Global Information Assurance Certification Paper

Securing BIND

How To Prevent Your DNS Server From being Hacked

Derek D. Martin

SANS GIAC Security Essentials Certification

Version 1.2d

Introduction

BIND is the Berkeley Internet Name Domain system, which is an implementation of DNS. DNS is not a peice of software; it is a specification of a protocol as outlined in RFC 1034 and defined by RFC 1035. It is a system which is made up of a distributed database of host names and IP addresses maintained by administrators of Internet sites around the globe, and which provides a means of efficiently mapping host names to IP addresses and vice-versa. BIND is one software package that implements DNS, and is by far the most common software used to provide DNS service on the Internet today. It runs on most, if not all flavors of Unix, including the free ones such as the BSDs and Linux, and even has a port to Windows. Having never used BIND on Windows, I will deal entirely with Unix-based implementations, though the concepts, and in some cases the procedures, will apply to Windows as well.

In writing this, I wanted to go into some detail about some issues regarding the security of DNS servers which, in my experience, are often glanced over or ignored entirely. My intention is to provide some brief background, describing some of the attacks that BIND has historically been subjected to, and then a more detailed description of some ways to completely lock down your installation of BIND. I will also focus on securing BIND from the perspective of how it interacts with the operating system; for the most part I will not discuss any of the security features of BIND itself. There are numerous other documents that discuss these features at length. For example, the man pages for `named(8)` and `named.conf(5)` talk about how and why to restrict zone transfers to specific hosts.

In his paper "BIND 8 Buffer Overflow in TSIG", Richard Biever talks about one of these methods, and describes a buffer overflow vulnerability that was discovered in its implementation. The Unix System Administration Handbook by Evi Nemeth, et. al., considered by some to be the bible of Unix System Administration, also talks about the DNSSec security extensions to BIND, and discusses other mechanisms that BIND implements for securing queries. Note that though a buffer overflow vulnerability was discovered in the TSIG exchange mechanism, it has been fixed, and this is not a reason not to use them! It *is* a reason to make sure that you are running the latest version of BIND, however.

What Are The Attacks?

Before we can protect ourselves, we need to know what it is that we are protecting ourselves from. Historically, BIND has been vulnerable to a wide array of attacks, most of which are attacks on the availability of the service, or arguably on the integrity of the service. I'll briefly discuss those first.

Denial of Service

A denial of service (often DoS) attack, generally, is when an attacker prevents some service that you are offering to the public from being available. One way a DoS can be effected against a BIND server is for a buffer overflow exploit to fail, due to some unexpected configuration or condition of the machine attacked, or because the code was written in such a way that the buffer overflown in an attack does not allow for an exploit which allows remote access. The result is generally that the named daemon is killed. There are also many other ways to effect a DoS attack.

An example of a DoS attack against BIND is detailed at Security Focus. When vulnerable versions of BIND are queried for compressed zone transfers, the named daemon crashes, resulting in a DoS attack. The solution to this type of attack (as with most vulnerabilities) is typically to upgrade to a non-vulnerable version, or apply a patch to the vulnerable version, as in this case.

Buffer Overflow/Remote Exploit

Historically, BIND has been subject to numerous buffer overflow vulnerabilities which allow an attacker to overwrite the execution stack of a process, and force it to execute arbitrary code with whatever privilege the compromised process had, and generally used to provide remote access to the attacker. A famous paper written by Aleph One called "Smashing The Stack For Fun And Profit" describes this process at length. The vulnerability discussed in Richard Biever's paper (mentioned above) allows this type of attack.

Cache Poison

Cache poisoning is an attack which allows the attacker to inject incorrect IP address information into the DNS cache of a remote DNS server, potentially causing all of the traffic intended for the spoofed site originating from the attacked site to be redirected to the attacker's site, or some other random target. More details are available in a paper entitled DNS Spoofing (Malicious Cache Poisoning) by Doug Sax. Some cache poisoning attacks can be remedied with more restrictive configurations, while others require upgrade to non-vulnerable versions.

Domain Hijack

Similar to the Cache Poison attack, domain hijacking allows an attacker to control the DNS for an entire site. This attack is often accomplished by completely taking over the site's master DNS server, or by social engineering, i.e. by swindling a site's domain registrar into believing that the attacker is responsible for the target site. Doug Sax's afforementioned DNS Spoofing paper discusses this as well.

How Do You Stop The Attacks?

There are a number of measures that can be taken to help you ensure that your DNS server is not compromised. When all are used together in conjunction, you ought to be able to pretty well guarantee that your machine will not be compromised (at least, based on the current known vulnerabilities of BIND). In the worst case scenario, an attacker might be able to effect a DoS attack against your server if an unreported (and therefore unfixed) vulnerability is attacked by someone trying to gain access to your system.

Run as non-root user

Most name servers have the worst damage done to them because named is typically run as root by default. However, it does not need to run as root, and there really is no good reason to run it as root. The administrator is provided two options to change the user and group that named runs as:

```
-u   specify the user that named runs as
-g   specify the group that named runs as
```

So, for example, if you created a user "named" and a group "named" for named to run as, you would start named (generally from an init script in `/etc/rc.*`) like this:

```
/usr/sbin/named -u named -g named
```

By not running named as root, you can greatly reduce the amount of damage that an attacker can do if your name server is compromised, assuming they can not find another way to become root. Security in depth is essential!

*Note: BIND 9 does not support the **-g** option. If you upgrade from BIND 8 to BIND 9, you will need to modify how you start named in your init script, so that it does not use **-g**.*

Permissions on Files and Directories

Most of the books and articles I've read that discuss DNS security overlook this aspect of securing the name server entirely. I also rarely have heard colleagues discuss file permissions when discussing the security of DNS. However, by carefully setting the ownership and permissions of your DNS server's relevant files, it is possible to greatly limit the damage an attacker can do, if you are also running your DNS server as a non-root user. If an attacker gains access to your DNS server, and your files are owned by the same user that named runs as, they can still do a lot of damage, because they can change your data, allowing them to re-direct traffic from your site to some other site, or they can delete the data entirely, effectively shutting down your server until you can restore from your most recent back-up.

If your DNS server is a slave, either for your domains or for someone else's, permissions will need to be set up a little less restrictive than the would be if it were only serving as a master DNS server. This is because in the former case, it will need to be able to write copies of the zone files it has downloaded from its respective master servers, whereas if it is only a master server, it will not need to write any zone files at all.

Master Server Only

In the case of the master server, you will want your permissions to be as restrictive as possible, in order to protect your zones from being over-written by attackers. Fortunately, because of the way named works, this is also possible with the master DNS server.

In order to ensure that the user which named runs as is not able to write files, you must look at both the files themselves, and the directories they are in. The directory that named zone files are in is often /var/named, though it may be different on your system. I'll use that as an example.

The ownership and permissions of `/var/named` should look like this:

```
drwxr-x---   3 root     named        1024 Mar 13 16:49 /var/named
```

The root user owns the files, and no one but root has write access to the directory. This prevents any other user from creating files in that directory, or deleting files that are in it. The directory needs to be both readable and executable by the group "named" so that the named daemon can read the files. If it can't read the files, named can't answer queries! Making the directory group-owned by named will allow the named daemon (which remember was running as group named in our example above) access to the files in that directory, but will not let it, or an attacker who has compromised it, create or delete files in that directory. Non-root users who are not in the named group will have no access to this directory, nor any of the files in it.

There's little harm in making the directory readable/executable by other users, if you want local users to be able to read these files without being either root, or named group. However, if you have untrusted users on the machine that provides your slave service, you may want to leave all of the permissions for the "other" category off, to deny any access to those users.

The zone files themselves will live in this directory, and have permissions similar to the directory but more appropriate for a regular file:

```
-r--r--r--   1 root     root        19484 Apr 10 16:59 db.yourdomain
-r--r--r--   1 root     root         2835 Aug 31  2000 named.ca
-r--r--r--   1 root     root          488 Aug 31  2000 named.local
-r--r--r--   1 root     root          577 Dec 20 19:18 rev.1.2.3
```

Again, these files need to be readable by everyone, but don't need to be writable by anyone. You can edit these files as root, who can write to these files whether they are writable or not. No files on the system need to be writable by the user named, nor should they be. With your permissions set up this way, your attacker will not be able to write any files here, even if they do manage to break in through an unpatched exploit in named, if you have not run your server as root.

Slave Server Only

If you are configuring your slave server, your options are a little more limited. The problem is that the user which named runs as must be able to write both temporary zone files while the zone transfer is in progress, and then must be able to write to the zone file after the transfer has completed. The risk of allowing this is mitigated a great deal by using the other teqniques outlined here, but it does increase your risk somewhat, and you really have no choice. You are more or less stuck configuring your permissions like this:

```
drwxrwx--T   3 root     named        1024 Mar 13 16:49 /var/named
```

For those unfamiliar with it, the 'T' on the end is the "sticky" bit. When set on a directory, it means that users who do not own a given file in the directory can not remove that file. It should be pointed out though, that if they have write access to the file itself, they can still overwrite the file, or truncate it to zero length. I'll explain why we set the sticky bit in a moment. Here is the command to set these permissions on the directory:

```
# chmod 1770 /var/named
```

Here, the '1' represents the sticky bit.

For the most part, you do not need to worry about the permissions of the files in this directory. If your secondary zone files already exist when you make this change, you will need to make sure that they are owned by named and have group ownership of named. For example:

```
-rw-r--r--   1 named    named        2835 Aug 31  2000 db.secondary
```

If they did not exist, you need do nothing, because your slave server will create all of the files it needs, with appropriate permissions. The two exceptions are the "cache" file, used to point the server at the root name servers, and the "local" file, which is used by the name server to resolve the address 127.0.0.1 to the name localhost. Since we don't need to download these files from a master, they don't need to be writable. We can protect them by making them owned by root, and read-only by all users.

```
-r--r--r--   1 root     root         2835 Aug 31  2000 named.ca
-r--r--r--   1 root     root          488 Aug 31  2000 named.local
```

This is where the sticky bit comes in above. If we did not set it, anyone in the named group could remove these files and write new ones.

Because the user named does not own the files, the sticky bit prevents this from happening.

Combination Master/Slave

The same rules apply to the directory as with the slave only server... the directory needs to be writable by the named daemon. Wherever possible, the files in this directory should be owned by root and read-only. Only the zone files for the slave domains need to be writable (and the named daemon will create these).

Run in a chroot-ed environment

After following the guidelines above, you can further limit the damage that a successful attacker can inflict by running named in a chroot-ed environment. The Unix `chroot(2)` system call changes the root directory of a process, such that the process can then no longer access any of the files above the specified root directory in the filesystem heirarchy. Because important files essential to running named (and virtually all programs) live in those directories, some preparations must be made.

Scott Wunsch has written an excellent paper called the Chroot-BIND HOWTO, which is part of the Linux Documentation Project. The document describes in some detail the step-by-step process of setting up BIND to run in a chroot-ed environment. While the focus of the LDP is Linux, this document applies generally to named (though you may need to adjust paths and some of the other details for your OS).

The author of this document suggests doing several things, which require recompiling BIND from source. By following the directions below, it should not be necessary to recompile. What follows is a much abbreviated summary of that document, with the appropriate modifications made so that you will not need to recompile BIND, and with some explanation of things I felt the author left out. It is essentially everything you need to know to set up a chroot environment for named to run in. I will assume that you already have a working installation of BIND, and that your zone files are in `/var/named`.

1. Create the directory structure

Choose a path for your chrooted environment. I chose /var/nroot for my named root. Then create all of the following directories in it:

```
# cd /var/nroot
# mkdir bin
# mkdir dev
# mkdir etc
# mkdir lib
# mkdir -p usr/sbin
# mkdir -p var/named
# mkdir var/run
```

2. Copy necessary files

Since the chroot-ed named will not have access to the rest of the filesystem, all of the files that it needs will need to be copied into the new root. You will need to make `/dev/null` by hand (or use your system's administration tools to create a copy of it), and copy the shared libraries and binaries needed by named. Note that the exact command to make a new `/var/nroot/dev/null` may be slightly different on your system, as will the major and minor numbers of the device if you are not running Linux. You may also need to create a copy of `/dev/zero`. See your man pages for details.

First, make the new `/dev/null`:

```
# mknod /var/nroot/dev/null c 1 3
# chmod 666 /var/nroot/dev/null
```

Now, copy the libraries, and other necessary files. You may need to use ldd to determine what libraries named is linked against on your system. Or, you could just recompile named statically from the sources available at the homepage of The Internet Software Consortium (ISC). You will also need to copy /etc/localtime (if your system has this file) so that named will log the correct time in log messages. On most Linux systems, the following should suffice:

```
# cp /etc/localtime /var/nroot/etc/
# cp /sbin/ldconfig /var/nroot/bin/
# cp /usr/sbin/named /var/nroot/usr/sbin/
# cp /usr/sbin/named-xfer /var/nroot/usr/sbin/
# cp -p /lib/libc*.so /var/nroot/lib
# cp -p /lib/ld*.so /var/nroot/lib
```

The last two commands copy the libraries. You also need to symllink the libraries you copied to their "common" names. For example, on a

recent Linux system, it is expected that the standard C library will be named libc.so.6, so you need to symlink that to the real library file. You also need to create similar symlinks for the dynamic link loader to whatever the name of the ld*.so file that you copied was. Before you do this step, look at the files in your /lib directory to see how the existing symlinks were created, and make sure you create the symlinks appropriately in the chroot environment. On my system, the commands would be:

```
# cd /var/nroot/lib
# ln -s libc-2.1.3.so libc.so.6
# ln -s ld-2.1.3.so ld-linux.so.2
```

On systems that use the GNU ld or one that behaves similarly, you may need to configure the linker by creating a file called etc/ld.so.cache, which you can do by running ldconfig in the new environment:

```
# chroot /var/nroot /bin/ldconfig
```

Finally, you must create a group file with the named group in it. This is the same group that you specified with the -g option when you started named above. This should suffice:

```
# grep '^named:' /etc/group > /var/nroot/etc/group
# chmod 444 /var/nroot/etc/group
```

3. Copy your named data

You need to copy your named data to the new environment. Many systems keep this data in `/var/named`. When we created our directory structure, we created this directory (`/var/nroot/var/named`). We copy the data into the new location:

```
# cp /etc/named.conf
# cp -rp /var/named/* /var/nroot/var/named/
```

4. Logging

Ordinarily on most systems, programs log messages to syslog via a socket, `/dev/log`. It may not be possible on your system to tell syslog to listen for messages on another socket, but on newer BSD-ish syslog daemons, you can specify additional sockets using the -a option. If you have such a syslogd, setting up logging for your chroot named is easy; just restart syslog, specifying an additional socket which lives somewhere in the chroot-ed directory tree, like this:

```
# syslogd -a /var/nroot/dev/log
```

If you do not have such a syslogd, then you can configure named to log to local files. You specify this via the `logging{}` statement in the `named.conf` file. See the man page for named.conf(5) for more details.

5. Using `ndc` (optional)

BIND comes with a program for controlling named, called ndc (name daemon control). Personally, I never use ndc, as I prefer to sigal named directly by using the kill command. However, if you wish to use ndc, you must tell it where its control socket is now located, in the chroot-ed directory tree. When you started up named, it should have been created in /var/nroot/var/run. So, when invoking ndc, you will need to use the following syntax:

```
# ndc -c /var/nroot/var/run/ndc [options] [command]
```

If you do not wish to specify the location of the control socket, you will need to obtain the source and re-compile, after making the modifications specified by Scott Wunsch in his Chroot-BIND HOWTO. If you frequently use the ndc command, you may find that it's worth your time to do this.

That's it! You should now have a working chroot-ed BIND, which will prevent even successful attackers from modifying any files on your filesystem above the chroot-ed home you've created for BIND.

Patch, patch, patch

It can not be said often enough: you must keep up with patches, both for BIND and for all the other software you run. All too often, when a system is compromised, the hole that the attacker gets in through is an old, well-known vulnerability. Do not fall into this trap! Stay on top of security vulnerabilities that are discovered in BIND by visiting ISC's website. They generally announce vulnerabilities as soon as they are released to the public, which generally occurs when a fix is available.

Also, remember that even if the machine on which BIND is running is only used as a DNS server, it probably has numerous other programs installed on it, which may or may not have vulnerabilities. Know the software you have installed; if you don't need a particular software, remove it! Even the worst code is not vulnerable to attack if it is not on the system. Make sure the programs you do need are up-to-date.

Use Defense In Depth

To quote Bruce Schneier, cryptographic expert and author of Secrets and Lies, "Security is a process, not a product." Even though taking the above measures may make your installation of BIND nearly impervious (at least, today), there is much more to securing your DNS server, i.e. the machine itself. A good defense uses a multi-tiered approach, which the authors of the SANS/GIAC Security Essentials curriculum have dubbed *"defense in depth."*

In addition to securing BIND itself, look at your whole security picture. Securing a server is a lot like securing a home. Make sure your server is behind a well-configured firewall to make it difficult to get inside your network, much like a fence around your home. Use host-based firewalling to keep out intruders who have made it past your outer defenses, much like you would use a home security system.

Passwords are the keys to a system; make sure you (and your users!) choose good, secure passwords for your system, just as you would choose strong locks for your doors. Audit all of these defenses regularly, using Intrusion Detection Systems (IDS), host-based vulnerability scanners such as SAINT, network vulnerability scanners such as nmap, and password auditing tools such as l0phtcrack or John the Ripper, as you would pay your home security company to audit your home security. Keep your OS and software patched, just as you would repair broken locks on your home. And, much like you would teach your family members to use your home security system, you must train and educate yourself, your co-workers, and your users. Sign up for mailing lists such as Bugtraq, which announces security vulnerabilities. Also become familiar with Security Focus which is an excellent security community web site which tracks vulnerabilities, exploits, security tools, commercial products, and more. I relied on it heavily to research historical vulnerabilities in BIND.

Also, though I do not discuss them here, BIND has a few internal methods of providing better security, restricting zone transfers. This can be done using IP-based authentication, or with TSIG (symmetric encryption-based) or DNSSEC (asymetric encryption-based) authentication. Look into these in the BIND documentation, and use them where appropriate for your site.

Sources

Mockapetris, P. "Domain Names - Concepts and Facilities." November 1987. URL: ftp://ftp.isi.edu/in-notes/rfc1034.txt

Mockapetris, P. "Domain Names - Implementation and Specification." November 1987. URL: ftp://ftp.isi.edu/in-notes/rfc1035.txt

Biever, Richard. "BIND 8 Buffer Overflow in TSIG." Feb 7, 2001. URL: http://www.sans.org/infosecFAQ/unix/BIND8.htm

Nemeth, Evi et. al. "The Unix System Administration Handbook Third Edition." Upper Saddle River, NJ. Prentice Hall, 2001. pp460-9.

Security Focus. "Multiple Vendor BIND 8.2.2-P5 Denial Of Service Vulnerability." November 1, 2000. URL: http://www.securityfocus.com/frames/?content=/vdb/%3Fid%3D1923

Aleph One. "Smashing the Stack For Fun And Profit." Phrack #49, Vol. VII (date unknown). URL: http://www.insecure.org/stf/smashstack.txt (ubiquitous).

Sax, Doug. "DNS Spoofing (Malicious Cache Poisoning)." November 12, 2000. URL: http://www.sans.org/infosecFAQ/firewall/DNS_spoof.htm

Wunsch, Scott. "Chroot-BIND HOWTO." 24 February 2001. URL: http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html

Internet Software Consortium. "ISC BIND." 2001. URL: http://www.isc.org/products/BIND/

Internet Software Consortium. "named(8)." 2001. ISC BIND manual page (provided with ISC BIND software).

Internet Software Consortium. "named.conf(5)." 2001. ISC BIND manual page (provided with ISC BIND software).

Schneier, Bruce. "Secrets & Lies (Digital Security In A Networked World)." New York. Wiley & Sons, 2000. pg xii.