] port ( integer | * ) ) ) [
      dscp integer ];
  request-expire boolean;
  request-ixfr boolean;
  request-nsid boolean;
  send-cookie boolean;
  tcp-only boolean;
  transfer-format ( many-answers | one-answer );
  transfer-source ( ipv4_address | * ) [ port ( integer |
      * ) ] [ dscp integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port (
      integer | * ) ] [ dscp integer ];
  transfers integer;
};
```

```
servfail-ttl ttlval;
sig-signing-nodes integer;
sig-signing-signatures integer;
sig-signing-type integer;
sig-validity-interval integer [ integer ];
sortlist { address_match_element; ... };
transfer-format ( many-answers | one-answer );
transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
    ] [ dscp integer ];
trust-anchor-telemetry boolean; // experimental
trusted-keys { string integer
    integer integer quoted_string;
    ... };
try-tcp-refresh boolean;
update-check-ksk boolean;
use-alt-transfer-source boolean;
v6-bias integer;
zero-no-soa-ttl boolean;
zero-no-soa-ttl-cache boolean;
zone string [ class ] {
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  allow-transfer { address_match_element; ... };
  allow-update { address_match_element; ... };
  allow-update-forwarding { address_match_element; ... };
  also-notify [ port integer ] [ dscp integer ] { (
      masters | ipv4_address [ port integer ] |
      ipv6_address [ port integer ] ) [ key string ];
      ... };
  alt-transfer-source ( ipv4_address | * ) [ port (
      integer | * ) ] [ dscp integer ];
  alt-transfer-source-v6 ( ipv6_address | * ) [ port (
      integer | * ) ] [ dscp integer ];
  auto-dnssec ( allow | maintain | off );
  check-dup-records ( fail | warn | ignore );
  check-integrity boolean;
  check-mx ( fail | warn | ignore );
  check-mx-cname ( fail | warn | ignore );
  check-names ( fail | warn | ignore );
  check-sibling boolean;
  check-spf ( warn | ignore );
  check-srv-cname ( fail | warn | ignore );
  check-wildcard boolean;
  database string;
  delegation-only boolean;
  dialup ( notify | notify-passive | passive | refresh |
```

```
    boolean );
dlz string;
dnssec-dnskey-kskonly boolean;
dnssec-loadkeys-interval integer;
dnssec-secure-to-insecure boolean;
dnssec-update-mode ( maintain | no-resign );
file quoted_string;
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { (
    ipv4_address | ipv6_address ) [ port integer ] [
    dscp integer ]; ... };
in-view string;
inline-signing boolean;
ixfr-from-differences boolean;
journal quoted_string;
key-directory quoted_string;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
masters [ port integer ] [ dscp integer ] { ( masters
    | ipv4_address [ port integer ] | ipv6_address [
    port integer ] ) [ key string ]; ... };
max-ixfr-log-size ( default | unlimited |
max-journal-size ( unlimited | sizeval );
max-records integer;
max-refresh-time integer;
max-retry-time integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
max-zone-ttl ( unlimited | ttlval );
min-refresh-time integer;
min-retry-time integer;
multi-master boolean;
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-source ( ipv4_address | * ) [ port ( integer | *
    ) ] [ dscp integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer
    | * ) ] [ dscp integer ];
notify-to-soa boolean;
pubkey integer
    integer
    integer
request-expire boolean;
request-ixfr boolean;
serial-update-method ( date | increment | unixtime );
server-addresses { ( ipv4_address | ipv6_address ); ... };
server-names { quoted_string; ... };
```

```
    sig-signing-nodes integer;
    sig-signing-signatures integer;
    sig-signing-type integer;
    sig-validity-interval integer [ integer ];
    transfer-source ( ipv4_address | * ) [ port ( integer |
        * ) ] [ dscp integer ];
    transfer-source-v6 ( ipv6_address | * ) [ port (
        integer | * ) ] [ dscp integer ];
    try-tcp-refresh boolean;
    type ( delegation-only | forward | hint | master | redirect
        | slave | static-stub | stub );
    update-check-ksk boolean;
    update-policy ( local | { ( deny | grant ) string (
        6to4-self | external | krb5-self | krb5-selfsub |
        krb5-subdomain | ms-self | ms-selfsub | ms-subdomain |
        name | self | selfsub | selfwild | subdomain | tcp-self
        | wildcard | zonesub ) [ string ] rrtypelist; ... };
    use-alt-transfer-source boolean;
    zero-no-soa-ttl boolean;
    zone-statistics ( full | terse | none | boolean );
  };
  zone-statistics ( full | terse | none | boolean );
};
```

ZONE

```
zone string [ class ] {
  allow-notify { address_match_element; ... };
  allow-query { address_match_element; ... };
  allow-query-on { address_match_element; ... };
  allow-transfer { address_match_element; ... };
  allow-update { address_match_element; ... };
  allow-update-forwarding { address_match_element; ... };
  also-notify [ port integer ] [ dscp integer ] { ( masters |
      ipv4_address [ port integer ] | ipv6_address [ port
      integer ] ) [ key string ]; ... };
  alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
      * ) ] [ dscp integer ];
  auto-dnssec ( allow | maintain | off );
  check-dup-records ( fail | warn | ignore );
  check-integrity boolean;
  check-mx ( fail | warn | ignore );
  check-mx-cname ( fail | warn | ignore );
  check-names ( fail | warn | ignore );
  check-sibling boolean;
  check-spf ( warn | ignore );
```

```
check-srv-cname ( fail | warn | ignore );
check-wildcard boolean;
database string;
delegation-only boolean;
dialup ( notify | notify-passive | passive | refresh | boolean ) ↩
   ;
dlz string;
dnssec-dnskey-kskonly boolean;
dnssec-loadkeys-interval integer;
dnssec-secure-to-insecure boolean;
dnssec-update-mode ( maintain | no-resign );
file quoted_string;
forward ( first | only );
forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
    | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
in-view string;
inline-signing boolean;
ixfr-from-differences boolean;
journal quoted_string;
key-directory quoted_string;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
masters [ port integer ] [ dscp integer ] { ( masters |
    ipv4_address [ port integer ] | ipv6_address [ port
    integer ] ) [ key string ]; ... };
max-journal-size ( unlimited | sizeval );
max-records integer;
max-refresh-time integer;
max-retry-time integer;
max-transfer-idle-in integer;
max-transfer-idle-out integer;
max-transfer-time-in integer;
max-transfer-time-out integer;
max-zone-ttl ( unlimited | ttlval );
min-refresh-time integer;
min-retry-time integer;
multi-master boolean;
notify ( explicit | master-only | boolean );
notify-delay integer;
notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
    dscp integer ];
notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
    [ dscp integer ];
notify-to-soa boolean;
pubkey integer integer
request-expire boolean;
request-ixfr boolean;
serial-update-method ( date | increment | unixtime );
server-addresses { ( ipv4_address | ipv6_address ); ... };
```

```
  server-names { quoted_string; ... };
  sig-signing-nodes integer;
  sig-signing-signatures integer;
  sig-signing-type integer;
  sig-validity-interval integer [ integer ];
  transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
      dscp integer ];
  transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
      ] [ dscp integer ];
  try-tcp-refresh boolean;
  type ( delegation-only | forward | hint | master | redirect |  ↩
     slave
      | static-stub | stub );
  update-check-ksk boolean;
  update-policy ( local | { ( deny | grant ) string ( 6to4-self |
      external | krb5-self | krb5-selfsub | krb5-subdomain | ms- ↩
         self
      | ms-selfsub | ms-subdomain | name | self | selfsub |  ↩
         selfwild
      | subdomain | tcp-self | wildcard | zonesub ) [ string ]
      rrtypelist; ... };
  use-alt-transfer-source boolean;
  zero-no-soa-ttl boolean;
  zone-statistics ( full | terse | none | boolean );
};
```

FILES

`/etc/named.conf`

SEE ALSO

ddns-confgen(8), named(8), named-checkconf(8), rndc(8), rndc-confgen(8), *BIND 9 Administrator Reference Manual*.

9.28 NAMED

named — Internet domain name server

Synopsis

named [-4 | -6] [-c *config-file*] [-d *debug-level*] [-D *string*] [-E *engine-name*] [-f] [-g] [-L *logfile*] [-M *option*] [-m *flag*] [-n *#cpus*] [-p *port*] [-s] [-S *#max-socks*] [-t *directory*] [-U *#listeners*] [-u *user*] [-v] [-V] [-X *lock-file*] [-x *cache-file*]

DESCRIPTION

named is a Domain Name System (DNS) server, part of the BIND 9 distribution from ISC. For more information on the DNS, see RFCs 1033, 1034, and 1035.

When invoked without arguments, **named** will read the default configuration file `/etc/named.conf`, read any initial data, and listen for queries.

OPTIONS

-4
Use IPv4 only even if the host machine is capable of IPv6. `-4` and `-6` are mutually exclusive.

-6
Use IPv6 only even if the host machine is capable of IPv4. `-4` and `-6` are mutually exclusive.

-c *config-file*
Use *config-file* as the configuration file instead of the default, `/etc/named.conf`. To ensure that reloading the configuration file continues to work after the server has changed its working directory due to to a possible `directory` option in the configuration file, *config-file* should be an absolute pathname.

-d *debug-level*
Set the daemon's debug level to *debug-level*. Debugging traces from **named** become more verbose as the debug level increases.

-D *string*
Specifies a string that is used to identify a instance of **named** in a process listing. The contents of *string* are not examined.

-E *engine-name*
When applicable, specifies the hardware to use for cryptographic operations, such as a secure key store used for signing.

When BIND is built with OpenSSL PKCS#11 support, this defaults to the string "pkcs11", which identifies an OpenSSL engine that can drive a cryptographic accelerator or hardware service module. When BIND is built with native PKCS#11 cryptography (--enable-native-pkcs11), it defaults to the path of the PKCS#11 provider library specified via "--with-pkcs11".

-f
Run the server in the foreground (i.e. do not daemonize).

-g
Run the server in the foreground and force all logging to `stderr`.

-L *logfile*
Log to the file `logfile` by default instead of the system log.

-M ***option***
Sets the default memory context options. Currently the only supported option is *external*, which causes the internal memory manager to be bypassed in favor of system-provided memory allocation functions.

-m ***flag***
Turn on memory usage debugging flags. Possible flags are *usage*, *trace*, *record*, *size*, and *mctx*. These correspond to the ISC_MEM_DEBUGXXXX flags described in <isc/mem.h>.

-n ***#cpus***
Create *#cpus* worker 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 worker thread will be created.

-p ***port***
Listen for queries on port *port*. If not specified, the default is port 53.

-s
Write memory usage statistics to stdout on exit.

> NOTE
>
> This option is mainly of interest to BIND 9 developers and may be removed or changed in a future release.

-S ***#max-socks***
Allow **named** to use up to *#max-socks* sockets. The default value is 4096 on systems built with default configuration options, and 21000 on systems built with "configure --with-tuning=large".

> WARNING
>
> 
>
> This option should be unnecessary for the vast majority of users. The use of this option could even be harmful because the specified value may exceed the limitation of the underlying system API. It is therefore set only when the default configuration causes exhaustion of file descriptors and the operational environment is known to support the specified number of sockets. Note also that the actual maximum number is normally a little fewer than the specified value because **named** reserves some file descriptors for its internal use.

-t ***directory***

Chroot to *directory* after processing the command line arguments, but before reading the configuration file.

> WARNING
>
> 
>
> This option should be used in conjunction with the `-u` option, as chrooting a process running as root doesn't enhance security on most systems; the way `chroot(2)` is defined allows a process with root privileges to escape a chroot jail.

-U ***#listeners***

Use *#listeners* worker threads to listen for incoming UDP packets on each address. If not specified, **named** will calculate a default value based on the number of detected CPUs: 1 for 1 CPU, and the number of detected CPUs minus one for machines with more than 1 CPU. This cannot be increased to a value higher than the number of CPUs. If `-n` has been set to a higher value than the number of detected CPUs, then `-U` may be increased as high as that value, but no higher. On Windows, the number of UDP listeners is hardwired to 1 and this option has no effect.

-u ***user***

Setuid to *user* after completing privileged operations, such as creating sockets that listen on privileged ports.

> NOTE
>
> On Linux, **named** uses the kernel's capability mechanism to drop all root privileges except the ability to `bind(2)` to a privileged port and set process resource limits. Unfortunately, this means that the `-u` option only works when **named** is run on kernel 2.2.18 or later, or kernel 2.3.99-pre3 or later, since previous kernels did not allow privileges to be retained after `setuid(2)`.

-v

Report the version number and exit.

-V

Report the version number and build options, and exit.

-X ***lock-file***

Acquire a lock on the specified file at runtime; this helps to prevent duplicate **named** instances from running simultaneously. Use of this option overrides the **lock-file** option in `named.conf`. If set to `none`, the lock file check is disabled.

-x *cache-file*
Load data from *cache-file* into the cache of the default view.

> WARNING
>
> 
>
> This option must not be used. It is only of interest to BIND 9 developers and may be removed or changed in a future release.

SIGNALS

In routine operation, signals should not be used to control the nameserver; **rndc** should be used instead.

SIGHUP
Force a reload of the server.

SIGINT, SIGTERM
Shut down the server.

The result of sending any other signals to the server is undefined.

CONFIGURATION

The **named** configuration file is too complex to describe in detail here. A complete description is provided in the *BIND 9 Administrator Reference Manual*.

named inherits the `umask` (file creation mode mask) from the parent process. If files created by **named**, such as journal files, need to have custom permissions, the `umask` should be set explicitly in the script used to start the **named** process.

FILES

/etc/named.conf
The default configuration file.

/var/run/named/named.pid
The default process-id file.

SEE ALSO

RFC 1033, *RFC 1034*, *RFC 1035*, named-checkconf(8), named-checkzone(8), rndc(8), lwresd(8), named.conf(5), *BIND 9 Administrator Reference Manual*.

9.29 NSEC3HASH

nsec3hash — generate NSEC3 hash

Synopsis

`nsec3hash` *`salt algorithm iterations domain`*

DESCRIPTION

nsec3hash generates an NSEC3 hash based on a set of NSEC3 parameters. This can be used to check the validity of NSEC3 records in a signed zone.

ARGUMENTS

salt
The salt provided to the hash algorithm.

algorithm
A number indicating the hash algorithm. Currently the only supported hash algorithm for NSEC3 is SHA-1, which is indicated by the number 1; consequently "1" is the only useful value for this argument.

iterations
The number of additional times the hash should be performed.

domain
The domain name to be hashed.

SEE ALSO

BIND 9 Administrator Reference Manual, *RFC 5155*.

9.30 NSLOOKUP

nslookup — query Internet name servers interactively

Synopsis

`nslookup` [`-option`] [name | -] [server]

DESCRIPTION

Nslookup is a program to query Internet domain name servers. **Nslookup** has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

ARGUMENTS

Interactive mode is entered in the following cases:

a. when no arguments are given (the default name server will be used)

b. when the first argument is a hyphen (-) and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

Options can also be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, and the initial timeout to 10 seconds, type:

```
nslookup -query=hinfo  -timeout=10
```

The `-version` option causes **nslookup** to print the version number and immediately exits.

INTERACTIVE COMMANDS

host [server]
Look up information for host using the current default server or using server, if specified. If host is an Internet address and the query type is A or PTR, the name of the host is returned. If host is a name and does not have a trailing period, the search list is used to qualify the name.

To look up a host not in the current domain, append a period to the name.

server *domain*

lserver *domain*
Change the default server to *domain*; `lserver` uses the initial server to look up information about *domain*, while `server` uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

root
not implemented

finger
not implemented

ls
not implemented

view
not implemented

help
not implemented

?
not implemented

exit
Exits the program.

set ***keyword[=value]***
This command is used to change state information that affects the lookups. Valid keywords are:

all
Prints the current values of the frequently used options to **set**. Information about the current default server and host is also printed.

class=*value*
Change the query class to one of:

IN
the Internet class

CH
the Chaos class

HS
the Hesiod class

ANY
wildcard

The class specifies the protocol group of the information.
(Default = IN; abbreviation = cl)

***[no]*debug**
Turn on or off the display of the full response packet and any intermediate response packets when searching.
(Default = nodebug; abbreviation = [no]deb)

***[no]*d2**
Turn debugging mode on or off. This displays more about what nslookup is doing.
(Default = nod2)

domain=*name*
Sets the search list to *name*.

***[no]*search**
If the lookup request contains at least one period but doesn't end with a trailing period, append the domain names in the domain search list to the request until an answer is received.
(Default = search)

port=*value*
Change the default TCP/UDP name server port to *value*.
(Default = 53; abbreviation = po)

querytype=*value*

type=*value*
Change the type of the information query.
(Default = A and then AAAA; abbreviations = q, ty)
Note: It is only possible to specify one query type, only the default behavior looks up both when an alternative is not specified.

***[no]*recurse**
Tell the name server to query other servers if it does not have the information.
(Default = recurse; abbreviation = [no]rec)

ndots=*number*
Set the number of dots (label separators) in a domain that will disable searching. Absolute names always stop searching.

retry=*number*
Set the number of retries to number.

timeout=*number*
Change the initial timeout interval for waiting for a reply to number seconds.

***[no]*vc**
Always use a virtual circuit when sending requests to the server.
(Default = novc)

***[no]*fail**
Try the next nameserver if a nameserver responds with SERVFAIL or a referral (nofail) or terminate query (fail) on such a response.
(Default = nofail)

RETURN VALUES

nslookup returns with an exit status of 1 if any query failed, and 0 otherwise.

IDN SUPPORT

If **nslookup** has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. **nslookup** appropriately converts character encoding of domain name before sending a request to DNS server or displaying a reply from the server. If you'd like to turn off the IDN support for some reason, define the `IDN_DISABLE` environment variable. The IDN support is disabled if the variable is set when **nslookup** runs or when the standard output is not a tty.

FILES

`/etc/resolv.conf`

SEE ALSO

dig(1), host(1), named(8).

9.31 NSUPDATE

nsupdate — Dynamic DNS update utility

Synopsis

`nsupdate` [`-d`] [`-D`] [`-i`] [`-L` *`level`*] [`-g` | `-o` | `-l` | `-y` *`[hmac:]keyname:secret`* | `-k` *`keyfile`*] [`-t` *`timeout`*] [`-u` *`udptimeout`*] [`-r` *`udpretries`*] [`-R` *`randomdev`*] [`-v`] [`-T`] [`-P`] [`-V`] [filename]

DESCRIPTION

nsupdate is used to submit Dynamic DNS Update requests as defined in RFC 2136 to a name server. This allows resource records to be added or removed from a zone without manually editing the zone file. A single update request can contain requests to add or remove more than one resource record.

Zones that are under dynamic control via **nsupdate** or a DHCP server should not be edited by hand. Manual edits could conflict with dynamic updates and cause data to be lost.

The resource records that are dynamically added or removed with **nsupdate** have to be in the same zone. Requests are sent to the zone's master server. This is identified by the MNAME field of the zone's SOA record.

Transaction signatures can be used to authenticate the Dynamic DNS updates. These use the TSIG resource record type described in RFC 2845 or the SIG(0) record described in RFC 2535 and RFC 2931 or GSS-TSIG as described in RFC 3645.

TSIG relies on a shared secret that should only be known to **nsupdate** and the name server. For instance, suitable key and server statements would be added to `/etc/named.conf` so that the name server can associate the appropriate secret key and algorithm with the IP address of the client application that will be using TSIG authentication. You can use **ddns-confgen** to generate suitable configuration fragments. **nsupdate** uses the `-y` or `-k` options to provide the TSIG shared secret. These options are mutually exclusive.

SIG(0) uses public key cryptography. To use a SIG(0) key, the public key must be stored in a KEY record in a zone served by the name server.

GSS-TSIG uses Kerberos credentials. Standard GSS-TSIG mode is switched on with the `-g` flag. A non-standards-compliant variant of GSS-TSIG used by Windows 2000 can be switched on with the `-o` flag.

OPTIONS

-d
Debug mode. This provides tracing information about the update requests that are made and the replies received from the name server.

-D
Extra debug mode.

-i
Force interactive mode, even when standard input is not a terminal.

-k *keyfile*
The file containing the TSIG authentication key. Keyfiles may be in two formats: a single file containing a `named.conf`-format **key** statement, which may be generated automatically by **ddns-confgen**, or a pair of files whose names are of the format `K{name}.+157.+{random}.key` and `K{name}.+157.+{random}.private`, which can be generated by **dnssec-keygen**. The `-k` may also be used to specify a SIG(0) key used to authenticate Dynamic DNS update requests. In this case, the key specified is not an HMAC-MD5 key.

-l
Local-host only mode. This sets the server address to localhost (disabling the **server** so that the server address cannot be overridden). Connections to the local server will use a TSIG key found in `/var/run/named/session.key`, which is automatically generated by **named** if any local master zone has set **update-policy** to **local**. The location of this key file can be overridden with the `-k` option.

-L *level*
Set the logging debug level. If zero, logging is disabled.

-p *port*
Set the port to use for connections to a name server. The default is 53.

-P
Print the list of private BIND-specific resource record types whose format is understood by **nsupdate**. See also the `-T` option.

-r *udpretries*
The number of UDP retries. The default is 3. If zero, only one update request will be made.

-R *randomdev*
Where to obtain randomness. If the operating system does not provide a `/dev/random` or equivalent device, the default source of randomness is keyboard input. `randomdev` specifies the name of a character device or file containing random data to be used instead of the default. The special value `keyboard` indicates that keyboard input should be used. This option may be specified multiple times.

-t *timeout*
The maximum time an update request can take before it is aborted. The default is 300 seconds. Zero can be used to disable the timeout.

-T

Print the list of IANA standard resource record types whose format is understood by **nsupdate**. **nsupdate** will exit after the lists are printed. The `-T` option can be combined with the `-P` option.

Other types can be entered using "TYPEXXXXX" where "XXXXX" is the decimal value of the type with no leading zeros. The rdata, if present, will be parsed using the UNKNOWN rdata format, (<backslash> <hash> <space> <length> <space> <hexstring>).

-u ***udptimeout***

The UDP retry interval. The default is 3 seconds. If zero, the interval will be computed from the timeout interval and number of UDP retries.

-v

Use TCP even for small update requests. By default, **nsupdate** uses UDP to send update requests to the name server unless they are too large to fit in a UDP request in which case TCP will be used. TCP may be preferable when a batch of update requests is made.

-V

Print the version number and exit.

-y ***[hmac:]keyname:secret***

Literal TSIG authentication key. *keyname* is the name of the key, and *secret* is the base64 encoded shared secret. *hmac* is the name of the key algorithm; valid choices are `hmac-md5`, `hmac-sha1`, `hmac-sha224`, `hmac-sha256`, `hmac-sha384`, or `hmac-sha512`. If *hmac* is not specified, the default is `hmac-md5` or if MD5 was disabled `hmac-sha256`.

NOTE: Use of the `-y` option is discouraged because the shared secret is supplied as a command line argument in clear text. This may be visible in the output from ps(1) or in a history file maintained by the user's shell.

INPUT FORMAT

nsupdate reads input from *filename* or standard input. Each command is supplied on exactly one line of input. Some commands are for administrative purposes. The others are either update instructions or prerequisite checks on the contents of the zone. These checks set conditions that some name or set of resource records (RRset) either exists or is absent from the zone. These conditions must be met if the entire update request is to succeed. Updates will be rejected if the tests for the prerequisite conditions fail.

Every update request consists of zero or more prerequisites and zero or more updates. This allows a suitably authenticated update request to proceed if some specified resource records are present or missing from the zone. A blank input line (or the **send** command) causes the accumulated commands to be sent as one Dynamic DNS update request to the name server.

The command formats and their meaning are as follows:

server servername [port]

Sends all dynamic update requests to the name server *servername*. When no server statement is provided, **nsupdate** will send updates to the master server of the correct zone. The MNAME field of that zone's SOA record will identify the master server for that zone. *port* is the port number on *servername* where the dynamic update requests get sent. If no port number is specified, the default DNS port number of 53 is used.

local address [port]
Sends all dynamic update requests using the local *address*. When no local statement is provided, **nsupdate** will send updates using an address and port chosen by the system. *port* can additionally be used to make requests come from a specific port. If no port number is specified, the system will assign one.

zone zonename
Specifies that all updates are to be made to the zone *zonename*. If no *zone* statement is provided, **nsupdate** will attempt determine the correct zone to update based on the rest of the input.

class classname
Specify the default class. If no *class* is specified, the default class is *IN*.

ttl seconds
Specify the default time to live for records to be added. The value *none* will clear the default ttl.

key [hmac:] keyname secret
Specifies that all updates are to be TSIG-signed using the *keyname secret* pair. If *hmac* is specified, then it sets the signing algorithm in use; the default is `hmac-md5` or if MD5 was disabled `hmac-sha256`. The **key** command overrides any key specified on the command line via `-y` or `-k`.

gsstsig
Use GSS-TSIG to sign the updated. This is equivalent to specifying `-g` on the command line.

oldgsstsig
Use the Windows 2000 version of GSS-TSIG to sign the updated. This is equivalent to specifying `-o` on the command line.

realm [realm_name]
When using GSS-TSIG use *realm_name* rather than the default realm in `krb5.conf`. If no realm is specified the saved realm is cleared.

check-names [yes_or_no]
Turn on or off check-names processing on records to be added. Check-names has no effect on prerequisites or records to be deleted. By default check-names processing is on. If check-names processing fails the record will not be added to the UPDATE message.

[prereq] nxdomain domain-name
Requires that no resource record of any type exists with name *domain-name*.

[prereq] yxdomain domain-name
Requires that *domain-name* exists (has as at least one resource record, of any type).

[prereq] nxrrset domain-name [class] type
Requires that no resource record exists of the specified *type*, *class* and *domain-name*. If *class* is omitted, IN (internet) is assumed.

[prereq] yxrrset domain-name [class] type
This requires that a resource record of the specified *type*, *class* and *domain-name* must exist. If *class* is omitted, IN (internet) is assumed.

[prereq] yxrrset domain-name [class] type data...
The *data* from each set of prerequisites of this form sharing a common *type*, *class*, and *domain-name* are combined to form a set of RRs. This set of RRs must exactly match the set of RRs existing in the zone at the given *type*, *class*, and *domain-name*. The *data* are written in the standard text representation of the resource record's RDATA.

[update] del[ete] domain-name [ttl] [class] [type [data...]]
Deletes any resource records named *domain-name*. If *type* and *data* is provided, only matching resource records will be removed. The internet class is assumed if *class* is not supplied. The *ttl* is ignored, and is only allowed for compatibility.

[update] add domain-name ttl [class] type data...
Adds a new resource record with the specified *ttl*, *class* and *data*.

show
Displays the current message, containing all of the prerequisites and updates specified since the last send.

send
Sends the current message. This is equivalent to entering a blank line.

answer
Displays the answer.

debug
Turn on debugging.

version
Print version number.

help
Print a list of commands.

Lines beginning with a semicolon are comments and are ignored.

EXAMPLES

The examples below show how **nsupdate** could be used to insert and delete resource records from the example.com zone. Notice that the input in each example contains a trailing blank line so that a group of commands are sent as one dynamic update request to the master name server for example.com.

```
# nsupdate
> update delete oldhost.example.com A
> update add newhost.example.com 86400 A 172.16.1.1
> send
```

Any A records for oldhost.example.com are deleted. And an A record for newhost.example.com with IP address 172.16.1.1 is added. The newly-added record has a 1 day TTL (86400 seconds).

```
# nsupdate
> prereq nxdomain nickname.example.com
> update add nickname.example.com 86400 CNAME somehost.example.com
> send
```

The prerequisite condition gets the name server to check that there are no resource records of any type for nickname.example.com. If there are, the update request fails. If this name does not exist, a CNAME for it is added. This ensures that when the CNAME is added, it cannot conflict with the long-standing rule in RFC 1034 that a name must not exist as any other record type if it exists as a CNAME. (The rule has been updated for DNSSEC in RFC 2535 to allow CNAMEs to have RRSIG, DNSKEY and NSEC records.)

FILES

/etc/resolv.conf
used to identify default name server

/var/run/named/session.key
sets the default TSIG key for use in local-only mode

K{name}.+157.+{random}.key
base-64 encoding of HMAC-MD5 key created by dnssec-keygen(8).

K{name}.+157.+{random}.private
base-64 encoding of HMAC-MD5 key created by dnssec-keygen(8).

SEE ALSO

RFC 2136, *RFC 3007*, *RFC 2104*, *RFC 2845*, *RFC 1034*, *RFC 2535*, *RFC 2931*, named(8), ddns-confgen(8), dnssec-keygen(8).

BUGS

The TSIG key is redundantly stored in two separate files. This is a consequence of nsupdate using the DST library for its cryptographic operations, and may change in future releases.

9.32 PKCS11-DESTROY

pkcs11-destroy — destroy PKCS#11 objects

Synopsis

`pkcs11-destroy` [`-m` *module*] [`-s` *slot*] **-i** *ID* | **-l** *label* [`-p` *PIN*] [`-w` *seconds*]

DESCRIPTION

pkcs11-destroy destroys keys stored in a PKCS#11 device, identified by their `ID` or `label`.

Matching keys are displayed before being destroyed. By default, there is a five second delay to allow the user to interrupt the process before the destruction takes place.

ARGUMENTS

-m *module*
Specify the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

-s *slot*
Open the session with the given PKCS#11 slot. The default is slot 0.

-i *ID*
Destroy keys with the given object ID.

-l *label*
Destroy keys with the given label.

-p *PIN*
Specify the PIN for the device. If no PIN is provided on the command line, **pkcs11-destroy** will prompt for it.

-w *seconds*
Specify how long to pause before carrying out key destruction. The default is five seconds. If set to `0`, destruction will be immediate.

SEE ALSO

pkcs11-keygen(8), pkcs11-list(8), pkcs11-tokens(8)

9.33 PKCS11-KEYGEN

pkcs11-keygen — generate keys on a PKCS#11 device

Synopsis

`pkcs11-keygen` **-a** *`algorithm`* [`-b` *`keysize`*] [`-e`] [`-i` *`id`*] [`-m` *`module`*] [`-P`] [`-p` *`PIN`*] [`-q`] [`-S`] [`-s` *`slot`*] label

DESCRIPTION

pkcs11-keygen causes a PKCS#11 device to generate a new key pair with the given `label` (which must be unique) and with `keysize` bits of prime.

ARGUMENTS

-a *algorithm*
Specify the key algorithm class: Supported classes are RSA, DSA, DH, ECC and ECX. In addition to these strings, the `algorithm` can be specified as a DNSSEC signing algorithm that will be used with this key; for example, NSEC3RSASHA1 maps to RSA, ECDSAP256SHA256 maps to ECC, and ED25519 to ECX. The default class is "RSA".

-b *keysize*
Create the key pair with `keysize` bits of prime. For ECC keys, the only valid values are 256 and 384, and the default is 256. For ECX kyes, the only valid values are 256 and 456, and the default is 256.

-e
For RSA keys only, use a large exponent.

-i *id*
Create key objects with id. The id is either an unsigned short 2 byte or an unsigned long 4 byte number.

-m *module*
Specify the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

-P
Set the new private key to be non-sensitive and extractable. The allows the private key data to be read from the PKCS#11 device. The default is for private keys to be sensitive and non-extractable.

-p *PIN*
Specify the PIN for the device. If no PIN is provided on the command line, **pkcs11-keygen** will prompt for it.

-q
Quiet mode: suppress unnecessary output.

-S
For Diffie-Hellman (DH) keys only, use a special prime of 768, 1024 or 1536 bit size and base (aka generator) 2. If not specified, bit size will default to 1024.

-s *slot*
Open the session with the given PKCS#11 slot. The default is slot 0.

SEE ALSO

pkcs11-destroy(8), pkcs11-list(8), pkcs11-tokens(8), dnssec-keyfromlabel(8)

9.34 PKCS11-LIST

pkcs11-list — list PKCS#11 objects

Synopsis

`pkcs11-list` [`-P`] [`-m` *`module`*] [`-s` *`slot`*] [-i *`ID`*] [-l *`label`*] [`-p` *`PIN`*]

DESCRIPTION

pkcs11-list lists the PKCS#11 objects with `ID` or `label` or by default all objects. The object class, label, and ID are displayed for all keys. For private or secret keys, the extractability attribute is also displayed, as either `true`, `false`, or `never`.

ARGUMENTS

-P
List only the public objects. (Note that on some PKCS#11 devices, all objects are private.)

-m *module*
Specify the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

-s *slot*
Open the session with the given PKCS#11 slot. The default is slot 0.

-i *ID*
List only key objects with the given object ID.

-l *label*
List only key objects with the given label.

-p *PIN*
Specify the PIN for the device. If no PIN is provided on the command line, **pkcs11-list** will prompt for it.

SEE ALSO

pkcs11-destroy(8), pkcs11-keygen(8), pkcs11-tokens(8)

9.35 PKCS11-TOKENS

pkcs11-tokens — list PKCS#11 available tokens

Synopsis

`pkcs11-tokens` [`-m` *`module`*] [`-v`]

DESCRIPTION

pkcs11-tokens lists the PKCS#11 available tokens with defaults from the slot/token scan performed at application initialization.

ARGUMENTS

-m *module*
Specify the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

-v
Make the PKCS#11 libisc initialization verbose.

SEE ALSO

pkcs11-destroy(8), pkcs11-keygen(8), pkcs11-list(8)

9.36 RNDC-CONFGEN

rndc-confgen — rndc key generation tool

Synopsis

`rndc-confgen` [`-a`] [`-A` *algorithm*] [`-b` *keysize*] [`-c` *keyfile*] [`-h`] [`-k` *keyname*] [`-p` *port*] [`-r` *randomfile*] [`-s` *address*] [`-t` *chrootdir*] [`-u` *user*]

DESCRIPTION

rndc-confgen generates configuration files for **rndc**. It can be used as a convenient alternative to writing the `rndc.conf` file and the corresponding **controls** and **key** statements in `named.conf` by hand. Alternatively, it can be run with the **-a** option to set up a `rndc.key` file and avoid the need for a `rndc.conf` file and a **controls** statement altogether.

OPTIONS

-a
Do automatic **rndc** configuration. This creates a file `rndc.key` in `/etc` (or whatever `sysconfdir` was specified as when BIND was built) that is read by both **rndc** and **named** on startup. The `rndc.key` file defines a default command channel and authentication key allowing **rndc** to communicate with **named** on the local host with no further configuration.

Running **rndc-confgen -a** allows BIND 9 and **rndc** to be used as drop-in replacements for BIND 8 and **ndc**, with no changes to the existing BIND 8 `named.conf` file.

If a more elaborate configuration than that generated by **rndc-confgen -a** is required, for example if rndc is to be used remotely, you should run **rndc-confgen** without the **-a** option and set up a `rndc.conf` and `named.conf` as directed.

-A ***algorithm***
Specifies the algorithm to use for the TSIG key. Available choices are: hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384 and hmac-sha512. The default is hmac-md5 or if MD5 was disabled hmac-sha256.

-b ***keysize***
Specifies the size of the authentication key in bits. Must be between 1 and 512 bits; the default is the hash size.

-c ***keyfile***
Used with the **-a** option to specify an alternate location for `rndc.key`.

-h
Prints a short summary of the options and arguments to **rndc-confgen**.

-k ***keyname***
Specifies the key name of the rndc authentication key. This must be a valid domain name. The default is `rndc-key`.

-p ***port***
Specifies the command channel port where **named** listens for connections from **rndc**. The default is 953.

-r ***randomfile***
Specifies a source of random data for generating the authorization. If the operating system does not provide a `/dev/random` or equivalent device, the default source of randomness is keyboard input. `randomdev` specifies the name of a character device or file containing random data to be used instead of the default. The special value `keyboard` indicates that keyboard input should be used.

-s ***address***
Specifies the IP address where **named** listens for command channel connections from **rndc**. The default is the loopback address 127.0.0.1.

-t ***chrootdir***
Used with the **-a** option to specify a directory where **named** will run chrooted. An additional copy of the `rndc.key` will be written relative to this directory so that it will be found by the chrooted **named**.

-u ***user***
Used with the **-a** option to set the owner of the `rndc.key` file generated. If **-t** is also specified only the file in the chroot area has its owner changed.

EXAMPLES

To allow **rndc** to be used with no manual configuration, run

rndc-confgen -a

To print a sample `rndc.conf` file and corresponding **controls** and **key** statements to be manually inserted into `named.conf`, run

rndc-confgen

SEE ALSO

rndc(8), rndc.conf(5), named(8), *BIND 9 Administrator Reference Manual*.

9.37 RNDC.CONF

`rndc.conf` — rndc configuration file

Synopsis

`rndc.conf`

DESCRIPTION

`rndc.conf` is the configuration file for **rndc**, the BIND 9 name server control utility. This file has a similar structure and syntax to `named.conf`. Statements are enclosed in braces and terminated with a semi-colon. Clauses in the statements are also semi-colon terminated. The usual comment styles are supported:

C style: /* */

C++ style: // to end of line

Unix style: # to end of line

`rndc.conf` is much simpler than `named.conf`. The file uses three statements: an options statement, a server statement and a key statement.

The `options` statement contains five clauses. The `default-server` clause is followed by the name or address of a name server. This host will be used when no name server is given as an argument to **rndc**. The `default-key` clause is followed by the name of a key which is identified by a `key` statement. If no `keyid` is provided on the rndc command line, and no `key` clause is found in a matching `server` statement, this default key will be used to authenticate the server's commands and responses. The `default-port` clause is followed by the port to connect to on the remote name server. If no `port` option is provided on the rndc command line, and no `port` clause is found in a matching `server` statement, this default port will be used to connect. The `default-source-address` and `default-source-address-v6` clauses which can be used to set the IPv4 and IPv6 source addresses respectively.

After the `server` keyword, the server statement includes a string which is the hostname or address for a name server. The statement has three possible clauses: `key`, `port` and `addresses`. The key name must match the name of a key statement in the file. The port number specifies the port to connect to. If an `addresses` clause is supplied these addresses will be used instead of the server name. Each address can take an optional port. If an `source-address` or `source-address-v6` of supplied then these will be used to specify the IPv4 and IPv6 source addresses respectively.

The `key` statement begins with an identifying string, the name of the key. The statement has two clauses. `algorithm` identifies the authentication algorithm for **rndc** to use; currently only HMAC-MD5 (for compatibility), HMAC-SHA1, HMAC-SHA224, HMAC-SHA256 (default), HMAC-SHA384 and HMAC-SHA512 are supported. This is followed by a secret clause which contains the base-64 encoding of the algorithm's authentication key. The base-64 string is enclosed in double quotes.

There are two common ways to generate the base-64 string for the secret. The BIND 9 program **rndc-confgen** can be used to generate a random key, or the **mmencode** program, also known as **mimencode**, can be used to generate a base-64 string from known input. **mmencode** does not ship with BIND 9 but is available on many systems. See the EXAMPLE section for sample command lines for each.

EXAMPLE

```
options {
  default-server  localhost;
  default-key     samplekey;
};
```

```
server localhost {
  key             samplekey;
};
```

```
server testserver {
  key   testkey;
  addresses { localhost port 5353; };
};
```

```
key samplekey {
  algorithm       hmac-sha256;
  secret          "6FMfj43Osz4lyb24OIe2iGEz9lf1llJO+lz";
};
```

```
key testkey {
  algorithm hmac-sha256;
  secret    "R3HI8P6BKw9ZwXwN3VZKuQ==";
};
```

In the above example, **rndc** will by default use the server at localhost (127.0.0.1) and the key called samplekey. Commands to the localhost server will use the samplekey key, which must

also be defined in the server's configuration file with the same name and secret. The key statement indicates that samplekey uses the HMAC-SHA256 algorithm and its secret clause contains the base-64 encoding of the HMAC-SHA256 secret enclosed in double quotes.

If **rndc -s testserver** is used then **rndc** will connect to server on localhost port 5353 using the key testkey.

To generate a random secret with **rndc-confgen**:

rndc-confgen

A complete `rndc.conf` file, including the randomly generated key, will be written to the standard output. Commented-out `key` and `controls` statements for `named.conf` are also printed.

To generate a base-64 secret with **mmencode**:

echo "known plaintext for a secret" | mmencode

NAME SERVER CONFIGURATION

The name server must be configured to accept rndc connections and to recognize the key specified in the `rndc.conf` file, using the controls statement in `named.conf`. See the sections on the `controls` statement in the BIND 9 Administrator Reference Manual for details.

SEE ALSO

rndc(8), rndc-confgen(8), mmencode(1), *BIND 9 Administrator Reference Manual*.

9.38 RNDC

rndc — name server control utility

Synopsis

`rndc` [`-b` *source-address*] [`-c` *config-file*] [`-k` *key-file*] [`-s` *server*] [`-p` *port*] [`-q`] [`-r`] [`-V`] [`-y` *key_id*] command

DESCRIPTION

rndc controls the operation of a name server. It supersedes the **ndc** utility that was provided in old BIND releases. If **rndc** is invoked with no command line options or arguments, it prints a short summary of the supported commands and the available options and their arguments.

rndc communicates with the name server over a TCP connection, sending commands authenticated with digital signatures. In the current versions of **rndc** and **named**, the only supported authentication algorithms are HMAC-MD5 (for compatibility), HMAC-SHA1, HMAC-SHA224, HMAC-SHA256 (default), HMAC-SHA384 and HMAC-SHA512. They use a shared secret on each end of the connection. This provides TSIG-style authentication for the command request

and the name server's response. All commands sent over the channel must be signed by a key_id known to the server.

rndc reads a configuration file to determine how to contact the name server and decide what algorithm and key it should use.

OPTIONS

-b ***source-address***
Use *source-address* as the source address for the connection to the server. Multiple instances are permitted to allow setting of both the IPv4 and IPv6 source addresses.

-c ***config-file***
Use *config-file* as the configuration file instead of the default, `/etc/rndc.conf`.

-k ***key-file***
Use *key-file* as the key file instead of the default, `/etc/rndc.key`. The key in `/etc/rndc.key` will be used to authenticate commands sent to the server if the *config-file* does not exist.

-s ***server***
server is the name or address of the server which matches a server statement in the configuration file for **rndc**. If no server is supplied on the command line, the host named by the default-server clause in the options statement of the **rndc** configuration file will be used.

-p ***port***
Send commands to TCP port *port* instead of BIND 9's default control channel port, 953.

-q
Quiet mode: Message text returned by the server will not be printed except when there is an error.

-r
Instructs **rndc** to print the result code returned by **named** after executing the requested command (e.g., ISC_R_SUCCESS, ISC_R_FAILURE, etc).

-V
Enable verbose logging.

-y ***key_id***
Use the key *key_id* from the configuration file. *key_id* must be known by **named** with the same algorithm and secret string in order for control message validation to succeed. If no *key_id* is specified, **rndc** will first look for a key clause in the server statement of the server being used, or if no server statement is present for that host, then the default-key clause of the options statement. Note that the configuration file contains shared secrets which are used to send authenticated control commands to name servers. It should therefore not have general read or write access.

COMMANDS

A list of commands supported by **rndc** can be seen by running **rndc** without arguments.

Currently supported commands are:

addzone *zone* [*class* [*view*]] *configuration*
Add a zone while the server is running. This command requires the **allow-new-zones** option to be set to **yes**. The *configuration* string specified on the command line is the zone configuration text that would ordinarily be placed in `named.conf`.

The configuration is saved in a file called *name*`.nzf`, where *name* is the name of the view, or if it contains characters that are incompatible with use as a file name, a cryptographic hash generated from the name of the view. When **named** is restarted, the file will be loaded into the view configuration, so that zones that were added can persist after a restart.

This sample **addzone** command would add the zone `example.com` to the default view:

`$` **rndc addzone example.com '{ type master; file "example.com.db"; };'**

(Note the brackets and semi-colon around the zone configuration text.)

See also **rndc delzone** and **rndc modzone**.

delzone [-clean] *zone* [*class* [*view*]]
Delete a zone while the server is running.

If the `-clean` argument is specified, the zone's master file (and journal file, if any) will be deleted along with the zone. Without the `-clean` option, zone files must be cleaned up by hand. (If the zone is of type "slave" or "stub", the files needing to be cleaned up will be reported in the output of the **rndc delzone** command.)

If the zone was originally added via **rndc addzone**, then it will be removed permanently. However, if it was originally configured in `named.conf`, then that original configuration is still in place; when the server is restarted or reconfigured, the zone will come back. To remove it permanently, it must also be removed from `named.conf`

See also **rndc addzone** and **rndc modzone**.

dnstap (-reopen | -roll [*number***])**
Close and re-open DNSTAP output files. **rndc dnstap -reopen** allows the output file to be renamed externally, so that **named** can truncate and re-open it. **rndc dnstap -roll** causes the output file to be rolled automatically, similar to log files; the most recent output file has ".0" appended to its name; the previous most recent output file is moved to ".1", and so on. If *number* is specified, then the number of backup log files is limited to that number.

dumpdb [-all|-cache|-zones|-adb|-bad|-fail] [*view ...***]**
Dump the server's caches (default) and/or zones to the dump file for the specified views. If no view is specified, all views are dumped. (See the **dump-file** option in the BIND 9 Administrator Reference Manual.)

flush
Flushes the server's cache.

flushname *name* [*view*]
Flushes the given name from the view's DNS cache and, if applicable, from the view's nameserver address database, bad server cache and SERVFAIL cache.

flushtree *name* **[***view***]**
Flushes the given name, and all of its subdomains, from the view's DNS cache, address database, bad server cache, and SERVFAIL cache.

freeze [*zone* **[***class* **[***view***]]]**
Suspend updates to a dynamic zone. If no zone is specified, then all zones are suspended. This allows manual edits to be made to a zone normally updated by dynamic update. It also causes changes in the journal file to be synced into the master file. All dynamic update attempts will be refused while the zone is frozen.

See also **rndc thaw**.

halt [-p]
Stop the server immediately. Recent changes made through dynamic update or IXFR are not saved to the master files, but will be rolled forward from the journal files when the server is restarted. If `-p` is specified **named**'s process id is returned. This allows an external process to determine when **named** had completed halting.

See also **rndc stop**.

loadkeys *zone* **[***class* **[***view***]]**
Fetch all DNSSEC keys for the given zone from the key directory. If they are within their publication period, merge them into the zone's DNSKEY RRset. Unlike **rndc sign**, however, the zone is not immediately re-signed by the new keys, but is allowed to incrementally re-sign over time.

This command requires that the **auto-dnssec** zone option be set to `maintain`, and also requires the zone to be configured to allow dynamic DNS. (See "Dynamic Update Policies" in the Administrator Reference Manual for more details.)

managed-keys *(status | refresh | sync)* **[***class* **[***view***]]**
When run with the "status" keyword, print the current status of the managed-keys database for the specified view, or for all views if none is specified. When run with the "refresh" keyword, force an immediate refresh of all the managed-keys in the specified view, or all views. When run with the "sync" keyword, force an immediate dump of the managed-keys database to disk (in the file `managed-keys.bind` or (*viewname*`.mkeys`).

modzone *zone* **[***class* **[***view***]]** *configuration*
Modify the configuration of a zone while the server is running. This command requires the **allow-new-zones** option to be set to **yes**. As with **addzone**, the *configuration* string specified on the command line is the zone configuration text that would ordinarily be placed in `named.conf`.

If the zone was originally added via **rndc addzone**, the configuration changes will be recorded permanently and will still be in effect after the server is restarted or reconfigured. However, if it was originally configured in `named.conf`, then that original configuration is still in place; when the server is restarted or reconfigured, the zone will revert to its original configuration. To make the changes permanent, it must also be modified in `named.conf`

See also **rndc addzone** and **rndc delzone**.

notify *zone* **[***class* **[***view***]]**
Resend NOTIFY messages for the zone.

notrace
Sets the server's debugging level to 0.

See also **rndc trace**.

nta [(-class *class* | -dump | -force | -remove | -lifetime *duration*)] *domain* [*view*]

Sets a DNSSEC negative trust anchor (NTA) for `domain`, with a lifetime of `duration`. The default lifetime is configured in `named.conf` via the `nta-lifetime` option, and defaults to one hour. The lifetime cannot exceed one week.

A negative trust anchor selectively disables DNSSEC validation for zones that are known to be failing because of misconfiguration rather than an attack. When data to be validated is at or below an active NTA (and above any other configured trust anchors), **named** will abort the DNSSEC validation process and treat the data as insecure rather than bogus. This continues until the NTA's lifetime is elapsed.

NTAs persist across restarts of the **named** server. The NTAs for a view are saved in a file called *name*`.nta`, where *name* is the name of the view, or if it contains characters that are incompatible with use as a file name, a cryptographic hash generated from the name of the view.

An existing NTA can be removed by using the `-remove` option.

An NTA's lifetime can be specified with the `-lifetime` option. TTL-style suffixes can be used to specify the lifetime in seconds, minutes, or hours. If the specified NTA already exists, its lifetime will be updated to the new value. Setting `lifetime` to zero is equivalent to `-remove`.

If the `-dump` is used, any other arguments are ignored, and a list of existing NTAs is printed (note that this may include NTAs that are expired but have not yet been cleaned up).

Normally, **named** will periodically test to see whether data below an NTA can now be validated (see the `nta-recheck` option in the Administrator Reference Manual for details). If data can be validated, then the NTA is regarded as no longer necessary, and will be allowed to expire early. The `-force` overrides this behavior and forces an NTA to persist for its entire lifetime, regardless of whether data could be validated if the NTA were not present.

The view class can be specified with `-class`. The default is class **IN**, which is the only class for which DNSSEC is currently supported.

All of these options can be shortened, i.e., to `-l`, `-r`, `-d`, `-f`, and `-c`.

querylog [on | off]
Enable or disable query logging. (For backward compatibility, this command can also be used without an argument to toggle query logging on and off.)

Query logging can also be enabled by explicitly directing the **queries category** to a **channel** in the **logging** section of `named.conf` or by specifying **querylog yes;** in the **options** section of `named.conf`.

reconfig
Reload the configuration file and load new zones, but do not reload existing zone files even if they have changed. This is faster than a full **reload** when there is a large number of zones because it avoids the need to examine the modification times of the zones files.

recursing
Dump the list of queries **named** is currently recursing on, and the list of domains to which iterative queries are currently being sent. (The second list includes the number of fetches currently active for the given domain, and how many have been passed or dropped because of the `fetches-per-zone` option.)

refresh ***zone*** **[*class* [*view*]]**
Schedule zone maintenance for the given zone.

reload
Reload configuration file and zones.

reload ***zone*** **[*class* [*view*]]**
Reload the given zone.

retransfer ***zone*** **[*class* [*view*]]**
Retransfer the given slave zone from the master server.

If the zone is configured to use **inline-signing**, the signed version of the zone is discarded; after the retransfer of the unsigned version is complete, the signed version will be regenerated with all new signatures.

scan
Scan the list of available network interfaces for changes, without performing a full **reconfig** or waiting for the **interface-interval** timer.

secroots [-] [*view* ...]
Dump the server's security roots and negative trust anchors for the specified views. If no view is specified, all views are dumped.

If the first argument is "-", then the output is returned via the **rndc** response channel and printed to the standard output. Otherwise, it is written to the secroots dump file, which defaults to `named.secroots`, but can be overridden via the `secroots-file` option in `named.conf`.

See also **rndc managed-keys**.

showzone ***zone*** **[*class* [*view*]]**
Print the configuration of a running zone.

See also **rndc zonestatus**.

sign ***zone*** **[*class* [*view*]]**
Fetch all DNSSEC keys for the given zone from the key directory (see the **key-directory** option in the BIND 9 Administrator Reference Manual). If they are within their publication period, merge them into the zone's DNSKEY RRset. If the DNSKEY RRset is changed, then the zone is automatically re-signed with the new key set.

This command requires that the **auto-dnssec** zone option be set to `allow` or `maintain`, and also requires the zone to be configured to allow dynamic DNS. (See "Dynamic Update Policies" in the Administrator Reference Manual for more details.)

See also **rndc loadkeys**.

signing [(-list | -clear ***keyid/algorithm*** **| -clear all | -nsec3param (** ***parameters*** **|**

List, edit, or remove the DNSSEC signing state records for the specified zone. The status of ongoing DNSSEC operations (such as signing or generating NSEC3 chains) is stored in the zone in the form of DNS resource records of type **sig-signing-type**. **rndc signing -list** converts these records into a human-readable form, indicating which keys are currently signing or have finished signing the zone, and which NSEC3 chains are being created or removed.

rndc signing -clear can remove a single key (specified in the same format that **rndc signing -list** uses to display it), or all keys. In either case, only completed keys are removed; any record indicating that a key has not yet finished signing the zone will be retained.

rndc signing -nsec3param sets the NSEC3 parameters for a zone. This is the only supported mechanism for using NSEC3 with **inline-signing** zones. Parameters are specified in the same format as an NSEC3PARAM resource record: hash algorithm, flags, iterations, and salt, in that order.

Currently, the only defined value for hash algorithm is `1`, representing SHA-1. The `flags` may be set to `0` or `1`, depending on whether you wish to set the opt-out bit in the NSEC3 chain. `iterations` defines the number of additional times to apply the algorithm when generating an NSEC3 hash. The `salt` is a string of data expressed in hexadecimal, a hyphen (`-') if no salt is to be used, or the keyword `auto`, which causes **named** to generate a random 64-bit salt.

So, for example, to create an NSEC3 chain using the SHA-1 hash algorithm, no opt-out flag, 10 iterations, and a salt value of "FFFF", use: **rndc signing -nsec3param 1 0 10 FFFF *zone***. To set the opt-out flag, 15 iterations, and no salt, use: **rndc signing -nsec3param 1 1 15 - *zone***.

rndc signing -nsec3param none removes an existing NSEC3 chain and replaces it with NSEC.

rndc signing -serial value sets the serial number of the zone to value. If the value would cause the serial number to go backwards it will be rejected. The primary use is to set the serial on inline signed zones.

stats
Write server statistics to the statistics file. (See the **statistics-file** option in the BIND 9 Administrator Reference Manual.)

status
Display status of the server. Note that the number of zones includes the internal **bind/CH** zone and the default **./IN** hint zone if there is not an explicit root zone configured.

stop [-p]
Stop the server, making sure any recent changes made through dynamic update or IXFR are first saved to the master files of the updated zones. If `-p` is specified **named**'s process id is returned. This allows an external process to determine when **named** had completed stopping.

See also **rndc halt**.

sync [-clean] [*zone* [*class* [*view*]]]
Sync changes in the journal file for a dynamic zone to the master file. If the "-clean" option is specified, the journal file is also removed. If no zone is specified, then all zones are synced.

thaw [*zone* [*class* [*view*]]]
Enable updates to a frozen dynamic zone. If no zone is specified, then all frozen zones are enabled. This causes the server to reload the zone from disk, and re-enables dynamic updates after the load has completed. After a zone is thawed, dynamic updates will no longer be refused. If the zone has changed and the **ixfr-from-differences** option is in use, then the journal file will be updated to reflect changes in the zone. Otherwise, if the zone has changed, any existing journal file will be removed.

See also **rndc freeze**.

trace
Increment the servers debugging level by one.

trace *level*
Sets the server's debugging level to an explicit value.

See also **rndc notrace**.

tsig-delete *keyname* [*view*]
Delete a given TKEY-negotiated key from the server. (This does not apply to statically configured TSIG keys.)

tsig-list
List the names of all TSIG keys currently configured for use by **named** in each view. The list includes both statically configured keys and dynamic TKEY-negotiated keys.

validation (on | off | status) [*view* ...]
Enable, disable, or check the current status of DNSSEC validation. Note **dnssec-enable** also needs to be set to **yes** or **auto** to be effective. It defaults to enabled.

zonestatus *zone* [*class* [*view*]]
Displays the current status of the given zone, including the master file name and any include files from which it was loaded, when it was most recently loaded, the current serial number, the number of nodes, whether the zone supports dynamic updates, whether the zone is DNSSEC signed, whether it uses automatic DNSSEC key management or inline signing, and the scheduled refresh or expiry times for the zone.

See also **rndc showzone**.

LIMITATIONS

There is currently no way to provide the shared secret for a `key_id` without using the configuration file.

Several error messages could be clearer.

SEE ALSO

rndc.conf(5), rndc-confgen(8), named(8), named.conf(5), ndc(8), *BIND 9 Administrator Reference Manual*.

A Release Notes

A.1 Release Notes for BIND Version 9.11.36

Introduction

BIND 9.11 (Extended Support Version) is a stable branch of BIND. This document summarizes significant changes since the last production release on that branch.

Please see the file `CHANGES` for a more detailed list of changes and bug fixes.

Download

The latest versions of BIND 9 software can always be found at https://www.isc.org/download/. There you will find additional information about each release, source code, and pre-compiled versions for Microsoft Windows operating systems.

License Change

With the release of BIND 9.11.0, ISC changed to the open source license for BIND from the ISC license to the Mozilla Public License (MPL 2.0).

The MPL-2.0 license requires that if you make changes to licensed software (e.g. BIND) and distribute them outside your organization, that you publish those changes under that same license. It does not require that you publish or disclose anything other than the changes you made to our software.

This requirement will not affect anyone who is using BIND, with or without modifications, without redistributing it, nor anyone redistributing it without changes. Therefore, this change will be without consequence for most individuals and organizations who are using BIND.

Those unsure whether or not the license change affects their use of BIND, or who wish to discuss how to comply with the license may contact ISC at https://www.isc.org/mission/contact/.

Notes for BIND 9.11.36

Security Fixes

- The **lame-ttl** option controls how long **named** caches certain types of broken responses from authoritative servers (see the security advisory for details). This caching mechanism could be abused by an attacker to significantly degrade resolver performance. The vulnerability has been mitigated by changing the default value of **lame-ttl** to **0** and overriding any explicitly set value with **0**, effectively disabling this mechanism altogether. ISC's testing has determined that doing that has a negligible impact on resolver performance while also preventing abuse. Administrators may observe more traffic towards servers issuing certain types of broken responses than in previous BIND 9 releases, depending on client query patterns. (CVE-2021-25219)

  ISC would like to thank Kishore Kumar Kothapalli of Infoblox for bringing this vulnerability to our attention. [GL #2899]

Notes for BIND 9.11.35

Security Fixes

- **named** failed to check the opcode of responses when performing zone refreshes, stub zone updates, and UPDATE forwarding. This could lead to an assertion failure under certain conditions and has been addressed by rejecting responses whose opcode does not match the expected value. [GL #2762]

Notes for BIND 9.11.34

This maintenance release of BIND 9.11 contains no significant changes, although some minor updates have been made (for example, to fix build issues on Solaris 11).

Notes for BIND 9.11.33

This maintenance release of BIND 9.11 contains no significant changes, although some minor updates have been made (for example, to eliminate compiler warnings emitted by GCC 11).

Notes for BIND 9.11.32

Feature Changes

- DNSSEC responses containing NSEC3 records with iteration counts greater than 150 are now treated as insecure. [GL #2445]
- The maximum supported number of NSEC3 iterations that can be configured for a zone has been reduced to 150. [GL #2642]
- The implementation of the ZONEMD RR type has been updated to match RFC 8976. [GL #2658]

Notes for BIND 9.11.31

Security Fixes

- A malformed incoming IXFR transfer could trigger an assertion failure in **named**, causing it to quit abnormally. (CVE-2021-25214)

  ISC would like to thank Greg Kuechle of SaskTel for bringing this vulnerability to our attention. [GL #2467]

- **named** crashed when a DNAME record placed in the ANSWER section during DNAME chasing turned out to be the final answer to a client query. (CVE-2021-25215)

  ISC would like to thank Siva Kakarla for bringing this vulnerability to our attention. [GL #2540]

- When a server's configuration set the **tkey-gssapi-keytab** or **tkey-gssapi-credential** option, a specially crafted GSS-TSIG query could cause a buffer overflow in the ISC implementation of SPNEGO (a protocol enabling negotiation of the security mechanism used for GSSAPI authentication). This flaw could be exploited to crash **named** binaries compiled for 64-bit platforms, and could enable remote code execution when **named** was compiled for 32-bit platforms. (CVE-2021-25216)

  This vulnerability was reported to us as ZDI-CAN-13347 by Trend Micro Zero Day Initiative. [GL #2604]

Feature Changes

- The ISC implementation of SPNEGO was removed from BIND 9 source code. Instead, BIND 9 now always uses the SPNEGO implementation provided by the system GSSAPI library when it is built with GSSAPI support. All major contemporary Kerberos/GSSAPI libraries contain an implementation of the SPNEGO mechanism. [GL #2607]

Notes for BIND 9.11.30

The BIND 9.11.30 release was withdrawn after a backporting bug was discovered during pre-release testing. ISC would like to acknowledge the assistance of Natan Segal of Bluecat Networks.

Notes for BIND 9.11.29

Bug Fixes

- An invalid direction field (not one of **N**, **S**, **E**, **W**) in a LOC record resulted in an INSIST failure when a zone file containing such a record was loaded. [GL #2499]

Notes for BIND 9.11.28

Security Fixes

- When **tkey-gssapi-keytab** or **tkey-gssapi-credential** was configured, a specially crafted GSS-TSIG query could cause a buffer overflow in the ISC implementation of SPNEGO (a protocol enabling negotiation of the security mechanism to use for GSSAPI authentication). This flaw could be exploited to crash **named**. Theoretically, it also enabled remote code execution, but achieving the latter is very difficult in real-world conditions. (CVE-2020-8625)

  This vulnerability was responsibly reported to us as ZDI-CAN-12302 by Trend Micro Zero Day Initiative. [GL #2354]

Notes for BIND 9.11.27

Bug Fixes

- Multiple threads could attempt to destroy a single RBTDB instance at the same time, resulting in an unpredictable but low-probability assertion failure in `free_rbtdb()`. This has been fixed. [GL #2317]

Notes for BIND 9.11.26

Feature Changes

- The default value of **max-recursion-queries** was increased from 75 to 100. Since the queries sent towards root and TLD servers are now included in the count (as a result of the fix for CVE-2020-8616), **max-recursion-queries** has a higher chance of being exceeded by non-attack queries, which is the main reason for increasing its default value. [GL #2305]
- The default value of **nocookie-udp-size** was restored back to 4096 bytes. Since **max-udp-size** is the upper bound for **nocookie-udp-size**, this change relieves the operator from having to change **nocookie-udp-size** together with **max-udp-size** in order to increase the default EDNS buffer size limit. **nocookie-udp-size** can still be set to a value lower than **max-udp-size**, if desired. [GL #2250]

Bug Fixes

- Handling of missing DNS COOKIE responses over UDP was tightened by falling back to TCP. [GL #2275]
- The CNAME synthesized from a DNAME was incorrectly followed when the QTYPE was CNAME or ANY. [GL #2280]
- Building with native PKCS#11 support for AEP Keyper has been broken since BIND 9.11.22. This has been fixed. [GL #2315]

Notes for BIND 9.11.25

Bug Fixes

- **named** acting as a resolver could incorrectly treat signed zones with no DS record at the parent as bogus. Such zones should be treated as insecure. This has been fixed. [GL #2236]
- After a Negative Trust Anchor (NTA) is added, BIND performs periodic checks to see if it is still necessary. If BIND encountered a failure while creating a query to perform such a check, it attempted to dereference a NULL pointer, resulting in a crash. [GL #2244]
- A problem obtaining glue records could prevent a stub zone from functioning properly, if the authoritative server for the zone were configured for minimal responses. [GL #1736]

Notes for BIND 9.11.24

Feature Changes

- DNS Flag Day 2020: The default EDNS buffer size has been changed from 4096 to 1232 bytes. According to measurements done by multiple parties, this should not cause any operational problems as most of the Internet "core" is able to cope with IP message sizes between 1400-1500 bytes; the 1232 size was picked as a conservative minimal number that could be changed by the DNS operator to an estimated path MTU minus the estimated header space. In practice, the smallest MTU witnessed in the operational DNS community is 1500 octets, the maximum Ethernet payload size, so a useful default for maximum DNS/UDP payload size on reliable networks would be 1400 bytes. [GL #2183]

Bug Fixes

- **named** reported an invalid memory size when running in an environment that did not properly report the number of available memory pages and/or the size of each memory page. [GL #2166]
- With multiple forwarders configured, **named** could fail the `REQUIRE(msg->state == (-1))` assertion in `lib/dns/message.c`, causing it to crash. This has been fixed. [GL #2124]

Notes for BIND 9.11.23

Bug Fixes

- Parsing of LOC records was made more strict by rejecting a sole period (`.`) and/or `m` as a value. These changes prevent zone files using such values from being loaded. Handling of negative altitudes which are not integers was also corrected. [GL #2074]
- Several problems found by OSS-Fuzz were fixed. (None of these are security issues.) [GL !3953] [GL !3975]

Notes for BIND 9.11.22

Security Fixes

- It was possible to trigger an assertion failure when verifying the response to a TSIG-signed request. This was disclosed in CVE-2020-8622.

  ISC would like to thank Dave Feldman, Jeff Warren, and Joel Cunningham of Oracle for bringing this vulnerability to our attention. [GL #2028]

- When BIND 9 was compiled with native PKCS#11 support, it was possible to trigger an assertion failure in code determining the number of bits in the PKCS#11 RSA public key with a specially crafted packet. This was disclosed in CVE-2020-8623.

  ISC would like to thank Lyu Chiy for bringing this vulnerability to our attention. [GL #2037]

- **update-policy** rules of type **subdomain** were incorrectly treated as **zonesub** rules, which allowed keys used in **subdomain** rules to update names outside of the specified subdomains. The problem was fixed by making sure **subdomain** rules are again processed as described in the ARM. This was disclosed in CVE-2020-8624.

  ISC would like to thank Joop Boonen of credativ GmbH for bringing this vulnerability to our attention. [GL #2055]

Bug Fixes

- Wildcard RPZ passthru rules could incorrectly be overridden by other rules that were loaded from RPZ zones which appeared later in the **response-policy** statement. This has been fixed. [GL #1619]

- LMDB locking code was revised to make **rndc reconfig** work properly on FreeBSD and with LMDB >= 0.9.26. [GL #1976]

Notes for BIND 9.11.21

Bug Fixes

- **named** could crash when cleaning dead nodes in `lib/dns/rbtdb.c` that were being reused. [GL #1968]

- Properly handle missing **kyua** command so that **make check** does not fail unexpectedly when CMocka is installed, but Kyua is not. [GL #1950]

- The validator could fail to accept a properly signed RRset if an unsupported algorithm appeared earlier in the DNSKEY RRset than a supported algorithm. It could also stop if it detected a malformed public key. [GL #1689]

Notes for BIND 9.11.20

Security Fixes

- It was possible to trigger an INSIST failure when a zone with an interior wildcard label was queried in a certain pattern. This was disclosed in CVE-2020-8619. [GL #1111] [GL #1718]

New Features

- **dig** and other tools can now print the Extended DNS Error (EDE) option when it appears in a request or a response. [GL #1835]

Bug Fixes

- When fully updating the NSEC3 chain for a large zone via IXFR, a temporary loss of performance could be experienced on the secondary server when answering queries for nonexistent data that required DNSSEC proof of non-existence (in other words, queries that required the server to find and to return NSEC3 data). The unnecessary processing step that was causing this delay has now been removed. [GL #1834]

- A data race in `lib/dns/resolver.c:log_formerr()` that could lead to an assertion failure was fixed. [GL #1808]

- Previously, **provide-ixfr no;** failed to return up-to-date responses when the serial number was greater than or equal to the current serial number. [GL #1714]

- **named-checkconf -p** could include spurious text in **server-addresses** statements due to an uninitialized DSCP value. This has been fixed. [GL #1812]

- The ARM has been updated to indicate that the TSIG session key is generated when named starts, regardless of whether it is needed. [GL #1842]

Notes for BIND 9.11.19

Security Fixes

- To prevent exhaustion of server resources by a maliciously configured domain, the number of recursive queries that can be triggered by a request before aborting recursion has been further limited. Root and top-level domain servers are no longer exempt from the **max-recursion-queries** limit. Fetches for missing name server address records are limited to 4 for any domain. This issue was disclosed in CVE-2020-8616. [GL #1388]

- Replaying a TSIG BADTIME response as a request could trigger an assertion failure. This was disclosed in CVE-2020-8617. [GL #1703]

Feature Changes

- Message IDs in inbound AXFR transfers are now checked for consistency. Log messages are emitted for streams with inconsistent message IDs. [GL #1674]

Bug Fixes

- When running on a system with support for Linux capabilities, **named** drops root privileges very soon after system startup. This was causing a spurious log message, "unable to set effective uid to 0: Operation not permitted", which has now been silenced. [GL #1042] [GL #1090]
- When **named-checkconf -z** was run, it would sometimes incorrectly set its exit code. It reflected the status of the last view found; if zone-loading errors were found in earlier configured views but not in the last one, the exit code indicated success. Thanks to Graham Clinch. [GL #1807]
- When built without LMDB support, **named** failed to restart after a zone with a double quote (") in its name was added with **rndc addzone**. Thanks to Alberto Fernández. [GL #1695]

Notes for BIND 9.11.18

Security Fixes

- DNS rebinding protection was ineffective when BIND 9 is configured as a forwarding DNS server. Found and responsibly reported by Tobias Klein. [GL #1574]

Known Issues

- We have received reports that in some circumstances, receipt of an IXFR can cause the processing of queries to slow significantly. Some of these are related to RPZ processing, others appear to occur where there are NSEC3-related changes (such as an operator changing the NSEC3 salt used in the hash calculation). These are being investigated. [GL #1685]

Notes for BIND 9.11.17

Feature Changes

- The **configure** option **--with-libxml2** now uses **pkg-config** to detect libxml2 library availability. You will either have to install **pkg-config** or specify the exact path where libxml2 has been installed on your system. [GL #1635]

Bug Fixes

- Fixed re-signing issues with inline zones which resulted in records being re-signed late or not at all.

Notes for BIND 9.11.16

Bug Fixes

- **named** crashed when it was queried for a nonexistent name in the CHAOS class. [GL #1540]

Notes for BIND 9.11.15

Bug Fixes

- Fixed a GeoIP2 lookup bug which was triggered when certain libmaxminddb versions were used. [GL #1552]
- Fixed several possible race conditions discovered by ThreadSanitizer.

Notes for BIND 9.11.14

Bug Fixes

- Fixed a bug that caused **named** to leak memory on reconfiguration when any GeoIP2 database was in use. [GL #1445]
- Fixed several possible race conditions discovered by ThreadSanitizer.

Notes for BIND 9.11.13

Security Fixes

- Set a limit on the number of concurrently served pipelined TCP queries. This flaw is disclosed in CVE-2019-6477. [GL #1264]

New Features

- Added a new statistics variable **tcp-highwater** that reports the maximum number of simultaneous TCP clients BIND has handled while running. [GL #1206]

Notes for BIND 9.11.12

None.

Notes for BIND 9.11.11

None.

Notes for BIND 9.11.10

New Features

- A SipHash 2-4 based DNS Cookie (RFC 7873) algorithm has been added. [GL #605]

  If you are running multiple DNS Servers (different versions of BIND 9 or DNS server from multiple vendors) responding from the same IP address (anycast or load-balancing scenarios), you'll have to make sure that all the servers are configured with the same DNS Cookie algorithm and same Server Secret for the best performance.

- DS records included in DNS referral messages can now be validated and cached immediately, reducing the number of queries needed for a DNSSEC validation. [GL #964]

Bug Fixes

- Interaction between DNS64 and RPZ No Data rule (CNAME *.) could cause unexpected results; this has been fixed. [GL #1106]
- **named-checkconf** now checks DNS64 prefixes to ensure bits 64-71 are zero. [GL #1159]
- **named-checkconf** could crash during configuration if configured to use "geoip continent" ACLs with legacy GeoIP. [GL #1163]
- **named-checkconf** now correctly reports a missing **dnstap-output** option when **dnstap** is set. [GL #1136]
- Handle ETIMEDOUT error on connect() with a non-blocking socket. [GL #1133]

Notes for BIND 9.11.9

New Features

- The new GeoIP2 API from MaxMind is now supported when BIND is compiled using **configure --with-geoip2**. The legacy GeoIP API can be used by compiling with **configure --with-geoip** instead. (Note that the databases for the legacy API are no longer maintained by MaxMind.)

  The default path to the GeoIP2 databases will be set based on the location of the **libmaxminddb** library; for example, if it is in `/usr/local/lib`, then the default path will be `/usr/local/share/GeoIP`. This value can be overridden in `named.conf` using the **geoip-directory** option.

  Some **geoip** ACL settings that were available with legacy GeoIP, including searches for **netspeed**, **org**, and three-letter ISO country codes, will no longer work when using GeoIP2. Supported GeoIP2 database types are **country**, **city**, **domain**, **isp**, and **as**. All of the databases support both IPv4 and IPv6 lookups. [GL #182]

Bug Fixes

- Glue address records were not being returned in responses to root priming queries; this has been corrected. [GL #1092]

Notes for BIND 9.11.8

Security Fixes

- A race condition could trigger an assertion failure when a large number of incoming packets were being rejected. This flaw is disclosed in CVE-2019-6471. [GL #942]

Notes for BIND 9.11.7

Security Fixes

- The TCP client quota set using the **tcp-clients** option could be exceeded in some cases. This could lead to exhaustion of file descriptors. This flaw is disclosed in CVE-2018-5743. [GL #615]

Feature Changes

- When **trusted-keys** and **managed-keys** are both configured for the same name, or when **trusted-keys** is used to configure a trust anchor for the root zone and **dnssec-validation** is set to `auto`, automatic RFC 5011 key rollovers will fail.

  This combination of settings was never intended to work, but there was no check for it in the parser. This has been corrected; a warning is now logged. (In BIND 9.15 and higher this error will be fatal.) [GL #868]

Notes for BIND 9.11.6

Security Fixes

- Code change #4964, intended to prevent double signatures when deleting an inactive zone DNSKEY in some situations, introduced a new problem during zone processing in which some delegation glue RRsets are incorrectly identified as needing RRSIGs, which are then created for them using the current active ZSK for the zone. In some, but not all cases, the newly-signed RRsets are added to the zone's NSEC/NSEC3 chain, but incompletely -- this can result in a broken chain, affecting validation of proof of nonexistence for records in the zone. [GL #771]
- **named** could crash if it managed a DNSSEC security root with **managed-keys** and the authoritative zone rolled the key to an algorithm not supported by BIND 9. This flaw is disclosed in CVE-2018-5745. [GL #780]
- **named** leaked memory when processing a request with multiple Key Tag EDNS options present. ISC would like to thank Toshifumi Sakaguchi for bringing this to our attention. This flaw is disclosed in CVE-2018-5744. [GL #772]
- Zone transfer controls for writable DLZ zones were not effective as the **allowzonexfr** method was not being called for such zones. This flaw is disclosed in CVE-2019-6465. [GL #790]

Feature Changes

- When compiled with IDN support, the **dig** and the **nslookup** commands now disable IDN processing when the standard output is not a tty (e.g. not used by human). The command line options +idnin and +idnout need to be used to enable IDN processing when **dig** or **nslookup** is used from the shell scripts.

Notes for BIND 9.11.5

Security Fixes

- **named** could crash during recursive processing of DNAME records when **deny-answer-aliases** was in use. This flaw is disclosed in CVE-2018-5740. [GL #387]

New Features

- Two new update policy rule types have been added **krb5-selfsub** and **ms-selfsub** which allow machines with Kerberos principals to update the name space at or below the machine names identified in the respective principals.

Feature Changes

- The **rndc nta** command could not differentiate between views of the same name but different class; this has been corrected with the addition of a **-class** option. [GL #105]

Bug Fixes

- When a negative trust anchor was added to multiple views using **rndc nta**, the text returned via **rndc** was incorrectly truncated after the first line, making it appear that only one NTA had been added. This has been fixed. [GL #105]

Notes for BIND 9.11.4

Security Fixes

- When recursion is enabled but the **allow-recursion** and **allow-query-cache** ACLs are not specified, they should be limited to local networks, but they were inadvertently set to match the default **allow-query**, thus allowing remote queries. This flaw is disclosed in CVE-2018-5738. [GL #309]

New Features

- **named** now supports the "root key sentinel" mechanism. This enables validating resolvers to indicate which trust anchors are configured for the root, so that information about root key rollover status can be gathered. To disable this feature, add **root-key-sentinel no;** to `named.conf`.
- Added the ability not to return a DNS COOKIE option when one is present in the request. To prevent a cookie being returned, add **answer-cookie no;** to `named.conf`. [GL #173]

  answer-cookie no is only intended as a temporary measure, for use when **named** shares an IP address with other servers that do not yet support DNS COOKIE. A mismatch between servers on the same address is not expected to cause operational problems, but the option to disable COOKIE responses so that all servers have the same behavior is provided out of an abundance of caution. DNS COOKIE is an important security mechanism, and should not be disabled unless absolutely necessary.

Removed Features

- **named** will now log a warning if the old BIND now can be compiled against libidn2 library to add IDNA2008 support. Previously BIND only supported IDNA2003 using (now obsolete) idnkit-1 library.

Feature Changes

- **dig +noidnin** can be used to disable IDN processing on the input domain name, when BIND is compiled with IDN support.
- Multiple **cookie-secret** clause are now supported. The first **cookie-secret** in `named.conf` is used to generate new server cookies. Any others are used to accept old server cookies or those generated by other servers using the matching **cookie-secret**.

Bug Fixes

- **named** now rejects excessively large incremental (IXFR) zone transfers in order to prevent possible corruption of journal files which could cause **named** to abort when loading zones. [GL #339]
- **rndc reload** could cause **named** to leak memory if it was invoked before the zone loading actions from a previous **rndc reload** command were completed. [RT #47076]

Notes for BIND 9.11.3

Security Fixes

- Addresses could be referenced after being freed during resolver processing, causing an assertion failure. The chances of this happening were remote, but the introduction of a delay in resolution increased them. This bug is disclosed in CVE-2017-3145. [RT #46839]

- update-policy rules that otherwise ignore the name field now require that it be set to "." to ensure that any type list present is properly interpreted. If the name field was omitted from the rule declaration and a type list was present it wouldn't be interpreted as expected.

Removed Features

- The ISC DNSSEC Lookaside Validation (DLV) service has been shut down; all DLV records in the dlv.isc.org zone have been removed. References to the service have been removed from BIND documentation. Lookaside validation is no longer used by default by **delv**. The DLV key has been removed from `bind.keys`. Setting **dnssec-lookaside** to **auto** or to use dlv.isc.org as a trust anchor results in a warning being issued.
- **named** will now log a warning if the old root DNSSEC key is explicitly configured and has not been updated. [RT #43670]

Protocol Changes

- BIND can now use the Ed25519 and Ed448 Edwards Curve DNSSEC signing algorithms described in RFC 8080. Note, however, that these algorithms must be supported in OpenSSL; currently they are only available in the development branch of OpenSSL at https://github.com/openssl/o [RT #44696]
- When parsing DNS messages, EDNS KEY TAG options are checked for correctness. When printing messages (for example, in **dig**), EDNS KEY TAG options are printed in readable format.

Feature Changes

- **named** will no longer start or accept reconfiguration if **managed-keys** or **dnssec-validation auto** are in use and the managed-keys directory (specified by **managed-keys-directory**, and defaulting to the working directory if not specified), is not writable by the effective user ID. [RT #46077]
- Previously, **update-policy local;** accepted updates from any source so long as they were signed by the locally-generated session key. This has been further restricted; updates are now only accepted from locally configured addresses. [RT #45492]

Bug Fixes

- Attempting to validate improperly unsigned CNAME responses from secure zones could cause a validator loop. This caused a delay in returning SERVFAIL and also increased the chances of encountering the crash bug described in CVE-2017-3145. [RT #46839]
- When **named** was reconfigured, failure of some zones to load correctly could leave the system in an inconsistent state; while generally harmless, this could lead to a crash later when using **rndc addzone**. Reconfiguration changes are now fully rolled back in the event of failure. [RT #45841]

- Some header files included <isc/util.h> incorrectly as it pollutes with namespace with non ISC_ macros and this should only be done by explicitly including <isc/util.h>. This has been corrected. Some code may depend on <isc/util.h> being implicitly included via other header files. Such code should explicitly include <isc/util.h>.
- Zones created with **rndc addzone** could temporarily fail to inherit the **allow-transfer** ACL set in the **options** section of `named.conf`. [RT #46603]
- **named** failed to properly determine whether there were active KSK and ZSK keys for an algorithm when **update-check-ksk** was true (which is the default setting). This could leave records unsigned when rolling keys. [RT #46743] [RT #46754] [RT #46774]

Notes for BIND 9.11.2

Security Fixes

- An error in TSIG handling could permit unauthorized zone transfers or zone updates. These flaws are disclosed in CVE-2017-3142 and CVE-2017-3143. [RT #45383]
- The BIND installer on Windows used an unquoted service path, which can enable privilege escalation. This flaw is disclosed in CVE-2017-3141. [RT #45229]
- With certain RPZ configurations, a response with TTL 0 could cause **named** to go into an infinite query loop. This flaw is disclosed in CVE-2017-3140. [RT #45181]

Feature Changes

- **dig +ednsopt** now accepts the names for EDNS options in addition to numeric values. For example, an EDNS Client-Subnet option could be sent using **dig +ednsopt=ecs:...**. Thanks to John Worley of Secure64 for the contribution. [RT #44461]
- Threads in **named** are now set to human-readable names to assist debugging on operating systems that support that. Threads will have names such as "isc-timer", "isc-sockmgr", "isc-worker0001", and so on. This will affect the reporting of subsidiary thread names in **ps** and **top**, but not the main thread. [RT #43234]
- DiG now warns about .local queries which are reserved for Multicast DNS. [RT #44783]

Bug Fixes

- Fixed a bug that was introduced in an earlier development release which caused multi-packet AXFR and IXFR messages to fail validation if not all packets contained TSIG records; this caused interoperability problems with some other DNS implementations. [RT #45509]
- Reloading or reconfiguring **named** could fail on some platforms when LMDB was in use. [RT #45203]
- Due to some incorrectly deleted code, when BIND was built with LMDB, zones that were deleted via **rndc delzone** were removed from the running server but were not removed from the new zone database, so that deletion did not persist after a server restart. This has been corrected. [RT #45185]

- Semicolons are no longer escaped when printing CAA and URI records. This may break applications that depend on the presence of the backslash before the semicolon. [RT #45216]
- AD could be set on truncated answer with no records present in the answer and authority sections. [RT #45140]

Notes for BIND 9.11.1

Security Fixes

- **rndc ""** could trigger an assertion failure in **named**. This flaw is disclosed in (CVE-2017-3138). [RT #44924]
- Some chaining (i.e., type CNAME or DNAME) responses to upstream queries could trigger assertion failures. This flaw is disclosed in CVE-2017-3137. [RT #44734]
- **dns64** with **break-dnssec yes;** can result in an assertion failure. This flaw is disclosed in CVE-2017-3136. [RT #44653]
- If a server is configured with a response policy zone (RPZ) that rewrites an answer with local data, and is also configured for DNS64 address mapping, a NULL pointer can be read triggering a server crash. This flaw is disclosed in CVE-2017-3135. [RT #44434]
- A coding error in the `nxdomain-redirect` feature could lead to an assertion failure if the redirection namespace was served from a local authoritative data source such as a local zone or a DLZ instead of via recursive lookup. This flaw is disclosed in CVE-2016-9778. [RT #43837]
- **named** could mishandle authority sections with missing RRSIGs, triggering an assertion failure. This flaw is disclosed in CVE-2016-9444. [RT #43632]
- **named** mishandled some responses where covering RRSIG records were returned without the requested data, resulting in an assertion failure. This flaw is disclosed in CVE-2016-9147. [RT #43548]
- **named** incorrectly tried to cache TKEY records which could trigger an assertion failure when there was a class mismatch. This flaw is disclosed in CVE-2016-9131. [RT #43522]
- It was possible to trigger assertions when processing responses containing answers of type DNAME. This flaw is disclosed in CVE-2016-8864. [RT #43465]
- Added the ability to specify the maximum number of records permitted in a zone (`max-records #;`). This provides a mechanism to block overly large zone transfers, which is a potential risk with slave zones from other parties, as described in CVE-2016-6170. [RT #42143]

Feature Changes

- **dnstap** now stores both the local and remote addresses for all messages, instead of only the remote address. The default output format for **dnstap-read** has been updated to include these addresses, with the initiating address first and the responding address second, separated by "-%gt;" or "%lt;-" to indicate in which direction the message was sent. [RT #43595]

- Expanded and improved the YAML output from **dnstap-read -y**: it now includes packet size and a detailed breakdown of message contents. [RT #43622] [RT #43642]

- If an ACL is specified with an address prefix in which the prefix length is longer than the address portion (for example, 192.0.2.1/8), **named** will now log a warning. In future releases this will be a fatal configuration error. [RT #43367]

Bug Fixes

- A synthesized CNAME record appearing in a response before the associated DNAME could be cached, when it should not have been. This was a regression introduced while addressing CVE-2016-8864. [RT #44318]

- **named** could deadlock if multiple changes to NSEC/NSEC3 parameters for the same zone were being processed at the same time. [RT #42770]

- **named** could trigger an assertion when sending NOTIFY messages. [RT #44019]

- Referencing a nonexistent zone in a **response-policy** statement could cause an assertion failure during configuration. [RT #43787]

- **rndc addzone** could cause a crash when attempting to add a zone with a type other than **master** or **slave**. Such zones are now rejected. [RT #43665]

- **named** could hang when encountering log file names with large apparent gaps in version number (for example, when files exist called "logfile.0", "logfile.1", and "logfile.1482954169"). This is now handled correctly. [RT #38688]

- If a zone was updated while **named** was processing a query for nonexistent data, it could return out-of-sync NSEC3 records causing potential DNSSEC validation failure. [RT #43247]

Maintenance

- The built-in root hints have been updated to include an IPv6 address (2001:500:12::d0d) for G.ROOT-SERVERS.NET.

Miscellaneous Notes

- Authoritative server support for the EDNS Client Subnet option (ECS), introduced in BIND 9.11.0, was based on an early version of the specification, and is now known to have incompatibilities with other ECS implementations. It is also inefficient, requiring a separate view for each answer, and is unable to correct for overlapping subnets in the configuration. It is intended for testing purposes but is not recommended for for production use. This was not made sufficiently clear in the documentation at the time of release.

Notes for BIND 9.11.0

Security Fixes

- It was possible to trigger a assertion when rendering a message using a specially crafted request. This flaw is disclosed in CVE-2016-2776. [RT #43139]
- getrrsetbyname with a non absolute name could trigger an infinite recursion bug in lwresd and named with lwres configured if when combined with a search list entry the resulting name is too long. This flaw is disclosed in CVE-2016-2775. [RT #42694]

New Features

- A new method of provisioning secondary servers called "Catalog Zones" has been added. This is an implementation of draft-muks-dnsop-dns-catalog-zones/.

  A catalog zone is a regular DNS zone which contains a list of "member zones", along with the configuration options for each of those zones. When a server is configured to use a catalog zone, all the zones listed in the catalog zone are added to the local server as slave zones. When the catalog zone is updated (e.g., by adding or removing zones, or changing configuration options for existing zones) those changes will be put into effect. Since the catalog zone is itself a DNS zone, this means configuration changes can be propagated to slaves using the standard AXFR/IXFR update mechanism.

  This feature should be considered experimental. It currently supports only basic features; more advanced features such as ACLs and TSIG keys are not yet supported. Example catalog zone configurations can be found in the Chapter 9 of the BIND Administrator Reference Manual.

  Support for master entries with TSIG keys has been added to catalog zones, as well as support for allow-query and allow-transfer.

- Added an **isc.rndc** Python module, which allows **rndc** commands to be sent from Python programs.
- Added support for DynDB, a new interface for loading zone data from an external database, developed by Red Hat for the FreeIPA project. (Thanks in particular to Adam Tkac and Petr Spacek of Red Hat for the contribution.)

  Unlike the existing DLZ and SDB interfaces, which provide a limited subset of database functionality within BIND - translating DNS queries into real-time database lookups with relatively poor performance and with no ability to handle DNSSEC-signed data - DynDB is able to fully implement and extend the database API used natively by BIND.

  A DynDB module could pre-load data from an external data source, then serve it with the same performance and functionality as conventional BIND zones, and with the ability to take advantage of database features not available in BIND, such as multi-master replication.

- Fetch quotas are now compiled in by default: they no longer require BIND to be configured with **--enable-fetchlimit**, as was the case when the feature was introduced in BIND 9.10.3.

These quotas limit the queries that are sent by recursive resolvers to authoritative servers experiencing denial-of-service attacks. They can both reduce the harm done to authoritative servers and also avoid the resource exhaustion that can be experienced by recursive servers when they are being used as a vehicle for such an attack.

  - `fetches-per-server` limits the number of simultaneous queries that can be sent to any single authoritative server. The configured value is a starting point; it is automatically adjusted downward if the server is partially or completely non-responsive. The algorithm used to adjust the quota can be configured via the `fetch-quota-params` option.
  - `fetches-per-zone` limits the number of simultaneous queries that can be sent for names within a single domain. (Note: Unlike "fetches-per-server", this value is not self-tuning.)

Statistics counters have also been added to track the number of queries affected by these quotas.

- Added support for **dnstap**, a fast, flexible method for capturing and logging DNS traffic, developed by Robert Edmonds at Farsight Security, Inc., whose assistance is gratefully acknowledged.

  To enable **dnstap** at compile time, the **fstrm** and **protobuf-c** libraries must be available, and BIND must be configured with `--enable-dnstap`.

  A new utility **dnstap-read** has been added to allow **dnstap** data to be presented in a human-readable format.

  rndc dnstap -roll causes **dnstap** output files to be rolled like log files -- the most recent output file is renamed with a `.0` suffix, the next most recent with `.1`, etc. (Note that this only works when **dnstap** output is being written to a file, not to a UNIX domain socket.) An optional numerical argument specifies how many backup log files to retain; if not specified or set to 0, there is no limit.

  rndc dnstap -reopen simply closes and reopens the **dnstap** output channel without renaming the output file.

  For more information on **dnstap**, see https://dnstap.info.

- New statistics counters have been added to track traffic sizes, as specified in RSSAC002. Query and response message sizes are broken up into ranges of histogram buckets: TCP and UDP queries of size 0-15, 16-31, ..., 272-288, and 288+, and TCP and UDP responses of size 0-15, 16-31, ..., 4080-4095, and 4096+. These values can be accessed via the XML and JSON statistics channels at, for example, http://localhost:8888/xml/v3/traffic or http://localhost:8888/json/v1/traffic.

  Statistics for RSSAC02v3 traffic-volume, traffic-sizes and rcode-volume reporting are now collected.

- A new DNSSEC key management utility, **dnssec-keymgr**, has been added. This tool is meant to run unattended (e.g., under **cron**). It reads a policy definition file (default `/etc/dnssec-policy.conf`) and creates or updates DNSSEC keys as necessary to ensure that a zone's keys match the defined policy for that zone. New keys are created whenever necessary to ensure rollovers occur correctly. Existing keys' timing metadata is adjusted as needed to set the correct rollover period, prepublication interval, etc. If the configured

policy changes, keys are corrected automatically. See the **dnssec-keymgr** man page for full details.

Note: **dnssec-keymgr** depends on Python and on the Python lex/yacc module, PLY. The other Python-based tools, **dnssec-coverage** and **dnssec-checkds**, have been refactored and updated as part of this work.

dnssec-keymgr now takes a -r *randomfile* option.

(Many thanks to Sebastián Castro for his assistance in developing this tool at the IETF 95 Hackathon in Buenos Aires, April 2016.)

- The serial number of a dynamically updatable zone can now be set using **rndc signing -serial *number zonename***. This is particularly useful with `inline-signing` zones that have been reset. Setting the serial number to a value larger than that on the slaves will trigger an AXFR-style transfer.

- When answering recursive queries, SERVFAIL responses can now be cached by the server for a limited time; subsequent queries for the same query name and type will return another SERVFAIL until the cache times out. This reduces the frequency of retries when a query is persistently failing, which can be a burden on recursive servers. The SERVFAIL cache timeout is controlled by `servfail-ttl`, which defaults to 1 second and has an upper limit of 30.

- The new **rndc nta** command can now be used to set a "negative trust anchor" (NTA), disabling DNSSEC validation for a specific domain; this can be used when responses from a domain are known to be failing validation due to administrative error rather than because of a spoofing attack. NTAs are strictly temporary; by default they expire after one hour, but can be configured to last up to one week. The default NTA lifetime can be changed by setting the `nta-lifetime` in `named.conf`. When added, NTAs are stored in a file (*viewname*`.nta`) in order to persist across restarts of the **named** server.

- The EDNS Client Subnet (ECS) option is now supported for authoritative servers; if a query contains an ECS option then ACLs containing `geoip` or `ecs` elements can match against the address encoded in the option. This can be used to select a view for a query, so that different answers can be provided depending on the client network.

- The EDNS EXPIRE option has been implemented on the client side, allowing a slave server to set the expiration timer correctly when transferring zone data from another slave server.

- A new `masterfile-style` zone option controls the formatting of text zone files: When set to `full`, the zone file will dumped in single-line-per-record format.

- **dig +ednsopt** can now be used to set arbitrary EDNS options in DNS requests.

- **dig +ednsflags** can now be used to set yet-to-be-defined EDNS flags in DNS requests.

- **dig +[no]ednsnegotiation** can now be used enable / disable EDNS version negotiation.

- **dig +header-only** can now be used to send queries without a question section.

- **dig +ttlunits** causes **dig** to print TTL values with time-unit suffixes: w, d, h, m, s for weeks, days, hours, minutes, and seconds.

- **dig +zflag** can be used to set the last unassigned DNS header flag bit. This bit is normally zero.
- **dig +dscp=*value*** can now be used to set the DSCP code point in outgoing query packets.
- **dig +mapped** can now be used to determine if mapped IPv4 addresses can be used.
- **nslookup** will now look up IPv6 as well as IPv4 addresses by default. [RT #40420]
- `serial-update-method` can now be set to `date`. On update, the serial number will be set to the current date in YYYYMMDDNN format.
- **dnssec-signzone -N date** also sets the serial number to YYYYMMDDNN.
- **named -L *filename*** causes **named** to send log messages to the specified file by default instead of to the system log.
- The rate limiter configured by the `serial-query-rate` option no longer covers NOTIFY messages; those are now separately controlled by `notify-rate` and `startup-notify-rate` (the latter of which controls the rate of NOTIFY messages sent when the server is first started up or reconfigured).
- The default number of tasks and client objects available for serving lightweight resolver queries have been increased, and are now configurable via the new `lwres-tasks` and `lwres-clients` options in `named.conf`. [RT #35857]
- Log output to files can now be buffered by specifying **buffered yes;** when creating a channel.
- **delv +tcp** will exclusively use TCP when sending queries.
- **named** will now check to see whether other name server processes are running before starting up. This is implemented in two ways: 1) by refusing to start if the configured network interfaces all return "address in use", and 2) by attempting to acquire a lock on a file specified by the `lock-file` option or the **-X** command line option. The default lock file is `/var/run/named/named.lock`. Specifying `none` will disable the lock file check.
- **rndc delzone** can now be applied to zones which were configured in `named.conf`; it is no longer restricted to zones which were added by **rndc addzone**. (Note, however, that this does not edit `named.conf`; the zone must be removed from the configuration or it will return when **named** is restarted or reloaded.)
- **rndc modzone** can be used to reconfigure a zone, using similar syntax to **rndc addzone**.
- **rndc showzone** displays the current configuration for a specified zone.
- When BIND is built with the **lmdb** library (Lightning Memory-Mapped Database), **named** will store the configuration information for zones that are added via **rndc addzone** in a database, rather than in a flat "NZF" file. This dramatically improves performance for **rndc delzone** and **rndc modzone**: deleting or changing the contents of a database is much faster than rewriting a text file.

  On startup, if **named** finds an existing NZF file, it will automatically convert it to the new NZD database format.

To view the contents of an NZD, or to convert an NZD back to an NZF file (for example, to revert back to an earlier version of BIND which did not support the NZD format), use the new command **named-nzd2nzf** [RT #39837]

- Added server-side support for pipelined TCP queries. Clients may continue sending queries via TCP while previous queries are processed in parallel. Responses are sent when they are ready, not necessarily in the order in which the queries were received.

  To revert to the former behavior for a particular client address or range of addresses, specify the address prefix in the "keep-response-order" option. To revert to the former behavior for all clients, use "keep-response-order { any; };".

- The new **mdig** command is a version of **dig** that sends multiple pipelined queries and then waits for responses, instead of sending one query and waiting the response before sending the next. [RT #38261]

- To enable better monitoring and troubleshooting of RFC 5011 trust anchor management, the new **rndc managed-keys** can be used to check status of trust anchors or to force keys to be refreshed. Also, the managed-keys data file now has easier-to-read comments. [RT #38458]

- An **--enable-querytrace** configure switch is now available to enable very verbose query trace logging. This option can only be set at compile time. This option has a negative performance impact and should be used only for debugging. [RT #37520]

- A new **tcp-only** option can be specified in **server** statements to force **named** to connect to the specified server via TCP. [RT #37800]

- The **nxdomain-redirect** option specifies a DNS namespace to use for NXDOMAIN redirection. When a recursive lookup returns NXDOMAIN, a second lookup is initiated with the specified name appended to the query name. This allows NXDOMAIN redirection data to be supplied by multiple zones configured on the server, or by recursive queries to other servers. (The older method, using a single **type redirect** zone, has better average performance but is less flexible.) [RT #37989]

- The following types have been implemented: CSYNC, NINFO, RKEY, SINK, TA, TALINK.

- A new **message-compression** option can be used to specify whether or not to use name compression when answering queries. Setting this to **no** results in larger responses, but reduces CPU consumption and may improve throughput. The default is **yes**.

- A **read-only** option is now available in the **controls** statement to grant non-destructive control channel access. In such cases, a restricted set of **rndc** commands are allowed, which can report information from **named**, but cannot reconfigure or stop the server. By default, the control channel access is *not* restricted to these read-only operations. [RT #40498]

- When loading a signed zone, **named** will now check whether an RRSIG's inception time is in the future, and if so, it will regenerate the RRSIG immediately. This helps when a system's clock needs to be reset backwards.

- The new **minimal-any** option reduces the size of answers to UDP queries for type ANY by implementing one of the strategies in "draft-ietf-dnsop-refuse-any": returning a single arbitrarily-selected RRset that matches the query name rather than returning all of the matching RRsets. Thanks to Tony Finch for the contribution. [RT #41615]

- **named** now provides feedback to the owners of zones which have trust anchors configured (**trusted-keys**, **managed-keys**, **dnssec-validation auto;** and **dnssec-lookaside auto;**) by sending a daily query which encodes the keyids of the configured trust anchors for the zone. This is controlled by **trust-anchor-telemetry** and defaults to yes.

Feature Changes

- The logging format used for **querylog** has been altered. It now includes an additional field indicating the address in memory of the client object processing the query.

  The ISC DNSSEC Lookaside Validation (DLV) service is scheduled to be disabled in 2017. A warning is now logged when **named** is configured to use this service, either explicitly or via `dnssec-lookaside auto;`. [RT #42207]

- The timers returned by the statistics channel (indicating current time, server boot time, and most recent reconfiguration time) are now reported with millisecond accuracy. [RT #40082]

- Updated the compiled-in addresses for H.ROOT-SERVERS.NET and L.ROOT-SERVERS.NET.

- ACLs containing **geoip asnum** elements were not correctly matched unless the full organization name was specified in the ACL (as in **geoip asnum "AS1234 Example, Inc.";**). They can now match against the AS number alone (as in **geoip asnum "AS1234";**).

- When using native PKCS#11 cryptography (i.e., **configure --enable-native-pkcs11**) HSM PINs of up to 256 characters can now be used.

- NXDOMAIN responses to queries of type DS are now cached separately from those for other types. This helps when using "grafted" zones of type forward, for which the parent zone does not contain a delegation, such as local top-level domains. Previously a query of type DS for such a zone could cause the zone apex to be cached as NXDOMAIN, blocking all subsequent queries. (Note: This change is only helpful when DNSSEC validation is not enabled. "Grafted" zones without a delegation in the parent are not a recommended configuration.)

- Update forwarding performance has been improved by allowing a single TCP connection to be shared between multiple updates.

- By default, **nsupdate** will now check the correctness of hostnames when adding records of type A, AAAA, MX, SOA, NS, SRV or PTR. This behavior can be disabled with **check-names no**.

- Added support for OPENPGPKEY type.

- The names of the files used to store managed keys and added zones for each view are no longer based on the SHA256 hash of the view name, except when this is necessary because the view name contains characters that would be incompatible with use as a file name. For views whose names do not contain forward slashes ('/'), backslashes ('\'), or capital letters - which could potentially cause namespace collision problems on case-insensitive filesystems - files will now be named after the view (for example, `internal.mkeys` or `external.nzf`). However, to ensure consistent behavior when upgrading, if a file using the old name format is found to exist, it will continue to be used.

- "rndc" can now return text output of arbitrary size to the caller. (Prior to this, certain commands such as "rndc tsig-list" and "rndc zonestatus" could return truncated output.)

- Errors reported when running **rndc addzone** (e.g., when a zone file cannot be loaded) have been clarified to make it easier to diagnose problems.

- When encountering an authoritative name server whose name is an alias pointing to another name, the resolver treats this as an error and skips to the next server. Previously this happened silently; now the error will be logged to the newly-created "cname" log category.

- If **named** is not configured to validate answers, then allow fallback to plain DNS on timeout even when we know the server supports EDNS. This will allow the server to potentially resolve signed queries when TCP is being blocked.

- Large inline-signing changes should be less disruptive. Signature generation is now done incrementally; the number of signatures to be generated in each quantum is controlled by "sig-signing-signatures *number*;". [RT #37927]

- The experimental SIT option (code point 65001) of BIND 9.10.0 through BIND 9.10.2 has been replaced with the COOKIE option (code point 10). It is no longer experimental, and is sent by default, by both **named** and **dig**.

  The SIT-related named.conf options have been marked as obsolete, and are otherwise ignored.

- When **dig** receives a truncated (TC=1) response or a BADCOOKIE response code from a server, it will automatically retry the query using the server COOKIE that was returned by the server in its initial response. [RT #39047]

- Retrieving the local port range from net.ipv4.ip_local_port_range on Linux is now supported.

- A new `nsip-wait-recurse` directive has been added to RPZ, specifying whether to look up unknown name server IP addresses and wait for a response before applying RPZ-NSIP rules. The default is **`yes`**. If set to **`no`**, **named** will only apply RPZ-NSIP rules to servers whose addresses are already cached. The addresses will be looked up in the background so the rule can be applied on subsequent queries. This improves performance when the cache is cold, at the cost of temporary imprecision in applying policy directives. [RT #35009]

- Within the `response-policy` option, it is now possible to configure RPZ rewrite logging on a per-zone basis using the `log` clause.

- The default preferred glue is now the address type of the transport the query was received over.

- On machines with 2 or more processors (CPU), the default value for the number of UDP listeners has been changed to the number of detected processors minus one.

- Zone transfers now use smaller message sizes to improve message compression. This results in reduced network usage.

- Added support for the AVC resource record type (Application Visibility and Control).

  Changed **rndc reconfig** behavior so that newly added zones are loaded asynchronously and the loading does not block the server.

- **minimal-responses** now takes two new arguments: `no-auth` suppresses populating the authority section but not the additional section; `no-auth-recursive` does the same but only when answering recursive queries.

- At server startup time, the queues for processing notify and zone refresh queries are now processed in LIFO rather than FIFO order, to speed up loading of newly added zones. [RT #42825]

- When answering queries of type MX or SRV, TLSA records for the target name are now included in the additional section to speed up DANE processing. [RT #42894]

- **named** can now use the TCP Fast Open mechanism on the server side, if supported by the local operating system. [RT #42866]

Bug Fixes

- Fixed a crash when calling **rndc stats** on some Windows builds: some Visual Studio compilers generate code that crashes when the "%z" printf() format specifier is used. [RT #42380]

- Windows installs were failing due to triggering UAC without the installation binary being signed.

- A change in the internal binary representation of the RBT database node structure enabled a race condition to occur (especially when BIND was built with certain compilers or optimizer settings), leading to inconsistent database state which caused random assertion failures. [RT #42380]

End of Life

BIND 9.11 (Extended Support Version) will be supported until at least December, 2021.

See https://kb.isc.org/docs/aa-00896 for details of ISC's software support policy.

Thank You

Thank you to everyone who assisted us in making this release possible.

B A Brief History of the DNS and BIND

Although the Domain Name System "officially" began in 1984 with the publication of RFC 920, the core of the new system was described in 1983 in RFCs 882 and 883. From 1984 to 1987, the ARPAnet (the precursor to today's Internet) became a testbed of experimentation for developing the new naming/addressing scheme in a rapidly expanding, operational network environment. New RFCs were written and published in 1987 that modified the original documents to incorporate improvements based on the working model. RFC 1034, "Domain Names-Concepts and Facilities," and RFC 1035, "Domain Names-Implementation and Specification," were published and became the standards upon which all DNS implementations are built.

The first working domain name server, called "Jeeves," was written in 1983-84 by Paul Mockapetris for operation on DEC Tops-20 machines located at the University of Southern California's Information Sciences Institute (USC-ISI) and SRI International's Network Information Center (SRI-NIC). A DNS server for Unix machines, the Berkeley Internet Name Domain (BIND) package, was written soon after by a group of graduate students at the University of California at Berkeley under a grant from the US Defense Advanced Research Projects Administration (DARPA).

Versions of BIND through 4.8.3 were maintained by the Computer Systems Research Group (CSRG) at UC Berkeley. Douglas Terry, Mark Painter, David Riggle, and Songnian Zhou made up the initial BIND project team. After that, additional work on the software package was done by Ralph Campbell. Kevin Dunlap, a Digital Equipment Corporation employee on loan to the CSRG, worked on BIND for 2 years, from 1985 to 1987. Many other people also contributed to BIND development during that time: Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss, Jim Bloom and Mike Schwartz. BIND maintenance was subsequently handled by Mike Karels and Øivind Kure.

BIND versions 4.9 and 4.9.1 were released by Digital Equipment Corporation (which became Compaq Computer Corporation and eventually merged with Hewlett-Packard). Paul Vixie, then a DEC employee, became BIND's primary caretaker. He was assisted by Phil Almquist, Robert Elz, Alan Barrett, Paul Albitz, Bryan Beecher, Andrew Partan, Andy Cherenson, Tom Limoncelli, Berthold Paffrath, Fuat Baran, Anant Kumar, Art Harkin, Win Treese, Don Lewis, Christophe Wolfhugel, and others.

In 1994, BIND version 4.9.2 was sponsored by Vixie Enterprises. Paul Vixie became BIND's principal architect/programmer.

BIND versions from 4.9.3 onward have been developed and maintained by Internet Systems Consortium and its predecessor, the Internet Software Consortium, with support provided by ISC's sponsors.

As co-architects/programmers, Bob Halley and Paul Vixie released the first production-ready version of BIND version 8 in May 1997.

BIND version 9 was released in September 2000 and is a major rewrite of nearly all aspects of the underlying BIND architecture.

BIND versions 4 and 8 are officially deprecated. No additional development is done on BIND version 4 or BIND version 8.

BIND development work is made possible today by the sponsorship of corporations who purchase professional support services from ISC (https://www.isc.org/contact/) and/or donate to our mission, and by the tireless efforts of numerous individuals.

C General DNS Reference Information

C.1 IPv6 ADDRESSES (AAAA)

IPv6 addresses are 128-bit identifiers, for interfaces and sets of interfaces, which were introduced in the DNS to facilitate scalable Internet routing. There are three types of addresses: *Unicast*, an identifier for a single interface; *Anycast*, an identifier for a set of interfaces; and *Multicast*, an identifier for a set of interfaces. Here we describe the global Unicast address scheme. For more information, see RFC 3587, "IPv6 Global Unicast Address Format."

IPv6 unicast addresses consist of a *global routing prefix*, a *subnet identifier*, and an *interface identifier*.

The global routing prefix is provided by the upstream provider or ISP, and roughly corresponds to the IPv4 *network* section of the address range. The subnet identifier is for local subnetting, much like subnetting an IPv4 /16 network into /24 subnets. The interface identifier is the address of an individual interface on a given network; in IPv6, addresses belong to interfaces rather than to machines.

The subnetting capability of IPv6 is much more flexible than that of IPv4; subnetting can be carried out on bit boundaries, in much the same way as Classless InterDomain Routing (CIDR), and the DNS PTR representation ("nibble" format) makes setting up reverse zones easier.

The interface identifier must be unique on the local link, and is usually generated automatically by the IPv6 implementation, although it is usually possible to override the default setting if necessary. A typical IPv6 address might look like: **2001:db8:201:9:a00:20ff:fe81:2b32**.

IPv6 address specifications often contain long strings of zeros, so the architects have included a shorthand for specifying them. The double colon ("::") indicates the longest possible string of zeros that can fit, and can be used only once in an address.

C.2 BIBLIOGRAPHY (AND SUGGESTED READING)

Requests for Comments (RFCs)

Specification documents for the Internet protocol suite, including the DNS, are published as part of the Request for Comments (RFCs) series of technical notes. The standards themselves are defined by the Internet Engineering Task Force (IETF) and the Internet Engineering Steering Group (IESG). RFCs can be obtained online at:

https://datatracker.ietf.org/doc/

Standards

[RFC1034] P.V. Mockapetris, *Domain Names --- Concepts and Facilities*, November 1987.

[RFC1035] P. V. Mockapetris, *Domain Names --- Implementation and Specification*, November 1987.

[RFC974] C. Partridge, *Mail Routing and the Domain System*, January 1986.

Proposed Standards

[RFC1995] M. Ohta, *Incremental Zone Transfer in DNS*, August 1996.

[RFC1996] P. Vixie, *A Mechanism for Prompt Notification of Zone Changes*, August 1996.

[RFC2136] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, *Dynamic Updates in the Domain Name System*, April 1997.

[RFC2181] R., R. Bush Elz, *Clarifications to the DNS Specification*, July 1997.

[RFC2308] M. Andrews, *Negative Caching of DNS Queries*, March 1998.

[RFC2671] P. Vixie, *Extension Mechanisms for DNS (EDNS0)*, August 1997.

[RFC2672] M. Crawford, *Non-Terminal DNS Name Redirection*, August 1999.

[RFC2845] P. Vixie, O. Gudmundsson, D. Eastlake, 3rd, and B. Wellington, *Secret Key Transaction Authentication for DNS (TSIG)*, May 2000.

[RFC2930] D. Eastlake, 3rd, *Secret Key Establishment for DNS (TKEY RR)*, September 2000.

[RFC2931] D. Eastlake, 3rd, *DNS Request and Transaction Signatures (SIG(0)s)*, September 2000.

[RFC3007] B. Wellington, *Secure Domain Name System (DNS) Dynamic Update*, November 2000.

[RFC3645] S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, and R. Hall, *Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)*, October 2003.

DNS Security Proposed Standards

[RFC3225] D. Conrad, *Indicating Resolver Support of DNSSEC*, December 2001.

[RFC3833] D. Atkins and R. Austein, *Threat Analysis of the Domain Name System (DNS)*, August 2004.

[RFC4033] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS Security Introduction and Requirements*, March 2005.

[RFC4034] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *Resource Records for the DNS Security Extensions*, March 2005.

[RFC4035] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *Protocol Modifications for the DNS Security Extensions*, March 2005.

Other Important RFCs About DNS Implementation

[RFC1535] E. Gavron, *A Security Problem and Proposed Correction With Widely Deployed DNS Software*, October 1993.

[RFC1536] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller, *Common DNS Implementation Errors and Suggested Fixes*, October 1993.

[RFC1982] R. Elz and R. Bush, *Serial Number Arithmetic*, August 1996.

[RFC4074] Y. Morishita and T. Jinmei, *Common Misbehaviour Against DNS Queries for IPv6 Addresses*, May 2005.

Resource Record Types

[RFC1183] C.F. Everhart, L. A. Mamakos, R. Ullmann, and P. Mockapetris, *New DNS RR Definitions*, October 1990.

[RFC1706] B. Manning and R. Colella, *DNS NSAP Resource Records*, October 1994.

[RFC1876] C. Davis, P. Vixie, T., and I. Dickinson, *A Means for Expressing Location Information in the Domain Name System*, January 1996.

[RFC2052] A. Gulbrandsen and P. Vixie, *A DNS RR for Specifying the Location of Services*, October 1996.

[RFC2163] A. Allocchio, *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping*, January 1998.

[RFC2168] R. Daniel and M. Mealling, *Resolution of Uniform Resource Identifiers using the Domain Name System*, June 1997.

[RFC2230] R. Atkinson, *Key Exchange Delegation Record for the DNS*, October 1997.

[RFC2536] D. Eastlake, 3rd, *DSA KEYs and SIGs in the Domain Name System (DNS)*, March 1999.

[RFC2537] D. Eastlake, 3rd, *RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)*, March 1999.

[RFC2538] D. Eastlake, 3rd and O. Gudmundsson, *Storing Certificates in the Domain Name System (DNS)*, March 1999.

[RFC2539] D. Eastlake, 3rd, *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)*, March 1999.

[RFC2540] D. Eastlake, 3rd, *Detached Domain Name System (DNS) Information*, March 1999.

[RFC2782] A. GulbrandsenP. VixieL. Esibov, *A DNS RR for specifying the location of services (DNS SRV)*, February 2000.

[RFC2915] M. MeallingR. Daniel, *The Naming Authority Pointer (NAPTR) DNS Resource Record*, September 2000.

[RFC3110] D. Eastlake, 3rd, *RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)*, May 2001.

[RFC3123] P. Koch, *A DNS RR Type for Lists of Address Prefixes (APL RR)*, June 2001.

[RFC3596] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi, *DNS Extensions to support IP version 6*, October 2003.

[RFC3597] A. Gustafsson, *Handling of Unknown DNS Resource Record (RR) Types*, September 2003.

DNS and the Internet

[RFC1101] P. V. Mockapetris, *DNS Encoding of Network Names and Other Types*, April 1989.

[RFC1123] Braden, *Requirements for Internet Hosts - Application and Support*, October 1989.

[RFC1591] J. Postel, *Domain Name System Structure and Delegation*, March 1994.

[RFC2317] H. Eidnes, G. de Groot, and P. Vixie, *Classless IN-ADDR.ARPA Delegation*, March 1998.

[RFC2826] Internet Architecture Board, *IAB Technical Comment on the Unique DNS Root*, May 2000.

[RFC2929] D. Eastlake, 3rd, E. Brunner-Williams, and B. Manning, *Domain Name System (DNS) IANA Considerations*, September 2000.

DNS Operations

[RFC1033] M. Lottor, *Domain administrators operations guide*, November 1987.

[RFC1537] P. Beertema, *Common DNS Data File Configuration Errors*, October 1993.

[RFC1912] D. Barr, *Common DNS Operational and Configuration Errors*, February 1996.

[RFC2010] B. Manning and P. Vixie, *Operational Criteria for Root Name Servers*, October 1996.

[RFC2219] M. Hamilton and R. Wright, *Use of DNS Aliases for Network Services*, October 1997.

[RFC8906] M. Andrews and R. Bellis, *A Common Operational Problem in DNS Servers: Failure to Communicate* , September 2020.

Internationalized Domain Names

[RFC2825] IAB and R. Daigle, *A Tangled Web: Issues of I18N, Domain Names, and the Other Internet protocols*, May 2000.

[RFC3490] P. Faltstrom, P. Hoffman, and A. Costello, *Internationalizing Domain Names in Applications (IDNA)*, March 2003.

[RFC3491] P. Hoffman and M. Blanchet, *Nameprep: A Stringprep Profile for Internationalized Domain Names*, March 2003.

[RFC3492] A. Costello, *Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*, March 2003.

Other DNS-related RFCs

[RFC1464] R. Rosenbaum, *Using the Domain Name System To Store Arbitrary String Attributes*, May 1993.

[RFC1713] A. Romao, *Tools for DNS Debugging*, November 1994.

[RFC1794] T. Brisco, *DNS Support for Load Balancing*, April 1995.

[RFC2240] O. Vaughan, *A Legal Basis for Domain Name Allocation*, November 1997.

[RFC2345] J. Klensin, T. Wolf, and G. Oglesby, *Domain Names and Company Name Retrieval*, May 1998.

[RFC2352] O. Vaughan, *A Convention For Using Legal Names as Domain Names*, May 1998.

[RFC3071] J. Klensin, *Reflections on the DNS, RFC 1591, and Categories of Domains*, February 2001.

[RFC3258] T. Hardie, *Distributing Authoritative Name Servers via Shared Unicast Addresses*, April 2002.

[RFC3901] A. Durand and J. Ihren, *DNS IPv6 Transport Operational Guidelines*, September 2004.

Obsolete and Unimplemented Experimental RFC

[RFC1712] C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni, *DNS Encoding of Geographical Location*, November 1994.

[RFC2673] M. Crawford, *Binary Labels in the Domain Name System*, August 1999.

[RFC2874] M. Crawford and C. Huitema, *DNS Extensions to Support IPv6 Address Aggregation and Renumbering*, July 2000.

Obsoleted DNS Security RFCs

[RFC2065] D. Eastlake, 3rd and C. Kaufman, *Domain Name System Security Extensions*, January 1997.

[RFC2137] D. Eastlake, 3rd, *Secure Domain Name System Dynamic Update*, April 1997.

[RFC2535] D. Eastlake, 3rd, *Domain Name System Security Extensions*, March 1999.

[RFC3008] B. Wellington, *Domain Name System Security (DNSSEC) Signing Authority*, November 2000.

[RFC3090] E. Lewis, *DNS Security Extension Clarification on Zone Status*, March 2001.

[RFC3445] D. Massey and S. Rose, *Limiting the Scope of the KEY Resource Record (RR)*, December 2002.

[RFC3655] B. Wellington and O. Gudmundsson, *Redefinition of DNS Authenticated Data (AD) bit*, November 2003.

[RFC3658] O. Gudmundsson, *Delegation Signer (DS) Resource Record (RR)*, December 2003.

[RFC3755] S. Weiler, *Legacy Resolver Compatibility for Delegation Signer (DS)*, May 2004.

[RFC3757] O. Kolkman, J. Schlyter, and E. Lewis, *Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag*, April 2004.

[RFC3845] J. Schlyter, *DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format*, August 2004.

Internet Drafts

Internet Drafts (IDs) are rough-draft working documents of the Internet Engineering Task Force. They are, in essence, RFCs in the preliminary stages of development. Implementors are cautioned not to regard IDs as archival, and they should not be quoted or cited in any formal documents unless accompanied by the disclaimer that they are "works in progress." IDs have a lifespan of six months after which they are deleted unless updated by their authors.

Other Documents About BIND

[1] Paul Albitz and Cricket Liu, *DNS and BIND*, Copyright © 1998 Sebastopol, CA: O'Reilly and Associates.

D BIND 9 DNS Library Support

D.1 BIND 9 DNS Library Support

This version of BIND 9 "exports" its internal libraries so that they can be used by third-party applications more easily (we call them "export" libraries in this document). Certain library functions are altered from specific BIND-only behavior to more generic behavior when used by other applications; to enable this generic behavior, the calling program initializes the libraries by calling **isc_lib_register()**.

In addition to DNS-related APIs that are used within BIND 9, the libraries provide the following features:

- The "DNS client" module. This is a higher-level API that provides an interface to name resolution, single DNS transaction with a particular server, and dynamic update. Regarding name resolution, it supports advanced features such as DNSSEC validation and caching. This module supports both synchronous and asynchronous mode.
- The "IRS" (Information Retrieval System) library. It provides an interface to parse the traditional `resolv.conf` file and more advanced, DNS-specific configuration file for the rest of this package (see the description for the `dns.conf` file below).
- As part of the IRS library, the standard address-name mapping functions, **getaddrinfo()** and **getnameinfo()**, are provided. They use the DNSSEC-aware validating resolver backend, and could use other advanced features of the BIND 9 libraries such as caching. The **getaddrinfo()** function resolves both A and AAAA RRs concurrently when the address family is unspecified.
- An experimental framework to support other event libraries than BIND 9's internal event task system.

Installation

```
$ make install
```

Normal installation of BIND also installs library object and header files. Root privilege is normally required.

To see how to build a custom application after the installation, see `lib/samples/Makefile-postinstall.in`.

Known Defects/Restrictions

- The "fixed" RRset order is not (currently) supported in the export library. To use "fixed" RRset order for, e.g., **named** while still building the export library even without the fixed-order support, build them separately:

```
$ ./configure --enable-fixed-rrset [other flags, but not --enable-exportlib]
$ make
$ ./configure --enable-exportlib [other flags, but not --enable-fixed-rrset]
$ cd lib/export
$ make
```

- RFC 5011 is not supported in the validating stub resolver of the export library. In fact, it is not clear whether it should be: trust anchors would be a system-wide configuration which would be managed by an administrator, while the stub resolver is used by ordinary applications run by a normal user.
- Not all common `/etc/resolv.conf` options are supported in the IRS library. The only available options in this version are **debug** and **ndots**.

The dns.conf File

The IRS library supports an "advanced" configuration file related to the DNS library, for configuration parameters that would be beyond the capability of the `resolv.conf` file. Specifically, it is intended to provide DNSSEC-related configuration parameters. By default the path to this configuration file is `/etc/dns.conf`. This module is very experimental and the configuration syntax or library interfaces may change in future versions. Currently, only the **trusted-keys** statement is supported, whose syntax is the same as the same statement in `named.conf`. (See Section 6.2 for details.)

Sample Applications

Some sample application programs using this API are provided for reference. The following is a brief description of these applications.

sample: a simple stub resolver utility

This sends a query of a given name (of a given optional RR type) to a specified recursive server and prints the result as a list of RRs. It can also act as a validating stub resolver if a trust anchor is given via a set of command-line options.

Usage: sample [options] server_address hostname

Options and Arguments:

-t RRtype
specifies the RR type of the query. The default is the A RR.

[-a algorithm] [-e] -k keyname -K keystring
specifies a command-line DNS key to validate the answer. For example, to specify the following DNSKEY of example.com:

```
example.com. 3600 IN DNSKEY 257 3 5 xxx
```

specify the options as follows:

```
-e -k example.com -K "xxx"
```

-e means that this key is a zone's "key signing key" (also known as "secure entry point"). When -a is omitted rsasha1 is used by default.

-s domain:alt_server_address
specifies a separate recursive server address for the specific "domain". Example: -s example.com:2001:db8::1234

server_address
is an IP(v4/v6) address of the recursive server to which queries are sent.

hostname
is the domain name for the query

sample-async: a simple stub resolver, working asynchronously

This is similar to "sample", but accepts a list of (query) domain names as a separate file and resolves the names asynchronously.

Usage: sample-async [-s server_address] [-t RR_type] input_file

Options and Arguments:

-s server_address
is an IPv4 address of the recursive server to which queries are sent. (IPv6 addresses are not supported in this implementation.)

-t RR_type
specifies the RR type of the queries. The default is the A RR.

input_file
is a list of domain names to be resolved; each line consists of a single domain name. For example:

```
www.example.com
mx.example.net
ns.xxx.example
```

sample-request: a simple DNS transaction client

sends a query to a specified server, and prints the response with minimal processing. It does not act as a "stub resolver": it stops the processing once it gets any response from the server, whether it's a referral or an alias (CNAME or DNAME) that would require further queries to get the ultimate answer. In other words, this utility acts as a very simplified **dig**.

Usage: sample-request [-t RRtype] server_address hostname

Options and Arguments:

-t RRtype
specifies the RR type of the queries. The default is the A RR.

server_address
is an IP(v4/v6) address of the recursive server to which the query is sent.

hostname
is the domain name for the query

sample-gai: getaddrinfo() and getnameinfo() test code

is a test program to check **getaddrinfo()** and **getnameinfo()** behavior. It takes a host name as an argument, calls **getaddrinfo()** with the given host name, and calls **getnameinfo()** with the resulting IP addresses returned by **getaddrinfo()**. If the dns.conf file exists and defines a trust anchor, the underlying resolver acts as a validating resolver, and **getaddrinfo()**/**getnameinfo()** fails with an EAI_INSECUREDATA error when DNSSEC validation fails.

Usage: sample-gai hostname

sample-update: a simple dynamic update client program

accepts a single update command as a command-line argument, sends an update request message to the authoritative server, and shows the response from the server. In other words, this is a simplified **nsupdate**.

Usage: sample-update [options] (add | delete) "update data"

Options and Arguments:

-a auth_server
is an IP address of the authoritative server that has authority for the zone containing the update name. This should normally be the primary authoritative server that accepts dynamic updates. It can also be a secondary server that is configured to forward update requests to the primary server.

-k keyfile
is a TSIG key file to secure the update transaction. The keyfile format is the same as that for the nsupdate utility.

-p prerequisite
is a prerequisite for the update; only one prerequisite can be specified. The prerequisite format is the same as that accepted by the nsupdate utility.

-r recursive_server
is an IP address of a recursive server that this utility uses. A recursive server may be necessary to identify the authoritative server address to which the update request is sent.

-z zonename
is the domain name of the zone that it contains.

(add | delete)
specifies the type of update operation. Either "add" or "delete" must be specified.

"update data"
specifies the data to be updated. A typical example of the data looks like "name TTL RRtype RDATA".

NOTE

In practice, either -a or -r must be specified. Others can be optional; the underlying library routine tries to identify the appropriate server and the zone name for the update.

Examples: assuming the primary authoritative server of the dynamic.example.com zone has an IPv6 address 2001:db8::1234,

```
$ sample-update -a sample-update -k Kxxx.+nnn+mmmm.key add "foo.dynamic. ↩
    example.com 30 IN A 192.168.2.1"
```

adds an A RR for foo.dynamic.example.com using the given key.

```
$ sample-update -a sample-update -k Kxxx.+nnn+mmmm.key delete "foo.dynamic ↩
    .example.com 30 IN A"
```

removes all A RRs for foo.dynamic.example.com using the given key.

```
$ sample-update -a sample-update -k Kxxx.+nnn+mmmm.key delete "foo.dynamic ↩
    .example.com"
```

removes all RRs for foo.dynamic.example.com using the given key.

nsprobe: domain/name server checker in terms of RFC 4074

checks a set of domains to ensure the name servers of the domains behave correctly in terms of RFC 4074. This is included in the set of sample programs to show how the export library can be used in a DNS-related application.

Usage: nsprobe [-d] [-v [-v...]] [-c cache_address] [input_file]

Options

-d
runs in "debug" mode. With this option, nsprobe dumps every RR it receives.

-v
increases verbosity of other normal log messages. This can be specified multiple times.

-c cache_address
specifies an IP address of a recursive (caching) name server. nsprobe uses this server to get the NS RRset of each domain and the A and/or AAAA RRsets for the name servers. The default value is 127.0.0.1.

input_file
is a file name containing a list of domain (zone) names to be probed. when omitted the standard input is used. Each line of the input file specifies a single domain name, such as "example.com". In general, this domain name must be the apex name of some DNS zone, unlike normal "host names" such as "www.example.com". nsprobe first identifies the NS RRsets for the given domain name, and sends A and AAAA queries to these servers for some widely used names under the zone; specifically, adding "www" and "ftp" to the zone name.

Library References

As of this writing, there is no formal "manual" for the libraries, except this document, header files (some of which provide pretty detailed explanations), and sample application programs.