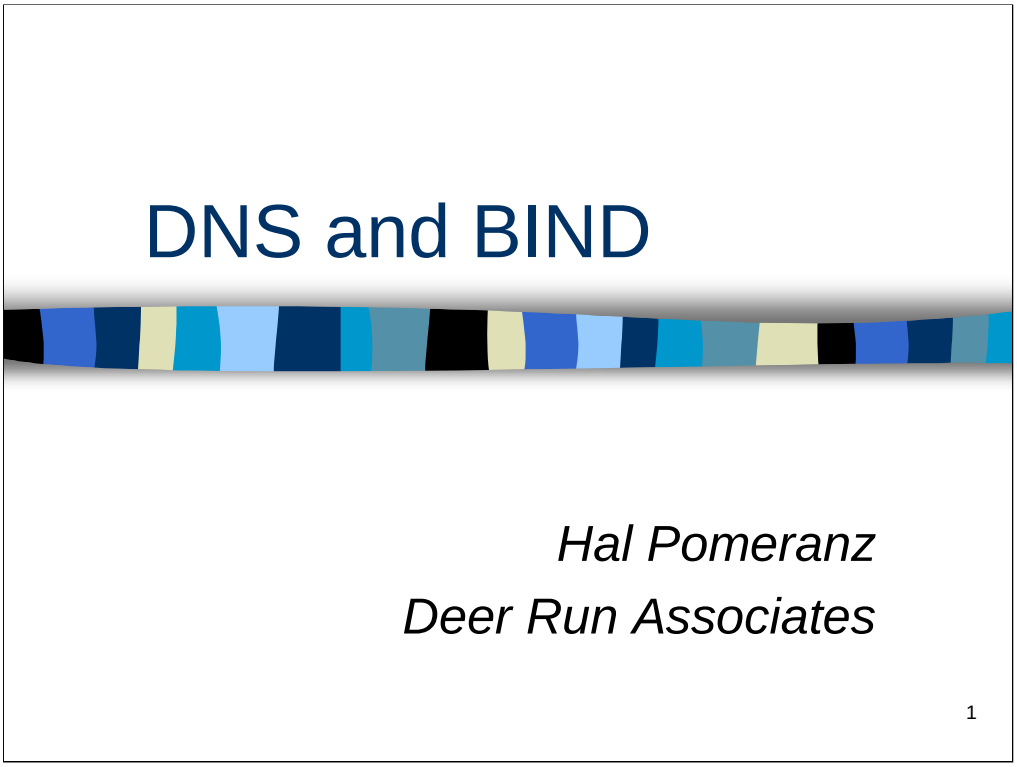
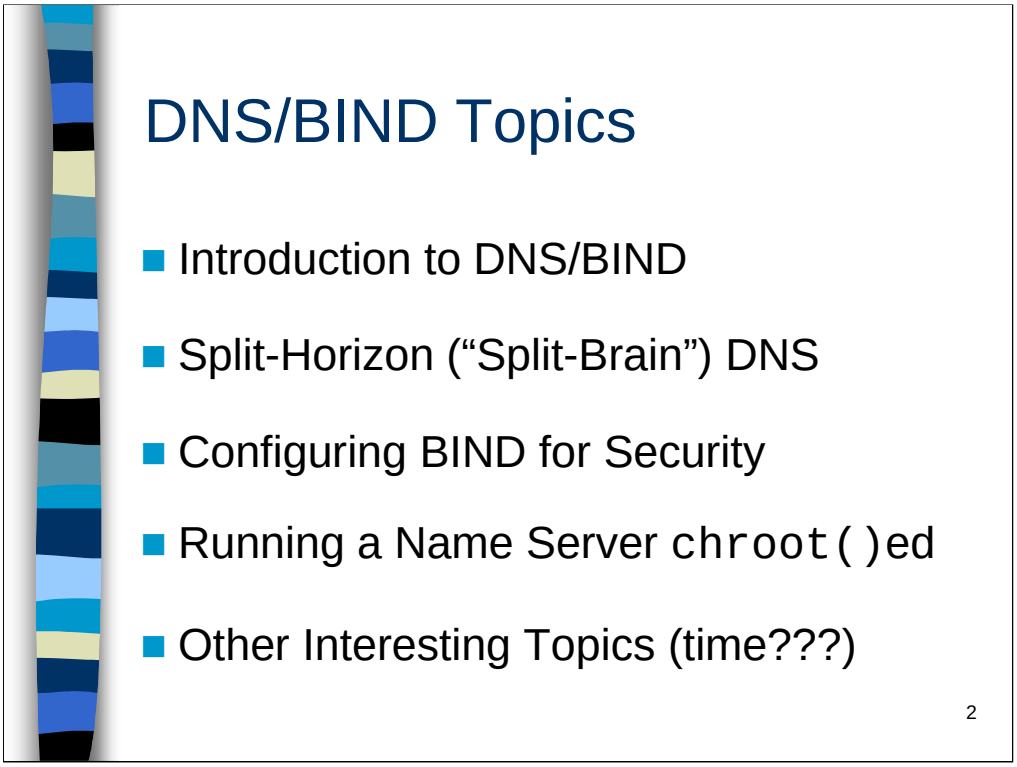


Hal Pomeranz * Founder/CEO * *hal@deer-run.com*
Deer Run Associates * PO Box 50638 * Eugene, OR 97405
+1 541-683-8680 (voice) * +1 541-683-8681 (fax)
http://www.deer-run.com/

More general information on DNS administration is available in the O'Reilly *DNS and BIND* book, as well as at *http://www.deer-run.com/~hal/dns-sendmail/*



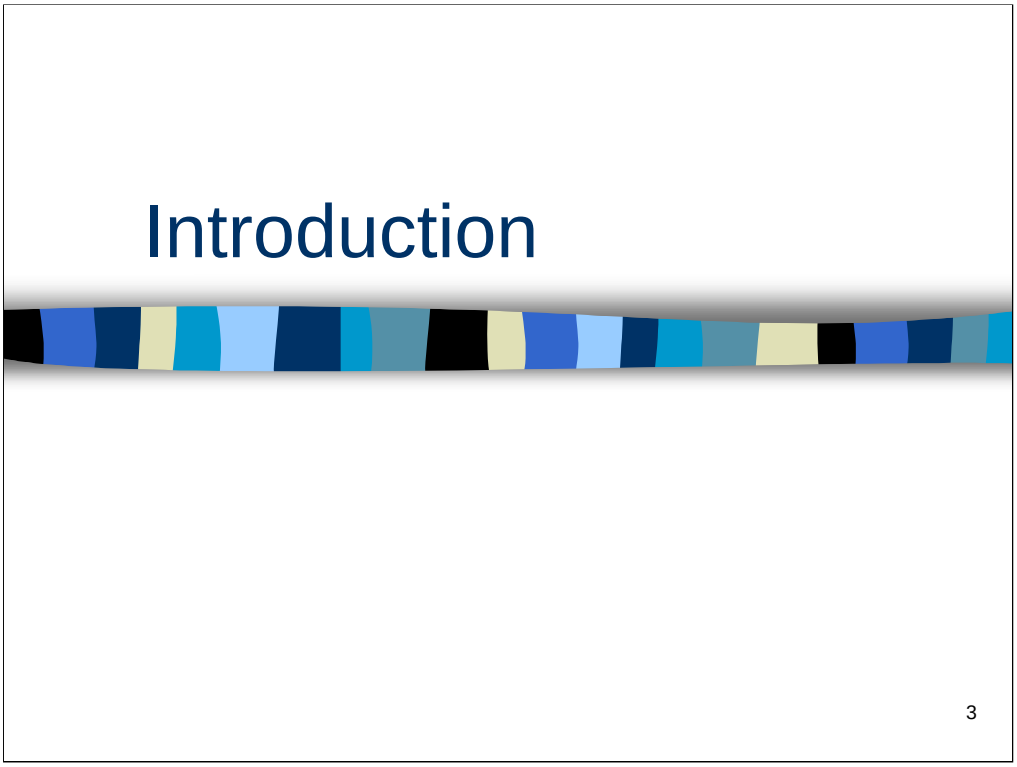
The *Introduction* is a quick introduction to the Domain Name Service and BIND plus an overview of common vulnerabilities in past and present DNS and BIND implementations.

Split-Horizon DNS discusses the theory behind presenting one version of your DNS information to the outside world and a completely different view internally– why and when this is useful and some architectural issues with such a configuration.

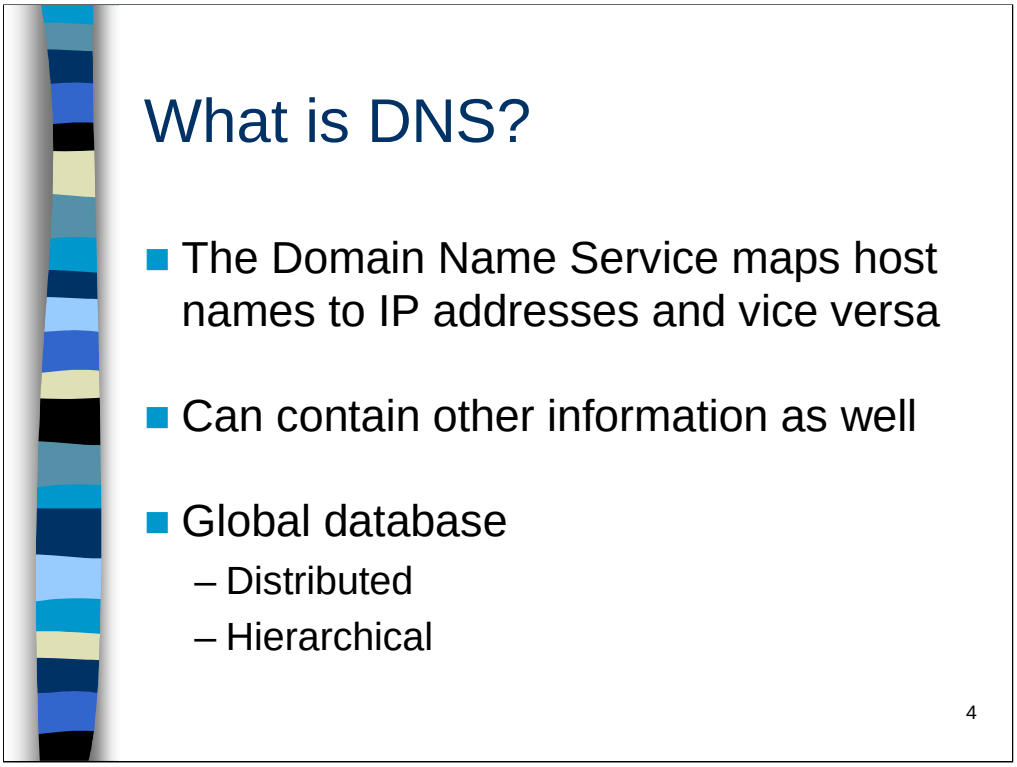
Configuring BIND for Security presents specific configuration examples for the DNS architecture introduced in the *Split-Horizon DNS* section, and introduces many of the new security features in BIND v8/v9.

Running a Name Server `chroot()`*ed* presents additional hints on administering BIND, including how to run BIND without superuser privileges and in a `chroot()`ed environment.

Finally, there's another section with a couple of *Other Interesting Topics* (not directly security related) that we can look at as time allows.



The *Introduction* is a quick introduction to the Domain Name Service and BIND plus an overview of common vulnerabilities in past and present DNS and BIND implementations.

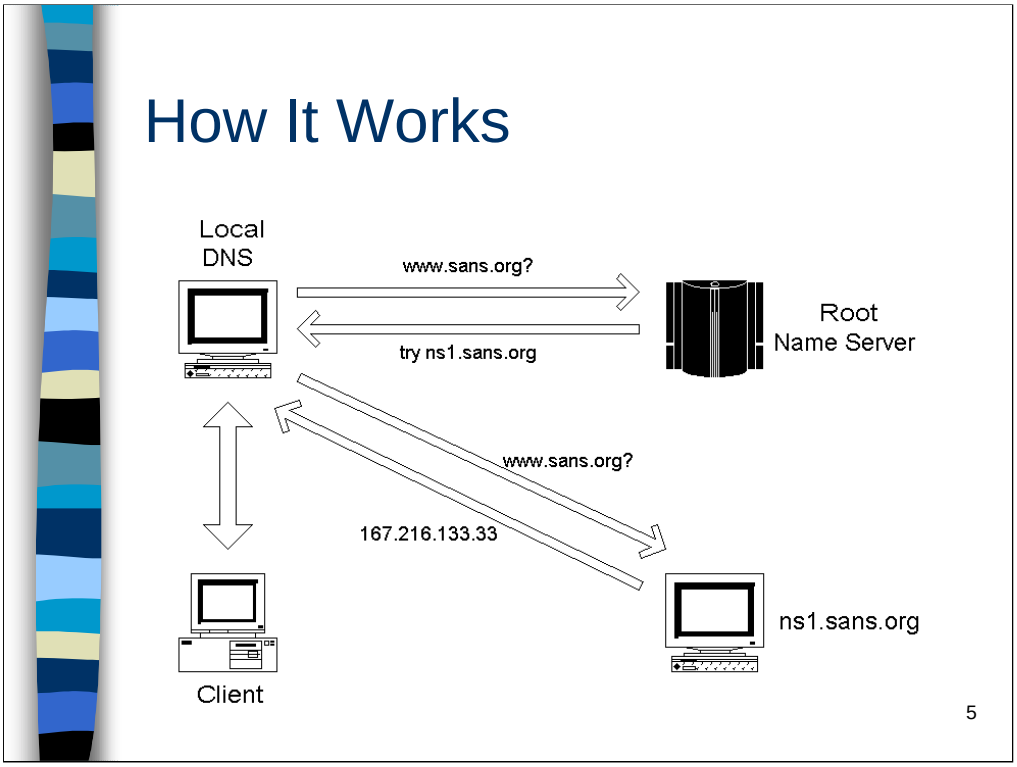


The Domain Name Service (DNS) is the mechanism that Internet hosts use to determine the IP address which corresponds to a given hostname. For example, if your Web browser wishes to reach the home page for the SANS Institute it must first determine the IP address for `www.sans.org`. This IP address is then used as the destination address in the packets which your client sends to communicate with the remote server.

Conversely, the Web server at `www.sans.org` will receive the IP address of your machine as the source of these packets and will most likely attempt to determine the hostname which corresponds to this IP address. Again, DNS is the mechanism which `www.sans.org` will use to make this determination.

DNS is also the mechanism which most companies use within their organization to distribute information about hosts. DNS can contain more information than just hostnames and IP addresses– for example DNS can store information about the type of machine and OS platform (HINFO records), general "free-form" information about machines (TXT records), and many other types of data. In particular, DNS usually provides both internal and external machines with information about how e-mail is to be routed to a given organization (MX records).

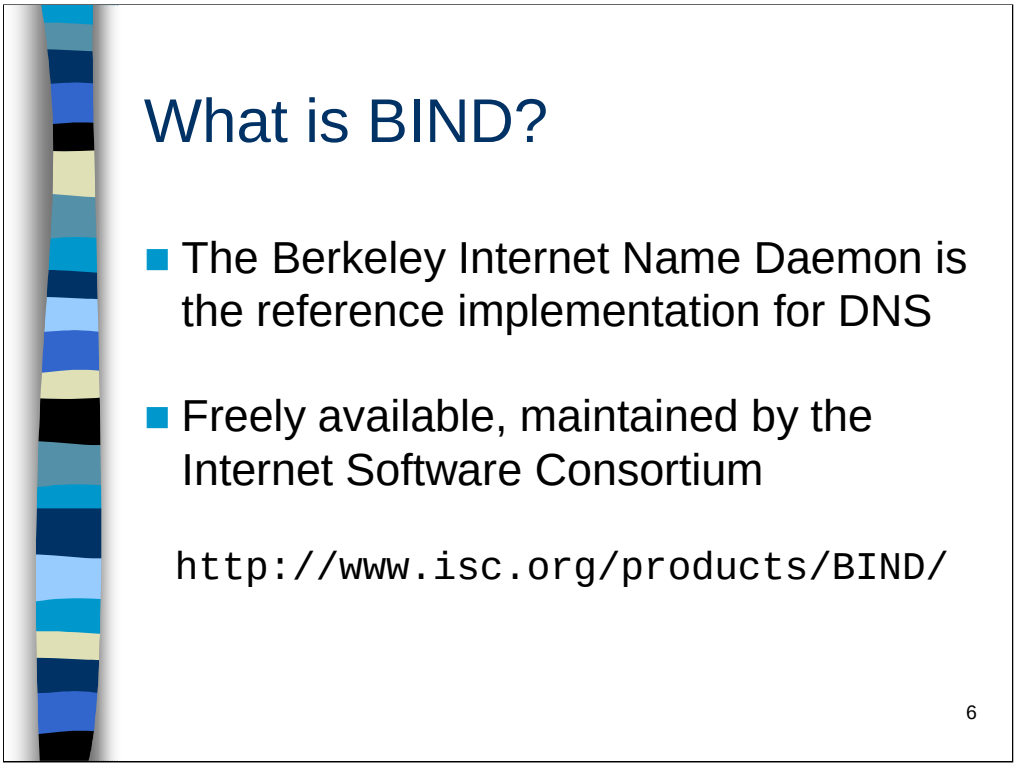
DNS is fundamentally a distributed database system– each organization maintains its own local information. These distributed collections of information are linked in a hierarchical fashion, which is more easily demonstrated pictorially… *(see next slide)*



Let us suppose that your local client wishes to learn the IP address of `www.sans.org`. Your client contacts a local name server which has been configured on the local client by the administrator (either statically or via DHCP, etc.). Your local DNS server actually does all of the work required to resolve the IP address and then will hand the result back to the client.

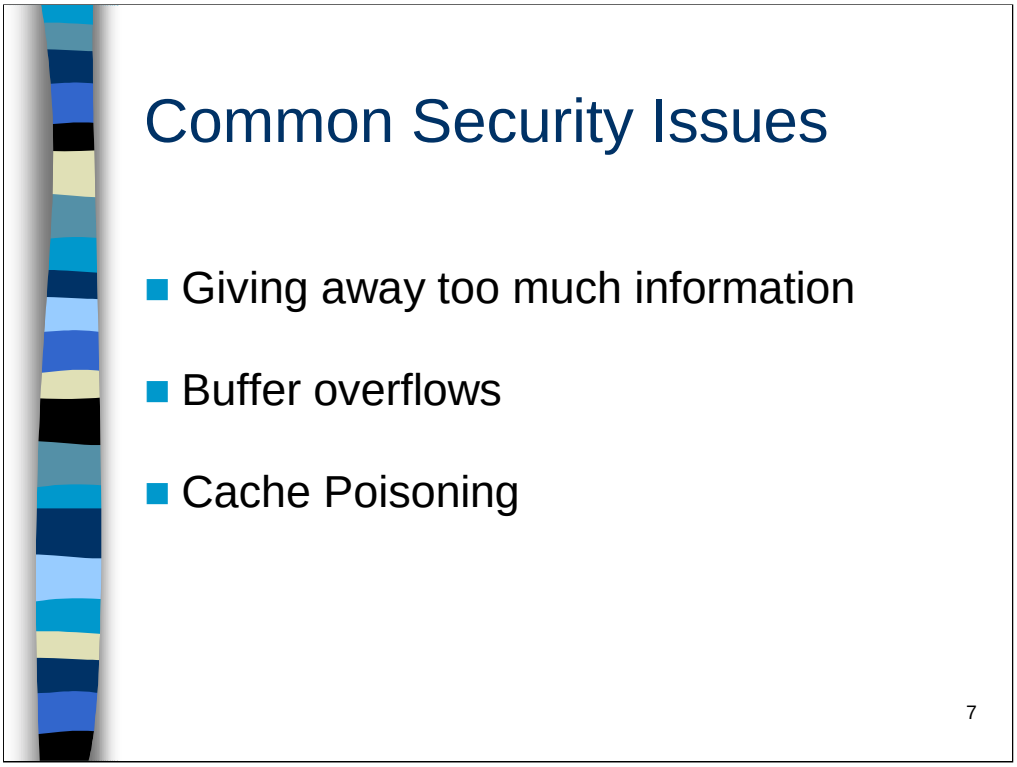
The local name server first attempts to contact one of the several *root name servers* that have been deployed on the Internet. Root name servers maintain a mapping between domains (`sans.org`) and name servers (`ns1.sans.org`)– when your local name server asks for the IP address of `www.sans.org`, it receives a *referral* from the root name servers which essentially says "unable to answer your question, but here is the name/address of somebody who can". In order to be able to contact a root name server, your local name server must be statically configured with the names and IP addresses of the available root name servers. This information is maintained by the InterNIC and downloaded by the administrator into a static file on the local name server.

Having received the names and IP addresses of the name servers for `sans.org` from the root name server, your local name server then contacts one of these machines and asks for the IP address of `www.sans.org`. The name server for `sans.org` returns the IP address to your local name server and the local name server hands the information back to your client.



BIND stands for Berkeley Internet Name Daemon. From its creation, BIND has been the reference implementation for DNS on the Internet and is the basis for the DNS implementation provided with most modern operating systems.

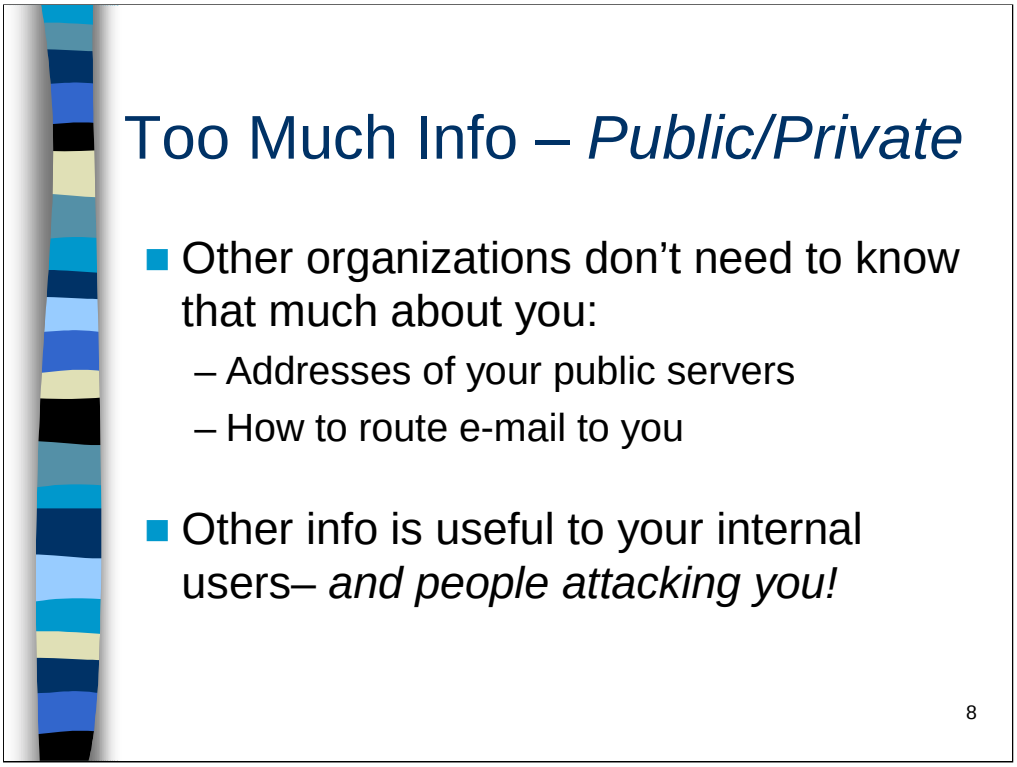
For much of the 80s and 90s, BIND was developed and maintained by Paul Vixie. When Paul co-founded the Internet Software Consortium, BIND was one of the first applications to be supported by this new group. Paul continues to be the primary maintainer for BIND v4 and BIND v8, but BIND v9 development is a team effort under the auspices of the ISC.



The primary risk with running DNS is that you give away too much information– information that can be used by people who wish to attack your systems and networks. Specific examples are given in the upcoming slides.

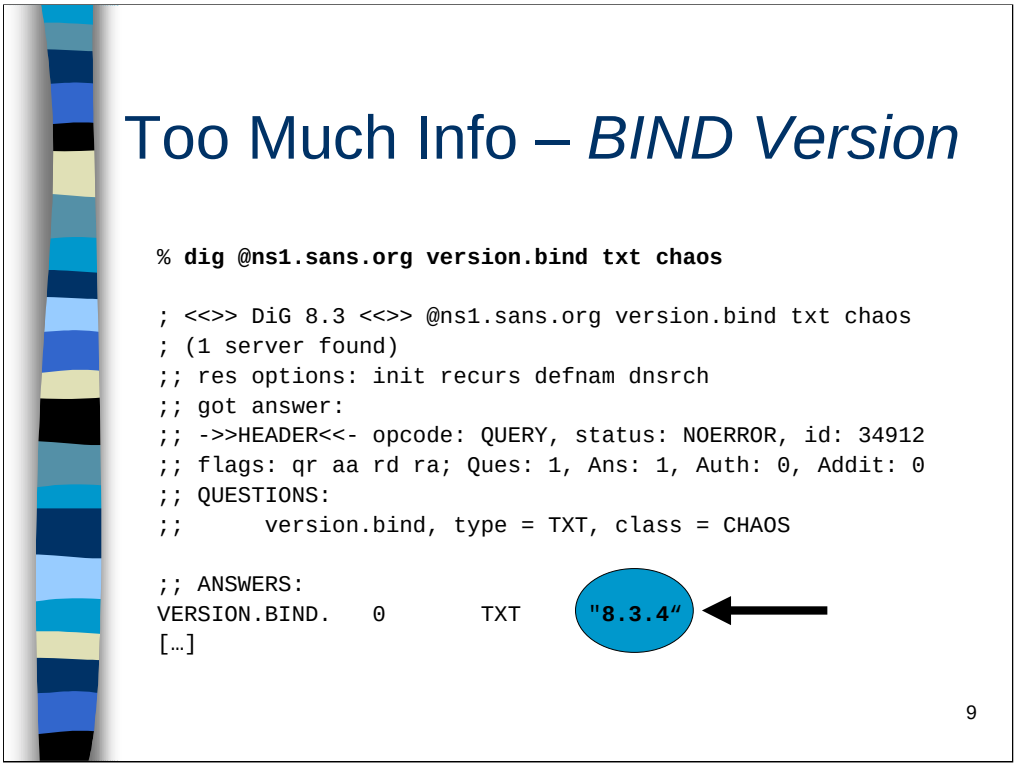
BIND has historically had buffer overflow problems in various releases. Some have led to root compromise attacks, others have simply been denial-of-service type attacks. The best defense against these attacks is to stay up to date on the version of BIND you are running, though the *Running a Name Server* section suggests how to configure BIND to run in a `chroot`()ed environment, which can help protect you in the event of an exploitable buffer overflow.

Cache poisoning occurs when a name server has been tricked into believing erroneous information from some external source. Sometimes this occurs by accident, but most often it is used by attackers who wish to embarrass an organization or exploit trust relationships based on hostname/address information. More on this in an upcoming slide.

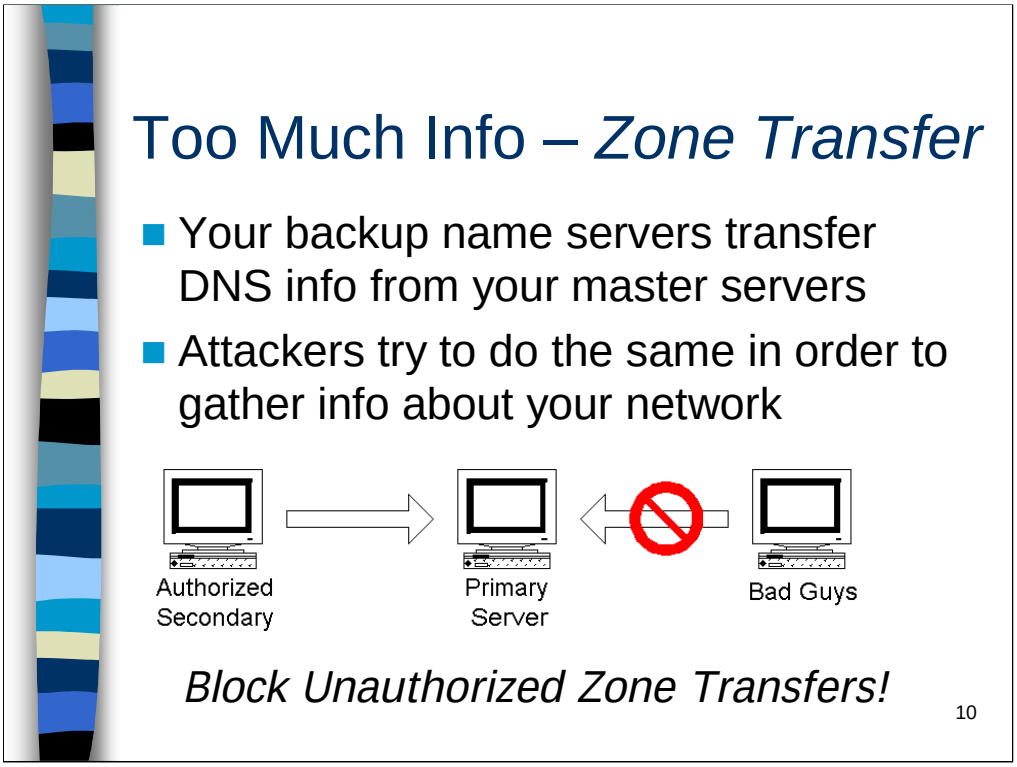


Your DNS database contains information about all of the machines in your organization. In particular, machine names and other information (HINFO and TXT records) may help an attacker locate machines which are most critical to the functioning of your organization or which can be easily targeted for attack– for example, a machine called `proxy.yourdomain.com` might be interesting to an attacker wishing to penetrate the interior of your network or anonymize their attacks on other Internet hosts.

Generally, the outside world needs to know very little information about your network. At a minimum, you need to advertise hostnames and IP addresses of a limited set of "public" servers: name servers, e-mail servers, Web and FTP servers, etc. In a perfect world, an organization would be able to present one set of information to the outside and reserve a full, rich set of information for internal consumption– this is the theory behind *split-horizon DNS* which will be covered in an upcoming section.

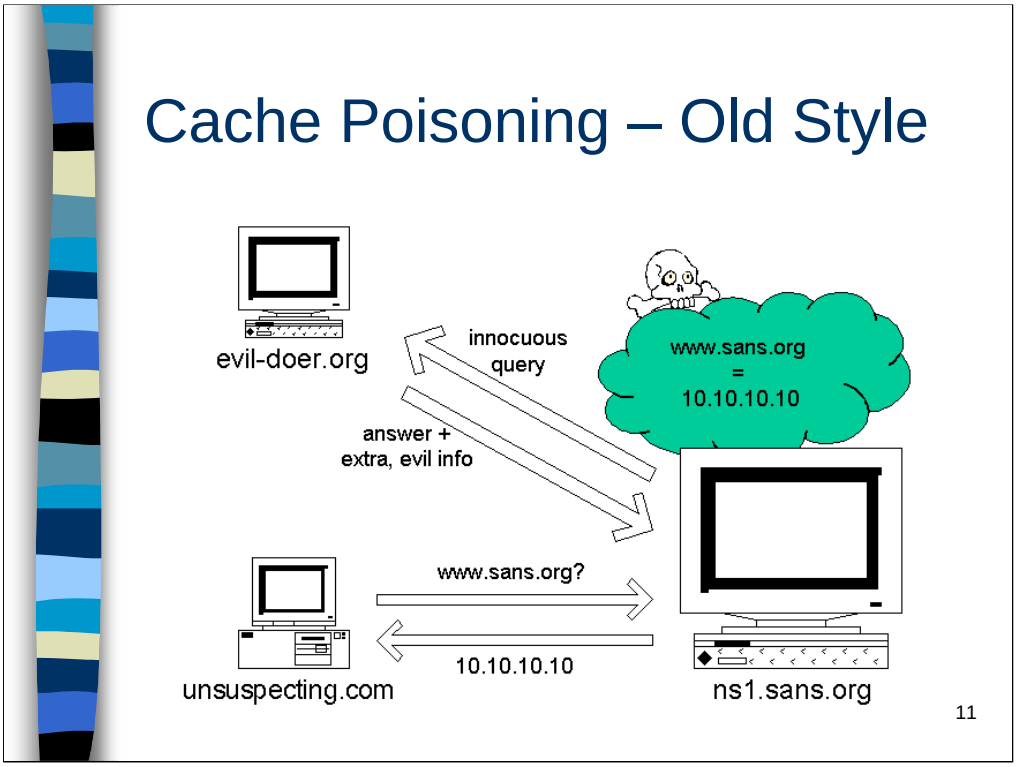


It is possible to query a running name server and retrieve the embedded version number string from the remote machine. Since there are known vulnerabilities in most older releases of BIND, this information can help an attacker target your machines. As we will see, BIND v8 (and later) allows the administrator to easily change the version number string to fool attackers.



Generally, each organization runs one master DNS server and one or more slave servers for redundancy. Periodically, the slaves must contact the master and download any updates to the local DNS database– this is referred to as a *zone transfer*. By default, name servers running BIND allow any remote system to perform a zone transfer– whether that system is a legitimate name server for that domain or not. Zone transfers can even be requested from the slave name servers for your domains.

Attackers often attempt zone transfers in order to gather information about your local network. If they succeed then they have instantly gotten all of the information about your internal hosts and networks with very little effort. Of course, a split-horizon DNS configuration can limit the amount of information an attacker will receive, but it is still a good idea to prevent unauthorized hosts from downloading your zone databases. In a later section we will see how to configure BIND to restrict zone transfers, but it is also a good idea to block this activity at your firewall as well.

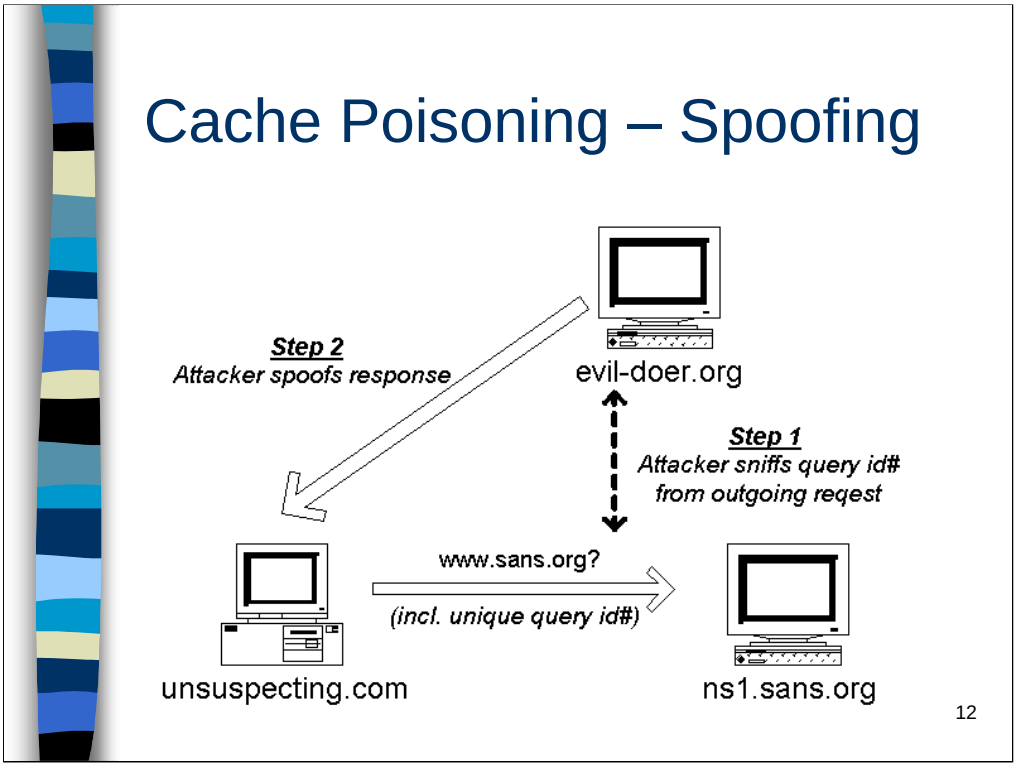


The classic cache poisoning attack is triggered when a vulnerable name server makes a query against some evil name server owned by the attacker. The attacker often triggers this initial query externally by connecting to some service on the vulnerable name server (or one of the hosts which use the vulnerable name server as their local name server)– this can cause an IP address to hostname lookup against the evil name server.

The vulnerable name server makes an innocent query against the evil name server and the evil name server hands back the proper response *plus* some extra information. Older versions of BIND (yet another reason to stay up to date on BIND releases) would put this extra information in their cache– potentially overwriting information that they had learned from their own local zone databases.

In the example above, the evil name server has poisoned the cache on `ns1.sans.org`, convincing this machine that the IP address of `www.sans.org` is `10.10.10.10` (an invalid IP address). When any other machine queries `ns1.sans.org` for the IP address of `www.sans.org` they will get the invalid address and be unable to reach the real Web server for sans.org. Imagine, however, that the attacker had used an address which corresponded to a hard-core pornographic site (embarrassment) or an IP address which corresponded to a Web site owned by the attacker (man in the middle attack).

Note: the idea for the design of this slide comes from Judy Novak (thanks Judy!)



Generally, the way to poison DNS caches these days is with a simple spoofing attack. DNS responses are essentially unauthenticated, so if an attacker can create and send a legitimate looking response before the "real" name server that was queried can respond, then the host doing the lookup will happily accept the forged response containing bogus information.

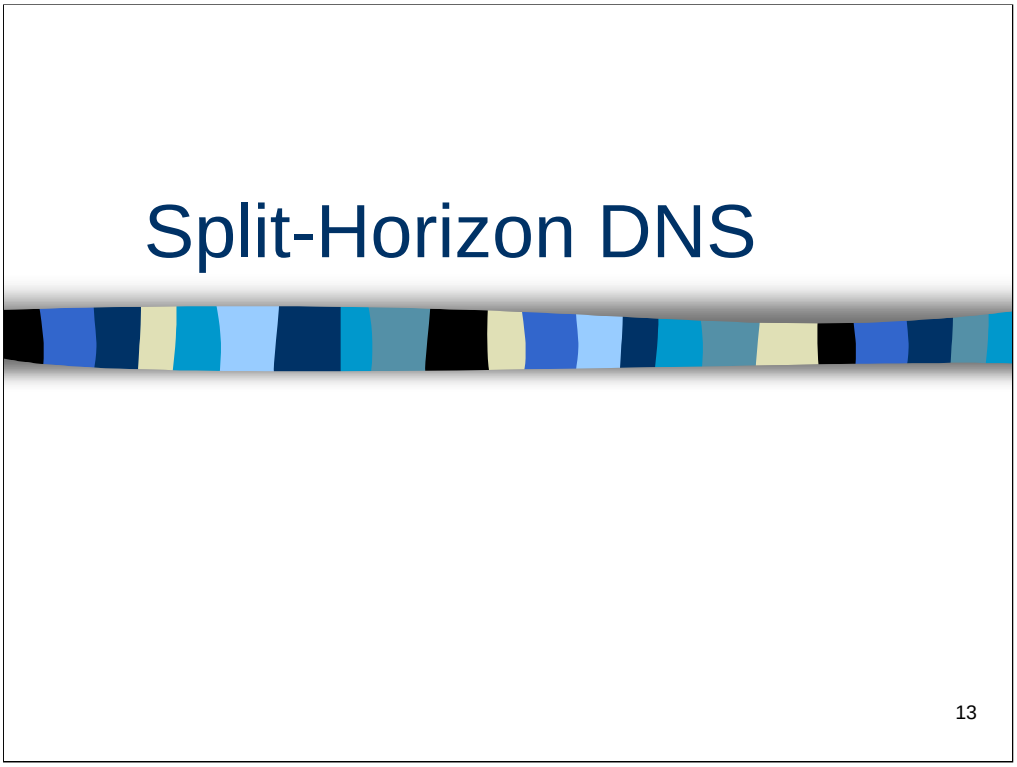
The only problem from the attacker's perspective is that each DNS query goes out with a unique query ID number– the attacker's bogus response must include this ID. So, the attacker must typically first sniff the outgoing request and then incorporate the stolen query ID number into the bogus request. Obviously, the attacker must also forge the source IP address of the bogus response to make it appear that the packet comes from the remote name server that was queried.

Early releases of BIND used very predictable query ID numbers– making it possible for attackers to spoof responses without first sniffing the outgoing query. More recent releases of BIND have better randomization algorithms (so stay up-to-date!).

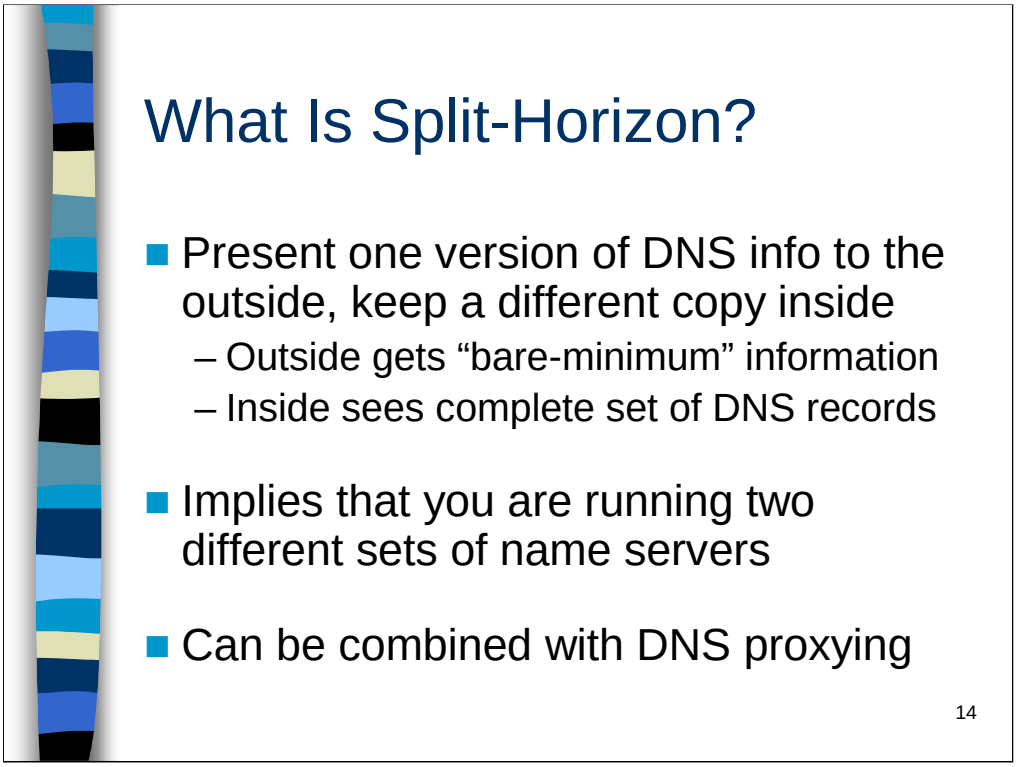
One of the draft standards working its way through the IETF is the "DNSSec" standard which implements digital signatures for authenticating DNS responses. Once DNSSec becomes widely implemented (BIND v9 actually includes a working DNSSec implementation), this spoofing issue should go away.

Zodiac is a tool which actually implements this sort of DNS spoofing. For more information on Zodiac, see:

`http://www.team-teso.net/projects/zodiac/`

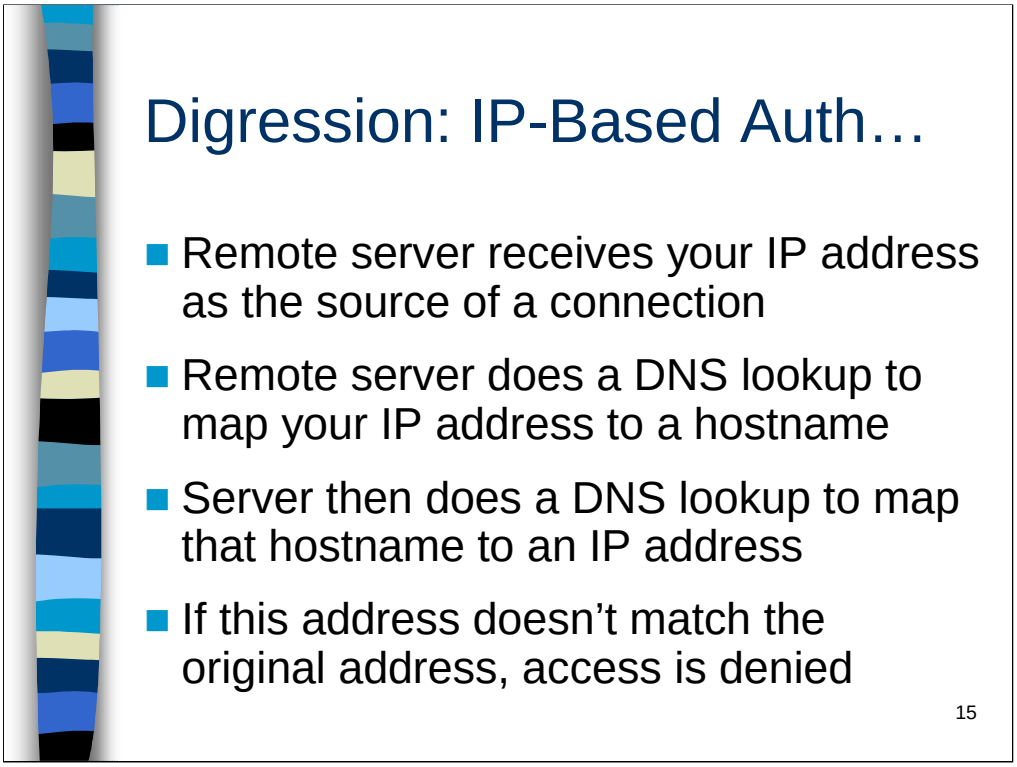


Split-Horizon DNS discusses the theory behind presenting one version of your DNS information to the outside world and a completely different view internally– why and when this is useful and some architectural issues with such a configuration.

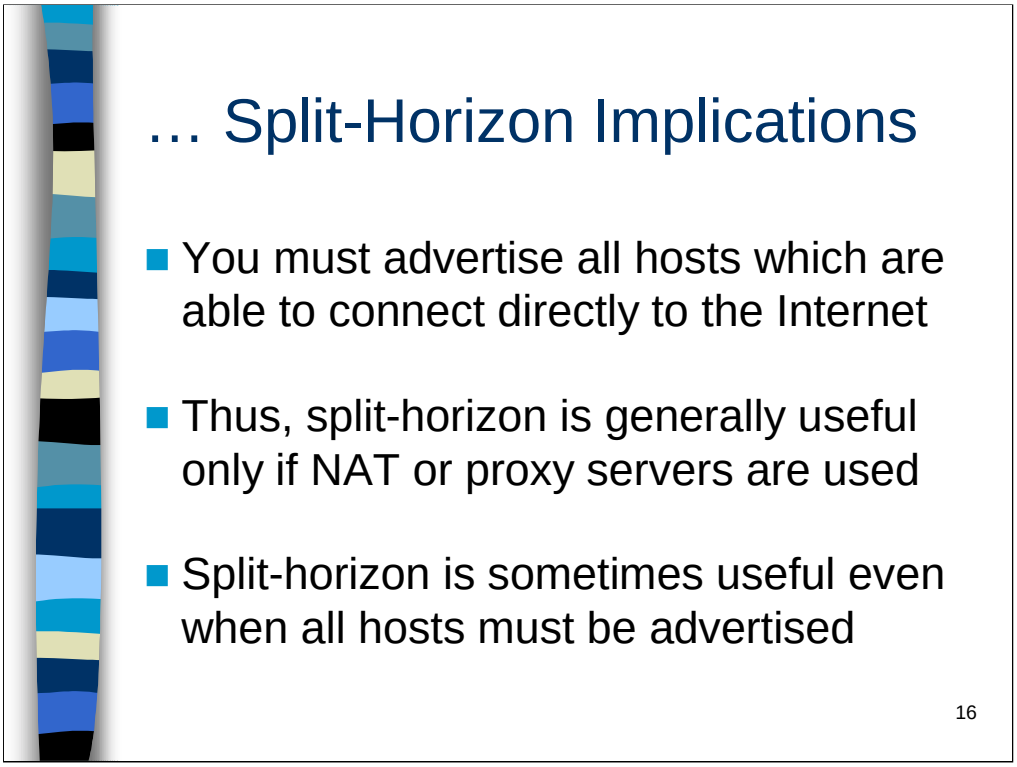


As discussed earlier, giving away too much DNS information to the outside world can help attackers map your networks or choose vulnerable or otherwise "interesting" machines to target for initial attacks. *Split-horizon* (sometimes called *split-brain*) *DNS* is a DNS configuration where an organization presents one set of DNS information to external organizations and reserves a second, separate set of DNS information for internal use. This is generally done by maintaining two different collections of name servers: an "external" set which publishes the limited amount of DNS information that external organizations need to interact with your company, and an "internal" set which holds your complete, rich set of DNS information. Note that while the separate DNS zone databases are generally maintained on two different sets of physical machines (and this is the configuration we shall describe in this course) it is possible under BIND v8 to run to different name servers with different zone databases on the same machine.

When your internal name servers wish to resolve external host names they must contact root name servers and name servers at other Internet-connected sites. This can open up your internal name servers to attack from the outside. For this reason, many organizations that run split-horizon DNS also employ a sort of DNS proxying (*slave forwarding name servers*) to "hide" their internal name servers completely from the outside world.



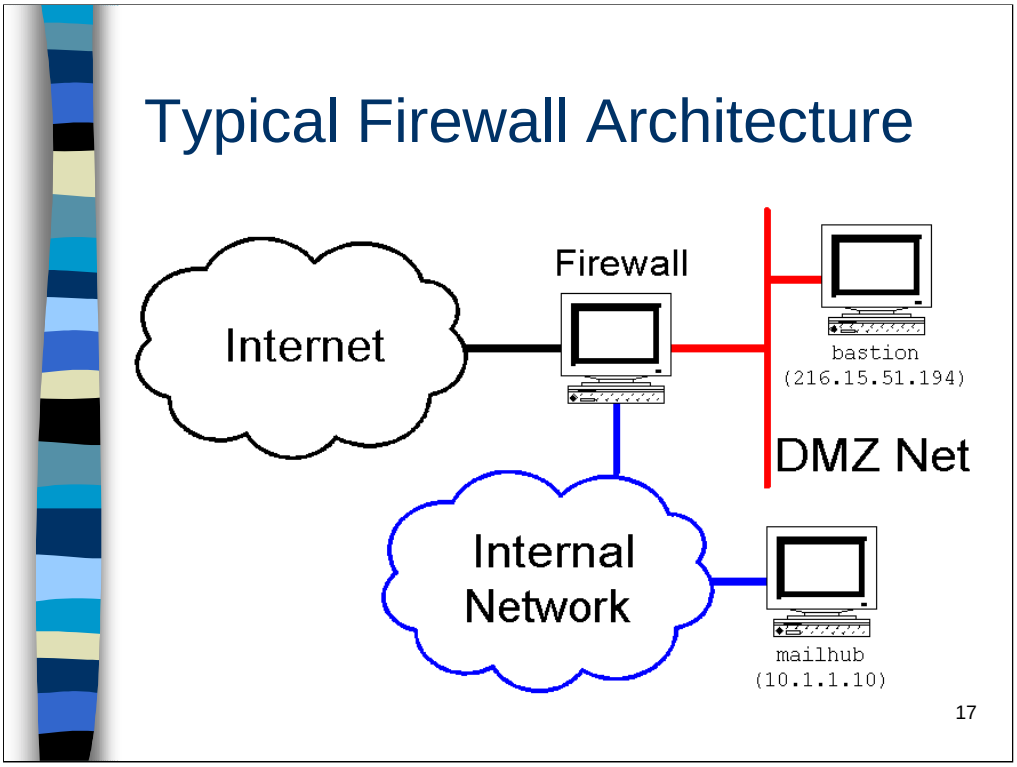
One common simple form of authentication used by some Internet servers (often FTP daemons, and services protected with TCP Wrappers) uses a combination of reverse (IP to hostname) and forward (hostname to IP) DNS lookups. A remote server will take the IP address it receives as the source of a connection and reverse that address to a hostname. Looking up this hostname yields an IP address. If the original address received as the source of the connection doesn't match the IP address retrieved from DNS, then the remote server assumes the connection comes from an attacker trying to spoof IP addresses and/or DNS information and will not allow the remote machine to connect.



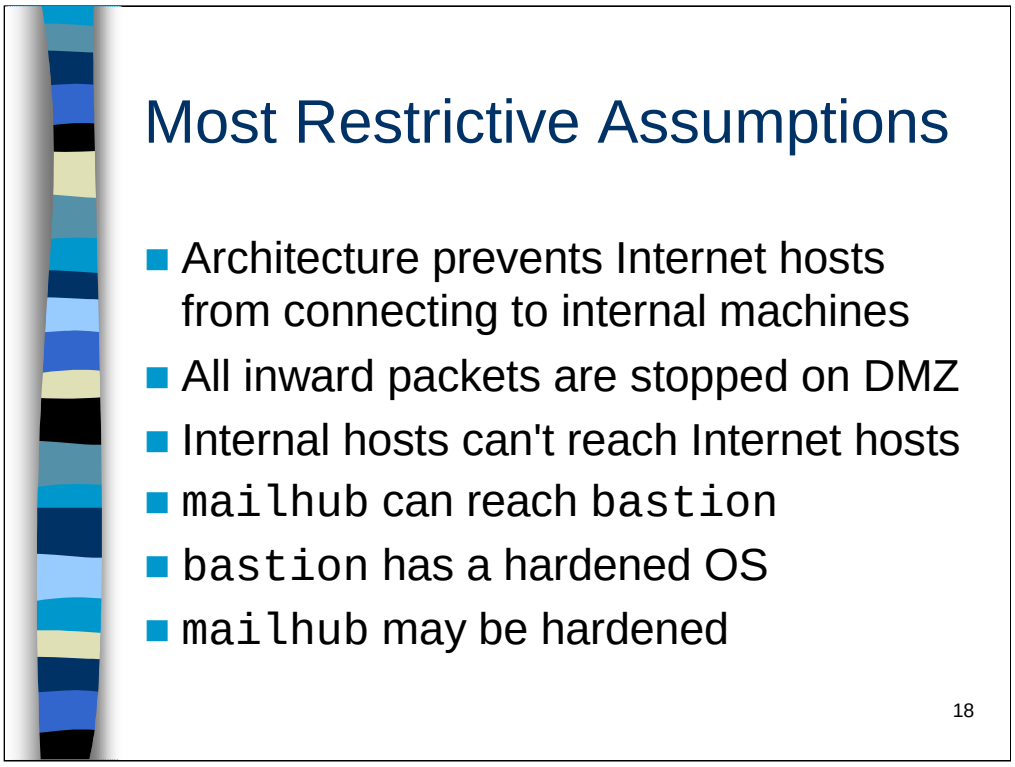
The implication of this form of authentication is that sites should maintain both forward and reverse DNS information for all hosts that are capable of connecting directly to the Internet. This information must be made available in your "external" DNS database in a split-horizon configuration.

Unfortunately, if all of your hosts can connect directly to the Internet, that means that you have to advertise information about all of your machines in your external split-horizon database– largely nullifying the usefulness of a split-horizon configuration. On the other hand, if your organization uses Network Address Translation (NAT) or proxy servers, you generally only have to advertise information about your NAT pool or proxy servers and split-horizon is useful.

Note that even if you have to advertise information about all of you internal hosts, you don't need to tell the outside world your HINFO or TXT records, so split-horizon can be somewhat useful. It is also sometimes useful to present one set of mail routing information (MX records) to the outside world and have a different configuration internally.

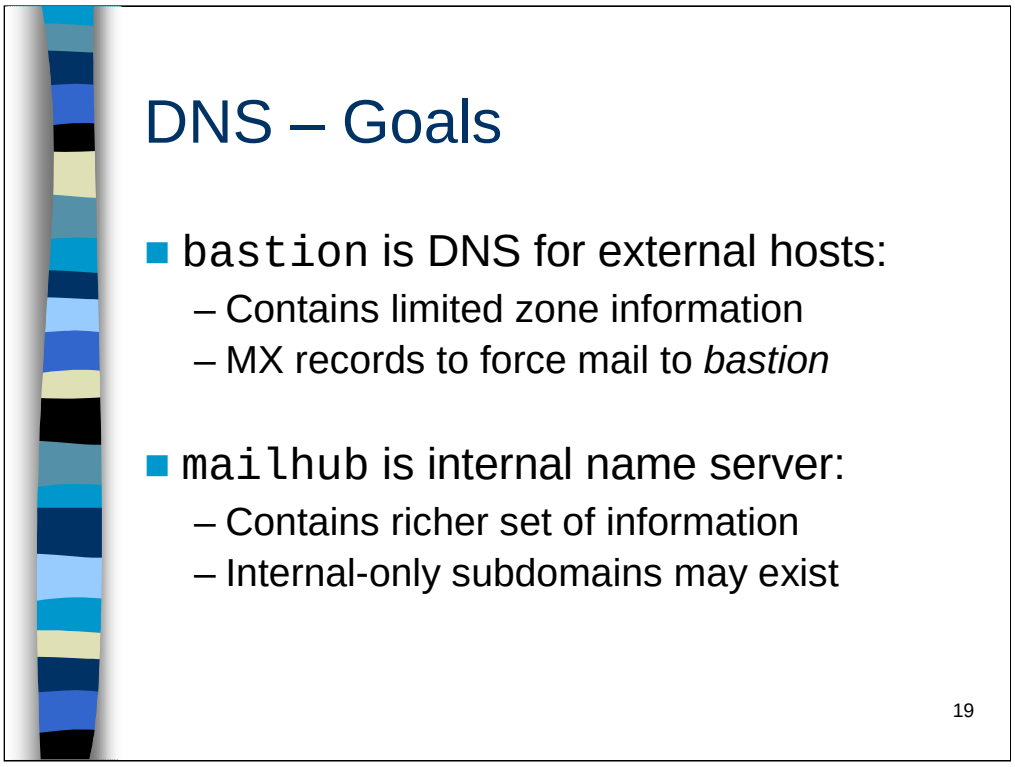


Here is a diagram of a simple, but fairly typical firewall configuration in use at many organizations today. The organization has a multi-legged firewall which connected the external Internet to both a semi-private de-militarized zone (DMZ) network and a private internal corporate network. The DMZ network is where an organization would put its Web and FTP servers and any other machines that the outside world needed to reach– and for purposes of this example a machine called `bastion` which will be the "external" DNS server for our split-horizon example. On the internal network there is a machine `mailhub` which acts as the primary server for the "internal" DNS.



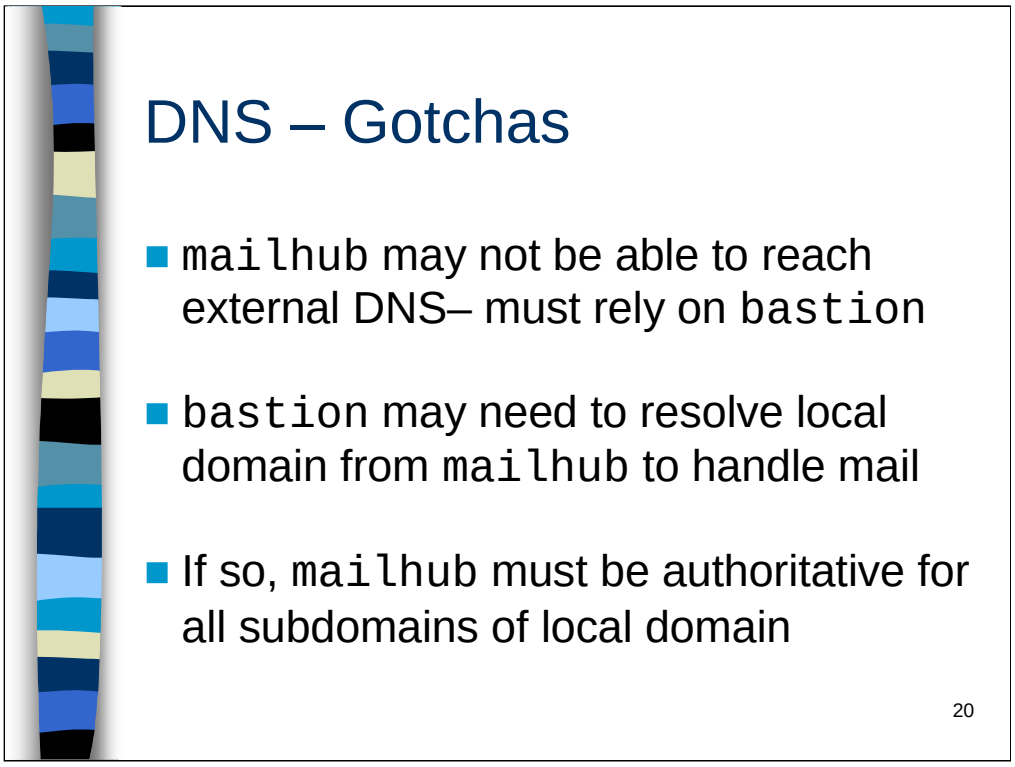
For purposes of example, we shall assume the most restrictive possible firewall configuration– assuming you have a working configuration for this environment, you can easily adapt the configuration for your own (less restrictive) firewall setup. In particular, we assume that hosts on the Internet can only route packets to hosts on the DMZ network and that the DMZ is the only network that is allowed to communicate with the internal corporate network. Internet machines are not allowed to communicate with "internal" machines and vice versa.

It's a good idea to spend effort removing dangerous services (read: everything not absolutely needed for mail and DNS transport) from the `bastion` host. Everybody in the world will be trying to break into this system. But it's also a good idea to harden your internal infrastructure servers (like the `mailhub` machine in this example) against both potential remote attacks and against the insider threat.



Again, the idea is that the external DNS for your organization will contain the bare minimum necessary for your organization to successfully conduct business on the Internet. It should not advertise internal hostnames, hardware or operating system information (HINFO and TXT records), and the like since this information could be useful to crackers trying to penetrate your network.

The internal DNS information at your site will contain complete information for your domain including useful host aliases, subdomain information, and system information. This DNS information will only be visible to hosts owned by your organization.



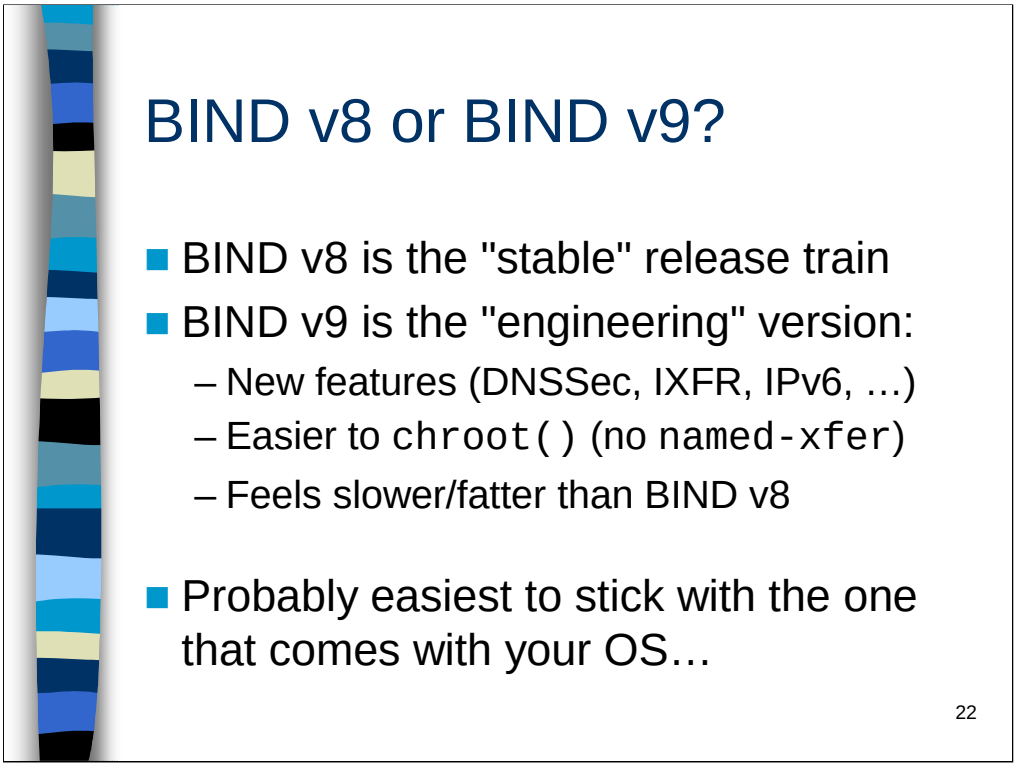
This is a very restrictive environment to work in (remember that there is a trade-off between security and ease-of-use). Since your internal hosts aren't able to reach other hosts on the Internet, they have to rely on the `bastion` host to make proxy DNS requests on their behalf.

On the other hand, `bastion` may not have enough DNS information locally to properly deliver inbound mail. It needs to be able to query the name server on the internal `mailhub` in order to get at the richer set of DNS information available to internal hosts. Note that there is no difficulty in having a host run a name server locally but get DNS information from a completely different machine.

One very unexpected item is that whatever internal DNS server your external `mailhub` relies on must have complete and authoritative information for every domain and subdomain in your organization. Because most of your DNS information will be for internal consumption only, the standard mechanisms for tracking down authoritative zone information break down, and so your external `mailhub` must be able to get authoritative information on its first query.

In reality, if the `bastion` is only exchanging mail with `mailhub`, you could put `mailhub` in `bastion`'s `/etc/hosts` file and have `bastion` just use its own name server for all other queries (for hosts outside the company). It's likely, however, that your mail architecture may change in the future and allowing `bastion` to resolve any internal hostname is probably a good idea for long-term maintainability.

Configuring BIND presents specific configuration examples for the DNS architecture introduced in the *Split-Horizon DNS* section, and introduces many of the new security features in BIND v8/v9.



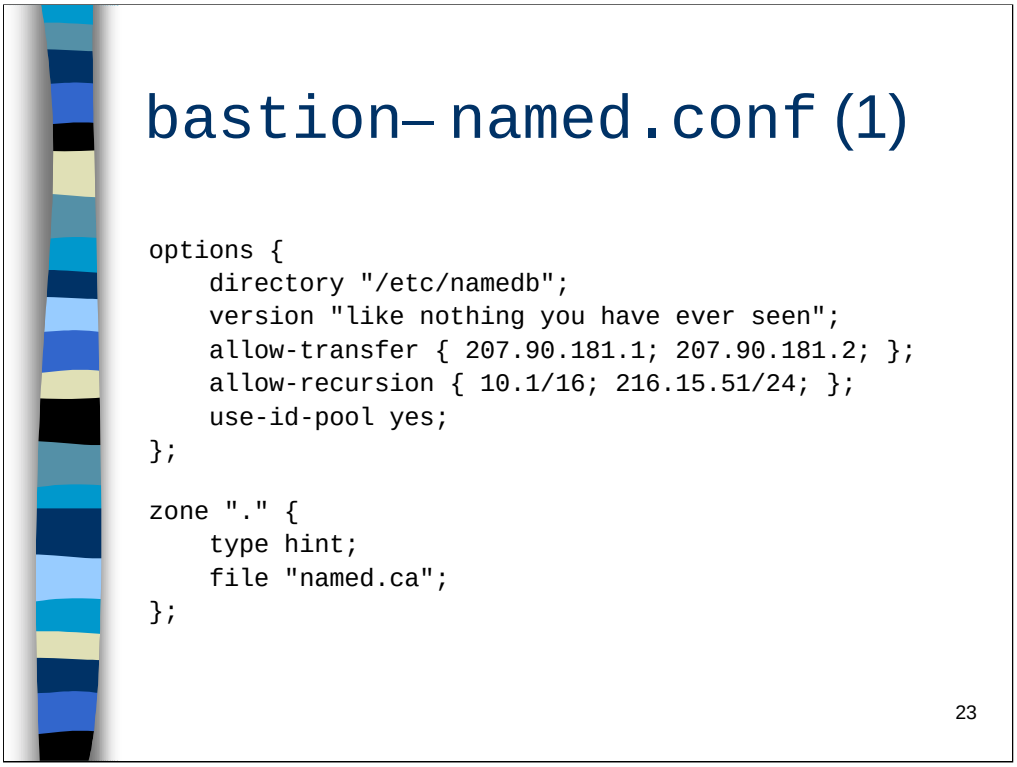
There are actually three different versions of BIND running around currently. BIND v4 is deprecated and only updates for critical security fixes are being produced. If you are still using BIND v4 for some reason, you should definitely upgrade to a newer version.

BIND v8 is the current "stable" release of BIND that ships with many of the commercial Unix distributions. BIND v9, on the other hand, is the version where new extensions to the DNS protocols are being implemented (although in some cases, this functionality is being back-ported to BIND v8 as well). Most of the Open Source operating systems seem to be shipping with BIND v9.

Generally speaking BIND v9 is "less stable" than BIND v8, but it seems to run with few or no problems. BIND v9 does appear to be noticeably slower and more resource intensive than BIND v8, though I've not done any formal benchmarking between the two versions.

As a minor note, it's generally easier to `chroot()` BIND v9 servers. This is because BIND v8 requires the external `named-xfer` program for doing zone transfers when you're a slave name server. So not only do you need to copy this program into your `chroot()` directory, but you also need the shared libraries that this program depends on. BIND v9 has the `named-xfer` functionality built into the master daemon, so there's no need for an extra program in the `chroot()` directory structure.

The reality is that it's probably easiest for most sites to use the version of BIND supplied by their OS vendor. If you prefer to build your own version from source, my recommendation is to run BIND v8 unless you particularly need the new features in BIND v9. I'm not convinced we've flushed as many security problems out of the BIND v9 source base as we have in BIND v8.



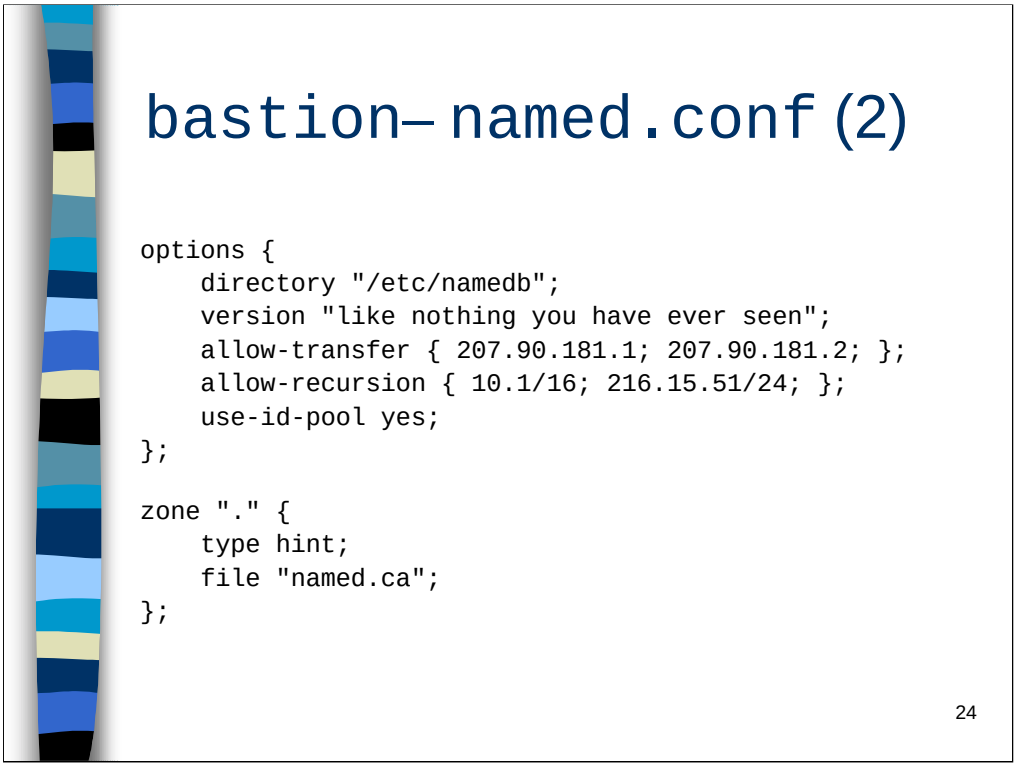
(slide 1 of 3)

This is the new BIND v8 syntax configuration file. If you have old v4 `named.boot` files, you can convert them to v8 `named.conf` files with the `named-bootconf.pl` script provided in the `src/bin/named` directory. Note that the format of the zone database files *did not* change, so you can keep using any old zone files you have lying around.

The `options` block applies globally to the rest of the configuration. The `directory` statement means that all other filenames are relative to `/etc/namedb` (e.g. `/etc/namedb/named.ca`). In fact, `directory` actually changes the value of `DESTRUN`, the default working directory for BIND.

As demonstrated earlier, it is possible for outsiders to query your running name server and find out what version of BIND you are running. Since certain versions of BIND have known vulnerabilities, you want to hide what version your name servers are running. The `version` option allows the administrator to specify an arbitrary string instead of the actual BIND version number. In this example the attacker will know you're running at least BIND v8, but not what version– theoretically, you could change the version string to be a valid version string for an earlier version (e.g. `"4.9.7"`) to confuse attackers.

You should use the `allow-transfer` option to restrict zone transfers to only those machines which are legitimate secondary servers for your domains. Note that you should also configure your firewall to block zone transfers from the outside world as an extra layer of security.

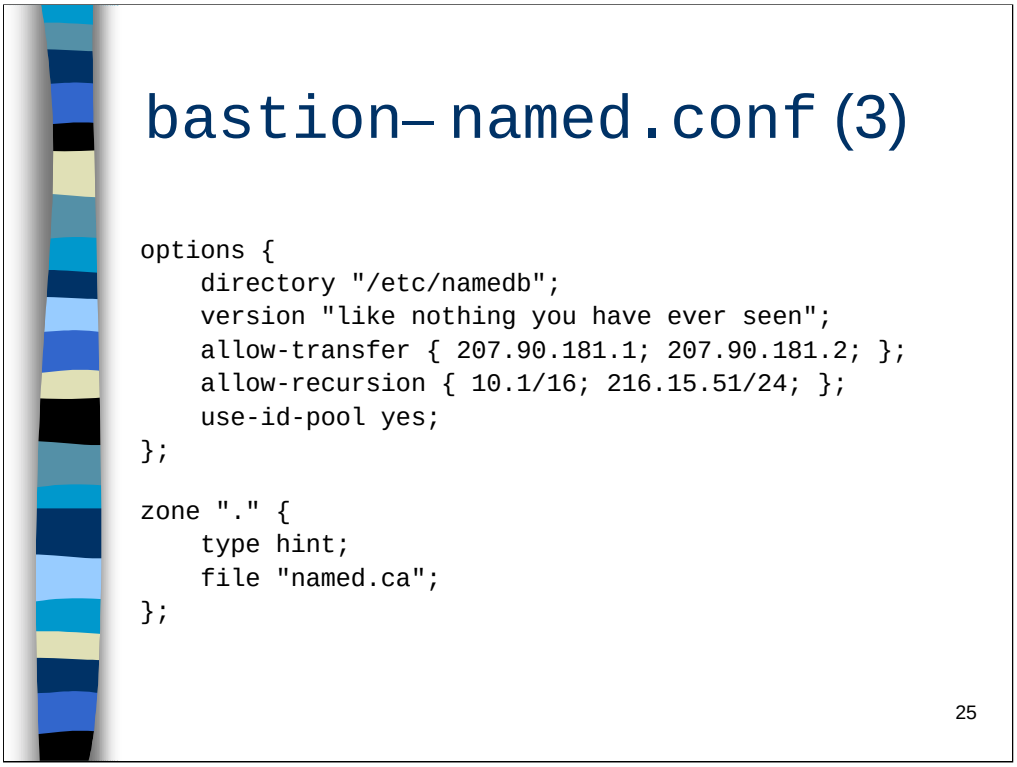


(slide 2 of 3)

Normally a name server receives a request from an external name server, responds with the best information it has, and does no further work-- this is a *non-recursive* (sometimes referred to as an *iterative*) *query*. However, client resolvers (and sometimes other name servers as we'll see in a moment) generally do *recursive queries* when communicating with their local name server– that is, they rely on their local name server to do all the work required to look up a given piece of information (including contacting root name servers and name servers at other organizations).

Generally, only machines that you own should be making recursive queries via your name server– the `allow-recursion` option specifies the ranges of IP addresses which are allowed to do recursive queries via this name server. If you allow outsiders to do recursive queries via your name servers, you make yourself more vulnerable to certain types of cache poisoning attacks.

Note that `version`, `allow-transfer`, and `allow-recursion` options are most appropriate for your external name servers. You can apply them to your internal name servers as well, but you may find this more of an administrative hassle than anything. Aside from having to maintain the lists of IP addresses in the `allow-transfer` and `allow-recursion` options, it is occasionally useful to be able to query your own internal name servers and find out what version of BIND they're running (so you know which ones to upgrade).

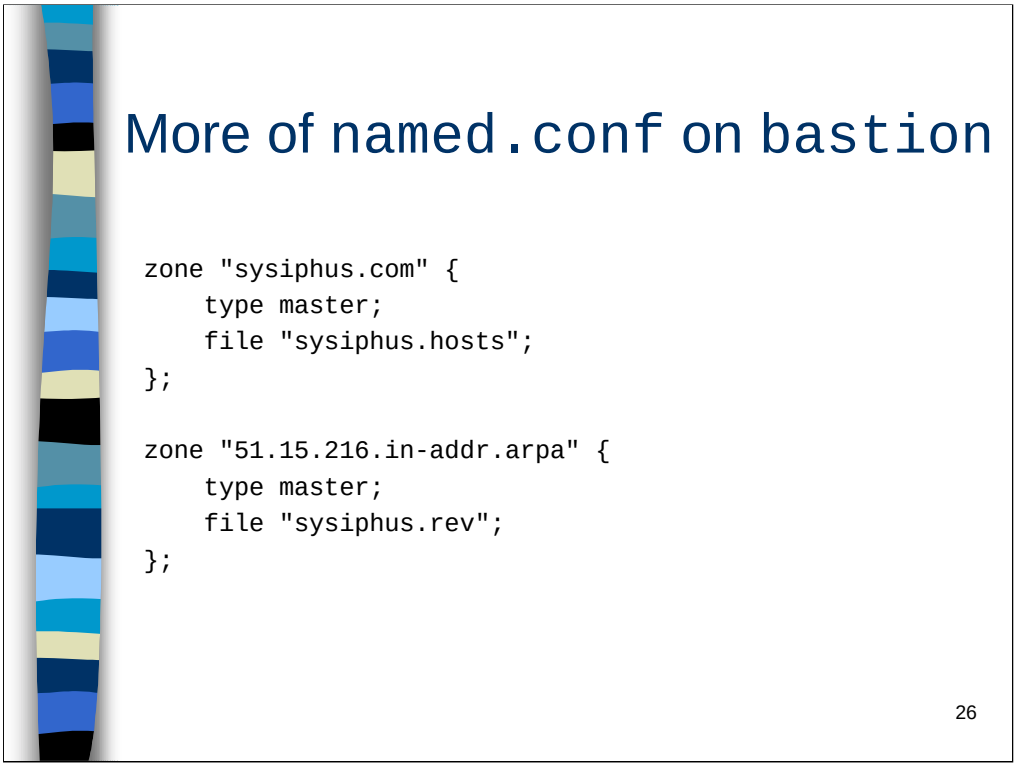


(slide 3 of 3)

The `use-id-pool` option became available in BIND v8.2. It causes your name server to use a better randomization algorithm for choosing query ID numbers (this is the default for BIND v9, so this option is not needed on BIND v9 servers). Remember that attackers sometimes try and guess query ID numbers as part of DNS spoofing attacks, so anything you can do to make their jobs harder is a good idea.

The latest version of the `named.ca` file is available from the InterNIC as

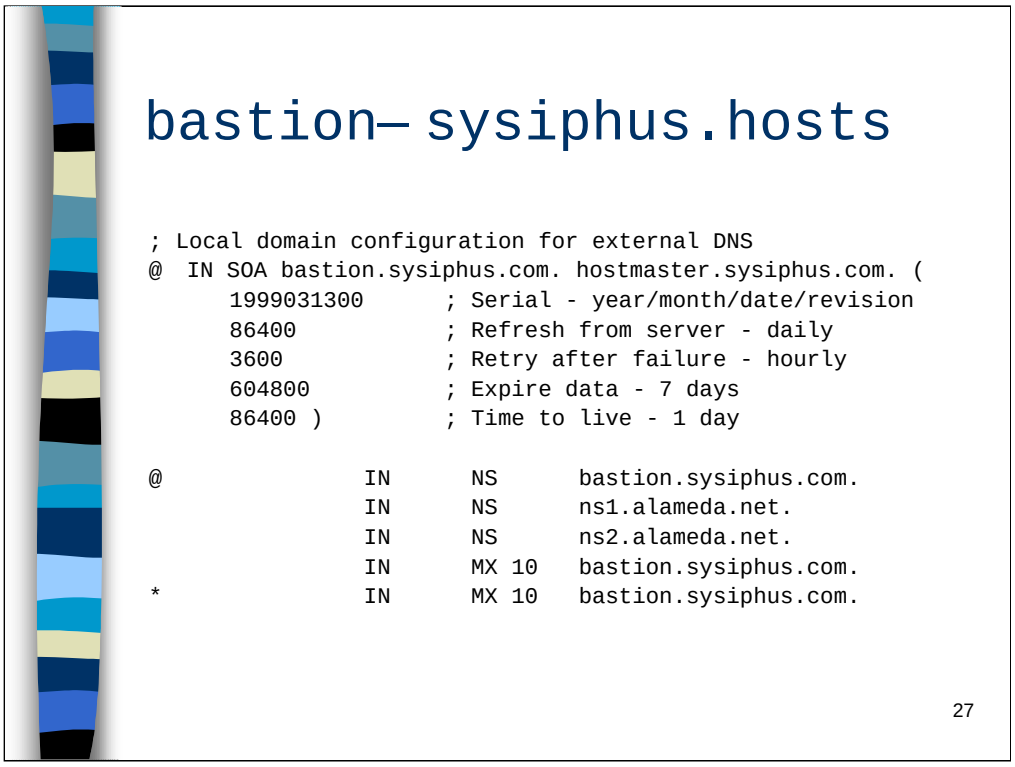
```
ftp://ftp.rs.internic.net/domain/named.ca
```



Now, for each zone, we specify whether this name server is the `master` server for the domain (`primary` server in BIND v4 speak) or a `slave` (`secondary` server). Then we give a filename (again, relative to `/etc/namedb`) where the zone data can be found.

The `in-addr.arpa` domain is where you maintain the mappings between IP addresses you own and hostnames. Read the domain from right to left– we are saying that bastion is the master server for all addresses in the `216.15.51.0` network.

Note that `sysiphus.com` is an example domain used in this course. Be sure to replace it with your local domain name when you get back to the office!

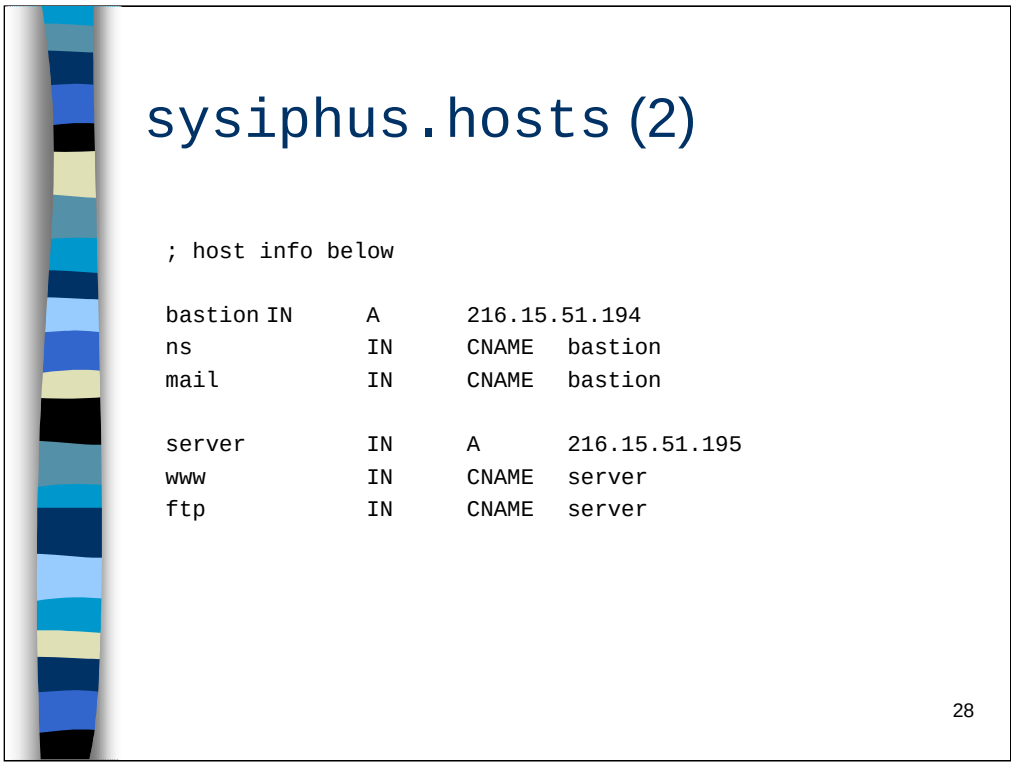


The SOA ("Start of Authority") record defines global properties for your domain and lists contact information for the domain (`hostmaster.sysiphus.com` instead of `hostmaster@sysiphus.com` since `@` is a restricted character).

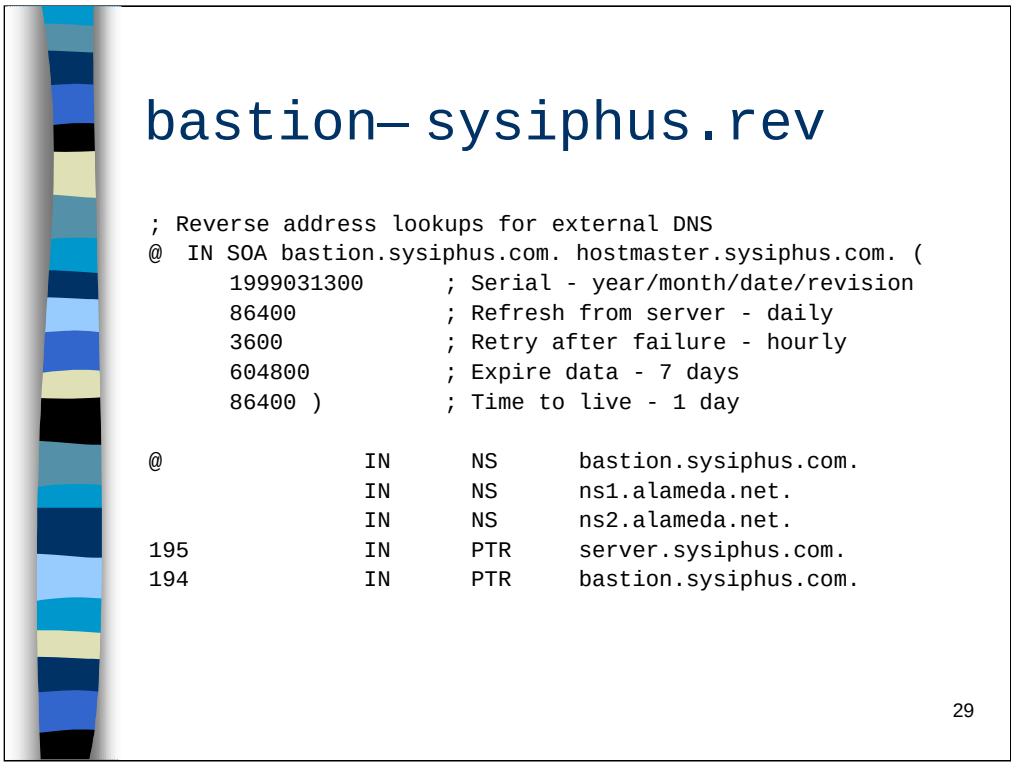
Generally speaking, the values in the SOA record above are appropriate for sites that don't make a great deal of DNS updates-- your external DNS should be relatively static. Don't forget to increment the serial number value every time you make changes to your DNS information (here we're using a date-based serial number: the last two digits are provided in case you make more than one update on a given day).

You should always have at least two advertised name servers for your domain, just in case the network partitions. If these hosts can be situated on two different major service provider networks, so much the better. Note that the name servers listed above belong to the author's (former) ISP– please do not advertise these name servers as slave servers for your domain!

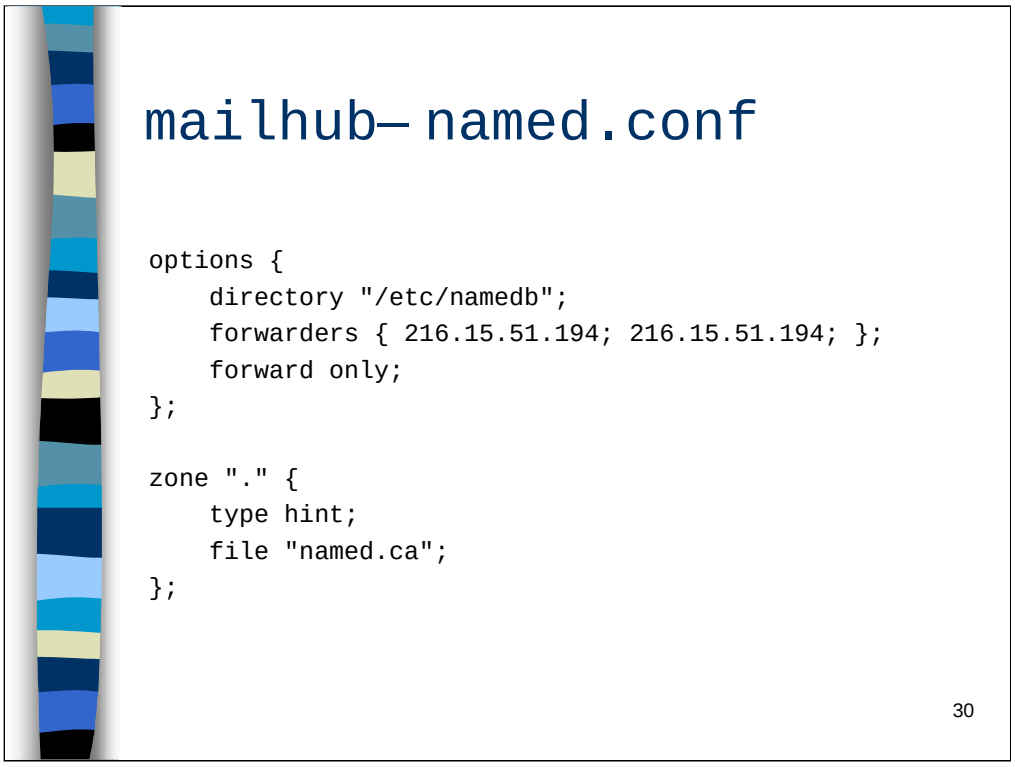
The MX ("Mail Exchanger") records are sending all mail for `sysiphus.com` and `*.sysiphus.com` to `bastion`. Wildcard MX records are dangerous if you're not using split-horizon DNS, but administratively convenient unless you have some automated mechanism for maintaining your DNS tables.



Here are some standard forward address records that might appear in your external DNS information. A `CNAME` (*canonical name*) record is a mechanism for setting up an alias for a given machine. Note that we use `CNAME`s to associate functional names with specific machines– we could just as easily have configured multiple `A` records all pointing at the same IP address. While religious debate runs hot and heavy about whether `CNAME`s or `A` records should be used in these situations, the reality is that *it fundamentally doesn't matter*– there is no substantial technical reason to prefer one over the other.



This file is very similar to the `sysiphus.hosts` file except that it contains `PTR` records instead of `A` and `CNAME` records. Note that the leftmost column of the `PTR` record entries reverses the order of the octets of the hosts' addresses. The trailing dot at the end of the `PTR` record is vitally important.

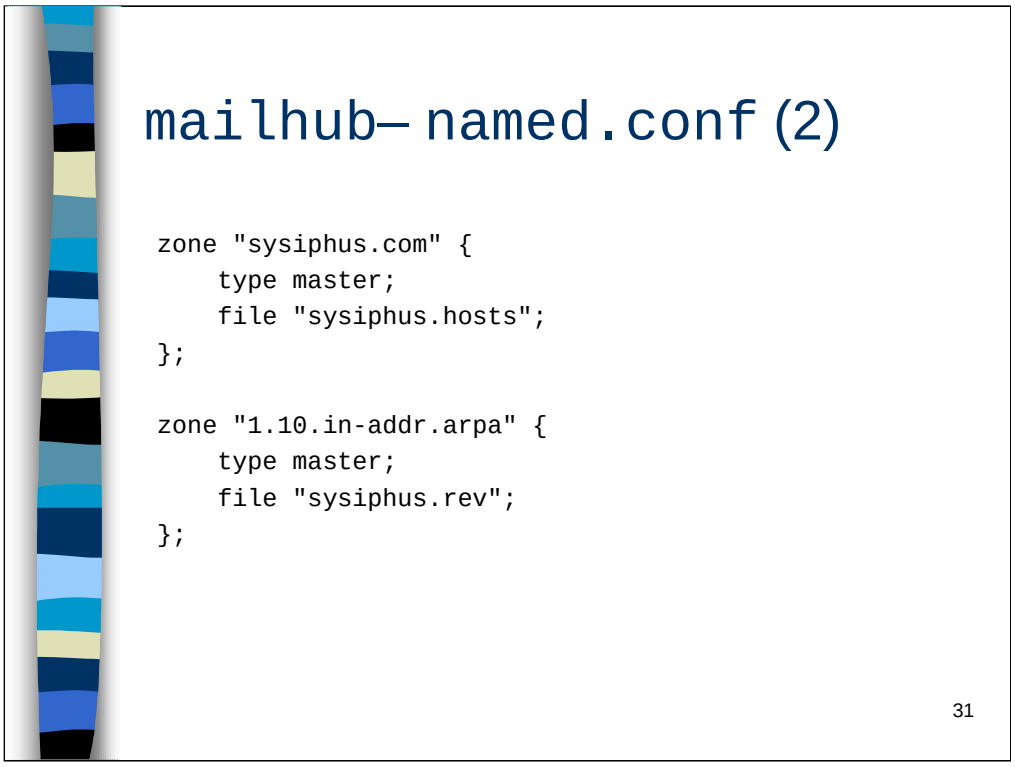


Remember that we are assuming our internal name server is unable to reach Internet connected hosts for DNS information. The `forwarders` lines cause the internal name server to send all external DNS queries to `bastion` to be resolved. In the event the query fails, the local server would normally attempt to contact a remote name server, but the "`forward only`" directive prevents this behavior.

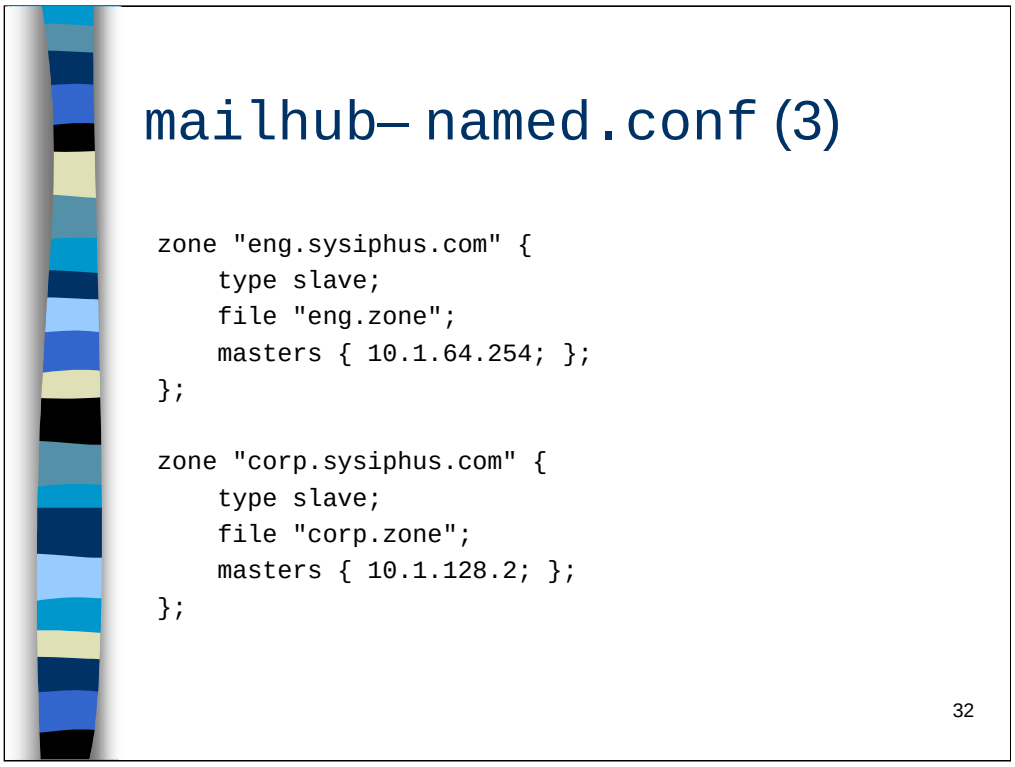
Note that we are listing the IP address of `bastion` twice in the `forwarders` directive. The timeout on `forwarders` queries is actually shorter than the timeout on the normal DNS queries made by the external machine `bastion`. Hence we list the IP address of `bastion` twice so that the local server will retry in the event it fails to get a response to its first query.

In the "real world" of course, you would probably have multiple external name servers to forward queries to in order to provide some redundancy. If you had name servers A, B, and C to list in your `forwarders` directive, you are probably better off listing their IP addresses `{ A; B; C; A; B; C; }` rather than `{ A; A; B; B; C; C; }`. After all, the most likely reason to not get a response to your first query is that machine A is down, so you want to try another name server first before re-trying server A.

All other configuration statements in this part of the file are the same as they were on the `bastion` host.

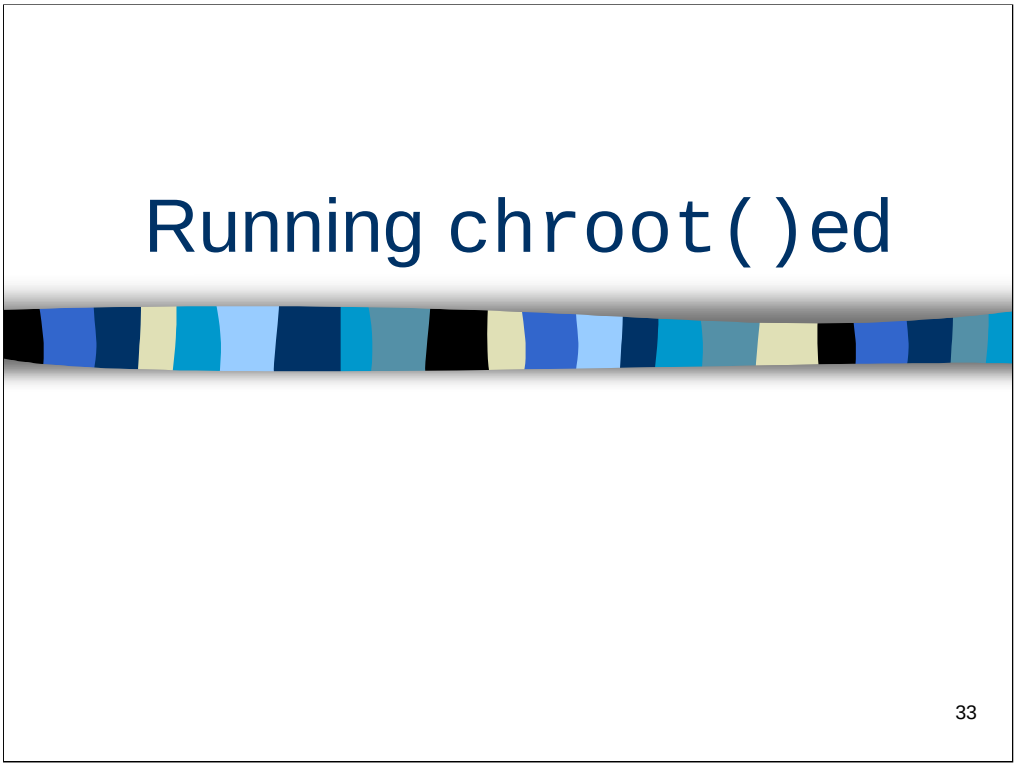


This section of the file defines the zone files for the internal copy of the `sysiphus.com` zone and the `in-addr.arpa` domain for the internal network (remember the zone file names are relative to `/etc/namedb`). The main drawback to split-horizon DNS is that some information is necessarily duplicated between the internal and external versions of the `sysiphus.com` zone– primarily address information for the external "public" servers. Some sites choose to develop scripts which automatically dump out the "external" zone information from the internal zone files in order to avoid mistakes where one copy of the information is updated and not the other.

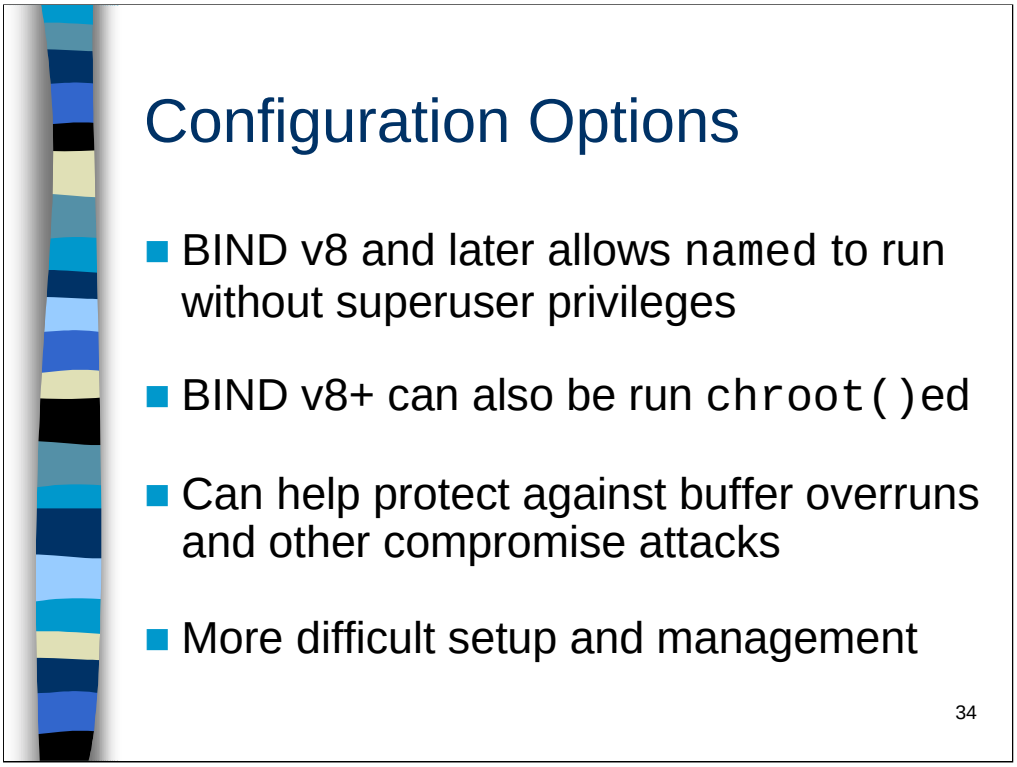


In this section of the file we see some internal domains that are not known to the outside world. In this case `mailhub` is a `slave` server of these domains and gets its zone information from some other internal `master` nameserver.

Remember the zone file names are relative to `/etc/namedb`.



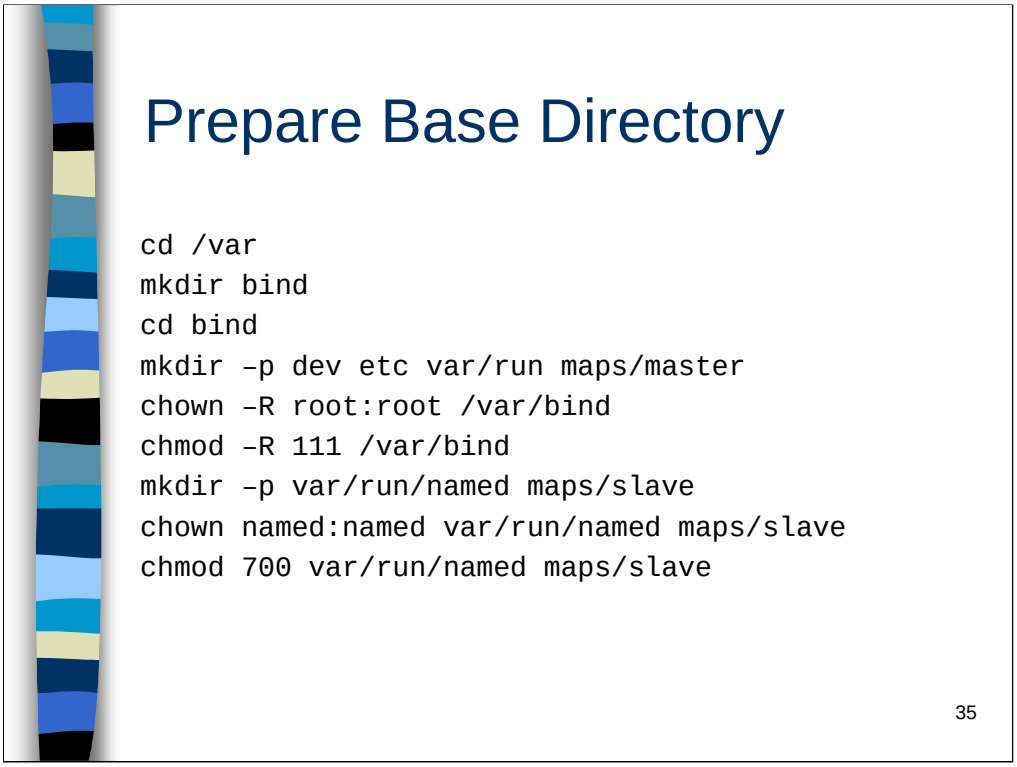
Running a Name Server presents additional hints on administering BIND, including how to run BIND without superuser privileges and in a `chroot()`ed environment.



One of the improvements that appeared with the release of BIND v8 was the ability to run your name server as some user other than root. This means that in the event of a successful buffer overflow attack or other remote compromise, the attacker will only have the privileges of some non-root user. This is a significant security enhancement.

BIND v8 and later also allow the name server to be run in a captive `chroot()`ed environment. This means the attacker will only be able to manipulate files in the `chroot()`ed environment even if they successfully subvert your name server daemon.

As we will see, setting up this environment is somewhat complicated– but not that bad if somebody figures it out for you!

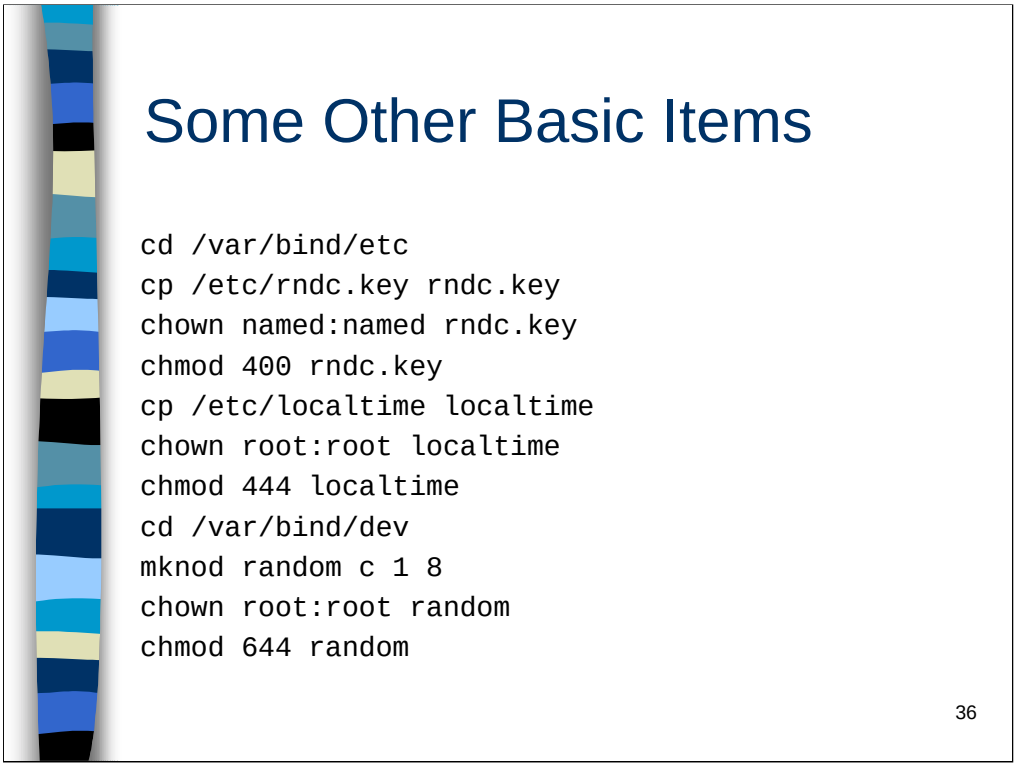


Note that in this section we will be showing a Red Hat Linux specific configuration procedure using BIND v9 (the default version that ships with Red Hat). Note also that this procedure assumes you have already set up an unprivileged `named` user and group ID, which is the default for Red Hat. You will need to be the super user to perform these configuration steps.

First we need to create the basic `chroot()`ed directory hierarchy we will be using for the name server. You can put this directory structure anywhere in your file system that you wish– in this case, we will be rooting our tree under `/var/bind`.

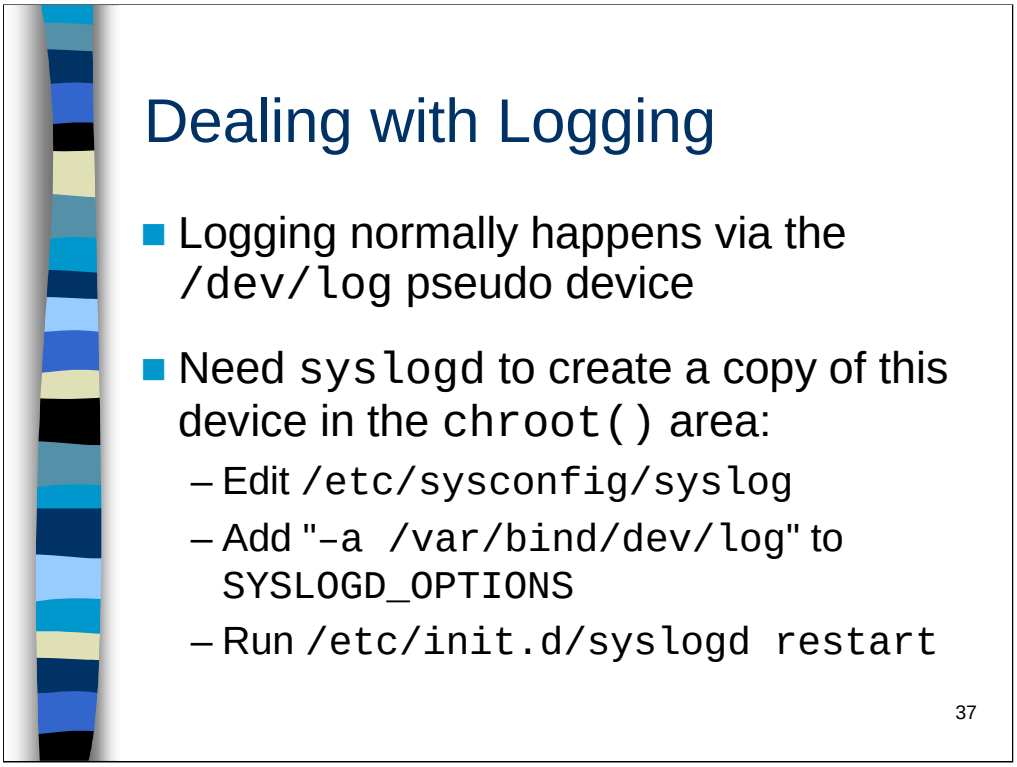
We wish to make the directory permissions as restrictive as possible. In particular, any directories that we don't want the name server to write into should be owned by `root` (since the name server will be running as our unprivileged `named` user). Also, most directories should simply be mode `111` (only the execute bit set) so that the `named` user can read files inside the directories, but not get directory listings.

However, there will need to be a couple of directories where the `named` user can actually write data. `/var/run/named` will be used as a default working directory for the name server (core files and debugging info go here) and is where the Red Hat named drops its `named.pid` file. We also need a directory where slave zone files can get written (assuming we're a slave name server– otherwise this directory is not needed).



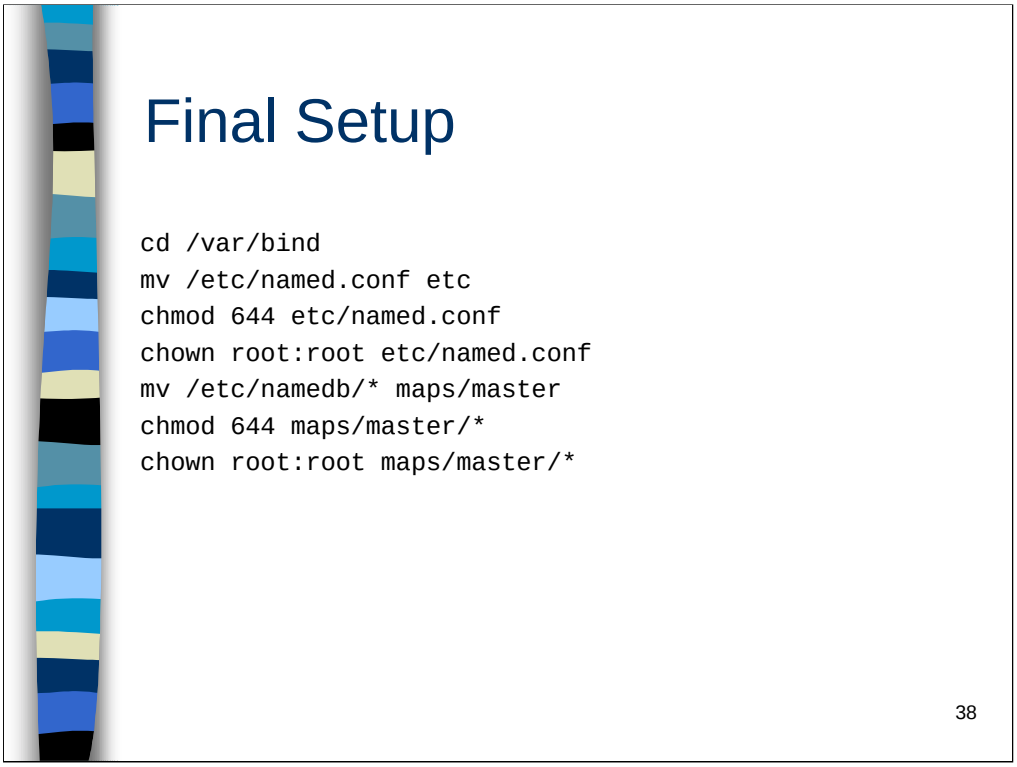
The name server needs some additional configuration files to function properly. The localtime file contains information about the machine's local time zone and ensures that log file time stamps appear to be correct for the local time zone. The rndc.key file is a random session key used by the secure DNS administration utility that ships with BIND v9. These files should just be copied out of the standard system configuration directories.

The name server also needs a copy of the system /dev/random device to function properly. The arguments to `mknod` are a `c` indicating that this is character-special device file (don't worry about what this means) and the major and minor device numbers appropriate for the device. All of this information (plus the proper permissions and ownership for this device) can be gathered by doing an `ls -l /dev/random` and then copying the parameters into the `mknod` command.



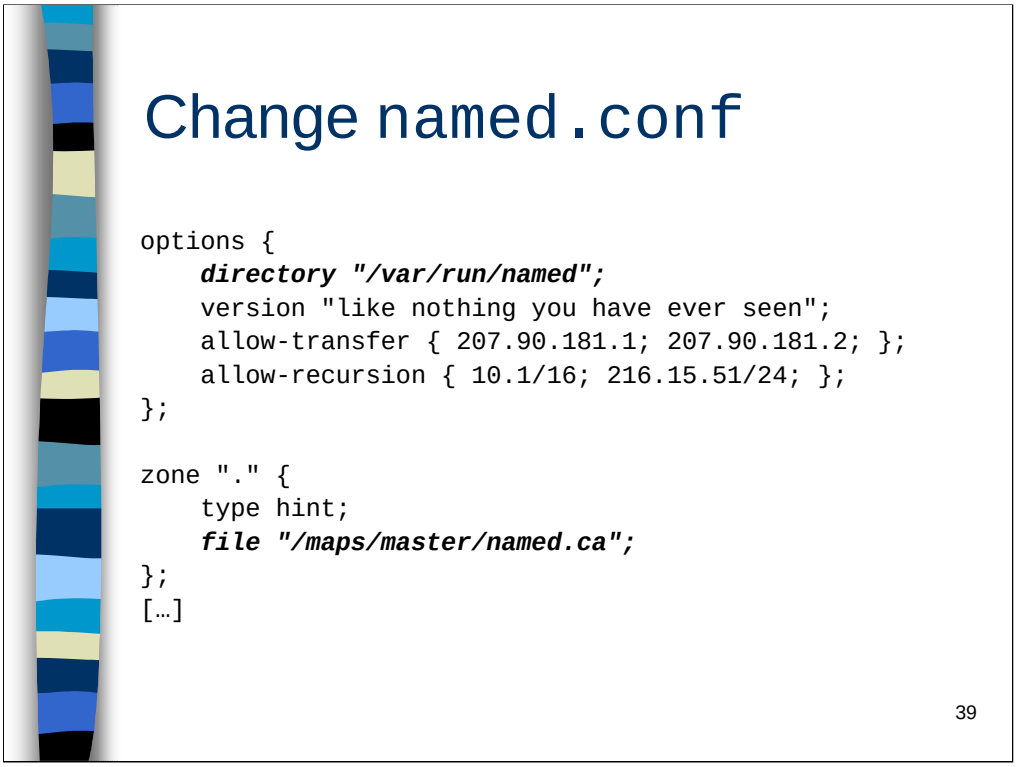
On many Unix systems, local logging happens via the `/dev/log` "device". Actually this is not a real device file at all, but instead is a named pipe created by the Syslog daemon. So for logging to happen in our `chroot()` directory, we need Syslog to create a copy of this device under `/var/bind/dev`.

Use the `-a` option to specify the location of an extra copy of this pseudo-device (any number of `-a` options can be specified on the command line). For Red Hat systems at least, this option can be specified via the `SYSLOGD_OPTIONS` variable in `/etc/sysconfig/syslog`. Once this file has been updated, just restart `syslogd`.



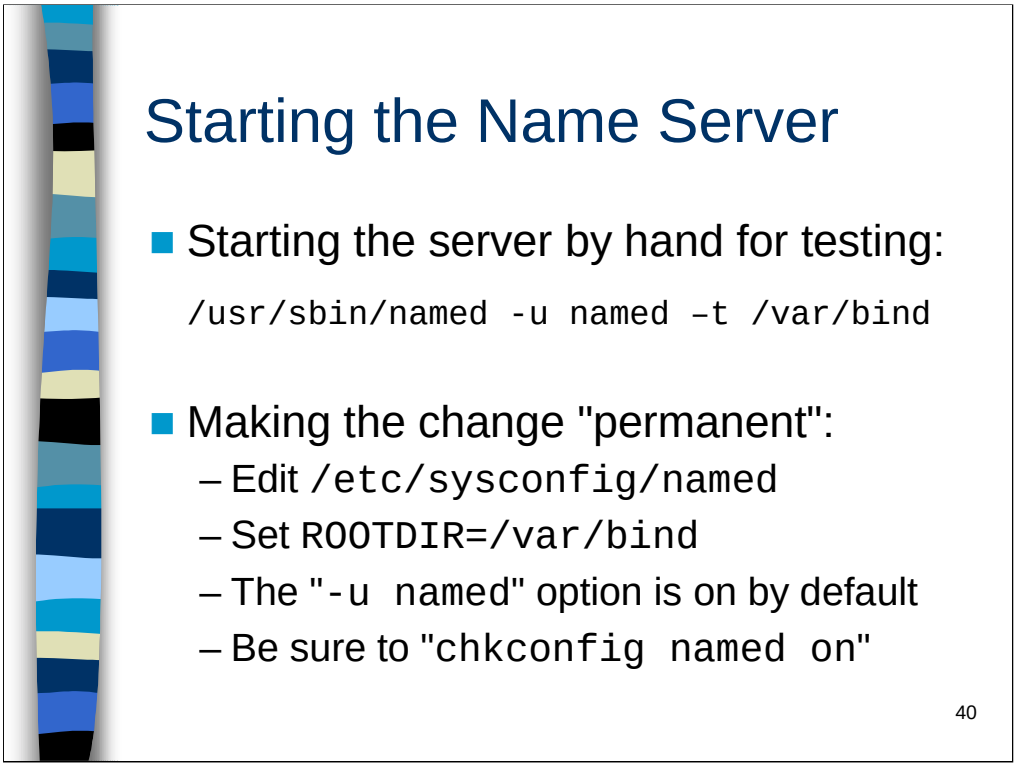
We need to move `named.conf` to the `chroot()`ed hierarchy since `named` `chroot()`s itself before even opening the `named.conf` file. It turns out we will have to make some modifications to `named.conf` for the directory structure in `chroot()`ed environment (see next slide).

We want to make sure that the primary zone files (stored in `/var/bind/maps/master`) and the `named.conf` file cannot be overwritten in the event of a name server compromise, so we make them owned by `root` and *not* the `named` user. However, these files must be *readable* by the `named` user, so that the name server can load information from them when it starts up.



Recall earlier we mentioned that the `directory` option actually changes the default working directory for BIND (the value of `DESTRUN` that was compiled into the binary). If we want our `chroot()`ed name server to write its debugging files in some particular location, we need to set the `directory` option appropriately. However, changing the `directory` option means we need to use absolute pathnames in all of our `zone` declarations, rather than relative pathnames.

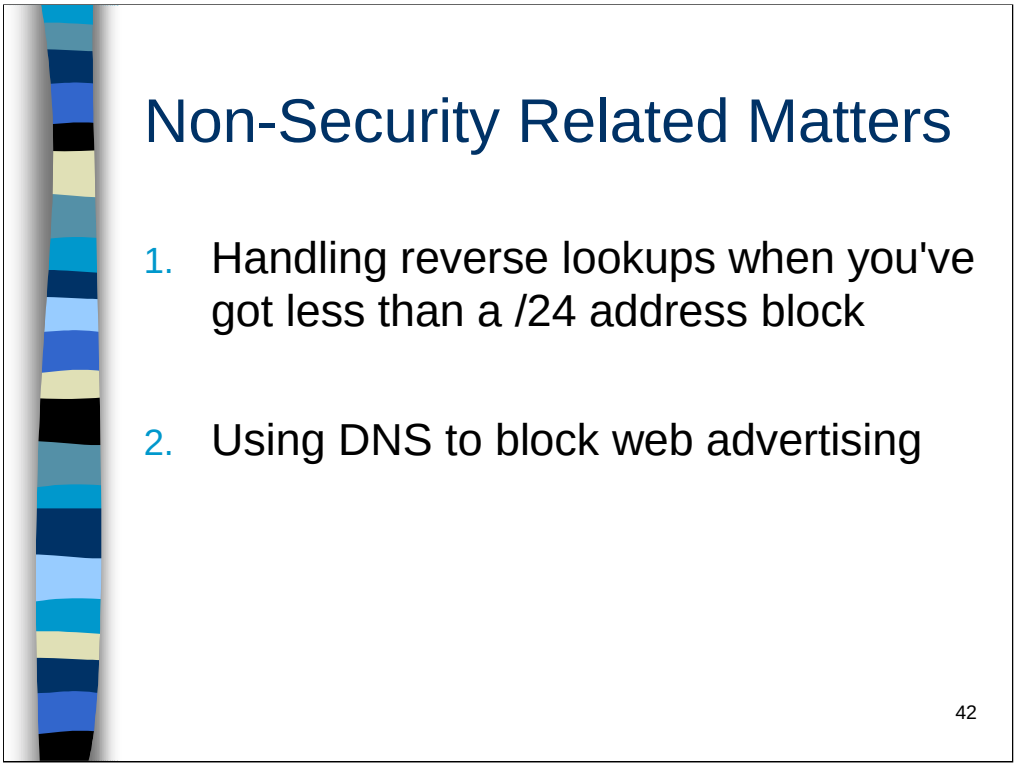
Note that since `named` `chroot()`s prior to reading `named.conf`, all of the pathnames listed here are rooted within the `/var/bind` directory– i.e., `/var/bind/var/run/named` and `/var/bind/maps/master/named.ca`.



You can start the `chroot()`ed named from the command line as shown above. The `–u` option specifies the user ID that the name server should run under, and `–t` specifies the root of the `chroot()`ed hierarchy.

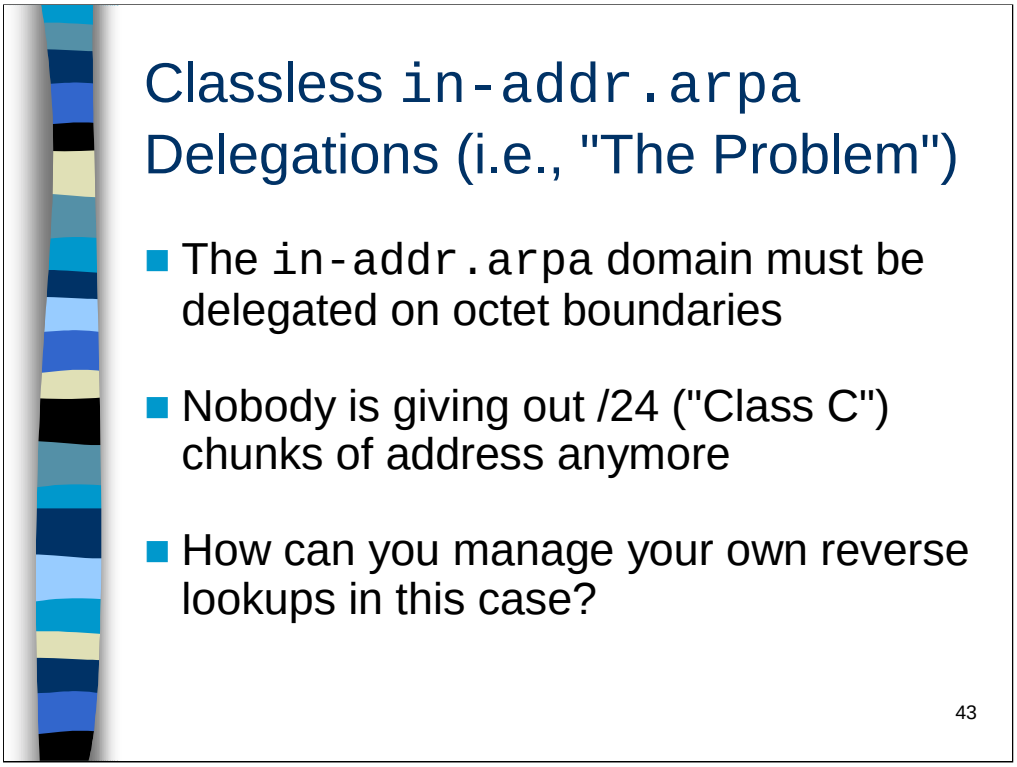
Note that you will have to make appropriate modifications to your boot scripts so that the `chroot()`ed name server is started at boot time as well. On Red Hat systems, you can set the `chroot()` directory path using the `ROOTDIR` variable in `/etc/sysconfig/named`. On Red Hat systems at least, starting the name server as an unprivileged user is the default configuration– you might have to make modifications on other Unix variants to run `named` as a non-root user.

Just a couple of other quick hacks that you might find useful…

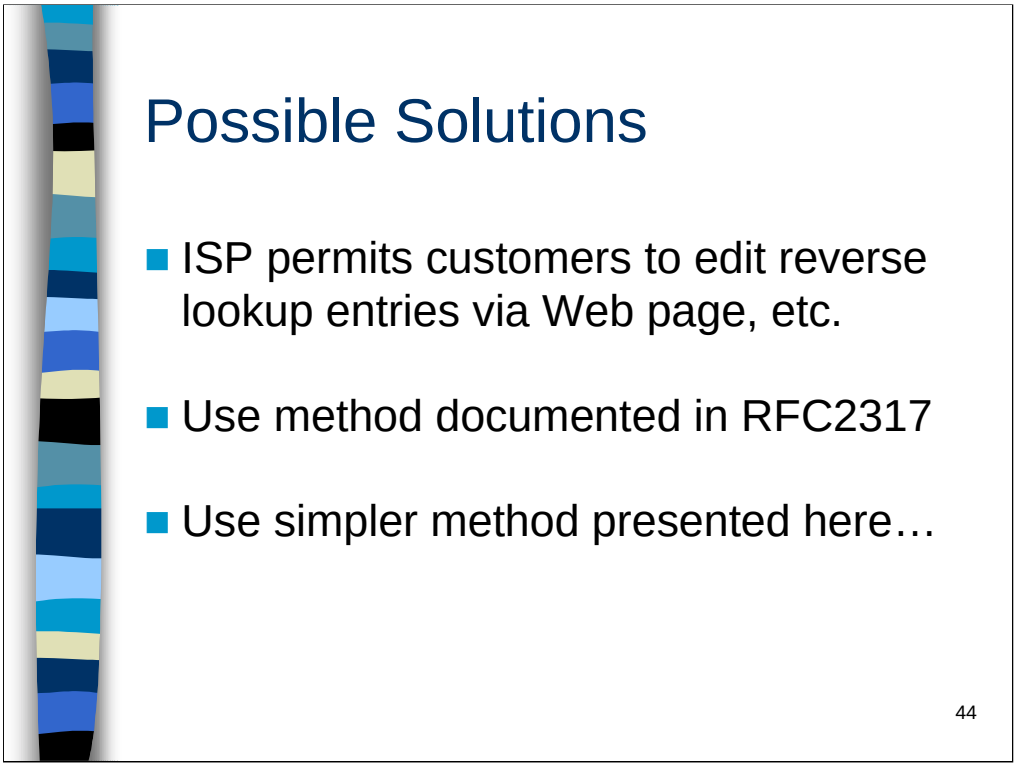


As Internet address space becomes more scarce, you see people getting smaller and smaller netblocks from their ISPs. One question I get frequently is how to handle reverse DNS lookups (IP address to hostname mappings in the `in-addr.arpa` domain) in this situation. So we'll look at the standard hacks for dealing with that.

I've recently gotten fed up with all the blinking, flashing, singing, dancing web advertising and pop-up windows on most Internet web sites. If you run your own local DNS servers, you can actually use DNS to filter out a huge amount of this nonsense. More on this at the end of this section.



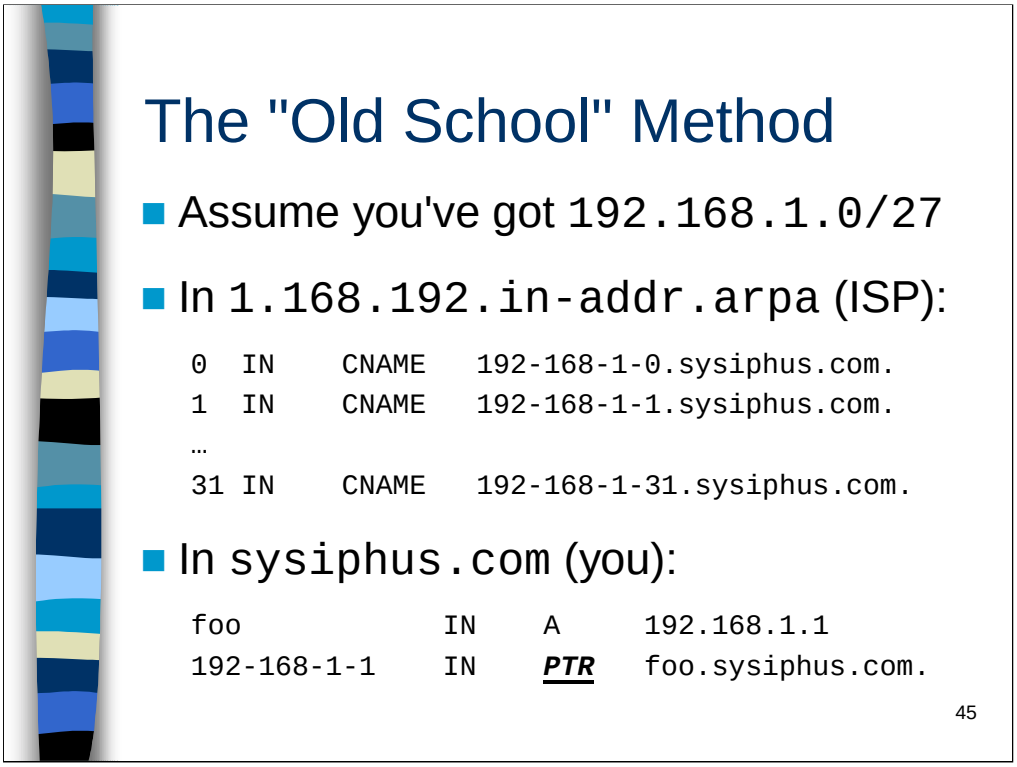
OK, suppose you've been handed a /27 netblock by your ISP (32 addresses) but you want to control your own reverse lookup domain. The bad news is that normal `in-addr.arpa` delegations can only happen on octet boundaries, so unless you've got a /24 or better, you're in trouble. However, some interesting hacks have developed over the years to deal with this issue…



Some ISPs just set up a web form (or other mechanism) to allow customers to manage their own reverse lookups. Basically this is a web form that results in information being changed on the remote ISP's servers. Assuming such access is handled in a secure fashion, this is not an unreasonable approach.

However, sometimes the ISP would prefer to not be bothered and offload the upload responsibility to the customers' servers. Customers may prefer to be more in charge of their own destiny as well. RFC2317 documents one mechanism that allows ISPs to offload this problem onto their customers. This RFC documents a procedure that is really just an updated version of a technique that's been around for some time.

Frankly, I think the procedure as documented in the RFC is unnecessarily tricky. Or maybe I just prefer the first way I learned to do this…



Let's suppose you've been given the netblock 192.168.1.0/27– in other words, you own IP addresses 192.168.1.0 through 192.168.1.31.

Now your ISP controls the DNS reverse lookup domain for this network, which is part of the `1.168.192.in-addr.arpa zone`. Your ISP populates their DNS zone file for this domain with a `CNAME` record for each individual IP address, which points to a record with a particular naming convention in their customer's normal DNS domain. The result is that when somebody goes to look up one of your IP addresses like 192.168.1.1 (aka `1.1.168.192.in-addr.arpa`), they get referred to a *forward lookup* in your site's normal domain.

Now in your normal domain zone file, whenever you add a host, you add not only the normal `A` record that associates a hostname with an IP address, but also a `PTR` record that matches the name listed in your ISP's `in-addr.arpa` file. This `PTR` record associates the weird `CNAME` from the ISP's configuration with the actual hostname of the machine.

In other words, reverse lookups of your IP address range now require remote sites to first contact your ISP's name server, get referred via the `CNAME` to do a lookup of your organization's local name servers, where they finally get the `PTR` record that they were expecting to complete the reverse lookup. While it seems strange to see a `PTR` record in a forward lookup file, it's perfectly legal.

Web advertising has really started to annoy me recently. There are more pop-up ads than ever, and it seems like the normal web banners are not only taking up more an more screen real estate but are also getting more annoying (more flashing, more movement, more distraction).

What's interesting is that most of these advertisements do not originate on the web sites that you're browsing. There are a relatively small number of companies that run massive server farms that make these ads available. So any particular web page you view may actually require you to do multiple DNS lookups and multiple HTTP connections, just for the purpose of downloading ads from some other site!

But the good news is that you can use DNS to stop this behavior. Basically, the trick is to put bogus zone entries for the domains that these advertising servers use into your local DNS server (if you don't run your own DNS servers, you could set up a caching-only server on your local machine and point your local resolv.conf file at that server). Now you can essentially black-hole all attempts to contact these ad servers by returning the loopback address (127.0.0.1) for any lookup in these domains. Since your local machine won't be running a web server, (or at least won't be running a web server that actually has the requested banner ad) you'll just get a "broken link" marker instead of the ad.

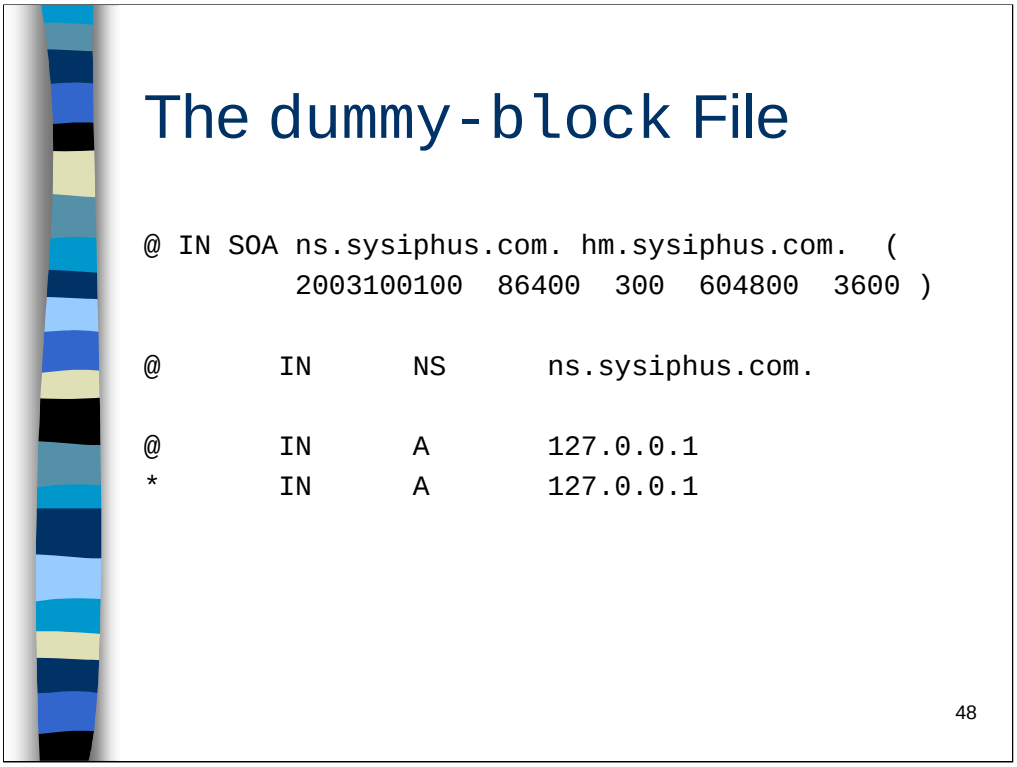
Plus, the name resolution happens on your local name servers (fast) and the HTTP connection happens over the loopback interface (very fast), so browsing speeds up enormously. And, of course, you're using less bandwidth because you're not actually downloading ads anymore.

Hmm, less ads, faster web page loads, and more bandwidth for other tasks… what's not to like?

The first step is to set up the bogus domain entries in your named.conf file. Here's an initial list that's worked pretty well for me, though if you want a larger list, drop me an email.

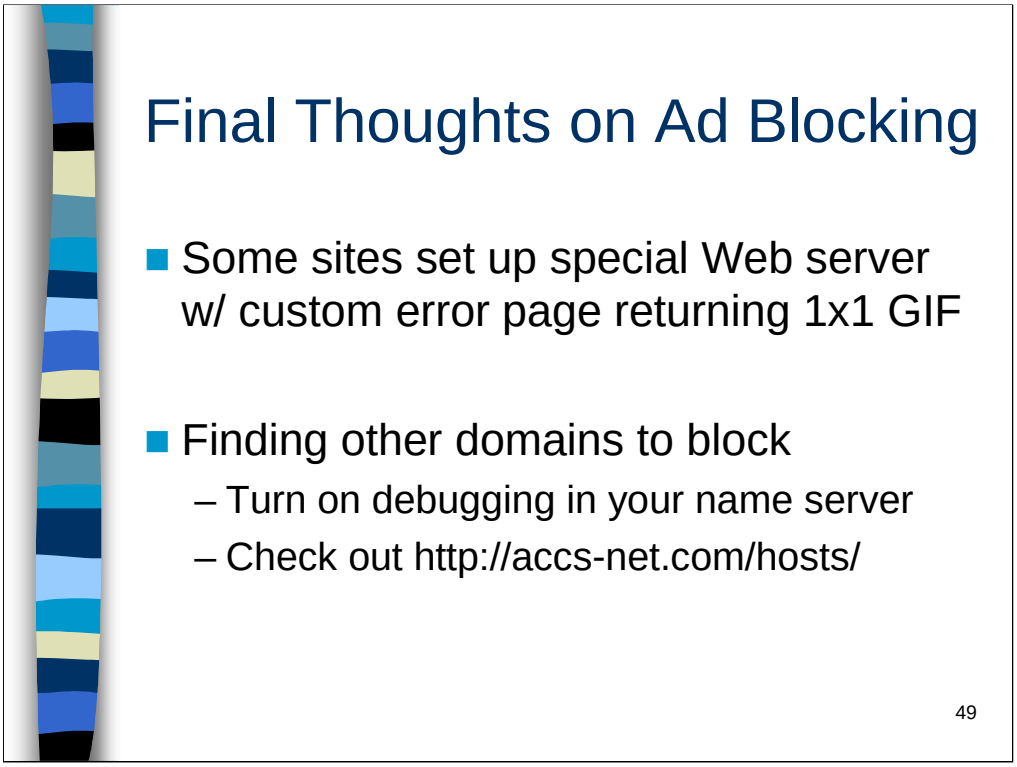
Each of the domains shown above is being pointed at the zone information in the same `dummy-block` file. The contents of this file are shown on the next slide.

Note that it's perfectly acceptable for multiple zones to all point at the same zone file, you just need to be careful how you write the zone file.



So the real trickery happens in the `dummy-block` file. Note that we make liberal use of the "@" symbol so that no matter which domain we specify in the `named.conf` file, the various records in the `dummy-block` file get expanded with the correct domain name.

But the really wacky part is the *wildcard A record* in the last line above. You don't normally see this in practice (and you should *never* use this in a normal zone file), but in this case it means that no matter what hostname gets looked up in the given domain, your name server will always return 127.0.0.1. This is exactly what we want in this case.



If the "broken link" tags bother you, you could redirect all of the lookups to the IP address of one of your local web servers, rather than 127.0.0.1. Then configure your web server to return a custom error page that is a 1x1 clear GIF or something similar. Expect this web server to see quite a bit of traffic…

You may be wondering how I figured out which domains to block. In my case, I simply turned on debugging on my local name server (`kill –USR1 ``cat /etc/named.pid``) and then watched the debugging logs to see what domains were being queried as I was browsing. Note that the debugging logs get big pretty quickly, so be sure to turn debugging off when you're through (`kill –USR2 ``cat /etc/named.pid``).

The good folks at the Gorilla Design Studio (*http://accs-net.com/hosts/*) have been maintaining a static hosts file that contains entries for various ad servers. You could either use this information by incorporating it into your local hosts file, or extract the domain names for use with the DNS technique presented here. Actually, the GDS folks also provide a Win32-based tool called DNSKong to allow you to use a similar DNS-based technique on Windows systems.

This space intentionally left blank…