Securing an Internet Name Server

CERT® Coordination Center

Allen Householder, CERT/CC
Brian King, CERT/CC

In collaboration with
Ken Silva, Verisign

Based in part on a presentation originally created by
Cricket Liu

August 2002

DNS overview

Domain name system (DNS) servers translate names suitable for use by people (such as www.example.com) into network addresses (e.g., 192.168.4.22) suitable for use by computers. There are a number of different name server software packages available today. Berkeley Internet Name Domain (BIND), produced by the Internet Software Consortium (http://www.isc.org), is the most widely deployed name server package, and is available on a wide variety of platforms. Other popular DNS packages include Microsoft DNS and djbdns.

The goal of this document is to discuss general name server security. However, in order to provide useful examples we have chosen to focus on BIND since it is the most commonly used software for DNS servers.

Risks to name servers

Name servers exposed to the Internet are subject to a wide variety of attacks:

- Attacks against the name server software may allow an intruder to compromise the server and take control of the host. This often leads to further compromise of the network.
- Denial of service attacks, even one directed at a single DNS server, may affect an entire network by preventing users from translating hostnames into the necessary IP addresses.
- Spoofing attacks that try to induce your name server to cache false resource records, and could lead unsuspecting users to unsavory sites.
- Information leakage from a seemingly innocent zone transfer could expose internal network topology information that can be used to plan further attacks.
- A name server could even be an unwitting participant in attacks on other sites.

While it is important for network administrators to secure any host connected to the Internet, they must give name servers special consideration due to the important role they play. The purpose of this document is to outline some common steps that can be taken to secure an Internet Name Server from various types of attacks.

Run a new version of your name server software

As with any piece of software, name server software evolves with each release. Virtually all older name servers have widely known vulnerabilities that can be exploited. Vulnerabilities that appear in one version are usually fixed in subsequent releases. While running the newest version of your name server software doesn't guarantee your server's security, it minimizes the possibility of exploitation. When upgrading, be sure to read the release notes to understand what has been fixed and what has been changed in the software. Upgrading the name server to a new version may require that changes be made to the configuration in order to provide the expected performance or to take advantage of new features.

At the time of this writing, the latest versions of BIND are:

- 9.2.0 (recommended) http://www.isc.org/products/BIND/bind9.html
- 8.3.1 http://www.isc.org/products/BIND/bind8.html
- 4.9.8 (deprecated) http://www.isc.org/products/BIND/bind4.html

Information on vulnerabilities in BIND, and which versions are affected, can be found at the ISC's BIND Vulnerabilities page

http://www.isc.org/products/BIND/bind-security.html

or by searching the CERT/CC's Vulnerability Notes Database

http://www.kb.cert.org/vuls/

Eliminate single points of failure

Denial of service attacks targeted at a name server, or an accidental network outage isolating the name server from the rest of the network can have far reaching effects. If the name servers are unreachable, users will be unable to reach other hosts since they will be unable to translate hostnames into IP addresses. This can affect both internal and public systems depending on the network architecture.

To combat denial of service attacks and prevent accidental service outages, it is necessary to eliminate single points of failure in the DNS infrastructure. Some common pitfalls to avoid are placing all of your name servers

- on a single subnet.
- behind a single router.
- behind a single leased line.
- within a single autonomous system.

In addition to mitigating the risks described above, you should ensure that you have multiple physical paths to your network. This is an important step to verify even if you have multiple service providers, since they may lease their physical plant from the same carriers. Without physical path diversity, your network is more susceptible to physical disruption (commonly known as "backhoe fade").

Even if all the systems listed in a public zone were to become unreachable from the Internet (as might be the case if all links to a site were saturated by a DDoS attack), there are a number of reasons an organization may wish to arrange for an off-site slave name server. Among these are

- to facilitate troubleshooting from remote networks, whether done by your own staff via alternate Internet connections, or by other network administrators attempting to diagnose why their users are unable to reach your systems.
- to maintain proper performance of SMTP mail relaying. If external mail servers are completely unable to resolve your domain, they may bounce the messages back to the senders as undeliverable. However, if your domain is resolvable but all mail exchangers are unreachable, external mail servers will usually queue the messages for later delivery.

Use separate servers

A number of steps can be taken to secure a name server that only has to provide DNS to a single audience since it can be optimized for a particular function. Therefore, it can be useful to use separate name servers, each configured to play a specific role. Frequently, different security polices can be applied to these servers such that even if one server is compromised the other server will continue to function normally.

Consider creating two kinds of name server, each optimized for a particular function:

> *Advertising* name server
> An advertising name server would commonly be used as an external name server that is authoritative for your DNS zones. It would "advertise" this DNS information to the Internet. Since it should not be queried for zones for which it is not authoritative, it should be configured as a non-recursive server. Thus, the server would only provide resolution for the zones for which it has authoritative information.
>
> *Resolving* name server
> A resolving name server would commonly be used to provide name resolution services to internal clients. It may or may not be configured as authoritative for internal zones. Since it must find DNS information requested by internal hosts (regardless of whether it is authoritative for that information or not), it should be configured as a recursive server. However, it should only answer queries from trusted sources (internal hosts), not from the Internet.

By adopting this strategy the external advertising name server is configured to provide little service other than answering queries for which it is authoritative. The internal resolving name server must provide recursion, and is therefore somewhat more susceptible to attack since it must accept data from other DNS servers. Additional protection of the resolving name server can be provided via other means such as packet filtering and restricting the server to respond only to known hosts.

In this manner, if the resolving server were to be compromised or its cache "poisoned", the advertising server's authoritative zone information would be unaffected, thus limiting the potential damage. Similarly, if the resolving name servers are also configured to be authoritative for internal zones, a compromise of the advertising name server(s) would not affect the normal operation of the internal clients of the resolving name server(s).

Filter traffic to your name server

Many organizations run their name servers on dedicated hosts. If the hosts that run the name servers don't provide any other services, there is no need for them to respond to non-DNS traffic. As such, all unnecessary traffic can be filtered out, thus reducing the possibility of the name server being compromised by a vulnerability in some other piece of software. (In addition to filtering unused services, it is good security practice to disable or remove any unnecessary software from the name server.)

For DNS servers providing public name resolution to the Internet (i.e., the *advertising* name server described above), everything but traffic from the Internet to 53/udp and 53/tcp on the server can be safely filtered. Name servers that only provide name resolution services to internal clients can be filtered to allow only internal clients to access 53/udp and 53/tcp on the name server, while allowing the server to make outbound queries to other name servers.

Table 1 summarizes the filtering recommendations above.

Function	Description	Source IP	Source Port	Destination IP	Destination Port
Public NS	Inbound queries	Any	`53/udp`, `53/tcp`, `>1023/udp`, `>1023/tcp`	Nameserver	`53/udp`, `53/tcp`
Public NS	Query replies	Nameserver	`53/udp`, `53/tcp`	Any	`53/udp`, `53/tcp`, `>1023/udp`, `>1023/tcp`
Internal NS	Queries from clients	Internal network	`>1023/udp`, `>1023/tcp`	Nameserver	`53/udp`, `53/tcp`
Internal NS	Replies to clients	Nameserver	`53/udp`, `53/tcp`	Internal network	`>1023/udp`, `>1023/tcp`
Internal NS	Outbound recursive queries	Nameserver	`53/udp`*, `53/tcp`*, `>1023/udp`*, `>1023/tcp`*	Any	`53/udp`, `53/tcp`
Internal NS	Replies to recursive queries	Any	`53/udp`, `53/tcp`	Nameserver	`53/udp`*, `53/tcp`*, `>1023/udp`*, `>1023/tcp`*

Table 1: Packet Filtering for Name servers

Note that it is a common misconception that that only 53/udp is used for DNS queries and that 53/tcp is only used for zone transfers. However, 53/tcp may be used for longer query responses that

* Source ports `53/udp` and `53/tcp` are required if your resolving name server is running BIND version 4.x software. BIND versions 8.x and 9.x default to using source ports `>1023/udp` and `>1023/tcp` for queries originating from the server.

won't fit into a single UDP packet. Thus it is necessary to allow traffic to both 53/udp and 53/tcp. Zone transfers via 53/tcp can be further restricted within the name server configuration itself. Furthermore, name servers that are configured to forward unresolved queries to another name server will appear as clients to the server they forward to.

Restrict zone transfers

Zone transfers are used to transfer DNS information from one name server to another. Restricting zone transfers is an important step in securing a name server. It prevents others from taxing your name server's resources, and it prevents intruders from listing the contents of your zones. An intruder who is able to complete a zone transfer can use that information to identify new targets on your network, such as routers and network infrastructure equipment, mail servers, other name servers, file servers, and anything else in your DNS records.

A common mistake is to restrict zone transfers from the primary master name server while neglecting to restrict transfers from slave servers. Since it is just as easy to transfer a zone from a slave as it is from the primary master, it is important to ensure that all authoritative name servers (regardless of their master or slave status) have restrictions placed on zone transfers. If possible, the most secure configuration is to disable zone transfers altogether and use a cryptographically secure method of transferring zone files to other authoritative servers.

The *host -l* command and *dig*'s *axfr* option can be useful for testing zone transfer restrictions since those commands are implemented as zone transfers.

Restricting zone transfers with BIND 4

With BIND 4.9.x, use the *xfrnets* directive:

```
xfrnets 192.168.4.154&255.255.255.255
```

This controls which name servers can transfer *any* zones from this name server.

Restricting zone transfers with BIND 8

With BIND 8 or 9, use the *allow-transfer* substatement, either at the server level:

```
options {
      allow-transfer { 192.168.4.154; };
};
```

or specific to a zone:

```
type master;
zone "example.com" {
      file "db.example.com";
      allow-transfer { 192.168.4.154; };
};
```

Authenticate zone transfers with TSIG

With BIND 8.2 and later name servers, you can use Transaction Signatures, or TSIG, to cryptographically authenticate and verify zone data. TSIG uses a shared-secret cryptographic signature to authenticate authoritative zone data. Use of TSIG requires that you configure a shared key on your master and slave name servers and then instruct them to use the key to sign communication with each other.

TSIG requires participating name servers to have their system clocks synchronized.

Configuring TSIG with BIND 8.2 and later

The following example tells the master name server to allow zone transfer requests for the `example.com` zone from any name server, as long as they are signed with the *tsig-signing* key[1]:

```
key tsig-signing. {
      algorithm hmac-md5;
      secret "mZiMNOUYQPMNwsDzrX2ENw==";
};
zone "example.com" {
      type master;
      file "db.example.com";
      allow-transfer { key tsig-signing; };
};
```

Correspondingly, the slave name server(s) must also contain the shared key:

```
key tsig-signing. {
      algorithm hmac-md5;
      secret "mZiMNOUYQPMNwsDzrX2ENw==";
};
server 192.168.4.47 {
      transfer-format many-answers;
      keys { tsig-signing.; };
};
zone "example.com" {
      type slave;
      file "bak.example.com";
      allow-transfer { none; };
};
```

[1] RFC-2845 makes a number of recommendations regarding key usage. Among these recommendations are that key names should be indicative of the hosts using them, and that distinct keys should be used for each pair of hosts using TSIG.

Restrict dynamic updates

Although dynamic updates are useful as a way to reduce the overhead typically associated with static DNS zone maintenance, they pose additional risk to the integrity of your zone data. This risk arises from the fact that an authorized updater can

- add new records to a zone.
- delete any or all the records from a zone.
- modify any or all records in a zone.

As a result, administrators of servers allowing dynamic updates should take precautions to restrict the potential for abuse. Dynamic updates can be limited in the following ways:

- by individual IP addresses
- by a list of TSIG keys

As with any IP address-based authentication mechanism, it is important to remember that IP addresses are easily spoofed. Furthermore, dynamic updates typically use UDP as the transport protocol, so it may only require a single forged packet to cause data corruption. Sites that use IP addresses as their primary authentication mechanism for dynamic DNS updates are therefore encouraged to implement anti-spoofing mechanisms to reduce the risk of unauthorized data modifications. This anti-spoofing function is typically performed at a packet filtering point such as a router or firewall.

Restricting dynamic updates with BIND 8 and 9

Dynamic updates are implemented in BIND versions 8 and 9. By default, these features are disabled. Enabling dynamic updates is done using the allow-update substatement:

```
zone "example.com" {
      type master;
      file "db.example.com";
      allow-update {
            localhost;
            key allowed-updater.;
      };
};
```

The above example restricts dynamic updates for the example.com zone to localhost (as authenticated by IP address), and to any updater using the allowed-updater key[*]. Note, however, that any user with the `allowed-updater` key will be able to add, delete, or modify any record in the entire `example.com` zone. If dynamic updates are in fact required for the entire zone, this may be acceptable.

[*] The `allowed-updater` key would, of course, need to be defined elsewhere in the configuration.

In the case where dynamic updates are only required for a few entries within a zone, though, it may be more appropriate to limit the updates to those particular records. Delegating a new zone containing only the record you wish to allow updates to will accomplish this. For example, if the only record in the `example.com` zone requiring dynamic updates is `www.example.com`, you could delegate a new zone for `www.example.com`. To do this, in the `db.example.com` zone file, you would replace any records for `www.example.com` with

```
IN    NS    ns1.example.com.
IN    NS    ns2.example.com.
```

Then, in the `named.conf` file on the primary master name server for `www.example.com`, you would add

```
zone "www.example.com" {
      type master;
      file "db.www.example.com";
      allow-update {
            key allowed-www-updater;
      };
};
```

Given this configuration, a user with the `allowed-www-updater` key would only be able to modify or delete the address of `www.example.com`, or create new subdomains of `www.example.com`. They would not be able to otherwise modify any records in the `example.com` zone, though.

Restricting dynamic updates with BIND 9

In BIND 9, the update-policy substatement allows administrators to restrict which domain names in a zone can be dynamically updated to a subset of a zone. Thus, it is possible to achieve a result similar to the previous example, but with finer control over what can and cannot be updated. For example:

```
zone "example.com" {
      type master;
      file "db.example.com";
      update-policy {
            grant allowed-www-updater self www.example.com A;
      };
};
```

This allows a client with the `allowed-www-updater` key to update only the address record for `www.example.com` without granting the ability to create new subdomains or to modify other record types (e.g., MX) associated with `www.example.com`.

Protecting against cache-poisoning

Name servers can operate in one of two ways in response to queries received:

- *recursive* queries are used when a client makes a request to a name server and the name server is expected to traverse the DNS hierarchy to find the answer. The name server in turn makes *non-recursive* queries to find the requested information.
- *non-recursive* queries are used when a name server asks another name server for information. The queried server will return either (a) an answer, (b) a referral to another name server, or (c) an error indicating that the queried name server has no information to fulfill the request.

By default, most name servers will allow recursive queries from any source.

Name servers that provide recursive resolution services to the Internet at large may be susceptible to cache poisoning. Cache poisoning occurs when a name server caches bogus data for a domain name. This can result in denial-of-service or man-in-the-middle attacks. By making a recursive query to a name server that provides recursion, an attacker can cause a name server to look up and cache information contained in zones under their control. Thus the victim name server is made to query their malicious name servers, resulting in the victim caching and serving bogus data.

To reduce this risk, BIND administrators have four options, depending on the software version in use:

1. Disable recursion entirely, if possible (BIND 4.9 and up).
2. Restrict the addresses that can make queries (of any type) to the name server (BIND 8 and up).
3. Restrict the addresses that can make recursive queries to the name server (BIND 8.2.1 and up).
4. Disable the fetching of glue records, if possible (BIND versions prior to 9 only).

Turning off recursion (BIND 4.9 and later)

Disabling recursion puts your name servers into a passive mode, telling them never to send queries on behalf of other name servers or resolvers. A totally non-recursive name server is protected from cache poisoning, since it will only answer queries directed to it — it doesn't send queries, and hence doesn't cache any data. Disabling recursion can also prevent attackers from bouncing denial of services attacks off your name server by querying for external zones.

BIND 4.9 supports recursion disabling via the options directive.

```
options no-recursion
```

In BIND 8 and 9, the same feature is disabled using the options statement:

```
options {
      recursion no;
};
```

Restricting queries using `allow-query` (BIND 8 and later)

As mentioned above, it is recommended that sites use separate name servers for (1) providing authoritative responses for their zones (*advertising*), and (2) providing recursive name resolution services to their internal systems. This allows for recursion to be completely disabled on the authoritative advertising name servers, while still providing the necessary recursive services to internal resolvers. This may not be practical in all environments, though, since one or more of the following may be true

- other name servers may forward unresolved queries to the name server in question for recursive resolution
- client resolvers may require recursive services of them.

Hence BIND versions 8 and 9 offer a variety of methods to restrict queries. (Unfortunately, BIND 4 does not offer these features.) The basic options allow the administrator to specify

- the addresses that should be allowed to make queries
- the zones those addresses should be permitted to query for

In an environment where both authoritative responses and recursive queries are required, it is commonly the case that while queries for records in authoritative zones can come from anywhere, queries for records outside of authoritative zones (i.e., recursive queries from clients) should only come from internal addresses.

For example, if your internal network uses the `192.168.4.0/24` address block but you need to provide authoritative responses for the example.com zone, you could use the `allow-query` substatement to restrict recursive services to that block while allowing any address to make requests in the `example.com` zone. The first step is to define an access list describing your internal network block:

```
acl internal { 192.168.4.0/24; };
```

Alternatively, if the name server is only to be used by other internal name servers as a local cache, the access list could be defined to include only those name servers that are permitted to use its services. In our example, the name servers to be permitted are at `192.168.4.154` and `192.168.4.47`:

```
acl internal { 192.168.4.154/32; 192.168.4.47/32; };
```

Next, make the server default to only providing service to the internal network:

```
options {
      ...
      allow-query { internal; };
      ...
};
```

Finally, override the default to allow queries from anywhere only for records in the `example.com` zone:

```
zone "example.com" {
      type master;
      file "db.example.com";
      allow-query { any; };
      ...
};
```

Restricting recursion using `allow-recursion` (BIND 8.2.1 and later)

Furthermore, in those configurations in which it is not possible to completely disable recursion on the name server, it is recommended that the name server be restricted to providing recursive resolution only to a limited set of addresses. BIND 8.2.1 and later allow you to restrict the IP addresses you accept recursive queries from. When this option is used, queries from other IP addresses will be processed as non-recursive, regardless of whether recursion was requested or not. This feature is activated through the use of the `allow-recursion` substatement.

Similar to our example from above, we first define the internal network block in an access list:

```
acl internal { 192.168.4.0/24; };
```

Then set the server default to allow recursion from internal addresses only:

```
options {
      ...
      allow-recursion { internal; };
      ...
};
```

This time, however, no special configuration is required in the `example.com` domain:

```
zone "example.com" {
      type master;
      file "db.example.com";
      ...
};
```

Restricting recursion using views (BIND 9 and later)

BIND 9 implements a new feature through which administrators can define different "views" for the data served by a name server. Using views, other name servers and resolver clients may see different responses based on their IP addresses.

In the following example, the `internalview` view allows recursion addresses defined in the `internal` access list, while the `externalview` view does not permit recursion:

```
view "internalview" {
      match-clients { internal; };
      recursion yes;
};
view "externalview" {
      match-clients { any; };
      recursion no;
};
```

Note that in this case we have not defined any zones within the views, so if the `example.com` domain were defined elsewhere in the `named.conf` file:

```
zone "example.com" {
      type master;
      file "db.example.com";
};
```

both the `internalview` and `externalview` views would see the same data for the `example.com` zone.

Turning off glue fetching (BIND 4.9 and 8 only)

If a name server is returning NS records in a response but does not have the corresponding A records to include as additional information, it may attempt to retrieve them. This glue fetching is the default behavior in BIND versions prior to 9, and is a potential source of cache poisoning. By disabling glue fetching it is possible to prevent this lookup, thereby eliminating the possibility that the name server might cache bogus A records retrieved in this manner.

To disable glue fetching, use the `no-fetch-glue` directive in BIND 4.9:

```
options no-fetch-glue
```

or the `fetch-glue` statement in BIND 8:

```
options {
      ...
      fetch-glue no;
      ...
};
```

Although BIND 9 allows the same syntax as BIND 8, the statement is obsolete since BIND 9 will not fetch glue under any circumstances.

More protection against spoofing (BIND 8)

BIND uses message IDs to identify incoming responses that were the result of outgoing queries. It is possible, though, that an attacker could predict the message ID of a query and spoof the response, thereby providing bogus information to the name server. BIND 8 allows administrators to randomize the message IDs used through the `use-id-pool` directive, thereby making it more difficult to spoof replies:

```
options {
        ...
        use-id-pool yes;
        ...
};
```

This option is obsolete in BIND 9 since all BIND 9 message IDs are randomized.

Run your name server as a user other than root (BIND 8.1.2 and up)

To ensure that the exploitation of a vulnerability in BIND doesn't give an attacker direct root access to your host, you can run named as a user other than root. Ideally, the user account running named would have as little privilege as possible; for example, only being able to read BIND related files, the named.pid file, and the like. BIND versions 8.1.2 and later support this feature.

The steps involved in running named as a non-root user are:

1. Create a new user and group to run named as (e.g., user = named, group = named).
2. Make sure the new user has read access to the zone files*.
3. If you are using dynamic updates, ensure that the new user has write access to any zone files for which dynamic updates are required.
4. Make sure this the new user can write to named's PID file.

    - The default PID file is `/var/run/named.pid`, but `/var/run/` is not usually writable by non-root users.
    - The default can be overridden with the `pid-file` option. For example, if `/var/named/` is writable by the new user, then the PID file could be put in this directory using:

      ```
      options {
      ```

* It is not necessary for the new user to be able to read the `named.conf` file, since named will still start up as root, read the `named.conf` file, then change to the unprivileged user.

```
        ...
        pid-file "/var/named/named.pid";
        ...
};
```

5. Start named with the `-u` option telling it to change to the new user. For example:

```
# named -u named
```

Run your name server in a `chroot()` "jail"

Further protection against attackers compromising the entire system through a BIND vulnerability can be obtained by running named in a `chroot()` "jail". Doing so effectively confines named to its own subdirectory so that a compromised named cannot expose the entire filesystem.

The changes required to implement named in a `chroot()` environment are more complicated than we can go into given the scope of this document. The basic steps are described below, but for additional resources on this topic please refer to

- Chapter 11 of DNS and BIND, 4th Edition http://www.oreilly.com/catalog/dns4/
- Chapter 10 of DNS and BIND, 3rd Edition http://www.oreilly.com/catalog/dns3/
- Chroot-BIND HOWTO http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html
- Dual `chroot`ed BIND/DNS servers http://www.etherboy.com/dns/chrootdns.html

The major steps required to implement named in a `chroot()` environment are:

1. Create a new directory for named to run in (e.g., `/var/named/`).
2. Create the appropriate subdirectories for named.
3. Copy the necessary files into the new directory. At minimum, this will include
    - the `named` binary.
    - the `named.conf` configuration file.
    - all zone files.
    - the `named-xfer` binary.

    Additional files, such as shared libraries, may be required.
4. Start named with the `-t` option to specify the `chroot()` directory

```
# named -u named -t /var/named
```

Example configurations

Here are some example configurations showing you how to put it all together.

- A BIND 8 or 9 name server, primary master for a zone, supporting no resolvers, not used as a forwarder, with IP-based authentication of slaves: (This configuration allows anyone to query this name server, but treats all queries as non-recursive)

```
acl slaves { 192.168.4.47; 192.168.4.201; };

options {
      directory "/var/named";
      recursion no;
      fetch-glue no; // for BIND 8 only
      allow-query { any; }; // the default
};
zone "example.com" {
      type master;
      file "db.example.com";
      allow-transfer { slaves; };
};
```

- Same as above, but with TSIG-key authenticated slaves:

```
key master-slave. {
      algorithm hmac-md5;
      secret "Niwn23hw=-whnOUn37.s97jwbaIaNuiaw083";
};

options {
      directory "/var/named";
      recursion no;
      fetch-glue no; // for BIND 8 only
      allow-query { any; }; // the default
};
zone "example.com" {
      type master;
      file "db.example.com";
      allow-transfer { key master-slave.; };
};
```

- A BIND 8 or 9 name server, primary master for a zone, that supports one or more resolvers:

```
acl internal { 192.168.4/24; };
acl slaves { 192.168.4.47; 192.168.4.201; };
options {
      directory "/var/named";
      recursion yes; // the default
      allow-query { internal; };
      use-id-pool yes; // for BIND 8 only
};
zone "example.com" {
      type master;
```

```
        file "db.example.com";
        allow-transfer { slaves; };
        allow-query { any; };
};
```

- A BIND 8.2.1+ or 9 name server, slave for a zone, that's used as a forwarder:

```
acl internal { 192.168.4/24; };
options {
        directory "/var/named";
        recursion yes; // the default
        allow-recursion { internal; };
        use-id-pool yes; // for BIND 8 only
};
zone "example.com" {
        type slave;
        masters { 192.168.4.154; };
        file "bak.example.com";
        allow-query { any; }; // the default
        allow-transfer { none; };
};
```

- A BIND 8 or 9 caching-only name server:

```
acl internal { 192.168.4/24; };
options {
        directory "/var/named";
        recursion yes; // the default
        use-id-pool yes; // for BIND 8 only
        allow-query { internal; };
};
zone "." {
        type hint;
        file "db.cache";
};
```

A split service name server configuration

- An advertising name server:

```
acl slaves { 192.168.4.47; 192.168.4.201; };
options {
        directory "/var/named";
        recursion no;
        fetch-glue no; // for BIND 8 only
        allow-query { any; }; // the default
};
zone "example.com" {
        type master;
        file "db.example.com";
        allow-transfer { slaves; };
};
```

- A resolving name server:

```
acl internal { 192.168.4/24; };
options {
      directory "/var/named";
      recursion yes; // the default
      use-id-pool yes; // for BIND 8 only
      allow-query { internal; };
};
zone "." {
      type hint;
      file "db.cache";
};
zone "example.com" {
      type slave;
      masters { 192.168.4.154; };
      file "bak.example.com";
      allow-transfer { internal; };
};
```

Follow relevant newsgroups and mailing lists

Unfortunately, new vulnerabilities are found in name servers all the time, however, vendors usually patch these vulnerabilities quickly. If you follow the relevant newsgroups and mailing lists closely, you'll learn quickly about the vulnerabilities and any necessary patches or reconfigurations.

- news:comp.protocols.dns.bind newsgroup (and mailto:bind-users@isc.org, its mailing list equivalent); join by sending mail to bind-users-request@isc.org

- bind-announce@isc.org, a lower volume announcement list; join by sending mail to bind-announce-request@isc.org)

- news:comp.protocols.tcp-ip.domains newsgroup

- The CERT Advisory mailing list; join by sending mail to cert-advisory-request@cert.org

- Internet Software Consortium (BIND publishers) http://www.isc.org/ and http://www.isc.org/products/BIND/

- Your UNIX vendor's security announcement mailing list

Appendix: Other documents on DNS security

Secure BIND Template
By Rob Thomas
http://www.cymru.com/~robt/Docs/Articles/secure-bind-template.html

Chroot-BIND HOWTO
By Scott Wunsch
http://www.linuxsecurity.com/docs/LDP/Chroot-BIND-HOWTO.html

DNS Tools
By D. J. Bernstein
http://cr.yp.to/djbdns.html

RFC-2845 Secret Key Transaction Authentication for DNS (TSIG)
http://www.ietf.org/rfc/rfc2845.txt?number=2845

DNS and BIND, 4th Edition
By Paul Albitz, Cricket Liu
http://www.oreilly.com/catalog/dns4/