Deploying DNS and Sendmail

Hal Pomeranz
Deer Run Associates

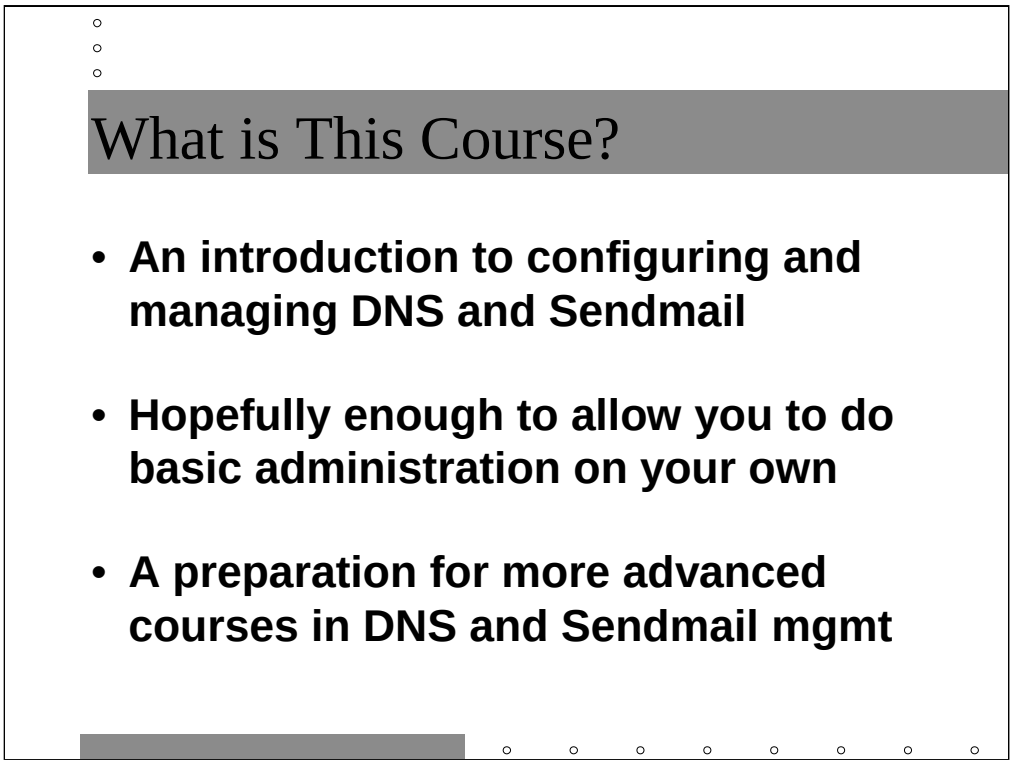
Hal Pomeranz * Founder/CEO * *hal@deer-run.com*
Deer Run Associates * PO Box 20370 * Oakland, CA 94620
+1 510-339-7740 (voice) * +1 510-339-3941 (fax)
http://www.deer-run.com/

Sample configuration files supplied in the Appendices can be downloaded as text files from `http://www.deer-run.com/~hal/dns-sendmail/`

Introduction

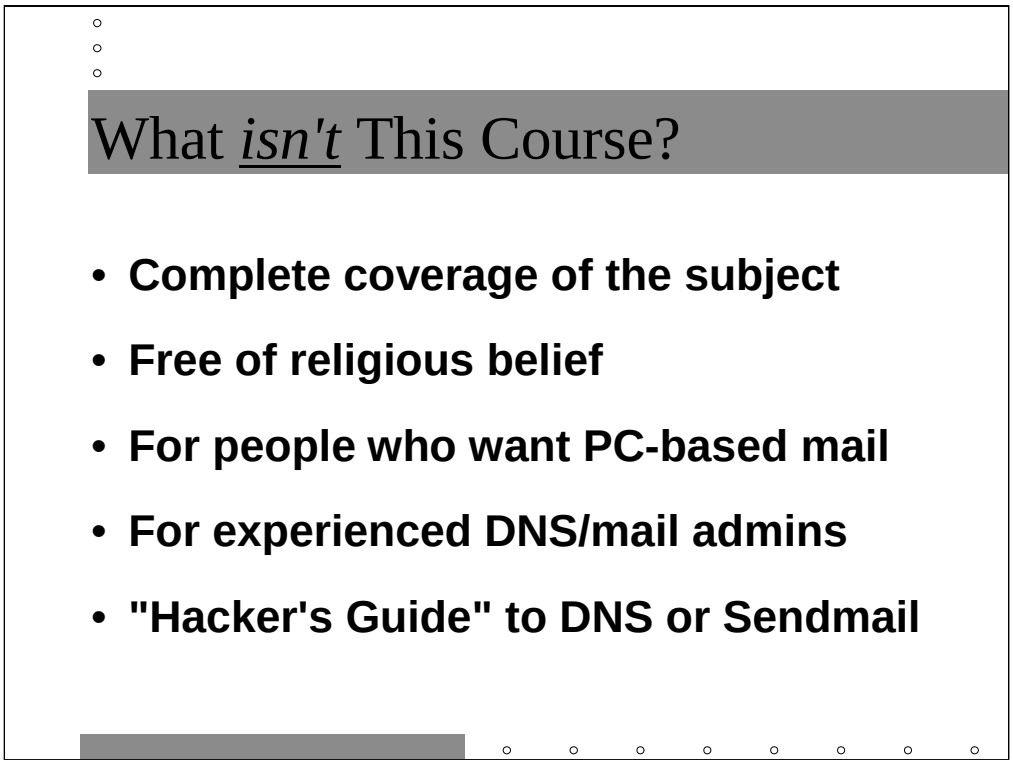
- ***About This Course***
- ***Agenda***

This space intentionally left blank.



This course will hopefully give you enough information to manage DNS and mail services for site, assuming you've been handed the job with no other training. It's not enough just to learn how to configure BIND and Sendmail, you have to understand something about good architectural principals, as well as have the knowledge to manage mailing lists, write your own mail programs and auto-responders, and deal with virtual domains.

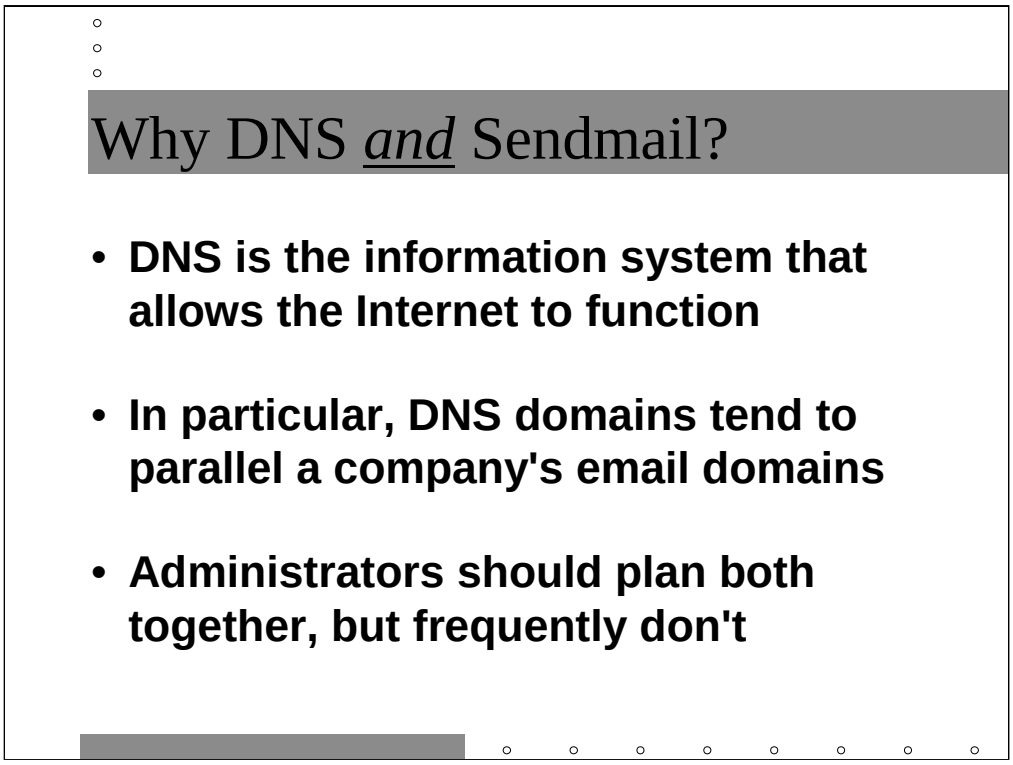
With any luck, this course will also prepare you to get additional knowledge on your own. There are many excellent books and other tutorials which cover more advanced topics in this area (including Sendmail courses from Eric Allman, the author of Sendmail, and DNS courses from Paul Vixie, the current maintainer of BIND).



This course should in no way be considered complete coverage of the subject matter. The O'Reilly Sendmail book alone is just over 1,000 pages. We'll be covering the areas that are "important"-- a subjective judgment by any standard. Doubtless other introductory courses would cover different material (and I've sat through several).

Note that your humble presenter has very strong opinions about deploying and managing DNS and mail services. I will attempt to note when there are differing opinions on a particular subject. One area which will hardly be discussed at all during this talk are PC-based mail systems such as Microsoft Exchange and Lotus Notes. This is one of those religious beliefs.

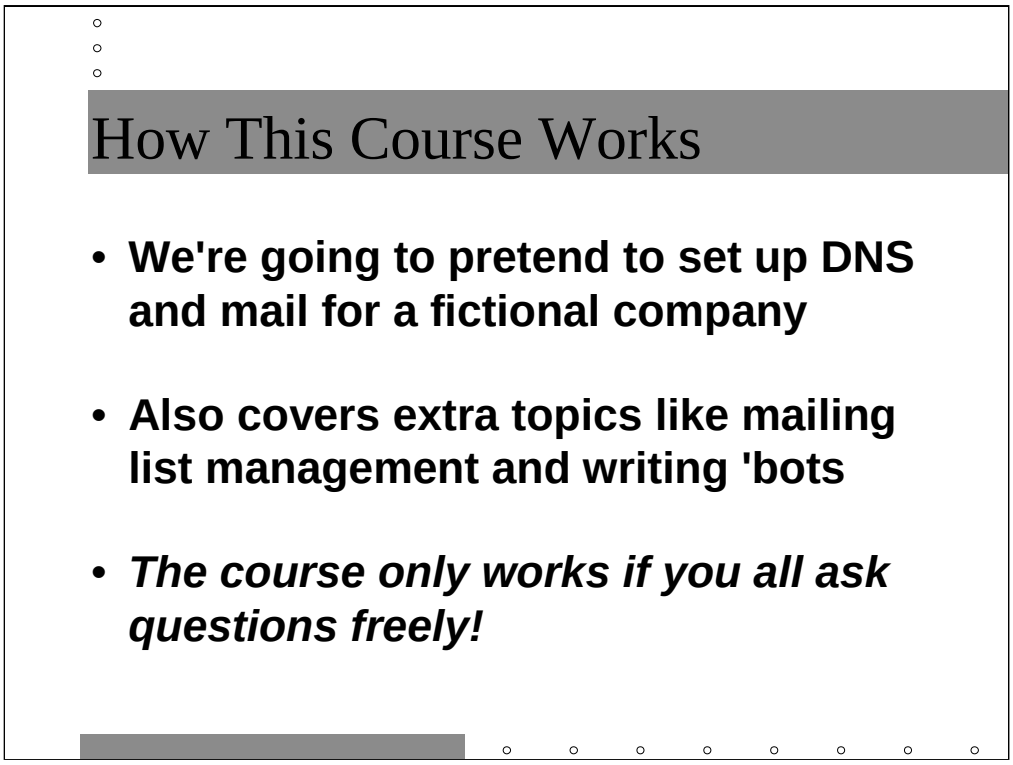
This course covers the most basic information regarding DNS and Sendmail. If you have any substantial amount of experience in this area, you're going to be bored. Seek a different course *now*. There are certainly many tricky and interesting things you can do with DNS and Sendmail, and this course covers none of them. I'd be happy to talk to anyone about nasty DNS and Sendmail hacks outside of this course.



DNS is the great invisible glue which binds the Internet. You tend to only notice it when it isn't working properly. When DNS isn't working you can't get to your favorite Web or FTP sites and they wouldn't let you connect even if you could.

While it is possible to have one set of domains from a DNS perspective and one from an email perspective, it generally makes life easier to use the same domain names for both. Later in this tutorial we will show how to hide DNS domains: that is have host names like `host.site.eng.domain.com` but have users receive email as `user@eng.domain.com`.

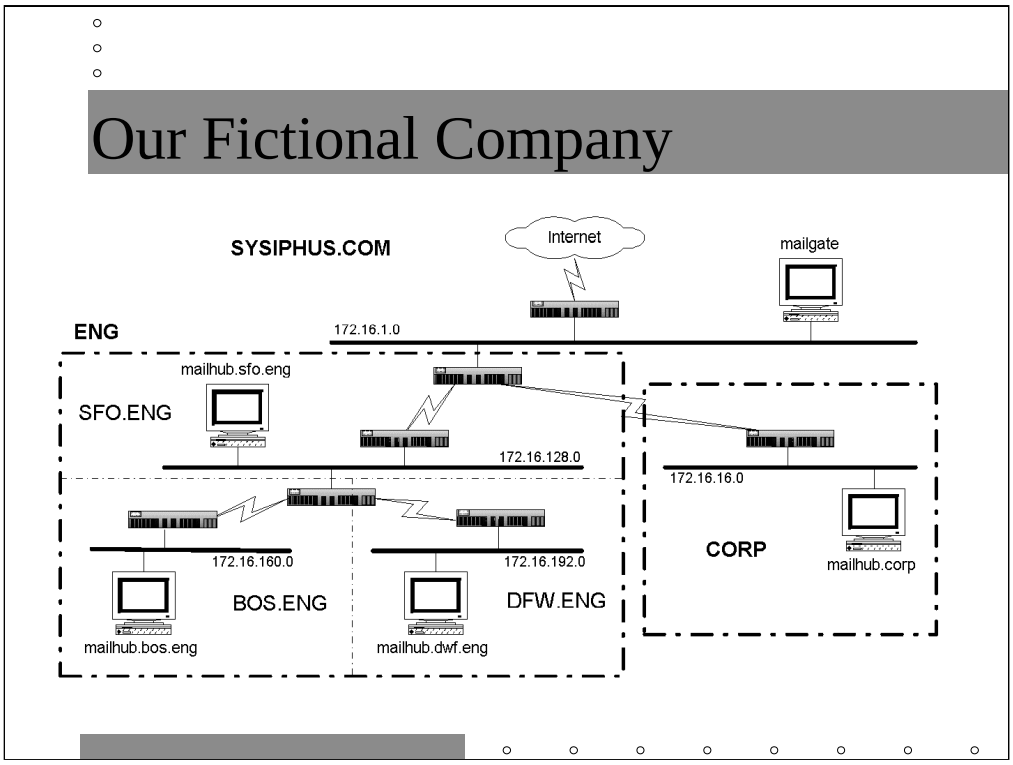
Because DNS domains and email domains should be related, an organization's email and DNS architecture should be planned together. Unfortunately, many sites grow their email and DNS infrastructure "organically" and end up with baroque configurations when a little forethought could have simplified things substantially.



We're going to do a "soup to nuts" install of a DNS and mail architecture for a fictional company. Obviously, one rarely gets to set up an entire network at once. Still, the hope is that you will at some point be faced with setting up DNS and mail in a situation that resembles one or more examples from our fictional company.

Once we've set up the basic DNS and mail infrastructure, we're going to talk more about mailing lists, mail programs, and auto responders. We'll also cover (time permitting) some more in-depth information about getting DNS and email through your company's firewall.

Asking questions is extremely important. Chances are at any given moment many folks in the class are wondering the same thing you are. This stuff is all deeply ingrained in your instructor-- it's your job to remind him that some of it is extremely non-obvious.

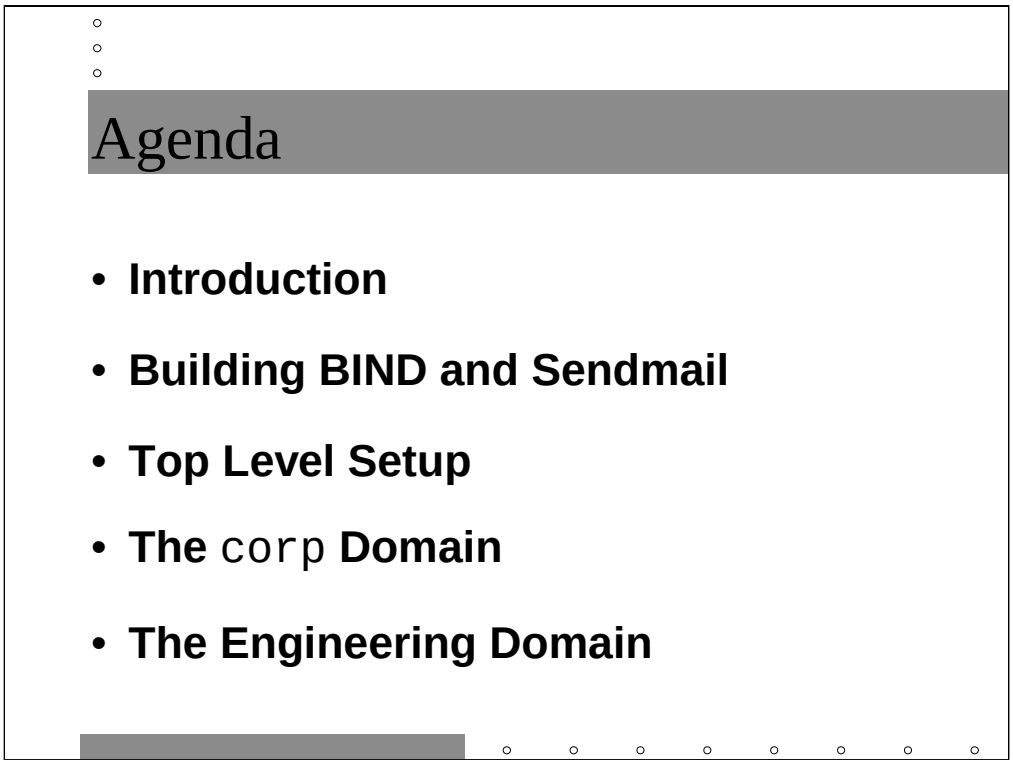


So, here's the network overview for Sysiphus Laborers, Incorporated (`sysiphus.com`). Actually, the domain name is owned by Deer Run Associates for use in examples in courses like this one. The name is derived from Greek mythology: Sysiphus was doomed to roll an enormous rock up a hill for eternity--just as he reached the top, the rock would slip away and roll down to the bottom of the hill. A nice metaphor for System/Network Administration...

At the top level, the machine `mailgate.sysiphus.com` is the primary DNS server for `sysiphus.com`. When we talk about getting email through firewalls, it will also be the main relay point for mail moving into and out of the company.

The `corp.sysiphus.com` domain contains on primary machine where mail for users in this domain is kept. These users are PC-based, so we're going to talk about the right way to do email with PCs here.

The Engineering domain is actually three different physical sites with separate administration for each site. However, they want all Engineering users to receive mail as `user@eng.sysiphus.com`, no matter what their physical location.



This is the schedule for the morning:

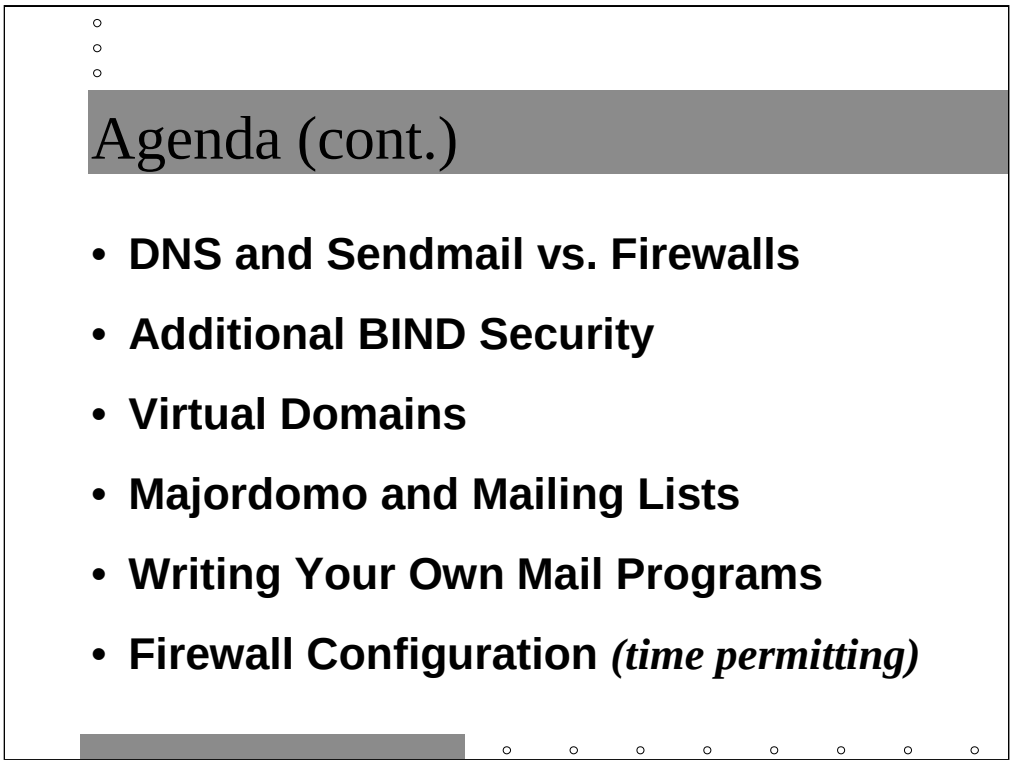
Introduction-- You're approaching the end of that section now.

Building BIND and Sendmail-- Where to get the standard DNS and Sendmail distributions and some hints on building and installing them.

Top Level Setup-- Setting up the primary DNS server for `sysiphus.com` and getting a basic Sendmail configuration in place.

The corp Domain-- Delegating your first subdomain, setting up email, and doing POP mail for PCs.

The Engineering Domain-- Lots more domain delegation, some words about aliases, and hiding DNS domains from mail.



For your enjoyment in the afternoon:

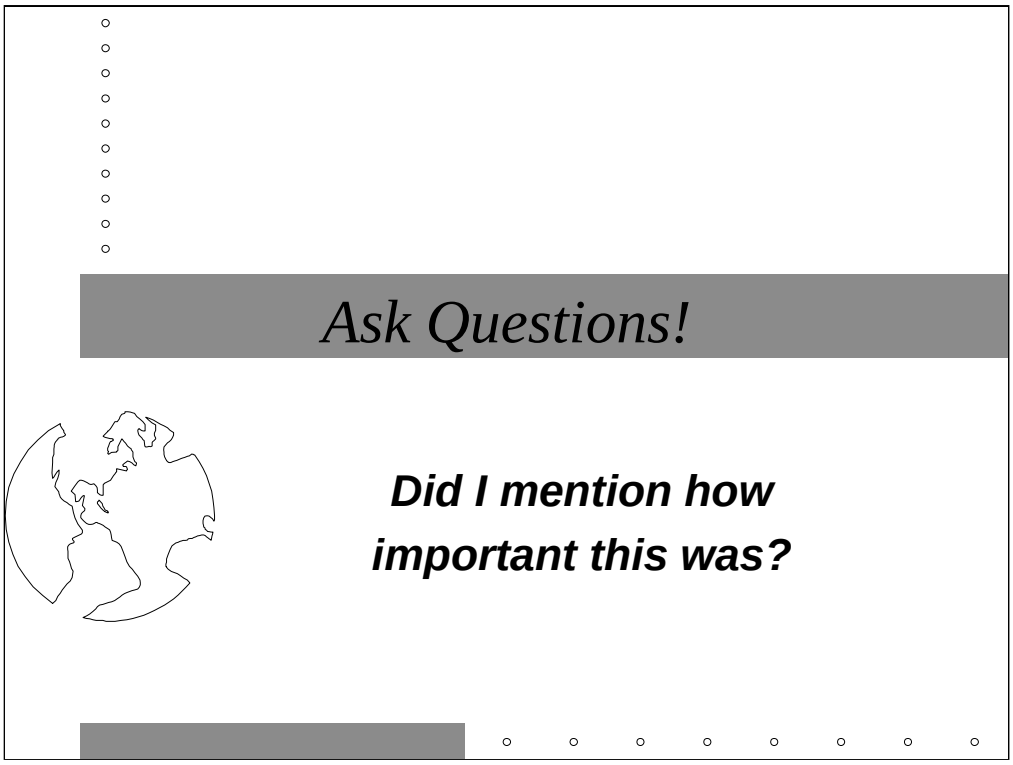
DNS and Sendmail vs. Firewalls-- Split-horizon DNS, clever mail routing tricks, and stopping the spread of unsolicited commercial email (spam)

Virtual Domains-- Pretending to be some other domain

Majordomo and Mailing Lists-- Using Majordomo to manage your own mailing lists

Writing Your Own Mail Programs-- Avoid the common mistakes that most first-time programmers make and write useful applications

Firewall Configuration-- Time allowing, drills down more on exactly what happens with DNS and Sendmail from a network perspective in the context of getting these services through your local firewall.



There is no such thing as a stupid question, merely a stupid instructor…

Building BIND and Sendmail

- ***Where to Get the Software***
- ***Building the Software***
- ***Install Process***

Before we can make much progress we have to get the latest versions of BIND and Sendmail installed on our systems. In general, Operating System vendors have long QA processes which make it impossible to stay current on Internet supported software. However, you generally need to run the latest software to avoid security problems and get the latest features. This means you need to get used to building this stuff on your own.

This section uses Solaris-specific commands as examples. The build and install process turns out not to be substantially different on other architectures.

BIND -- Build Process

- **Get BIND source code**

```
ftp.isc.org
/isc/bind/src/cur/bind-8/bind-*.tar.gz
```

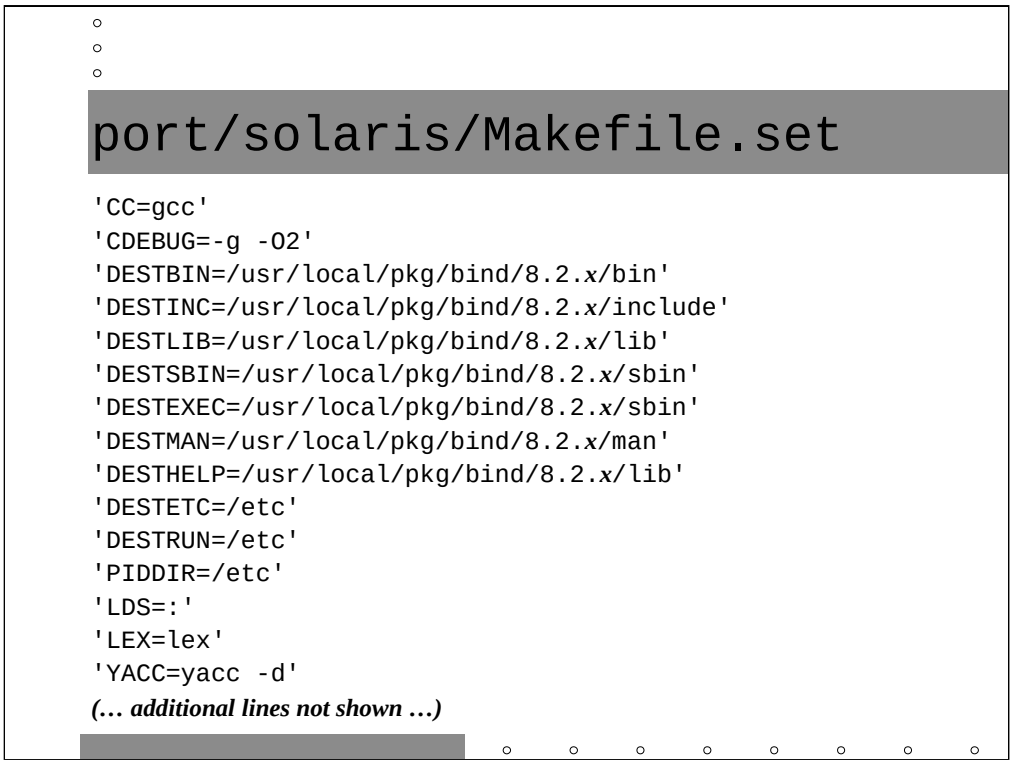
- **Unpack and build**

```
% zcat bind-src.tar.gz | tar xf -
% cd src
% make DST=../obj-solaris SRC=`pwd` links
% cd ../obj-solaris
% vi port/solaris/Makefile.set    (see next slide)
% make depend                     (lots of output)
% make                            (lots of output)
```

BIND stands for Berkeley Internet Name Daemon. The code is currently maintained by the Internet Software Consortium (ISC), and development is being led by Paul Vixie.

The BIND v8 distribution is actually three tarfiles-- one containing the core sources, one with contributed sources, and one with documentation. You want all three.

First unpack the tar file you downloaded (note you need the GNU version of `zcat` here, `ftp://ftp.prep.ai.mit.edu/pub/gnu/gzip*`). You don't want to build the objects in the same directory with the sources (you probably want to have several different binaries for different platforms), so use `make … links` to build a separate link tree. `make depend` causes the build scripts to figure out what architecture you're on and set up the `Makefiles` appropriately.



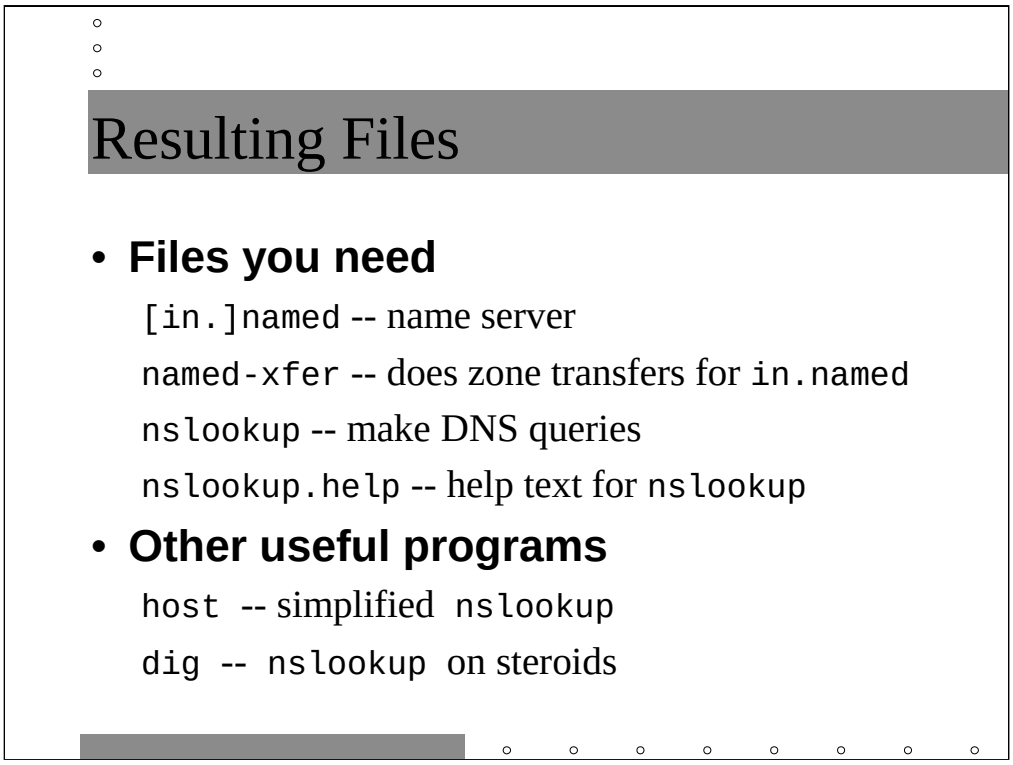
If you don't change the parameters in `Makefile.set`, BIND will install itself by overwriting your OS binaries. We're going to do this later, but you definitely want this process under your complete control, so install the binaries in their own repository for now.

Note that `DESTLIB` and `DESTINC` are not in `Makefile.set` by default. You need to add these lines yourself (note the quotes!).

`DESTETC` and `PIDDIR` control the location of the named configuration file and the `named.pid` file, respectively.

`DESTRUN` is the default working directory for BIND. When BIND starts executing, the name server process changes it's current directory to the value of `DESTRUN`. This means that if the process dumps core, the core file will appear in this directory. This is also the directory where the name server looks for its DNS database files (aka *zone files*) by default and where the name server will dump debugging information if coerced. As we will see, BIND provides a configuration option for changing the compiled-in value of `DESTRUN` on the fly.

`Makefile.set` comes with `LEX` set to `flex` (the GNU version of `lex`) and `YACC` set to `byacc` (Berkeley `yacc`). Reset these to the default OS version.



The BIND release builds a binary, `named`, which is the actual name server daemon. Solaris refers to this program as `in.named`. `named` periodically calls `named-xfer` when it needs to do a wholesale transfer of domain information from another nameserver, e.g. when your name server is acting as a secondary for another domain.

`nslookup` is a tool for actually making DNS queries of a local or remote name server. Make sure you install the version of `nslookup` from the release of BIND you are installing and to throw away the Sun version, or vast confusion will result.

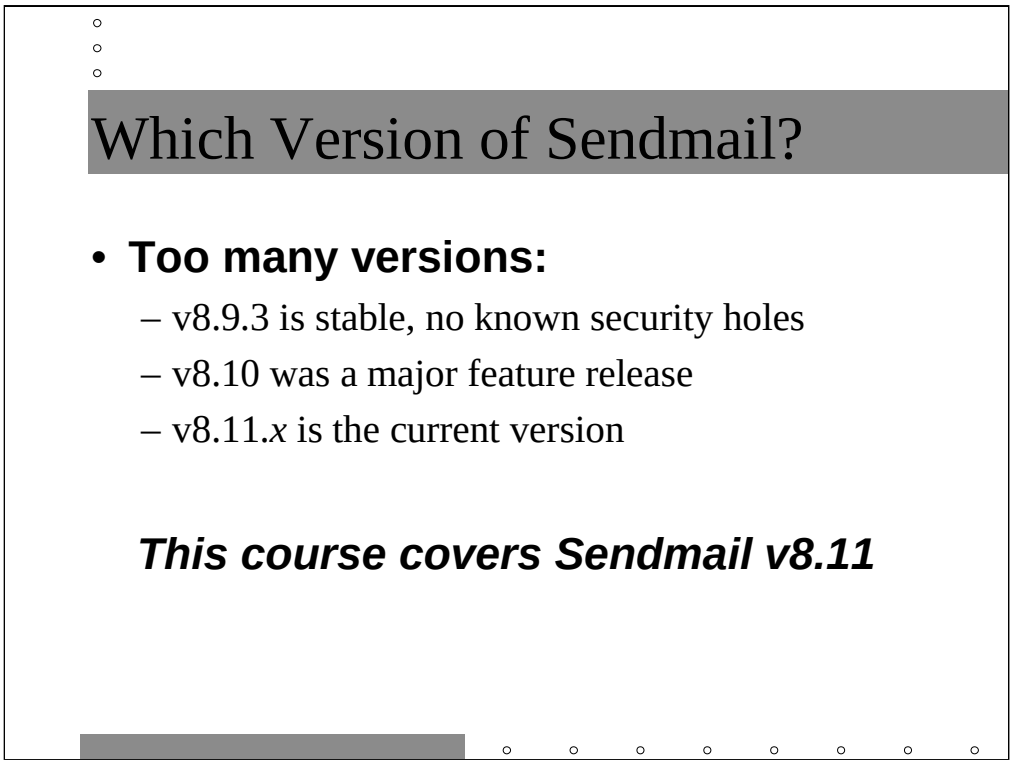
The `dig` and `host` programs are alternate means of making DNS queries. In particular, `dig` gives back a great deal of useful debugging information once you learn how to read the output.

BIND -- Installing Files

```
% /bin/su
Password:
# make install
   (lots of output)
# cd /usr/sbin
# cp /usr/local/pkg/bind/8.2.x/sbin/named \
       in.named_from_usr-local-pkg
# rm in.named
# ln -s in.named_from_usr-local-pkg in.named
# rm named-xfer
# ln -s /usr/local/pkg/bind/8.2.x/sbin/named-xfer .
# rm nslookup
# ln -s /usr/local/pkg/bind/8.2.x/sbin/nslookup .
```

Note that we are removing the Solaris binaries.

We want to install the new `in.named` locally so that it's available at boot time before NFS gets a chance to start up. We go through the rigmarole with the `in.named_from_usr-local-pkg` just to make it clear to other admins that we're not using the stock Solaris `in.named`.

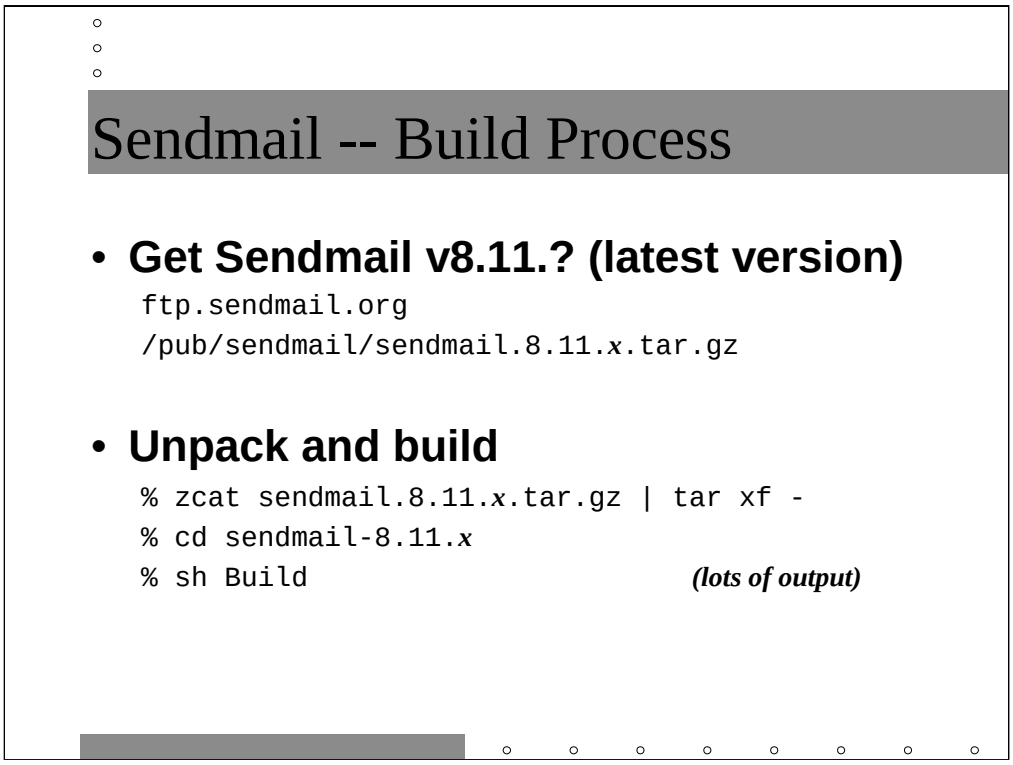


In the past, it was usually the case that administrators always wanted to be running the latest version of Sendmail– mostly because new versions of Sendmail were created whenever a security problem was discovered. At this time, however, Sendmail v8.9.3 has been in use for a long time and we still don't know of any security problems in this version of the code. v8.9.3 is a known stable release that is still in use on a lot of systems.

Sendmail v8.10 was the largest feature release in Sendmail history– made possible by the work of the engineering team at Sendmail.COM. While many of these new features are useful, new features/code implies that new bugs may have been added as well. Sendmail v8.10 was followed by v8.11 (v8.11 includes crypto support and support for SMTP AUTH)– the latest version as of this writing is v8.11.3.

If you don't need all of the features that were introduced in Sendmail v8.10 and v8.11, then perhaps you may be more comfortable staying with v8.9.3. This course uses Sendmail v8.11.3 for its examples because we want to make use of the new anti-spam features in the latest version. For more information on new features introduced in Sendmail v8.10, check out:

```
http://www.sendmail.net/allabout810.shtml
```



Frankly, the build process for recent versions of Sendmail is trivial. The compiled binary ends up in a directory called `obj.<system type>`, where `<system type>` is a string like "`OpenBSD.2.8.i386`".

A couple of notes if you're building Sendmail v8.9.3 from source:

• The source code and `Build` script are located in a sub-directory called `sendmail-8.9.3/src`, rather than the top-level directory

• You may want to edit the OS configuration file for your system in the `sendmail-8.9.3/BuildTools/OS` directory. In particular, you may want to add the `-DUSE_VENDOR_CF_PATH` switch to the `confENVDEF` declaration in the appropriate file. This forces Sendmail to look for its configuration file in the directory that your OS vendor chooses, rather than the Sendmail default (`/etc/sendmail.cf`). This may make administration easier for you.

Note that starting with Sendmail v8.10, the convention is to put all Sendmail-related configuration files in the directory `/etc/mail`, so the `USE_VENDOR_CF_PATH` switch isn't used anymore.

Sendmail -- Installing Files

```
% /bin/su
Password:
# cp obj.SunOS.5.8.sun4/sendmail \
     /usr/lib/sendmail
# chmod 4555 /usr/lib/sendmail
# chown root:bin /usr/lib/sendmail
# chmod g-w /etc /etc/mail
```

Again, we are disposing of the binary provided with Solaris. You will have to replace the `sendmail.cf` file that your vendor provides-- we'll be talking about how to do this in later sections.

Note that, starting with Sendmail v8.9, Sendmail will refuse to start if either the `/etc` or `/etc/mail` directories are group writable. On Solaris machines and some others, these directories are normally group writable, but turning off this attribute doesn't appear to impact the system.

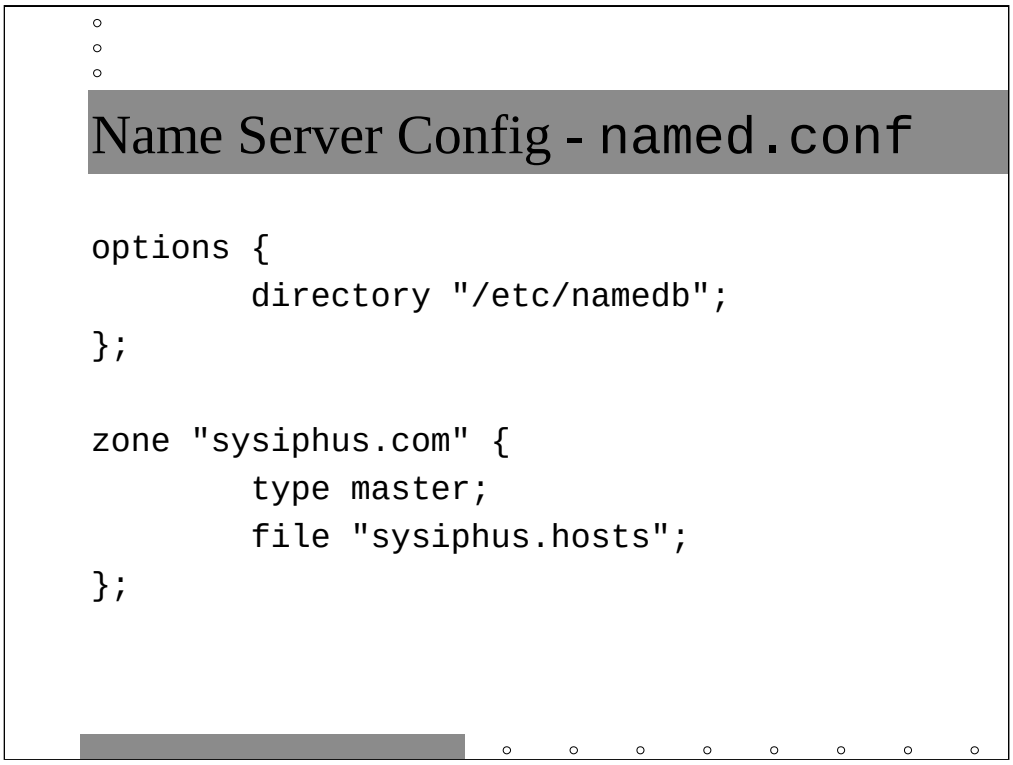
Actually, what's happening is that recent versions of Sendmail institute a number of checks on the OS environment and will refuse to execute if they detect problems. Generally, you should follow the advice that Sendmail gives you in this respect. However, if you get to a point where Sendmail is refusing to operate unless you make a change that breaks your operating system, you can always set the `DontBlameSendmail` flag in the Sendmail configuration file to turn off these checks (at the cost of running Sendmail in a potentially insecure environment).

Top Level Setup

- ***Basic DNS Configuration***
- ***Running and Testing DNS***
- ***Basic Sendmail Configuration***

In this section we're going to cover basic DNS configuration and installation. This includes an introduction to the `named.conf` file-- the primary configuration file for the BIND v8 `in.named`-- as well as our first glimpse at DNS zone database files. Later sections will revisit these files and demonstrate additional functionality.

Once we've got DNS set up to our satisfaction, we'll take our first look at creating Sendmail configurations.



This is slide 1 of 2.

In this portion of the `named.conf` file, we set the `directory` option to `/etc/namedb`. This changes the default working directory of the named process (`DESTRUN` from `Makefile.set`), meaning that this is where named will now dump core file as well as debugging info and that this is where named will look for its zone files. Effectively, this means that all file names given in other sections of this `named.conf` file will be relative to `/etc/namedb`.

Next we define the `sysiphus.com` zone. We are the `master` server for the domain, which means the DNS information for this domain is maintained on this machine. The DNS information for this domain is kept in the file `sysiphus.hosts` (which lives in `/etc/namedb`, see above). The contents of the `sysiphus.hosts` file will appear later in this section.

Note that the older versions of BIND used "`primary`" instead of "`master`". You may here people refer to being the "primary nameserver" for a domain. No need to be confused , they just mean "master namesever".

named.conf (cont.)

```
zone "16.172.in-addr.arpa" {
        type master;
        file "sysiphus.rev";
};

zone "." {
        type hint;
        file "named.ca";
};
```

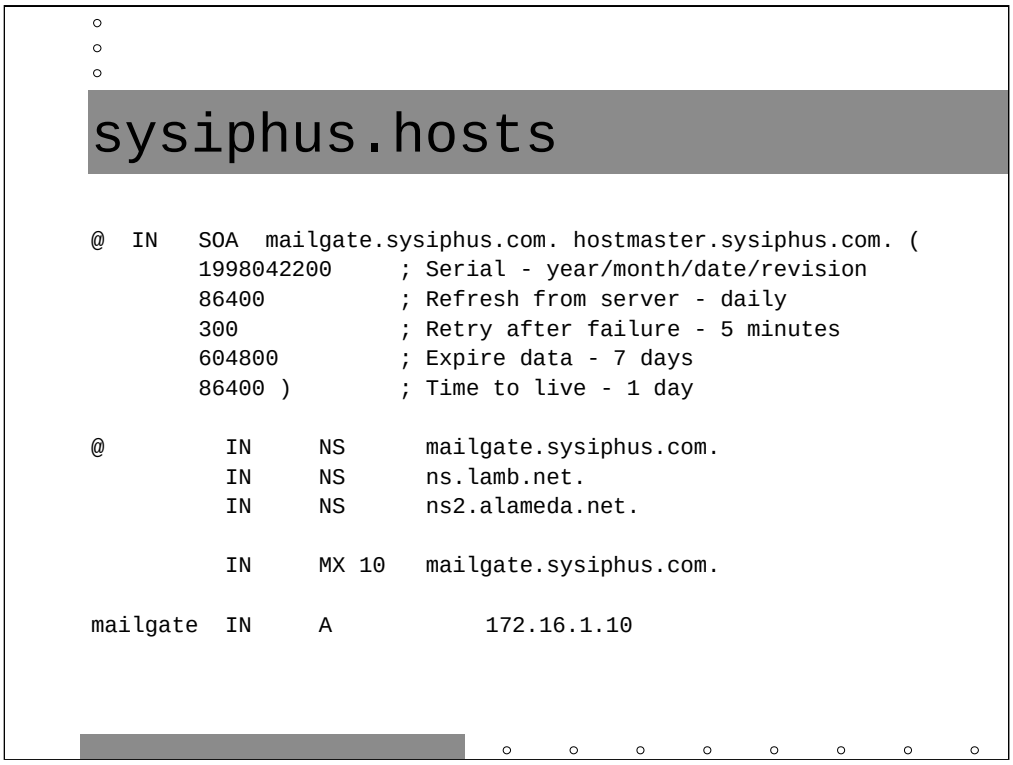
This is slide 2 of 2.

The next zone name looks odd. `in-addr.arpa` is a very old hack which was created to allow DNS clients to look up IP addresses and get back hostnames. Drop the `in-addr.arpa` part and reverse the two numbers: this declaration is for the network `172.16.0.0`. By configuring ourselves as the master for this domain, we're claiming that this server knows the hostnames for addresses that are somewhere within this netblock. The address to hostname mappings are kept in a different file from the hostname to address mappings for `sysiphus.com` hosts.

Zone `"."` is a special zone, generally referred to as the "root" of the DNS system. Servers at the root are responsible for providing information about which servers serve which zones. For example, when your local name server wants to look up `www.cisco.com`, it first has to contact a root server to find out what machine to talk to in order to get information for the `cisco.com` domain. Your name server then talks to the name server for `cisco.com` to get the address of `www.cisco.com`.

The `named.ca` file contains names and addresses of all the currently running root name servers. This file is maintained by the InterNIC (Internet Network Information Center) and is available at:

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



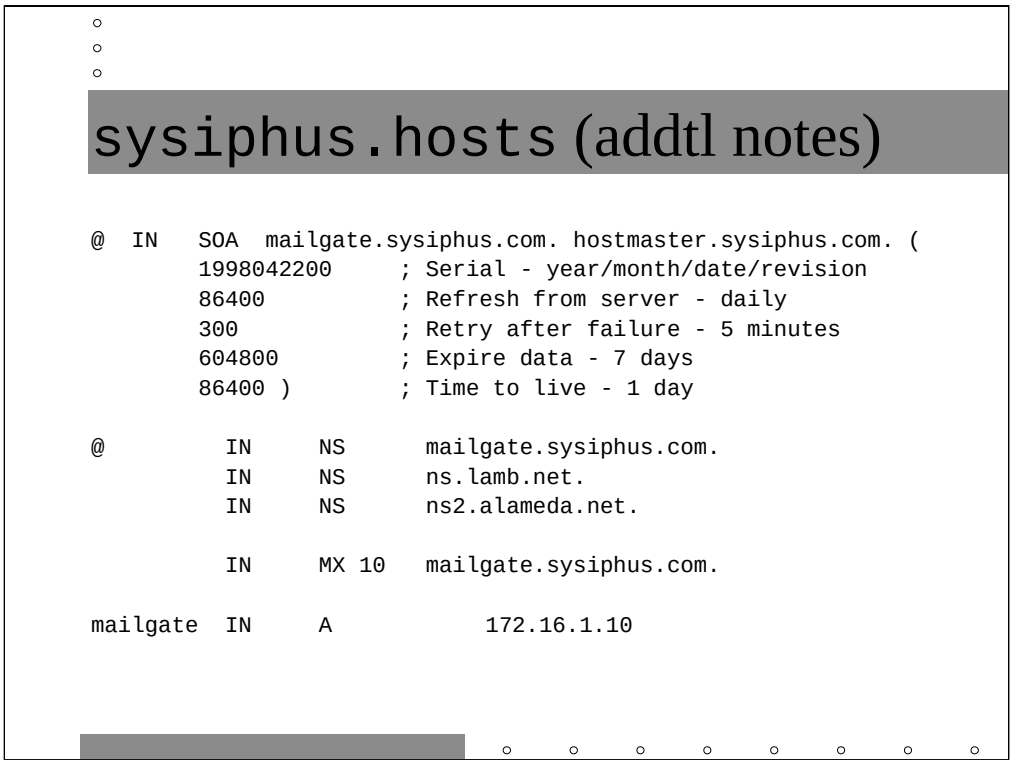
(slide 1 of 4)

Every zone database file begins with an `SOA` (Start of Authority) record. The first hostname listed is the name of the master name server for the domain. The second hostname isn't a hostname at all– it's an email address, `hostmaster@sysiphus.com`, for the local domain administrator. The `@` character is a special symbol which gets expanded to mean "the current domain" (`sysiphus.com` in this case). We can see `@` markers in a couple of locations in this file.

The first number in the `SOA` record is the serial number for the zone. You can use any numbering scheme you want just so long as every time you make a change you increase the value of the serial number in the `SOA` record. If you don't increase the serial number value then your changes won't get propagated. Forgetting to update the serial number after zone changes is an extremely common error.

The next two values say how often slave servers should poll the master for new information and how frequently they should retry if they fail to get connected. If the slave server cannot contact the master within the given expire time (the fourth value in the SOA record) it will invalidate its own information and stop answering queries for the given zone-- basically the slave decides that at some point its data is so old that it *must* be invalid. Setting up slave servers (covered in later sections) is important for disaster recovery.

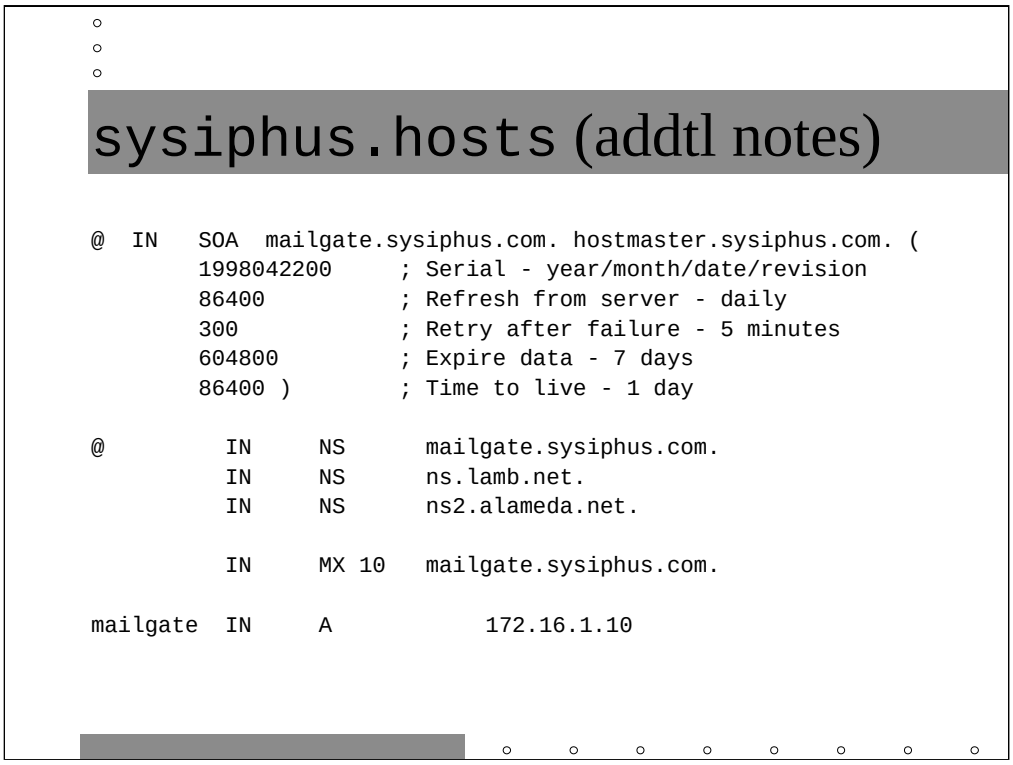
Note that starting with BIND v8.1.2, updates on the master server result in the known slave servers receiving a special "notify" message which tells them that the zone data on the primary has been changed. As long as the slave server is running a recent version of BIND (servers older than v8.1.2 receive the notify packets but don't understand or do anything with them), the slave immediately contacts the master server and download the new zone information. This means that the middle three values of the SOA record are becoming much less important. in modern releases of BIND.



(slide 2 of 4)

The last value in the SOA record is the time-to-live (TTL) value for data from this zone. It specifies how long data in the `sysiphus.com` zone to live in the caches of other name servers. Caching reduces the load on remote nameservers: if 500 people in your company all go to `www.playboy.com`, your server only has to do one lookup, and Playboy's name servers have to answer 499 less queries.

In general the settings for your server refresh period and TTL are dependent on how dynamic your DNS data is. If you're making lots of changes then you want your slave servers to synch up frequently (hourly perhaps) and you want TTL values to be low. On the other hand, if your DNS data is more static you can significantly reduce server load by tuning these values back. You may want to make the intervals smaller when you're doing a major network upgrade or renumbering and then back off to longer intervals when your network "returns to normal".

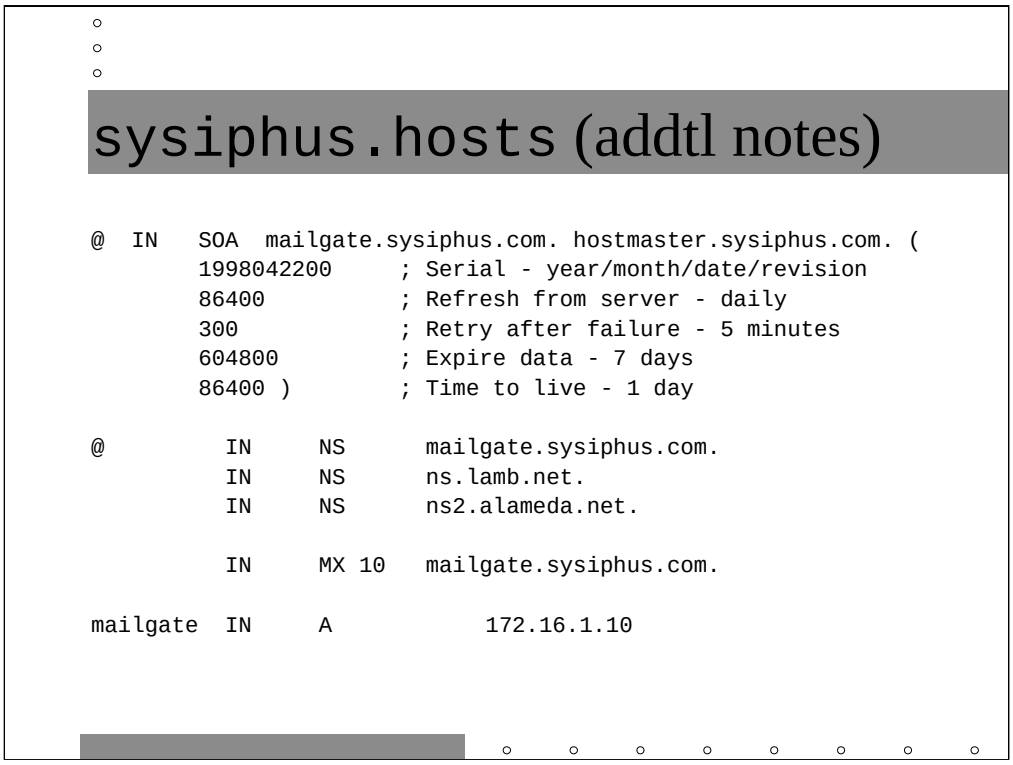


(slide 3 of 4)

The next three entries in the file are `NS` (Name Server) records. They list the known name servers for the domain. It's a good idea to have secondary name servers for your domain that are somewhere other than on your network (in case your Internet link goes down). Most ISPs will be slave servers for their customers at no additional charge. *Please do not list `ns.lamb.net` and `ns2.alameda.net` in your zone files! They're my ISP, not yours!*

The next record is an `MX` (Mail eXchanger) record. This one says "direct all mail for `user@sysiphus.com` addresses to `mailgate.sysiphus.com`. The numeric value "10" is a preference value-- the *lower* the number, the *more preferred* a given server is. If we had `MX` records at priority 10 and priority 100, all mail would go to the priority 10 server unless it was unreachable. When all lower numbered servers were unreachable, the mail would go to the priority 100 server. You can have multiple servers at the same priority, in which case the name server hands out the names of the mail server in order until the list is exhausted and then starts over again with the first server.

The last record in the file is an `A` (Address) record. It says the address of "`mailhub(.sysiphus.com)`" is `172.16.1.10`.

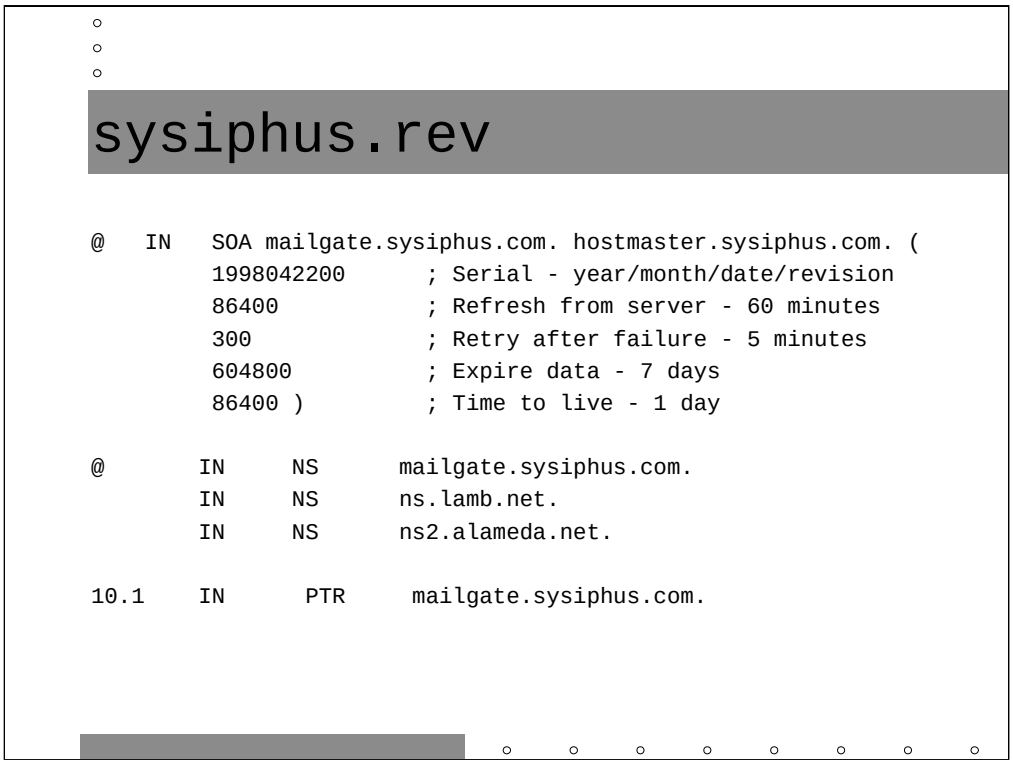


(slide 4 of 4)

You've probably noticed at this point that two of the three `NS` records as well as the `MX` record have nothing in the first column. If no name is specified before the `IN`, then the value from the previous line is used. We actually could have dropped the `@` in front of the first `NS` record, since we specified `@` in front of the `SOA` record.

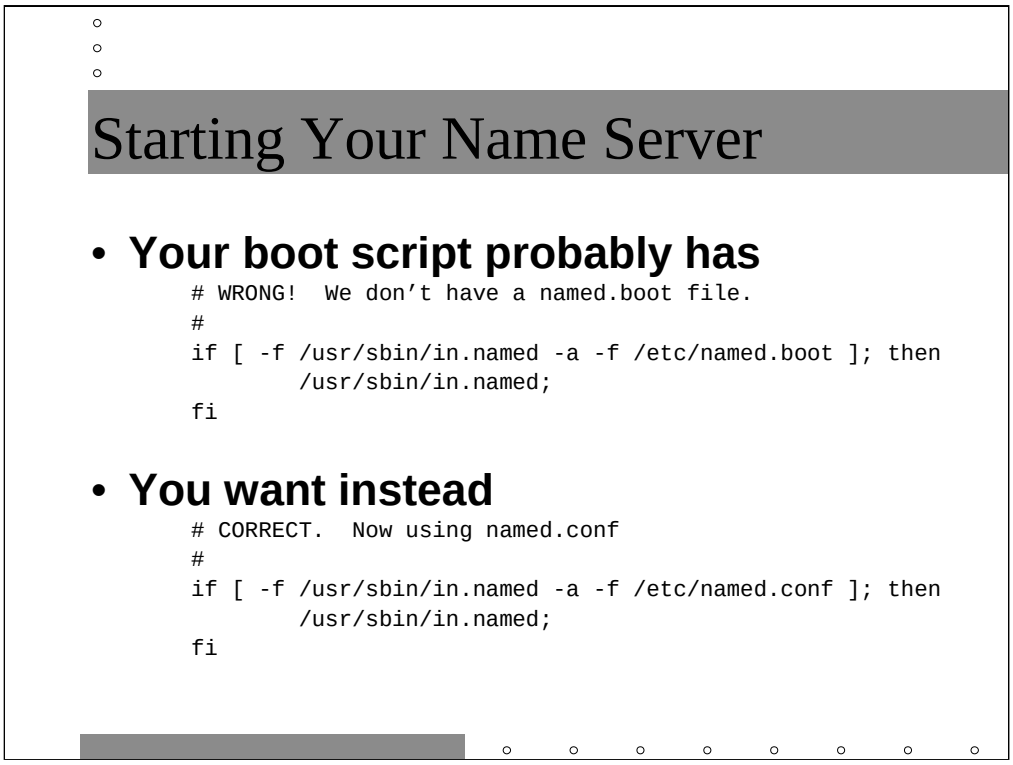
The next important piece of syntax to note is all of the trailing "." at the end of each entry. If there is no trailing ".", then the local domain name is appended. `mailgate.sysiphus.com` would become `mailgate.sysiphus.com.sysiphus.com` and `ns.lamb.net` would become `ns.lamb.net.sysiphus.com`, both of which are nonsensical.

In many cases, this automatic appending of domain names is useful. For example in the single `A` record in this domain: `mailgate` becomes `mailgate.sysiphus.com`. There is a very clever reason why we're doing unqualified names in the first column and fully-qualified names with dots in the last column, but you'll have to wait until we get to the *Virtual Domains* section to find out what that is.



Recall that the `sysiphus.rev` file holds the IP address to hostname mappings. It too has an `SOA` record and group of NS records, just like the `sysiphus.hosts` file has. Generally, it's a good idea to have the same set of servers be name servers for both the forward (hostname to IP address) and reverse (IP address to hostname) zones for a given organization, if for no other reason than it reduces confusion on the part of local administrators.

The `PTR` (Pointer-- a bad name, IMHO) record is the logical inverse of the `A` record from the forward zone file. Recall that we read the domain name for the `16.172.in-addr.arpa` zone "backwards"? Well, you read `PTR` records backwards too: the address of `mailgate.sysiphus.com` is `172.16.1.10`. Again, note the trailing "." on `mailgate.sysiphus.com`!

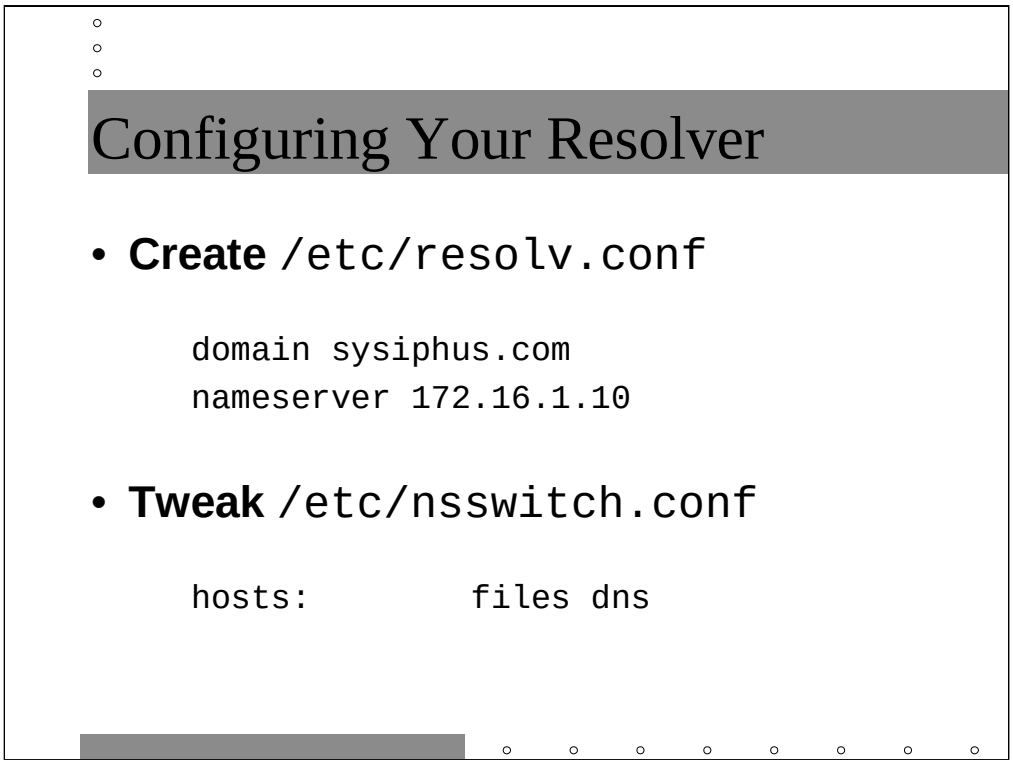


Congratulations! You've now completed the basic DNS configuration steps! There are, unfortunately, a few things left to do.

Older versions of BIND (that's v4 and earlier-- there was no BIND v5-7) use a different config file called `named.boot`. Chances are your boot script looks for `named.boot` before starting the name server. You need to tweak this script to look for `/etc/named.conf` instead. You can execute `/usr/sbin/in.named` as root to get the name server started without rebooting.

Btw, `named.boot` has a completely different syntax from `named.conf`. If you have old `named.boot files` lying around, there's a script called `named.bootconf.pl` in the BIND v8 distribution which converts `named.boot` files to `named.conf` files.

Also, btw, the format of the zone database files *did not* change, so you can keep using any old files you have lying around.

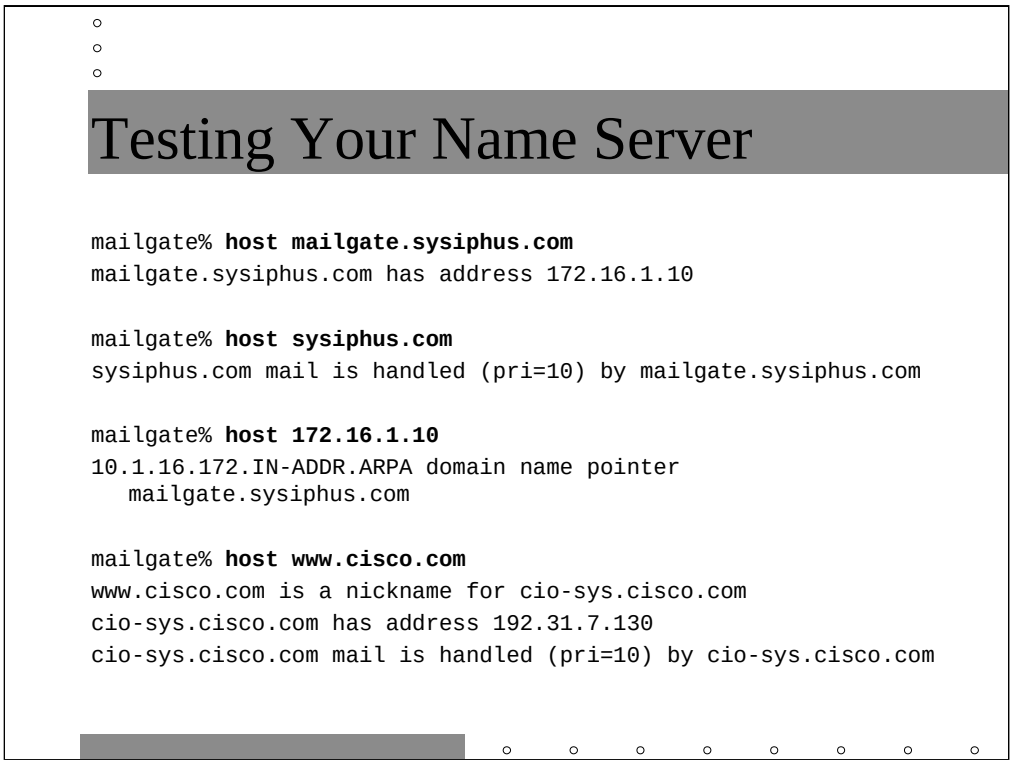


Great, DNS is configured and the name server is even going to start up at boot time. Now you have to teach your machine to look up hostnames using DNS.

What's going on here is that there are really two pieces to this DNS equation. We've already seen the basics of configuring a name server. The other piece is what's generally referred to as a "resolver" or DNS client. The `nslookup`, `host`, and `dig` programs are all resolvers. More interestingly, there are a bunch of library routines buried in your system (in the Unix universe, the outermost expression of these routines are the `gethostbyname()` and `gethostbyaddr()` functions) which programs incorporate to do hostname lookups. Depending on the configuration of your machine's resolver, hostname lookups can happen in a flat file (like `/etc/hosts`), via DNS, or via something like NIS[+] or Hesiod, or even by some combination of these.

To configure the resolver on a Unix machine, create a file called `/etc/resolv.conf`. The `domain` directive lists the machine's local domain and then you may have one or more `nameserver` lines which give the IP address (can't use hostnames because we wouldn't know how to look them up!) of name servers for the given domain.

Some Unix systems also require you to tweak an OS configuration file (like `/etc/nsswitch.conf` on Solaris systems) to tell the OS you're going to be using DNS. Consult your vendor documentation.



Allrighty, everything should be ready to go from a DNS perspective. Let's use the `host` command to verify a few things.

In the first case, we simply verify that we can do a simple hostname to address lookup. This also verifies that your nameserver is running and your resolver configured properly.

In the next example we look up the records for the `sysiphus.com`. By default, the `host` command doesn't print out `SOA` or `NS` records, but we do see the priority 10 `MX` record pointing at `mailgate.sysiphus.com`.

The third example tests the `in-addr.arpa` domain configuration.

Finally we make sure we can resolve external names. This verifies that we can get out to the Internet and that our `named.ca` file is in good shape.

Creating Sendmail Config Files

```
% cd /path/to/sendmail/src
% cd cf
% mkdir sysiphus-configs
% cd sysiphus-configs
% vi toplevel.mc      (see next slide)
% m4 toplevel.mc > toplevel.cf
% /bin/su
Password:
# cp toplevel.cf /etc/mail/sendmail.cf
# echo sysiphus.com >/etc/mail/local-host-names
# echo mailgate.sysiphus.com >>/etc/mail/local-host-names
# /etc/init.d/sendmail stop
# /etc/init.d/sendmail start
```

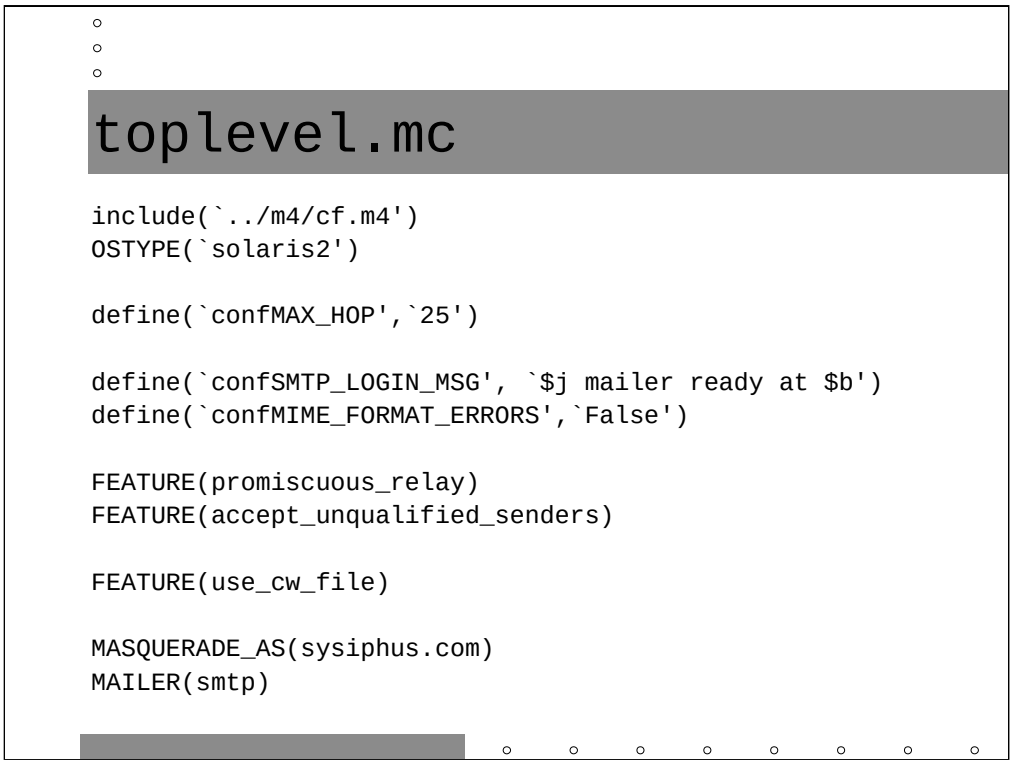
In the bad old days, when monsters and giants roamed the earth, lowly mail administrators had to learn an arcane language to configure Sendmail. You can still see this language if you happen to look at your local `sendmail.cf` file. The good news is that v8 Sendmail comes with an exceedingly handy mechanism for generating `sendmail.cf` files from a more human-readable configuration language.

Yes, some day you should learn how to hack `sendmail.cf` files directly, but for 9 out of 10 jobs, you don't even have to look at them. We're not going to look at a Sendmail ruleset for this entire course-- take Eric Allman or Rob Kolstad's course if you want to be done to death by `sendmail.cf` files.

Inside the Sendmail source directory is a subdirectory `cf` which contains a number of directories and files containing `m4` macros. `m4` is another artifact of the ancient days of Unix-- think of it as a less functional version of the C pre-processor (if you know what that is). It turns out you probably use `m4` nearly every day and don't know it: the `syslog.conf` file is in the `m4` macro language and you'll see some similarities with our `m4` files which generate `sendmail.cf`s.

Your best bet is to create one or more subdirectories of your own in Sendmail's `cf` hierarchy. Put your configuration files there and give them descriptive names. Chances are you'll go back to them from time to time to tweak your mail configurations.

More on the `/etc/mail/local-host-names` nonsense in the next couple of slides.



(slide 1 of 3)

The first line of the file pulls in the standard set of `m4` macros used to generate `sendmail.cf` files. On the second line we define our `sendmail.cf` file as running on a Solaris system (this sets OS specific values like directory names).

Sendmail attempts to stop messages from looping forever. Normally a message is rejected if it's taken 18 or more hops (a hop is generally being passed from one machine to another). We can set this value higher by setting the `MAX_HOP` variable as shown.

When you telnet to port 25 on a machine running Sendmail, you see something like:

```
220 mailgate.sysiphus.com ESMTP Sendmail \
        8.8.5/8.8.5; …
```

which you will note gives the version of Sendmail. Since there are known bugs with many older versions of Sendmail and surely more to be discovered in the current version, going around advertising your version of Sendmail is a bad idea. The new `SMTP_LOGIN_MSG` we set looks like

```
220 mailgate.sysiphus.com ESMTP mailer \
        ready at <date>
```

which doesn't even advertise that we're running Sendmail.

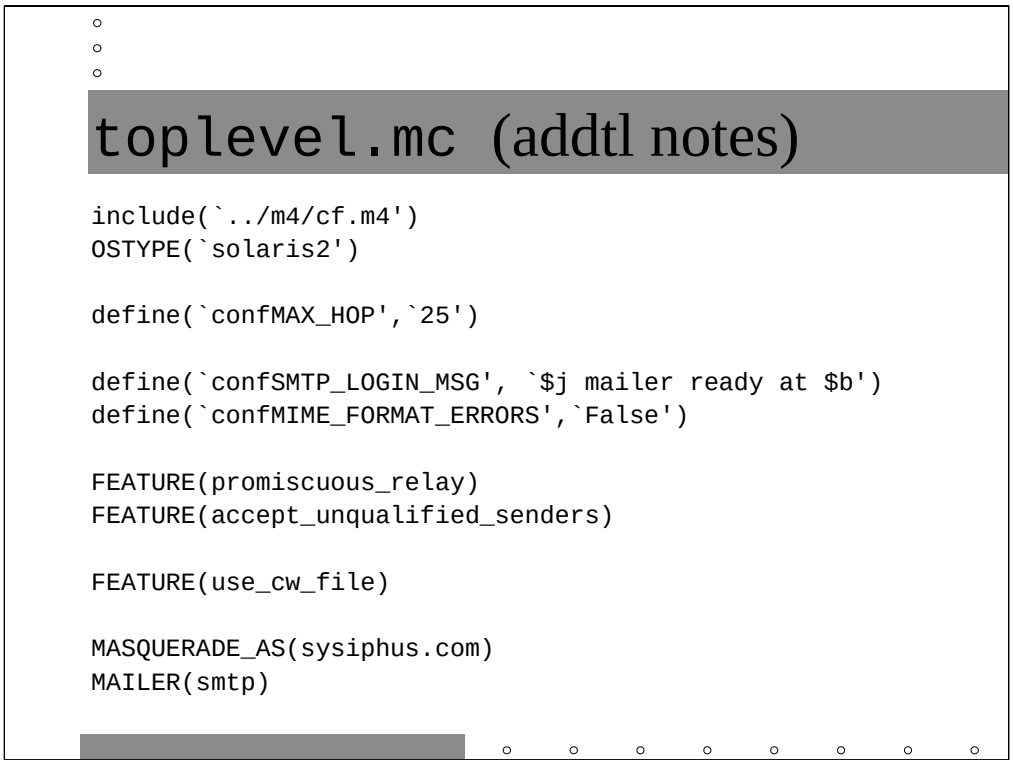
toplevel.mc (addtl notes)

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')

define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')

FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)

FEATURE(use_cw_file)

MASQUERADE_AS(sysiphus.com)
MAILER(smtp)
```

(slide 2 of 3)

Sendmail v8 by default formats error messages (bounces, etc.) as MIME multipart documents. Set `MIME_FORMAT_ERRORS` to `False` if your site doesn't widely use MIME compliant mail readers.

The growth of spam on the Internet has made it necessary for Sendmail to change many of its previous behaviors with the latest v8.9. Relaying, spam, and related issues will be discussed more completely in the section *DNS and Sendmail vs. Firewalls*, but for now we will disable the new Sendmail v8.9 behaviors with `FEATURE(promiscuous_relay)`-- your mail server will accept messages from any sender to any recipient (including messages from outside your company to recipients outside your company)-- and `FEATURE(accept_unqualified_senders)`-- Sendmail will accept messages from addresses that do not include a domain name.



(slide 3 of 3)

The `use_cw_file` feature causes Sendmail to look for a file called `/etc/mail/local-host-names` (Prior to Sendmail v8.10, the file was called `sendmail.cw` which is why the feature has the name it does). This file contains a list of all domains that should be considered "local" to this machine. Without this feature you have to modify your Sendmail configuration every time you inherit a new domain, which turns out to be a pain. With `use_cw_file` you just update the `local-host-names` file and restart Sendmail.

The `MASQUERADE_AS` directive says that all mail sent from this machine should have return addresses like `user@sysiphus.com`. Without this directive, the addresses would look like `user@mailgate.sysiphus.com`.

Which brings us to the question, what names should go in the `local-host-names` file? Since we're masquerading as `sysiphus.com`, we should definitely put that in. We should never emit mail that appears to come from `user@mailgate.sysiphus.com`, but it wouldn't hurt us to put that name in `local-host-names`. Always try to follow the "be strict in what you send out, liberal in what you accept" rule.

`MAILER(smtp)` means that this machine will be doing standard SMTP (Simple Mail Transfer Protocol-- the standard protocol for moving mail around the Internet) mail. There are other mailers, such as UUCP, to support various archaic systems. We won't use any mailers but SMTP in this tutorial. The general rule of thumb is that `MAILER` declarations should always go last in your m4 files.

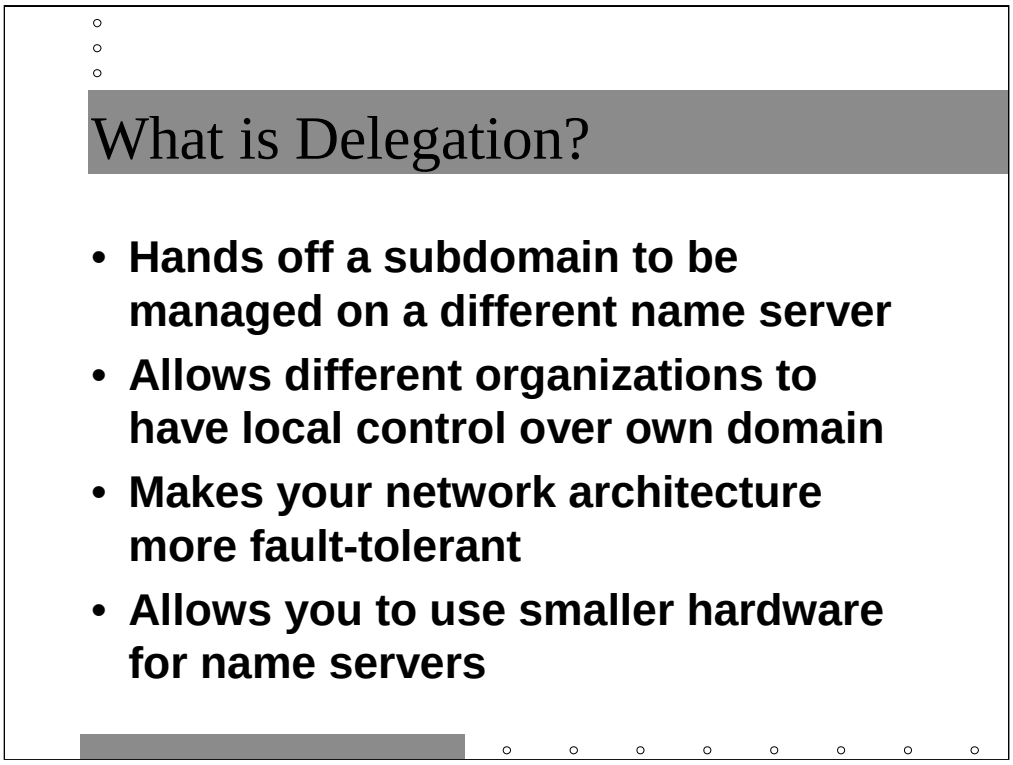
The `corp` Subdomain

- ***Delegating Your First Domain***
- ***Configuring the `corp` Domain***
- ***POP/IMAP***

Congratulations! You just suffered through the basics of configuring DNS and Sendmail and emerged relatively unscathed.

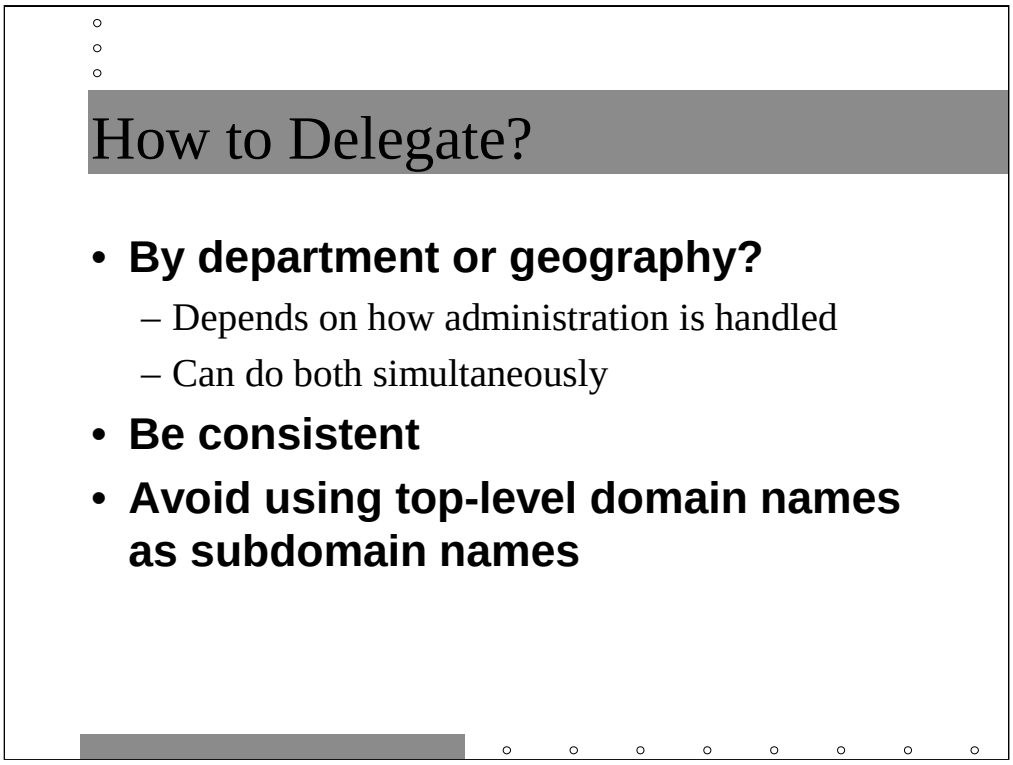
In this section we'll complicate our initial example by creating a new subdomain called `corp.sysiphus.com` and then *delegating* that domain to another server. Delegating a domain means handing it off to another (administrative) group to manage on their own. This is how DNS scales for large companies.

We'll also digress a bit in this section to talk about POP (Post Office Protocol) mail and IMAP-- good systems for allowing PC users to read mail from a Unix server.



As we just mentioned, delegation is a mechanism for handing off control of part of your domain to a different machine and a different group of administrators. By having their own local domain with their own servers, DNS administration can be more efficient and the organization won't need to suffer as badly if there is a problem on the top level name server or the name server belonging to some other organization. More than anything, delegation eliminates some political arguments (although it can cause others).

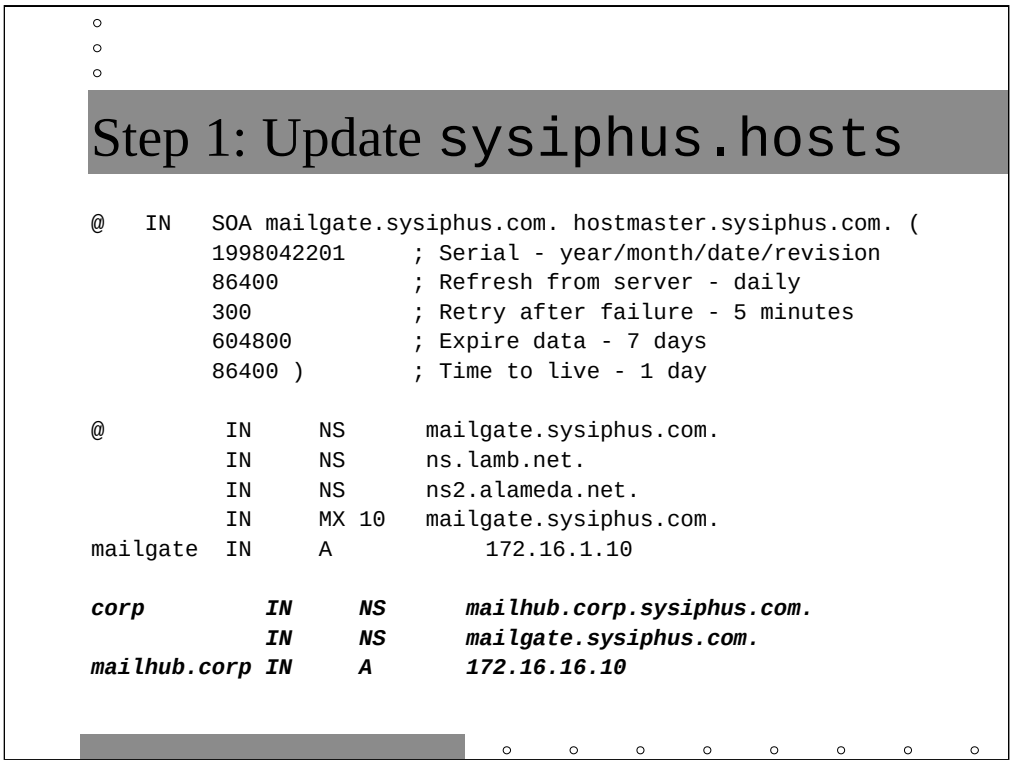
Another nice aspect of subdomains and delegation is that you can have the same hostname in different domains, e.g. `mailhub.eng.sysiphus.com` and `mailhub.corp.sysiphus.com`. This makes delegation a particularly useful technique for integrating companies you might acquire-- just give the new company its own subdomain and you don't have to worry about telling them they have to change all their important machine names.



One political argument that delegating subdomains causes is how the subdomains should be named. You can go for functional names (`eng.sysiphus.com` and `corp.sysiphus.com`) or geographical names (`boston.sysiphus.com`, `washington.sysiphus.com`, etc.) or both (`sfo.eng.sysiphus.com`). The right answer to this question is "name them based on how your administrative staff is organized". Remember that the primary function of delegation is to spread the administrative load appropriately. Chances are, your administration is primarily arranged on a functional level and then perhaps on a geographical basis.

Whatever you do, be consistent! Part of the goal here is to make your domain names memorable by humans so they can connect to your servers and address email properly. Don't have a `boston.sysiphus.com` and an `eng.sysiphus.com` and a `corp.washington.sysiphus.com`!

Also, avoid using `com`, `edu`, `net`, `org`, `mil`, `int`, `gov`, `arpa`, etc. as subdomain names. Until you understand more about how DNS servers do lookups (which we won't discuss in this course), you'll just have to take my word for this one.



The first step in delegating a domain is adding `NS` records for the subdomain in the higher level zone database (note that we remembered to update the serial number after we made this change!). In this case we're saying that `mailhub.corp.sysiphus.com` and `mailgate.sysiphus.com` will be name servers for `corp.sysiphus.com`. We're going to make `mailhub` the master server and `mailgate` a slave server (more on this shortly, promise!) but the actual order of the `NS` records doesn't imply anything.

Then we have to add an `A` record for `mailhub.corp`(`.sysiphus.com`--remember the current domain name is appended if there's no trailing "."). It's not always the case that the upper level name server is a slave server for the subdomain, and the top level machine might have no way of knowing what the IP address is of the machine it just delegated the domain to. When an outside server asks the upper level server who owns the `corp.sysiphus.com` domain, it wouldn't be able to give a complete answer without the IP address and the delegation would break down. Always add the extra A record! These A records are generally referred to as *glue records.*

Btw, if you look at the `named.ca` file, it's entirely made up of glue records. The file is nothing but a listing of NS records for the root zone and corresponding IP addresses.

Step 2: Update `sysiphus.rev`

```
@   IN   SOA mailgate.sysiphus.com. hostmaster.sysiphus.com. (
         1998042201      ; Serial - year/month/date/revision
         86400           ; Refresh from server - daily
         300             ; Retry after failure - 5 minutes
         604800          ; Expire data - 7 days
         86400 )         ; Time to live - 1 day

@        IN     NS      mailgate.sysiphus.com.
         IN     NS      ns.lamb.net.
         IN     NS      ns2.alameda.net.

16       IN     NS      mailhub.corp.sysiphus.com.
         IN     NS      mailgate.sysiphus.com.

10.1     IN     PTR     mailgate.sysiphus.com.
```

We're also going to delegate a portion of our network space to this new domain. In this case, we're giving them the `172.16.16.0` network. Giving out chunks of address space is tricky-- you want to have enough room to grow, but not waste space that you're never going to use. It's also best to hand out address space based on multiples of powers of 2-- it's a big win when you start summarizing network routing information (if you don't know what this means, don't sweat it).

Note that there's no glue record in this file. This is an `in-addr.arpa` zone so A records don't belong here. We only need the one in the forward zone file.

Otherwise the update is similar to the one in the previous slide-- including updating the serial number!

Step 3: Create `mailhub named.conf`

```
options {
        directory "/etc/namedb";
};

zone "corp.sysiphus.com" {
        type master;
        file "corp.hosts";
};
```

(continued on next slide)

Next we need to start configuring BIND on `mailhub.corp.sysiphus.com` by creating a `named.conf` file. Once again, we're putting all of our zone files in `/etc/namedb`. This machine is going to be the master server for `corp.sysiphus.com`.

Step 3 (continued)

```
zone "16.16.172.in-addr.arpa" {
        type master;
        file "corp.rev";
};

zone "." {
        type hint;
        file "named.ca";
};
```

We're also the master server for the `172.16.16.0` network reverse domain. And of course we need a `named.ca` file.

Note that this file is substantially the same as the `named.conf` file on `mailgate.sysiphus.com`. This stuff isn't rocket science-- the same principals apply to configurations throughout your enterprise. Life's going to get more complicated quickly, though.

Step 4: Create `corp.hosts`

```
@ IN SOA mailhub.corp.sysiphus.com. hostmaster.corp.sysiphus.com. (
         1998042200      ; Serial - year/month/date/revision
         86400           ; Refresh from server - daily
         300             ; Retry after failure - 5 minutes
         604800          ; Expire data - 7 days
         86400 )         ; Time to live - 1 day

@       IN      NS      mailhub.corp.sysiphus.com.
        IN      NS      mailgate.sysiphus.com.
        IN      MX 10   mailhub.corp.sysiphus.com.

mailhub IN      A       172.16.16.10
```

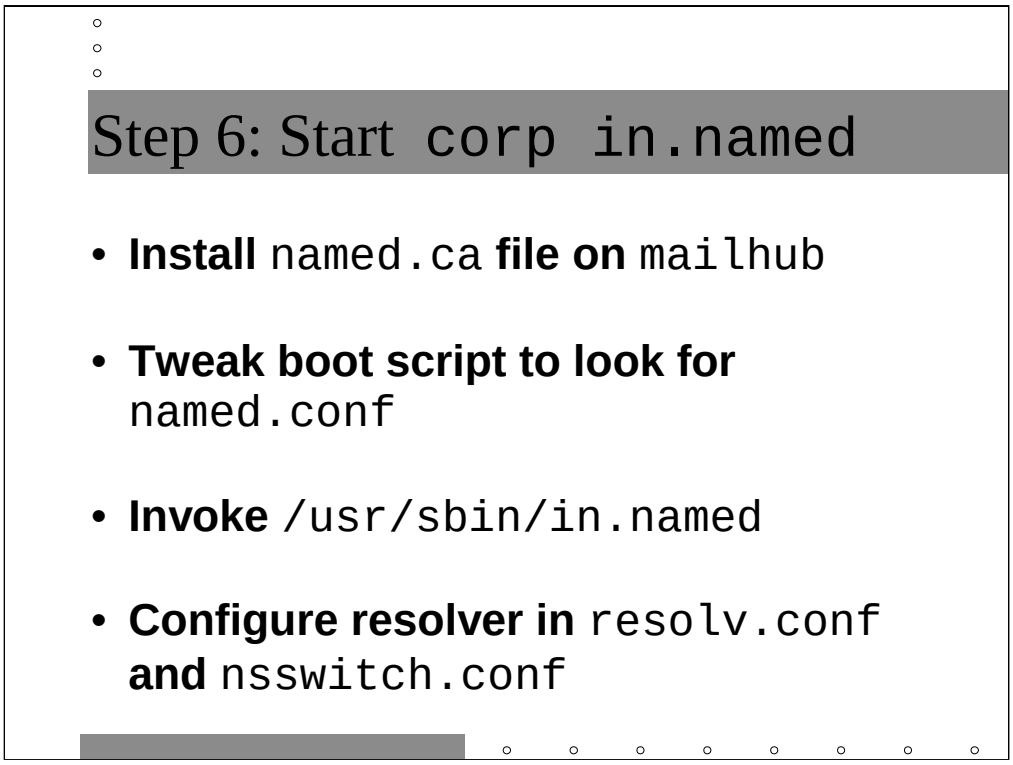
Again this file is nearly the same as the corresponding file on `mailgate.sysiphus.com`. We've changed the primary name server and email information in the `SOA` record, modified the `NS` and `MX` records appropriately, and put in the `A` record for `mailhub.corp.sysiphus.com`.

Why don't we need glue in this file so we can find `mailgate.sysiphus.com`? We can always do a normal DNS lookup to find the address of this machine: ask a root nameserver who's the DNS server for `sysiphus.com` and then do the lookup there. `mailgate` couldn't do the same for us because it's "above" us in the DNS hierarchy-- it has nobody to ask because it's the one who's supposed to know already!

Step 5: Create `corp.rev`

```
@ IN SOA mailhub.corp.sysiphus.com. hostmaster.corp.sysiphus.com. (
         1998042200      ; Serial - year/month/date/revision
         86400           ; Refresh from server - daily
         300             ; Retry after failure - 5 minutes
         604800          ; Expire data - 7 days
         86400 )         ; Time to live - 1 day

@        IN      NS      mailhub.corp.sysiphus.com.
         IN      NS      mailgate.sysiphus.com.

10       IN      PTR     mailhub.corp.sysiphus.com.
```

Again this file is nearly the same as the corresponding file on `mailgate`. Note that there's only one octet listed in the `PTR` record for this zone because we're configuring `16.16.172.in-addr.arpa`, or network `172.16.16.0` (in network speak, this is a "class C" network as opposed to a "class B" network like `172.16.0.0`).



Now we need to get the `mailhub.corp` name server running. Download the `named.ca` file and tweak the boot script to look for `named.conf` instead of `named.boot`. Then invoke `/usr/sbin/in.named` from the command line.

You will also want to configure the resolver on `mailhub.corp`. The `resolv.conf` file for this machine would be

```
domain corp.sysiphus.com
nameserver 172.16.16.10
nameserver 172.16.1.10
```

Note that we list both `mailhub.corp` and `mailgate` as name servers for redundancy.

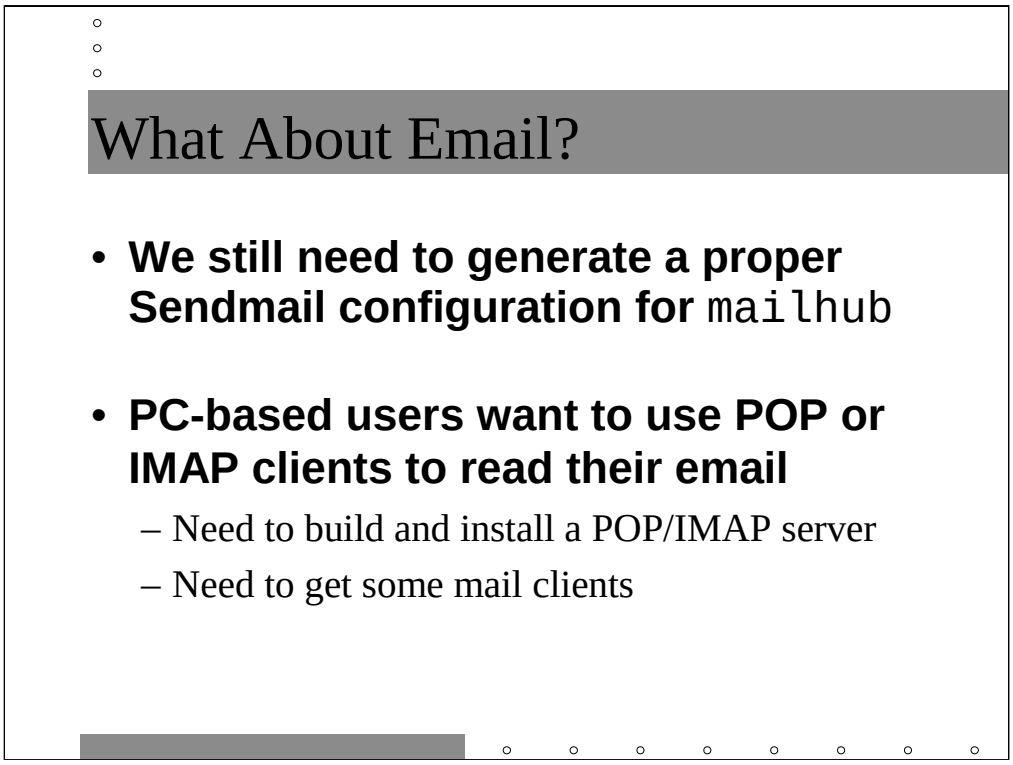
Step 7: Modify `mailgate named.conf`

```
zone "corp.sysiphus.com" {
      type slave;
      file "corp-hosts.zone";
      masters { 172.16.16.10; };
};

zone "16.16.172.in-addr.arpa" {
      type slave;
      file "corp-rev.zone";
      masters { 172.16.16.10; };
};
```

Now we need to go back to `mailgate` and configure the new slave zones it has just inherited. The definition for a slave zone is similar to the definition for a master zone except for the new `masters` definition. The list after the `masters` keyword is the IP address (not hostname!) of the machine the slave should get zone updates from. Note that "masters" is a misnomer since it is actually permissible to get zone updates from another slave server (can be good for spreading the load around).

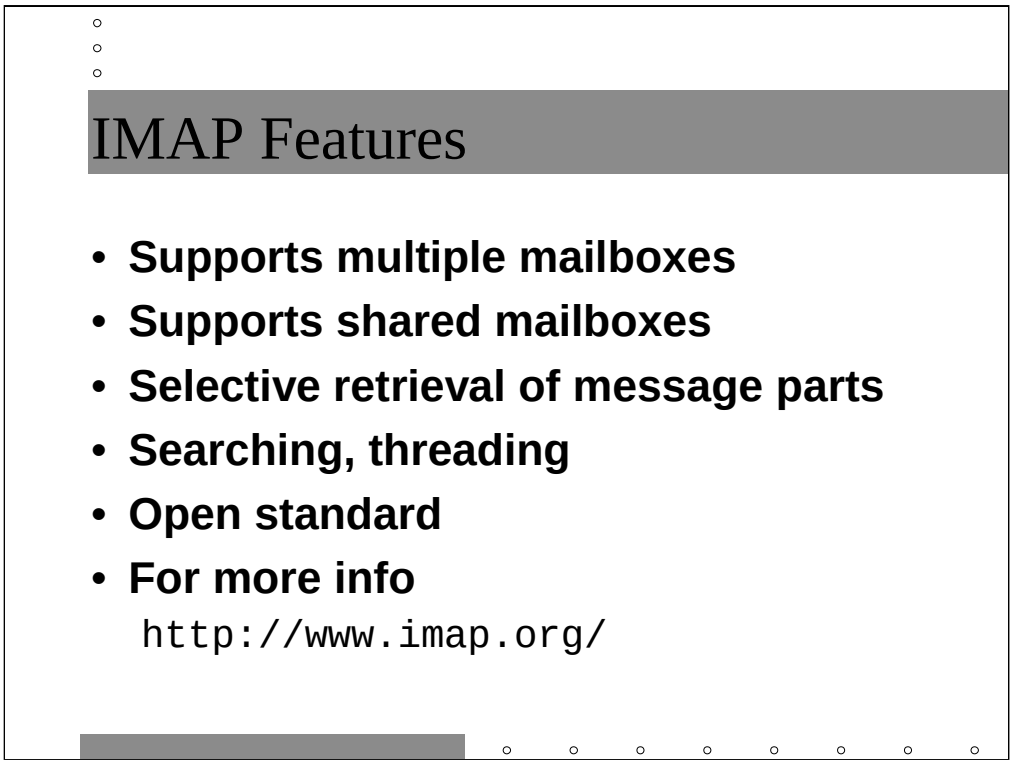
Note that the zone files are not ASCII files in the same format as the regular master zone files. Don't go editing slave zone files! Make updates on the master only (and don't forget to update the serial number!).

That's it for configuring DNS for a zone delegation. It gets automatic once you get the hang of it.



We still need to generate a `sendmail.cf` for `mailhub.corp`, but don't worry because it's going to look a lot like the one for `mailgate`.

The we need to come up with a solution to allow the user community to get their email from `mailhub.corp`. In most companies, the corporate folks tend to be PC users (meaning Macs or Wintel machines). POP (the Post Office Protocol) is an old an established means for providing email services to PCs, and IMAP is the new standard which has some nice additional functionality. In general, most IMAP servers include POP support as well.

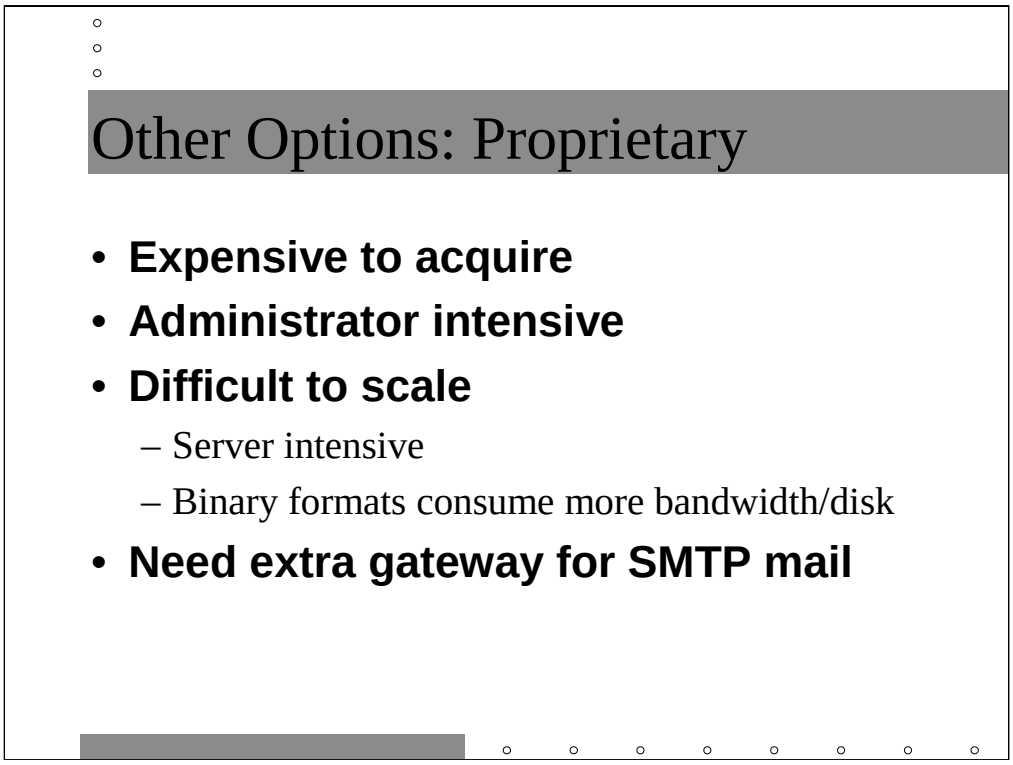


One major benefit to IMAP over POP is that users can maintain multiple separate mail folders, possibly on different servers. Users can share mailboxes, with a fairly high granularity of access control.

From protocol perspective, IMAP allows users to only fetch parts of a message (just the headers, or perhaps the body of a message but not a binary attachment). This makes IMAP useful over low-bandwidth connections. The IMAP protocol also supports searching.

IMAP generally assumes a long-running connection with the mail server. This is different from POP mail where the user connects to fetch their mail and then disconnects immediately. Expect IMAP servers to need more CPU power and memory as compared to your existing POP servers.

Links to the currently available IMAP servers/clients and related software can be found at The IMAP Connection (`http://www.imap.org/`) which is maintained at the University of Washington (one of the two popular free IMAP implementations comes out of the University-- the other is "Cyrus" from Carnegie Mellon).



Many sites have one or more communities of users that read email with proprietary email systems such as Microsoft Exchange or Lotus Notes. While such systems may have more integrated "groupware" type functionality, they also come with a host of problems that aren't present in standards-based email systems.

First and foremost, these proprietary systems usually cost a lot of money. Aside from the basic software cost, many organizations end up spending a lot of money on consulting help to get the systems installed properly. The systems also tend to require many more administrators per user as compared to standards-based systems.

Being monolithic, "closed" applications, there really is no good way to scale these products other than buying bigger servers-- something which rapidly reaches a point of diminishing marginal returns. Because users are used to shipping Word documents and other binary formats around as email messages, more network bandwidth and disk space are consumed.

Also note that when these organizations wish to send email via the Internet, SMTP functionality is usually an add-on product (and often extremely buggy). Also, these users will tend to continue to send Word documents as email messages, even though the recipients (often members of your own company) will be unable to read them, being Unix text mail users.

Sendmail config for `mailhub`

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')

define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')

FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)

FEATURE(use_cw_file)

MASQUERADE_AS(corp.sysiphus.com)
MAILER(smtp)
```

This file looks exactly like the `m4` file for `mailgate` except that `mailhub.corp` masquerades as `corp.sysiphus.com`. The `local-host-names` file on `mailhub.corp` should contain `corp.sysiphus.com` and `mailhub.corp.sysiphus.com`. Putting the version with the machine name into the `local-host-names` file is particularly important on `mailhub.corp` because users can easily mis-configure their POP clients to send out email with the machine name in the return address.

Building UWash IMAP Server

- **Get sources**

  `ftp://ftp.cac.washington.edu/imap/imap.tar.Z`

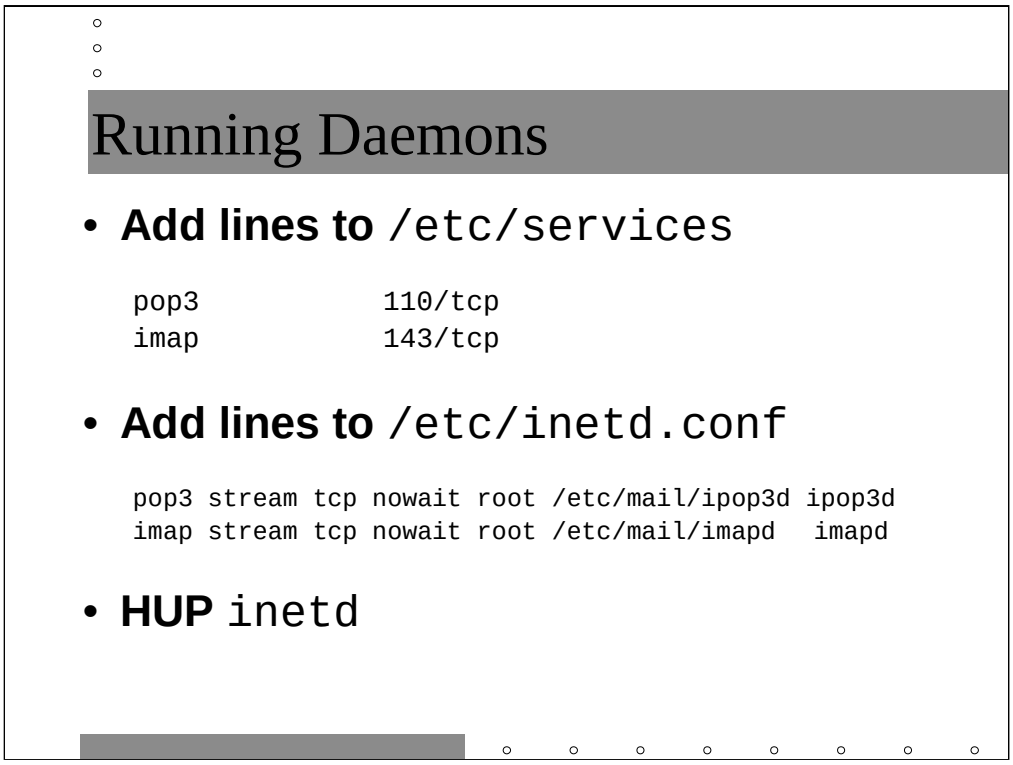
- **Unpack, build, install**

```
% zcat imap.tar.Z | tar xf -
% cd imap-4.x
% make <tag>                          (lots of output)
% cp imapd/imapd ipopd/ipop3d /etc/mail
```

The University of Washington provides a free IMAP server that includes POP support. Most sites that use Open Source IMAP/POP servers use either the UWash server or Cyrus from Carnegie Mellon. Cyrus has the distinct disadvantage, however, that it doesn't keep mailboxes in standard Unix `/var/mail` format (for performance reasons). This means that you and your users must only access the server via POP or IMAP-- not always desirable. The UWash server is also generally easier to install and get working.

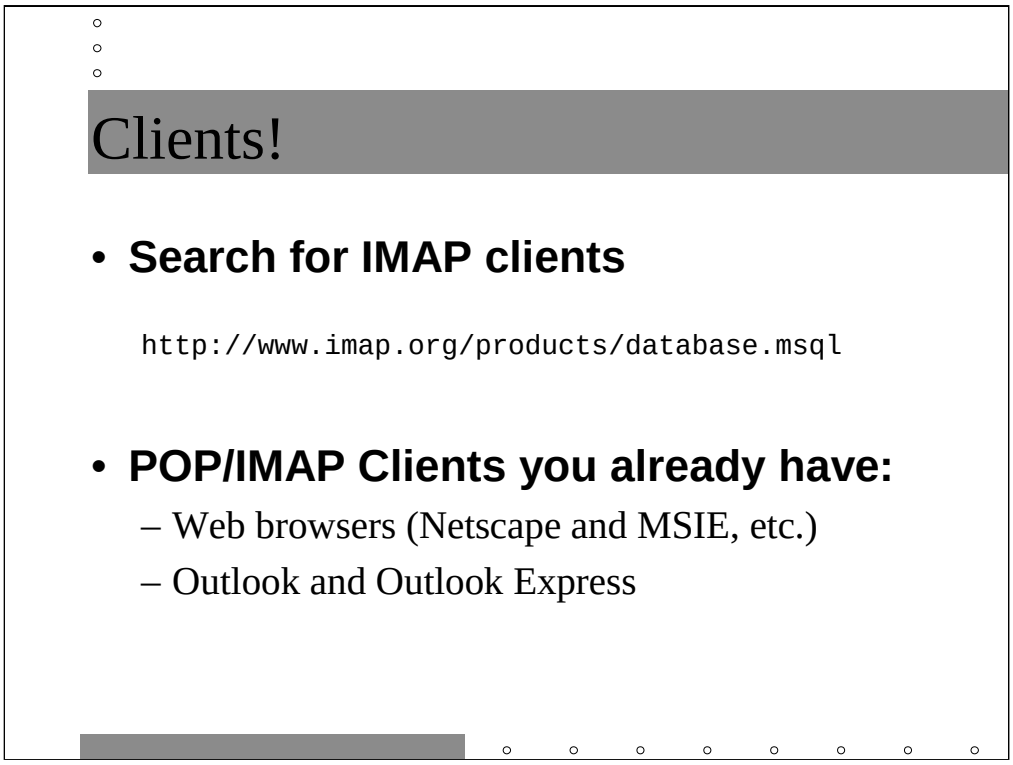
Suck down the source, unpack it, and run the `make` with an appropriate argument for your system. There are dozens of three letter OS/compiler tags documented in the toplevel `Makefile` (for example, the normal Solaris tag is `sol`, but Solaris with GCC is `gso`… go figure). The make command then creates the appropriate `Makefiles` to build the software and then does the build.

After you build the daemons, you probably want to install them on a non-NFS mounted directory, so you don't end up hanging all your users if your NFS server goes away. `/etc/mail` is a fine place for the binaries on Solaris systems.



The POP/IMAP daemons get run out of `inetd` so you don't have to have a separate daemon process running on your system all the time. You can run the daemons in standalone mode if your performance is really suffering, but most users probably won't even notice the difference.

Don't forget you need to `kill -HUP` the `inetd` process to get the changes to take effect.



The University of Washington IMAP.org site has a searchable database which lists all of the know IMAP clients (and servers and bunches of other stuff). If you're searching for just the right thing for your site, it's a good starting point.

In general, your users probably already have POP/IMAP clients installed on their systems. Most users will probably be happy enough using their favorite Web browser or Outlook to read their email.

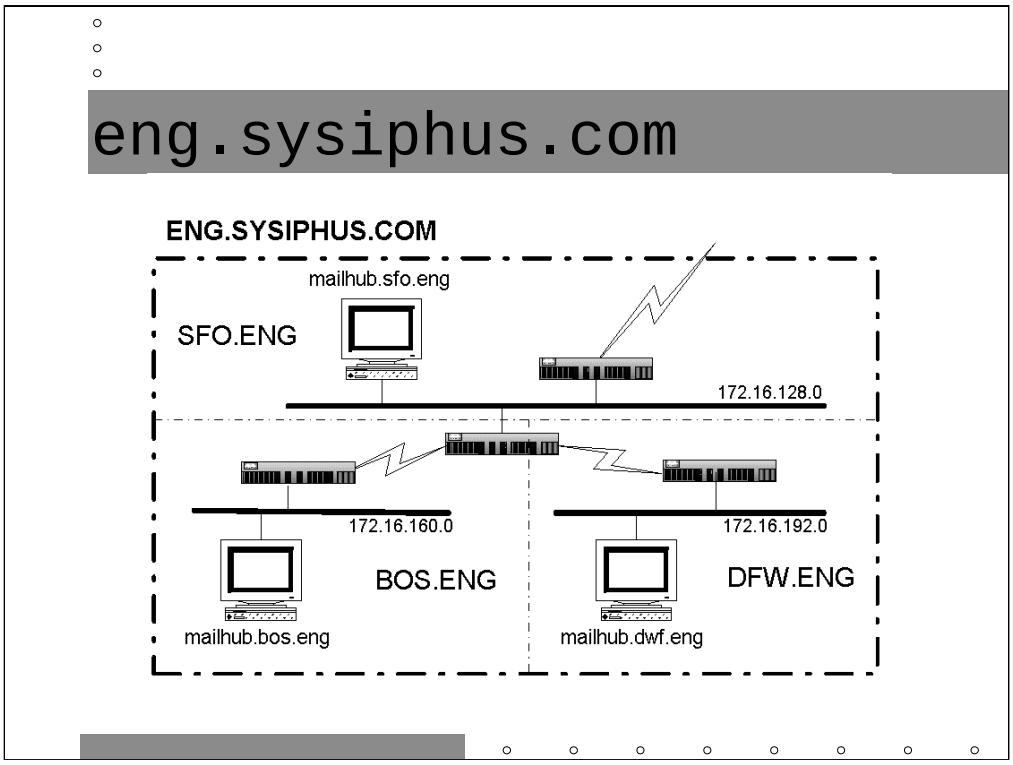
The Engineering Subdomain

- ***More Delegation!***
- ***Unix Mail Clients***
- ***Stupid Alias Tricks***

In setting up our hypothetical Engineering domain, we're going to do two levels of delegation. That will be enough delegation examples to last anybody a lifetime.

Then there will be some hints on configuring UNIX workstations as mail clients.

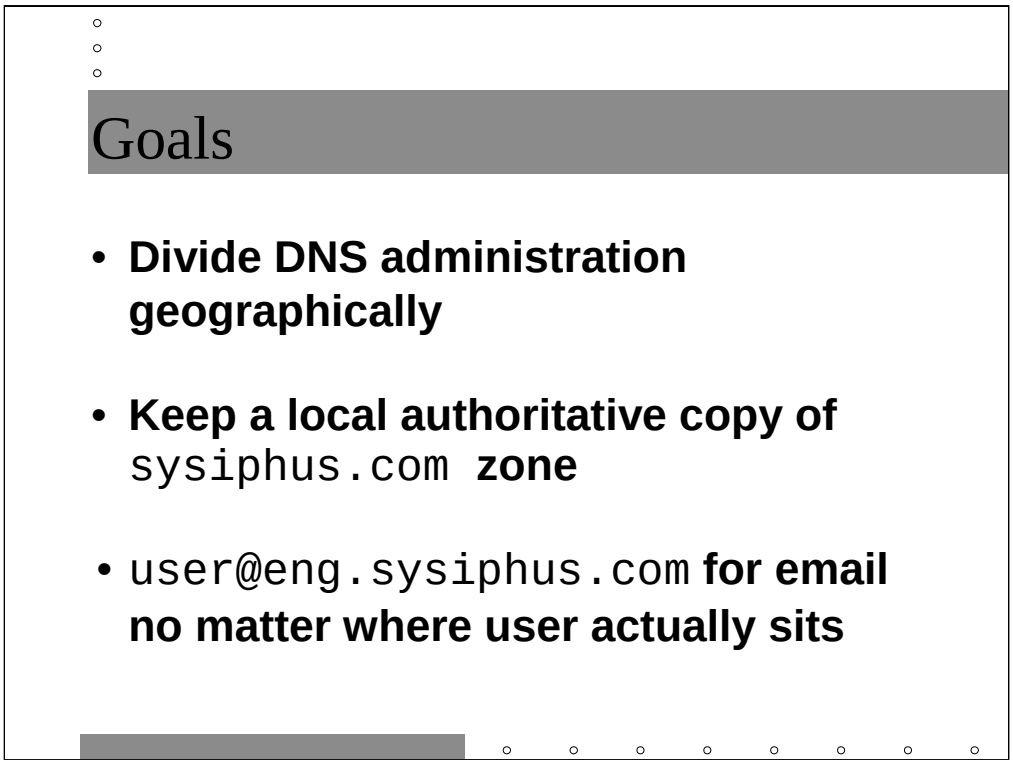
Finally we'll spend some time talking about email aliases, including some hints on using aliases while hiding DNS subdomains.



Let's look at that network diagram of the Engineering domain again.

The Engineering domain is made up of three geographically distant sites-- one in the San Francisco Bay Area, one in Boston, and one in the area around Dallas, Texas. As an easy to remember shorthand, we'll named each of these regions by its closest major airport (`SFO`, `BOS`, and `DFW`).

The trick is that we want all mail for this domain to be addressed as `user@eng.sysiphus.com` regardless of whether the user sits in San Francisco, Boston, or Dallas. This means that somehow we're going to have to maintain a mapping of users to physical location. More on this to come.



Each of the SFO, BOS, and DFW sites have their own local admins, so we want to further subdomain the Engineering domain by geography. Since the Engineering domain connects to the rest of the company in SFO, we're going to let the SFO admins manage the top level eng.sysiphus.com domain in addition to their own geographic domain, sfo.eng.sysiphus.com. Any of the Engineering sites could be the master for eng.sysiphus.com, we're just picking this one more or less arbitrarily.

Let's say the network connection between SFO and the rest of the company is relatively low bandwidth or unstable. The SFO DNS server is going to get set up as a slave server of the sysiphus.com domain to help make our DNS setup more fault-tolerant.

And, as mentioned in the previous slide, we need to get the mail delivery worked out properly.

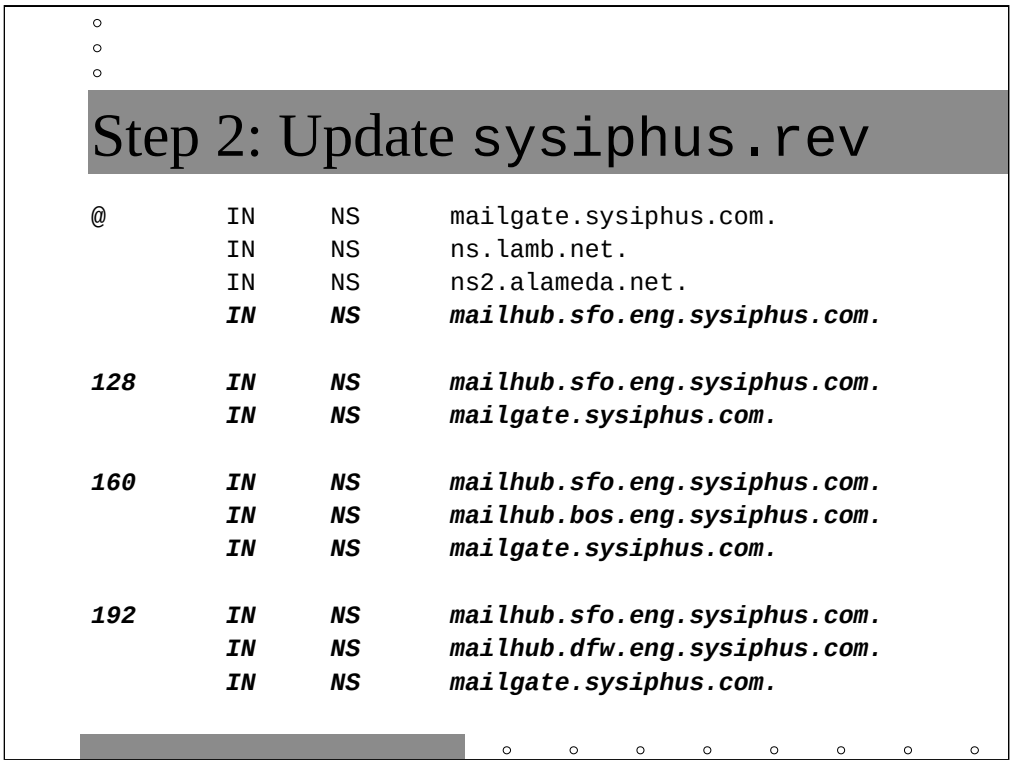
Step 1: Update `sysiphus.hosts`

```
@         IN     NS      mailgate.sysiphus.com.
          IN     NS      ns.lamb.net.
          IN     NS      ns2.alameda.net.
          IN     NS      mailhub.sfo.eng.sysiphus.com
          IN     MX 10   mailgate.sysiphus.com.
mailgate  IN     A          172.16.1.10

eng                IN     NS    mailhub.sfo.eng.sysiphus.com.
                   IN     NS    mailhub.bos.eng.sysiphus.com.
                   IN     NS    mailhub.dfw.eng.sysiphus.com.
                   IN     NS    mailgate.sysiphus.com.
mailhub.sfo.eng    IN     A     172.16.128.10
mailhub.bos.eng    IN     A     172.16.160.10
mailhub.dfw.eng    IN     A     172.16.192.10
```

Note that this is only part of `mailgate`'s `sysiphus.hosts` file-- we're not showing the deleted the `SOA` record and delegations for the `corp` domain to make everything fit on the slide.

Here we delegate the `eng` domain just as we did the `corp` domain. All of the various `SFO`, `BOS`, and `DFW` machines, as well as `mailgate` will be name serves for this domain. The `SFO` server will be the master server.

Then we add glue records for all of the Engineering name servers.



Again this is only part of the `sysiphus.rev` file with the `SOA` record and `corp` network delegation not shown.

Each site has its own class C network. The nameserver at each site will be the master server for its own `in-addr.arpa` domain. In addition, the `SFO` server is a slave server for the other two sites' networks. `mailgate` is also a slave server for all `in-addr.arpa`. domains.

Why did we make `SFO` a slave server for the two other sites' networks but not make `DFW` and `BOS` back each other (and `SFO`) up? Consider that `BOS` and `DFW` have to talk through `SFO` to reach one another. Chances are, most problems which will take out connectivity to the `SFO` nameserver will also take out connectivity between `BOS` and `DFW`. It won't do any good to have a backup nameserver in a site that's unreachable.

Note also that we've once again assigned netblocks according to powers of 2. The `SFO` network is at `172.16.128.0` (128 is 2^7) and the other netblocks are each 32 (2^5) network numbers away.

Step 3: `mailhub.sfo named.conf`

```
options { directory "/etc/namedb"; };

zone "." { type hint;
           file "named.ca"; };

zone "eng.sysiphus.com" {
        type master;
        file "eng.hosts";
};

zone "sfo.eng.sysiphus.com" {
        type master;
        file "sfo-eng.hosts";
};
```

(continued on next slide)

OK., `mailhub.sfo.eng.sysiphus.com` (quite a mouthful-- you can see why most sites don't delegate domains much deeper than a couple of levels) is the master server for both `eng.sysiphus.com` and `sfo.eng.sysiphus.com`. Once again we're putting all of our zone files in `/etc/namedb`, including our very own copy of `named.ca`.

Step 3 (continued)

```
zone "sysiphus.com" {
        type slave;
        file "sysiphus.zone";
        masters { 172.16.1.10; };
};

zone "bos.eng.sysiphus.com" {
        type slave;
        file "bos-eng.zone";
        masters { 172.16.160.10; };
};

zone "dfw.eng.sysiphus.com" {
        type slave;
        file "dfw-eng.zone";
        masters { 172.16.192.10; };
};
```

(continued from previous slide -- continues on next slide)

`mailhub.sfo` is also a slave server for several domains. It gets `sysiphus.com` updates from the master DNS server for the domain, `mailgate.sysiphus.com`. The DNS updates for the other Engineering zones come from their respective masters.

Step 3 (continued)

```
zone "128.16.172.in-addr.arpa" {
        type master;
        file "sfo-eng.rev";
};

zone "160.16.172.in-addr.arpa" {
        type slave;
        file "bos-eng-rev.zone";
        masters { 172.16.160.10; };
};

zone "192.16.172.in-addr.arpa" {
        type slave;
        file "dfw-eng-rev.zone";
        masters { 172.16.192.10; };
};
```

`mailhub.sfo` also serves several `in-addr.arpa` domains. It is the primary server for its own `172.16.128.0` network, and the slave server for the other two Engineering sites' networks (zone updates from the servers at those sites).

Note that this is by far the largest `named.conf` file we've seen to date. There is theoretically no limit to the number of domains, subdomains, and reverse `in-addr.arpa` domains a given machine can serve. Of course, `in.named` keeps all of its DNS data in memory, so servers with even just a few large domains can run out of memory fairly quickly and become unusably slow. Also, many slave servers for a given domain can cause the master server to become unavailable when the slave servers all attack at the same time for their zone updates.

Step 4: Create `eng.hosts`

```
@ IN SOA mailhub.sfo.eng.sysiphus.com. hostmaster.eng.sysiphus.com. (
         1998042200      ; Serial - year/month/date/revision
         86400           ; Refresh from server - daily
         300             ; Retry after failure - 5 minutes
         604800          ; Expire data - 7 days
         86400 )         ; Time to live - 1 day

@        IN      NS      mailhub.sfo.eng.sysiphus.com.
         IN      NS      mailhub.bos.eng.sysiphus.com.
         IN      NS      mailhub.dfw.eng.sysiphus.com.
         IN      NS      mailgate.sysiphus.com.

         IN      MX 10   mailhub.sfo.eng.sysiphus.com.
         IN      MX 10   mailhub.bos.eng.sysiphus.com.
         IN      MX 10   mailhub.dfw.eng.sysiphus.com.
```

(continued on next slide)

Here's the top of the `eng.hosts` file on `mailhub.sfo`. This file contains the authoritative info for the `eng.sysiphus.com` domain.

Note that in the `SOA` record, the primary name server is `mailhub.sfo.eng`, but the email contact is `hostmaster@eng.sysiphus.com`.

All three Engineering site nameservers serve the `eng.sysiphus.com` zone, as does `mailgate.sysiphus.com`. Don't panic, the glue records are on the next slide.

All three Engineering site servers also accept mail for `eng.sysiphus.com` at equal priority level. Often times you list multiple `MX` servers for redundancy. In our case, this is mostly here to support our goal of `user@eng.sysiphus.com` mail addressing.

Step 4 (continued)

```
sfo      IN       NS       mailhub.sfo.eng.sysiphus.com.
         IN       NS       mailhub.bos.eng.sysiphus.com.
         IN       NS       mailhub.dfw.eng.sysiphus.com.

bos      IN       NS       mailhub.sfo.eng.sysiphus.com.
         IN       NS       mailhub.bos.eng.sysiphus.com.
         IN       NS       mailhub.dfw.eng.sysiphus.com.

dfw      IN       NS       mailhub.sfo.eng.sysiphus.com.
         IN       NS       mailhub.bos.eng.sysiphus.com.
         IN       NS       mailhub.dfw.eng.sysiphus.com.

mailhub.sfo   IN  A        172.16.128.10
mailhub.bos   IN  A        172.16.160.10
mailhub.dfw   IN  A        172.16.192.10
```

(continued from previous slide)

Next we delegate the `SFO`, `BOS`, and `DFW` zones. Each zone's name server is primary for that domain and the other two name servers are slaves.

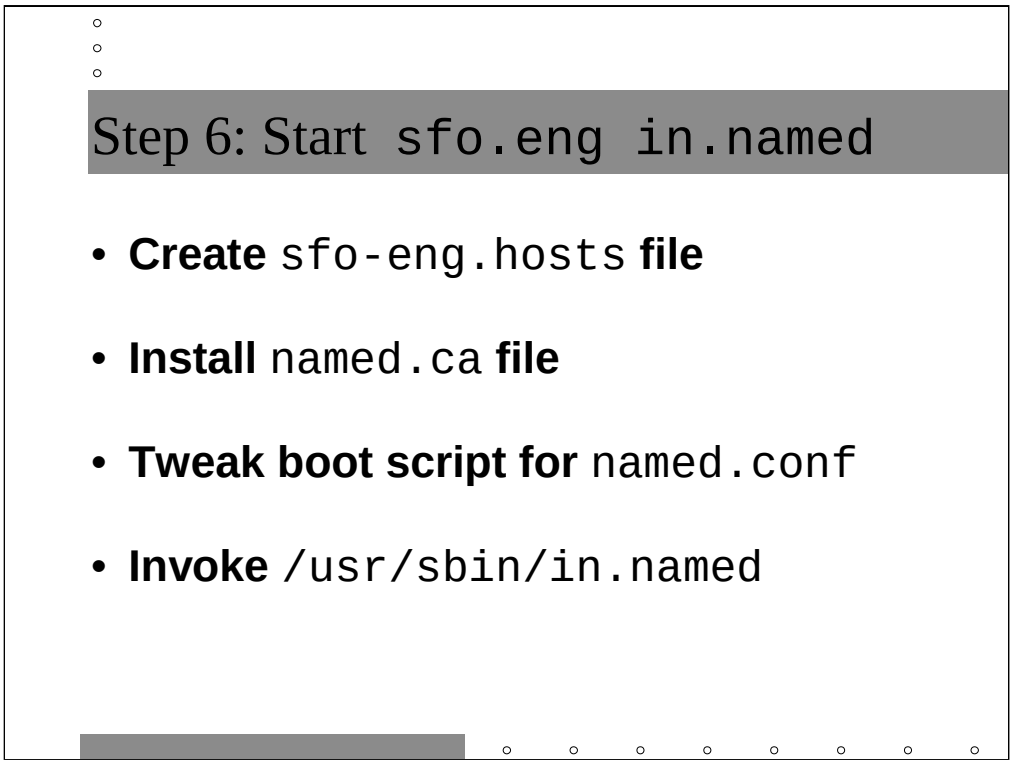
Finally, we have the glue records for the various name servers in the different Engineering domains.

Creating the `sfo-eng.hosts file` (which contains DNS info for the sfo.eng.sysiphus.com domain) is left as an exercise to the reader. Actually, all of the configuration files for all domains and subdomains of `sysiphus.com` are included in the back of this tutorial as Appendices.

Step 5: Create `sfo-eng.rev`

```
@ IN SOA mailhub.sfo.eng.sysiphus.com. hostmaster.eng.sysiphus.com. (
         1998042200      ; Serial - year/month/date/revision
         86400           ; Refresh from server - daily
         300             ; Retry after failure - 5 minutes
         604800          ; Expire data - 7 days
         86400 )         ; Time to live - 1 day

@       IN      NS      mailhub.sfo.eng.sysiphus.com.
        IN      NS      mailgate.sysiphus.com.

10      IN      PTR     mailhub.sfo.eng.sysiphus.com.
```

Here's the `in-addr.arpa` domain file for the SFO network. Note that by simply changing all `sfo`s to `bos` or `dfw`, this file is equally appropriate for each of the other Engineering networks.



Again, there's a copy of all of the config files for the `SFO`, `BOS`, and `DFW` zones in the back of this tutorial. You have to download the `named.ca` file for yourself!

Once you've got the config and zone files installed on the Engineering servers, modify your boot script to look for `named.conf` and fire up the name server on each machine.

When configuring `resolv.conf`, you want something like

```
domain sfo.eng.sysiphus.com
nameserver 172.16.128.10
nameserver 172.16.160.10
nameserver 172.16.192.10
```

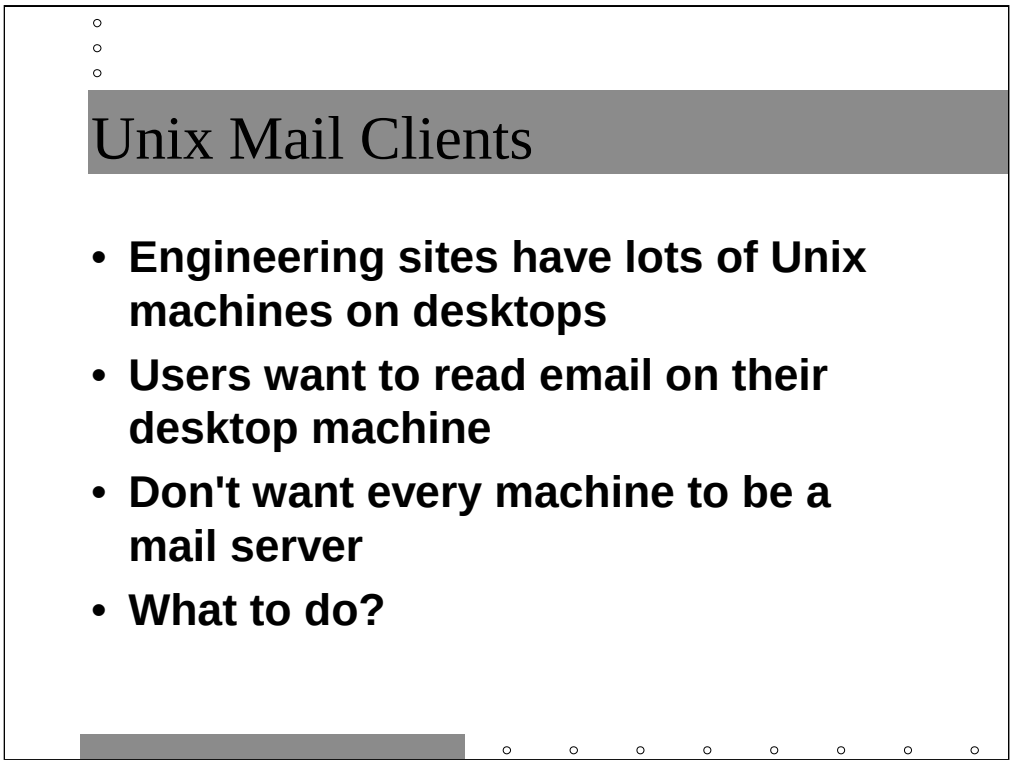
Change the `domain` specifier as appropriate for each machine, and list the local name server first in the list of name servers so that machine gets tried first.

Step 7: Modify `mailgate named.conf`

```
zone "eng.sysiphus.com" {
       type slave;
       file "eng-hosts.zone";
       masters { 172.16.128.10; };
};

zone "128.16.172.in-addr.arpa" {
       type slave;
       file "sfo-eng-rev.zone";
       masters { 172.16.128.10; };
};
```

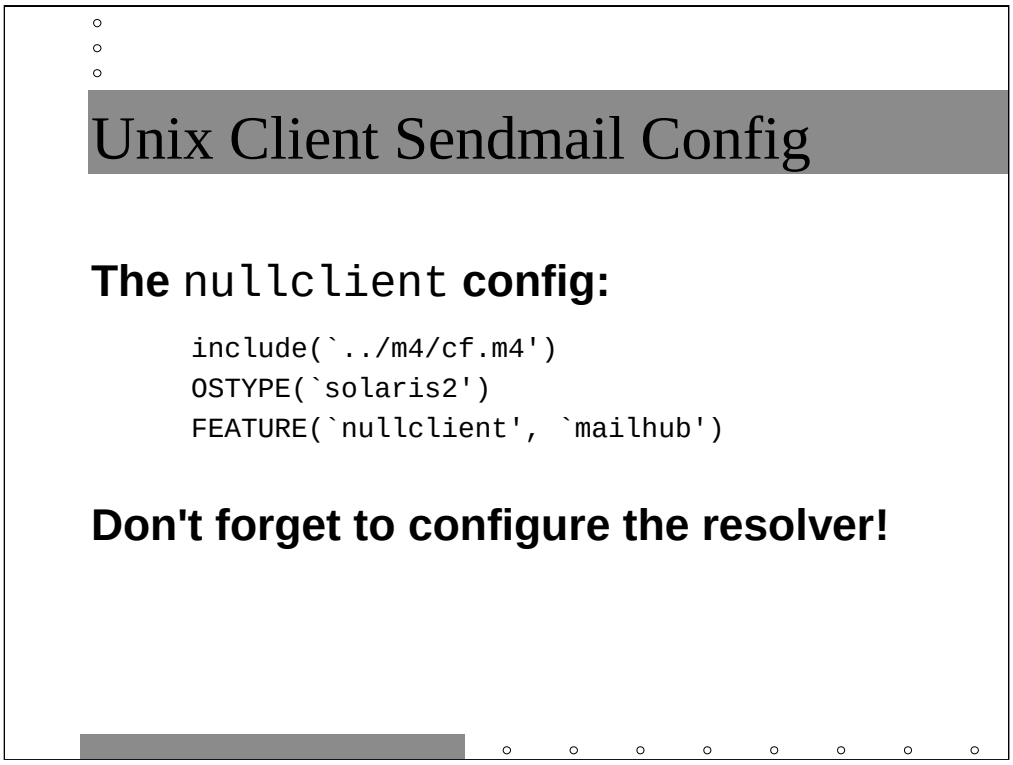
(… similar entries for other in-addr.arpa zones …)

`mailgate`'s `named.conf` now needs to get slave zone blocks for the `eng.sysiphus.com` domain as well as the three `in-addr.arpa` domains it delegated away to the Engineering sites. Two of those `in-addr.arpa` domains are not shown here due to space issues.



Engineering sites tend to have lots of various types of Unix workstations on people's desktops. Users tend to want to read mail on their desktop so that they can easily use GUI-based mail tools (`exmh`, etc.) or just not have to log into a different machine to get their mail. Unix mail programs, though, generally require a copy of the user's mailbox on whatever machine they're being run on. This means we somehow have to get the user's mailbox to their desktop.

You definitely don't want each of these machines configured to be a mail server-- it would be an administrative nightmare! In general, most sites configure their Unix workstations to rely on a central server (or group of servers) to handle mail for the site. The desktop machines mount the mail spool directory from this central machine via NFS (this is dangerous as we'll discuss in a moment) or use a separate program to fetch the mail down to the local machine. Desktop clients have their Sendmail configured to forward all mail to the central server for processing.

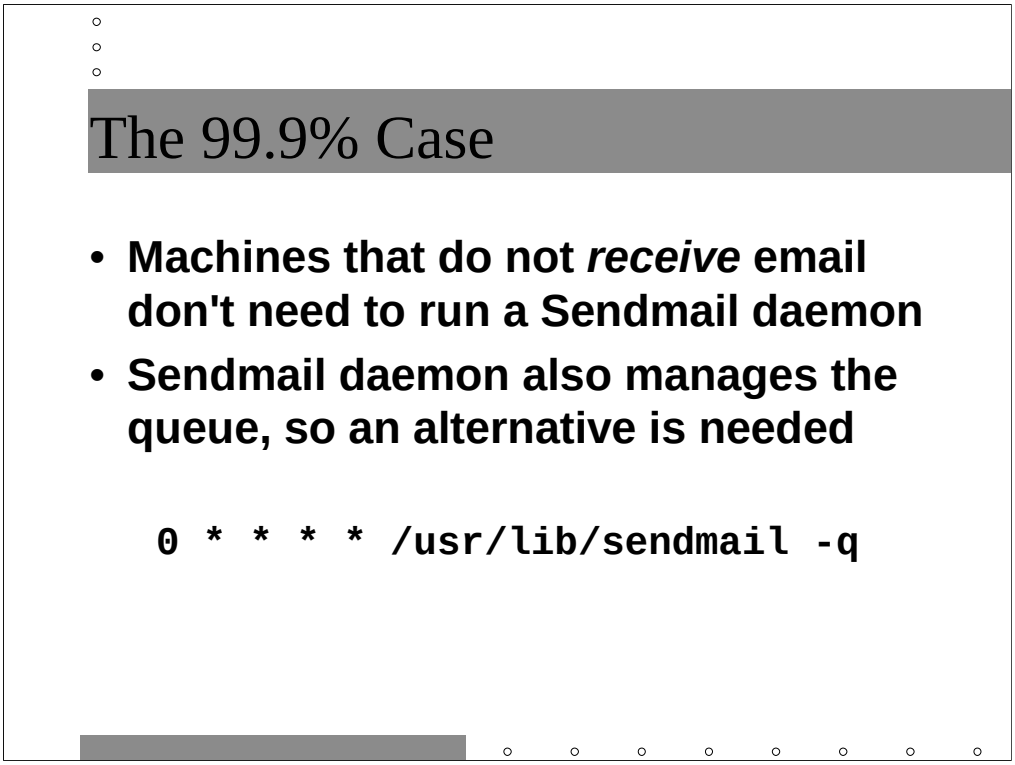


The `nullclient` FEATURE causes the local Sendmail daemon on each workstation to do nothing with mail it receives except to forward that mail to the central mail hub for delivery. We don't want to deliver mail locally because the desktop workstations won't have their own mail spool directories.

You must also make sure you have the correct `resolv.conf` and `nsswitch.conf` (or other similar file) settings on each machine. Again, the `resolv.conf` file for SFO machines should look like

```
domain sfo.eng.sysiphus.com
nameserver 172.16.128.10
nameserver 172.16.160.10
nameserver 172.16.192.10
```

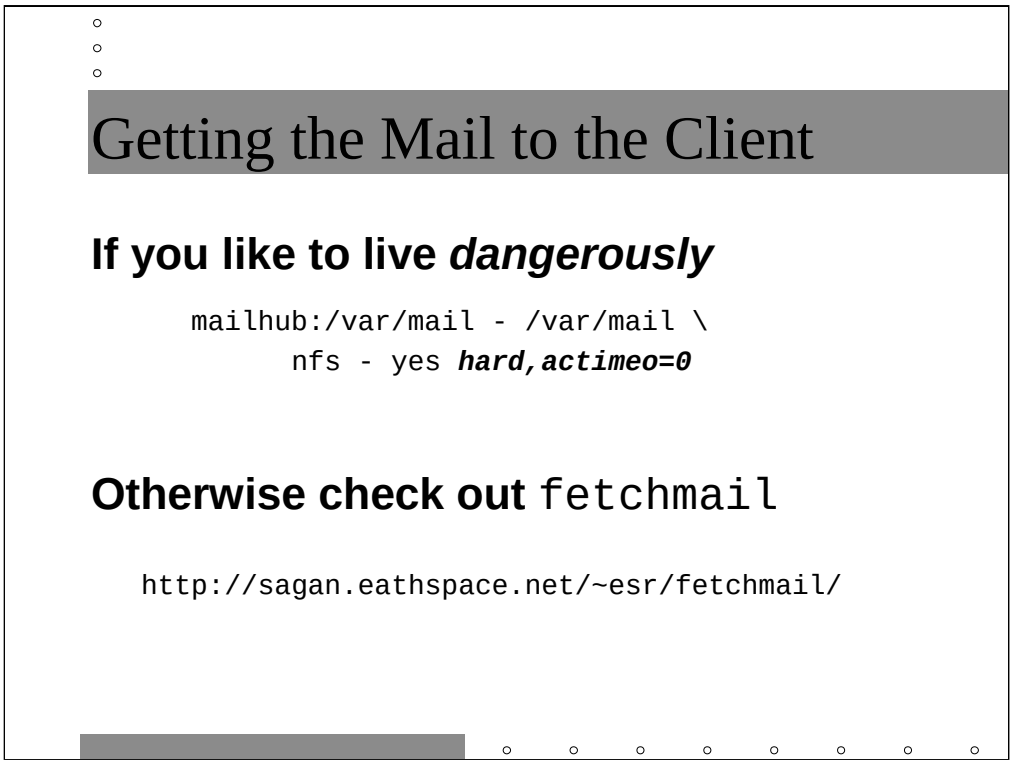
and the files for other domains should be similar (except for the domain name and name server order).



So far, we've really only been talking about the configuration of our company's mail servers. However, the vast majority of machines in your company *are not* mail servers– that is, they do not receive, store, or relay email for any other hosts. Since these machines do not ever receive email, they do not need to be running a Sendmail daemon (the primary purpose of which is to listen on port `25/tcp` for incoming email)– this is a big win from a security perspective. Processes which generate email on the local system always invoke Sendmail directly to send their email to other machines.

However, that the Sendmail daemon is also responsible for periodically flushing any queued messages from the mail queues. If you turn off the daemon on most of your machines, you will need to invoke Sendmail periodically on these hosts to run the queue– above we see an example of a line from root's crontab file which accomplishes this mission.

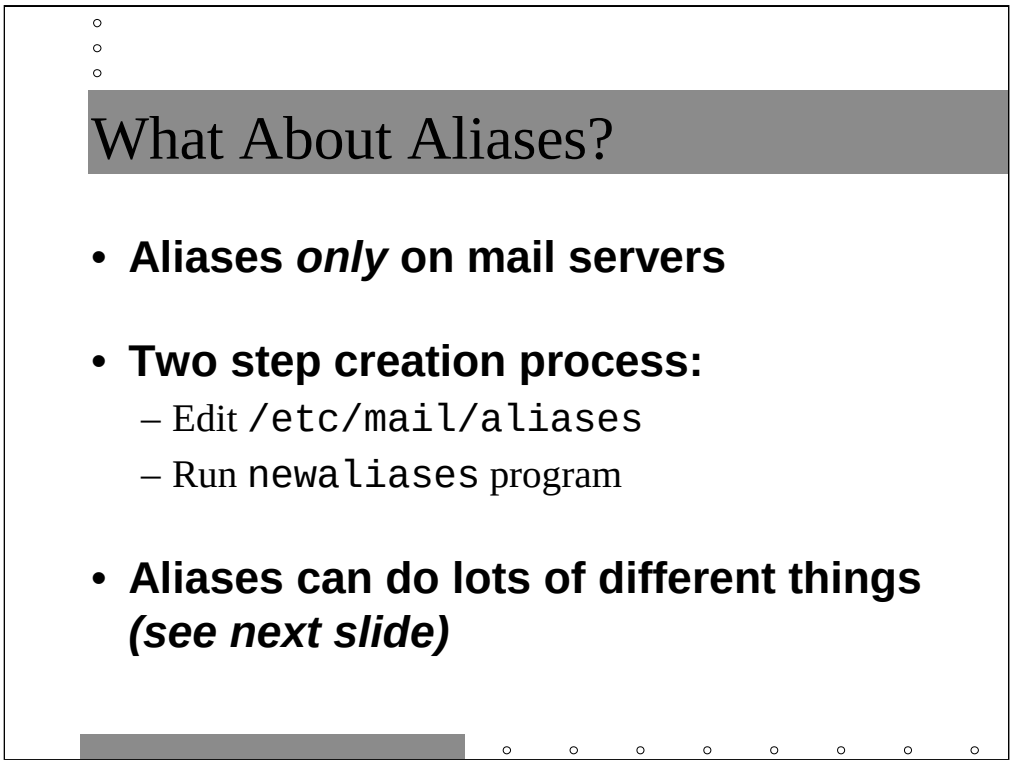
Note that it is also possible to run Sendmail as a daemon which runs the queue periodically (see the description of the `-q` flag in the manual page) but doesn't listen on port 25 (*don't* use `-bd`).



Many sites use NFS to make a central server's mail spool available to desktop clients. The problem is that locking a file between two processes across an NFS link is not a solved problem. As long as you have a completely heterogenous network and the volume of mail your users get is small, you may not notice any problems. As volume starts ramping up, however, the probability that users will start having corrupted mailboxes goes up.

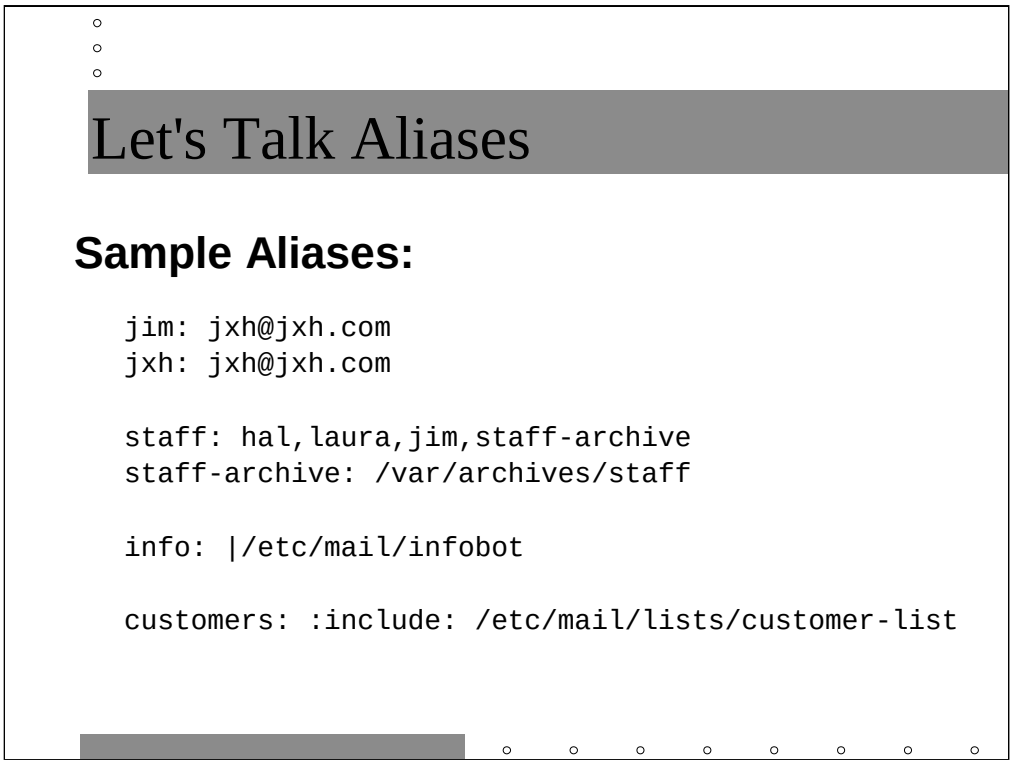
If you are going to use NFS to distribute mailboxes to desktops, make sure you use the mount options shown above. The `hard` option causes programs opening files in the mail directory to get I/O errors if the remote server isn't responding. If you don't mount `/var/mail` with the `hard` option, your mail programs can get confused and corruption will result. `actimeo=0` stops NFS on your machine from caching file attributes (size, last modification, etc.) at all. This hurts performance, but again prevents confusion and data loss.

As an alternative to NFS, you should consider using a program like `fetchmail`. `fetchmail` acts like an IMAP (or POP) client to the central server and safely moves the user's mail down to their desktop machine so their local client can operate on it. Since the IMAP server that `fetchmail` interacts with and the Sendmail server which delivers mail to the user are running on the same machine, you don't have the NFS locking problem.



Aliases are a mechanism for creating an email address that points to a different user or group of users, or possibly feeds mail to a program or into a text file. In general, aliases are only appropriate for machines where mail delivery happens because aliases are expanded only when the mail is just about to be delivered.

To create an alias, you edit the text file `/etc/mail/aliases` (`/etc/aliases` on some systems) and add aliases at will. Once you've updated the file, you need to run the `newaliases` program which creates a hashed NDBM-style database of your aliases so that Sendmail can look them up faster. Note that on most systems, `newaliases` is actually just a symbolic link to the Sendmail program. When Sendmail is invoked as `newaliases`, it rebuilds the alias NDBM files.



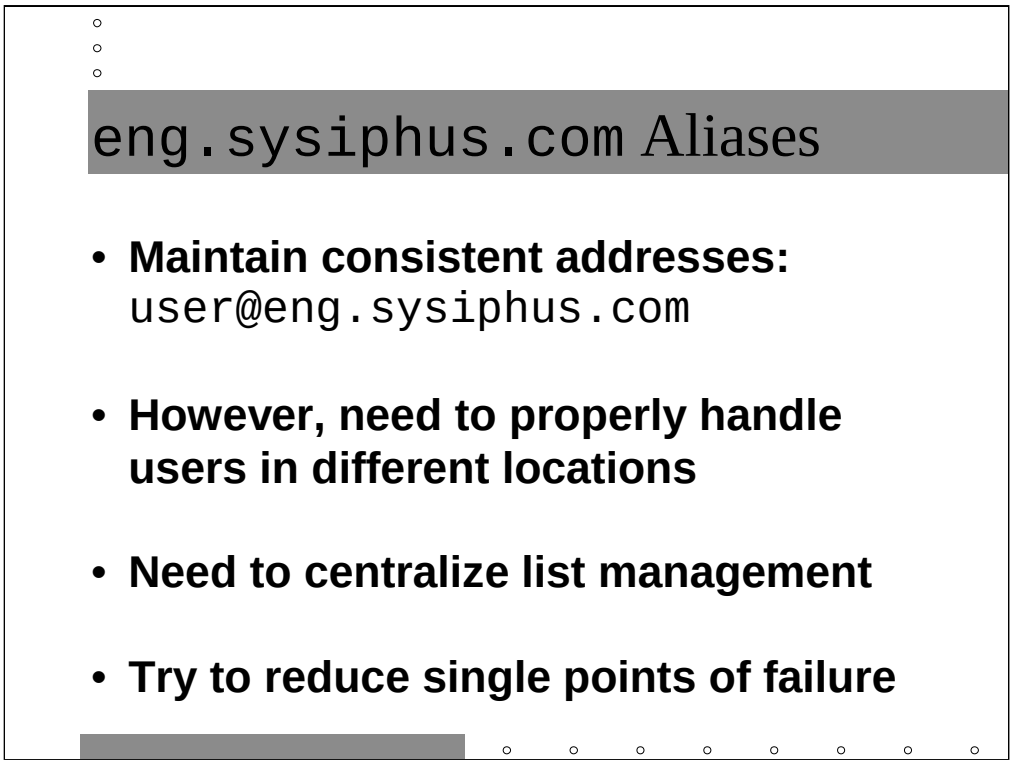
In the first example, we're saying that the addresses `jim@eng.sysiphus.com` and `jxh@eng.sysiphus.com` both map to an external email alias `jxh@jxh.com`. Any mail going to either of the two `eng.sysiphus.com` addresses will not be delivered on any mail server locally, but instead be fired away to the `jxh.com` domain.

Next we see an alias for a group of people who get mail for `staff@eng.sysiphus.com`. Note that last address in the list which is actually an alias itself. The `staff-archive` alias dumps all mail to the `staff` mailing list into an archive file so that requests to staff don't get lost. Note that this archive file will grow without bound, so you need an external process which moves the old file out of the way periodically and compresses it.

Also note that we use the `staff-archive` alias in another alias before it gets defined elsewhere in the file. Remember that the flat aliases file gets converted into a NDBM file and re-ordered anyway. Sendmail does the right thing handling multiple levels of aliasing.

The `info` alias causes all mail sent to `info@eng.sysiphus.com` to get piped to a program (perhaps some sort of autoresponder script). We'll talk more about mail programs later.

The last alias demonstrates how to keep a list of users for an alias in an external file. This may be a good mechanism to allow unprivileged users to maintain their own mailing lists, but wait for our section on Majordomo before coming to any conclusions.



Now, we've got a problem with aliases and mail delivery in general in the Engineering domain. Each of the sites wants to believe that it's `eng.sysiphus.com` in a mail addressing sense.

The first thing we need to do is to make sure mail gets directed properly to users at the various sites. In order to do this, we're going to maintain three different files of aliases. For each site, there's going to be a file which lists all users at that site and has aliases like:

```
user:     user@site.eng.sysiphus.com
```

We'll see why this is useful in a moment.

We also want each site to be able to expand certain aliases locally but the same on all mail hubs in Engineering. For example, there might be an alias for `all@eng.sysiphus.com`. We want the mailhub at each site to have the same notion of who's on this list and how to expand the list locally without routing the mail to another machine (which could, after all, be down).

Unfortunately, certain types of aliases need to be managed on one machine only. A prime candidate for this sort of handling is Majordomo-related aliases (wait and see why). We're going to force mail for all mailing lists managed by Majordomo to go to the `SFO mailhub` for processing.

Sound complicated? Well, maybe it is, but the solution is straightforward…

sfo-eng.mc File

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')
define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')
FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)

# Make sure this is one whole line before m4 gets it!
define(`ALIAS_FILE', `/etc/mail/aliases,/etc/mail/aliases-bos, \
    /etc/mail/aliases-dfw,/etc/mail/aliases-majordomo')
FEATURE(use_cw_file)

MASQUERADE_AS(eng.sysiphus.com)
MAILER(smtp)
```

This looks pretty much like the `m4` file for the mailhub.corp machine except for the `MASQUERADE_AS` configuration and that nasty `ALIAS_FILE` thingy.

The trick we're exploiting for our mail routing is the fact that Sendmail can have arbitrarily many alias files. In fact, you don't even have to use `/etc/mail/aliases` at all if you don't want. You can define your own set of alias file names to your heart's content (some sites do this to put all of their aliases in some directory other than `/etc/mail`).

In our case, we're going to keep `/etc/mail/aliases` as the place where all of the shared global aliases (like `all@eng.sysiphus.com`) go. Every mailhub in Engineering is going to have a copy of the same aliases file (use `rdist` to ship them around).

Then we're going to create the three aliases files we talked about in the last slide which map users to sites:

```
user:     user@site.eng.sysiphus.com
```

Because the `SFO` config picks up the `BOS` and `DFW` aliases files, whenever the `SFO` server gets mail for `user@eng.sysiphus.com` where user lives in `BOS` or `DFW`, it will expand the alias and re-route the mail to the correct site! `SFO` doesn't suck in the `SFO` aliases file because that would cause mailing loops. Still you can safely `rdist` all three site aliases files to all servers because they will be configured to only pick up the non-local site aliases.

`aliases-majordomo` contains aliases for all of the Majordomo-related aliases maintained at `SFO`.

bos-eng.mc File

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')
define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')
FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)

# Make sure this is one whole line before m4 gets it!
define(`ALIAS_FILE', `/etc/mail/aliases,/etc/mail/aliases-dfw, \
    /etc/mail/aliases-sfo,/etc/mail/aliases-listpointers')
FEATURE(use_cw_file)

MASQUERADE_AS(eng.sysiphus.com)
MAILER(smtp)
```

The BOS Sendmail config is similar to the SFO config, and nearly identical to the DFW config which I'm not going to show you.

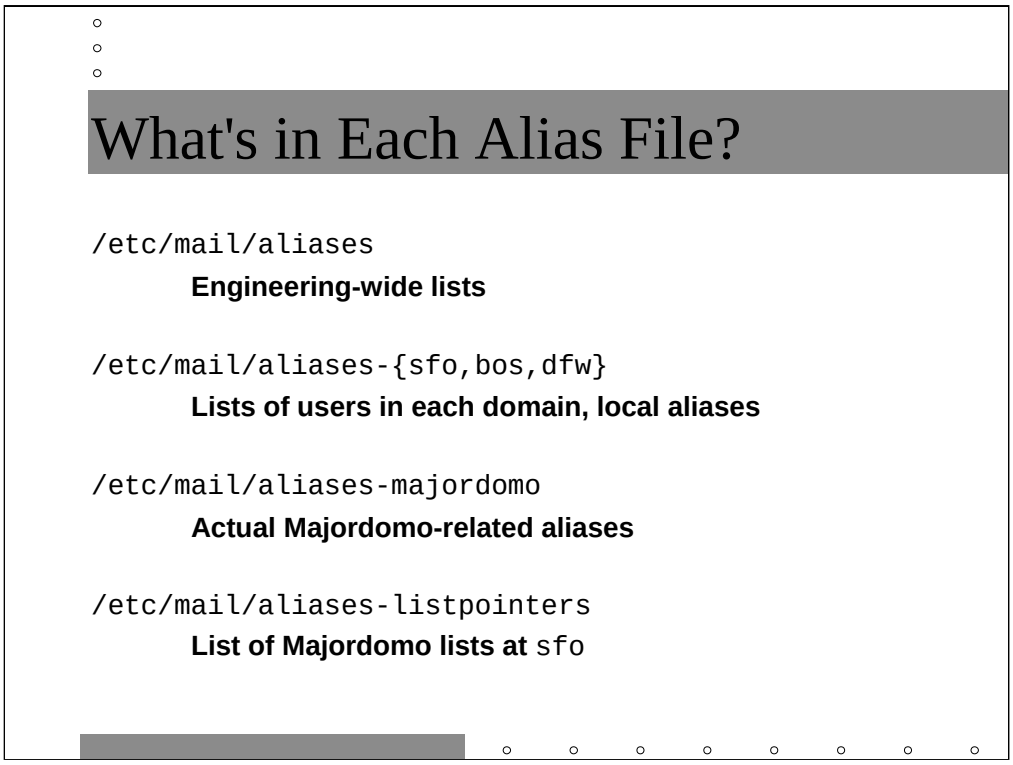
The BOS Sendmail uses the global aliases file like the other mailhubs, and picks up the site aliases for the other sites (`aliases-sfo` and `aliases-dfw`). BOS, however, doesn't maintain the Majordomo aliases locally, so it picks up a different file called `aliases-listpointers`. For every mailing list `list` that SFO manages with Majordomo, the `aliases-listpointers` has an entry

```
list:     list@sfo.eng.sysiphus.com
```

Again this aliases file is shipped around to all mail hubs in Engineering (even SFO, though the SFO Sendmail config never sees it).

A word about `local-host-names` files is appropriate here. Each `mailhub` needs to list both `eng.sysiphus.com` and `eng.site.sysiphus.com` as being local. This is so the redirection to `user@site.eng` or `list@site.eng` will work and the mail will get delivered properly.

You may also wish to add other `machine.site.eng.sysiphus.com` entries in `local-host-names` if you've got machines that aren't properly masquerading their email addresses (perhaps machines outside of your administrative control).



Let's recap those alias files again. These files should all be maintained on one server (though you could manage different files on different machines if you wanted to make yourself crazy) and `rdist`ed around to the other mailhubs. `SFO` might be the best place to manage all of the alias files since the admins there are going to spend extra time managing the Majordomo-related aliases anyway.

`/etc/mail/aliases` contains global Engineering-wide aliases which all of the machines should be able to expand locally. These would be aliases like `all@eng.sysiphus.com` which contained a list of all usernames in the Engineering domain.

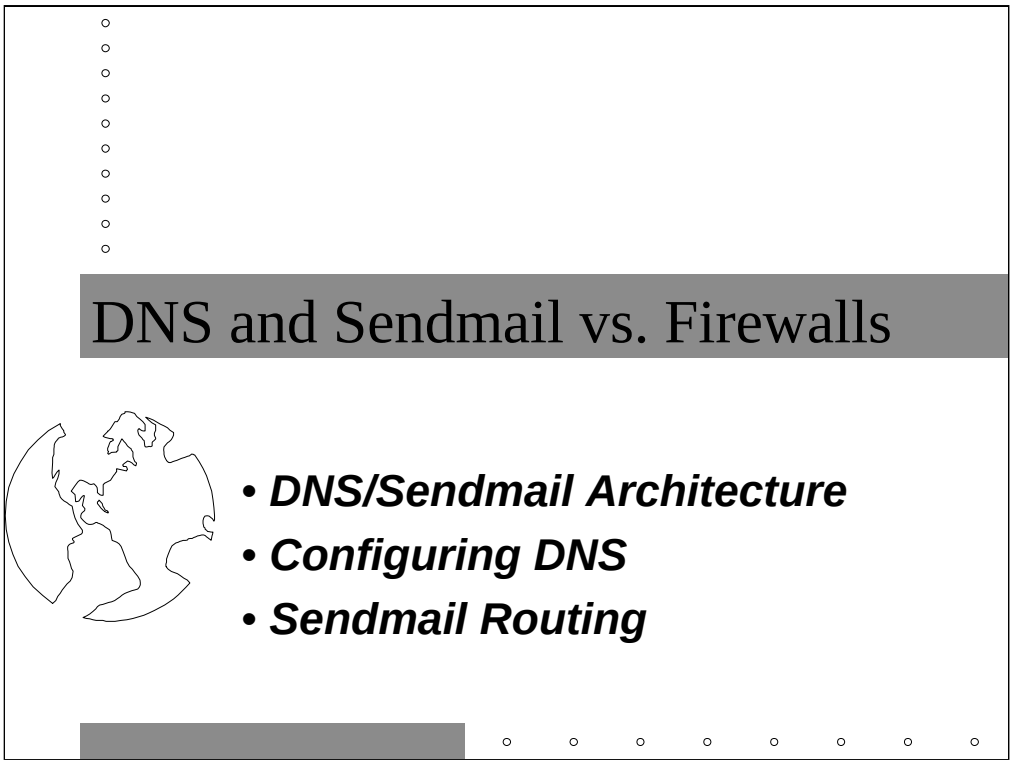
`/etc/mail/aliases-site` contain `username` to `username@site` mappings like

```
user:     user@sfo.eng.sysiphus.com
```

Note that the `all@eng.sysiphus.com` might expand to a group of user names, some of which get expanded to `user@site type aliases`.

`/etc/mail/aliases-majordomo` contains all of the aliases related to lists managed by Majordomo. This file is only read by the SFO Sendmail. The `BOS` and `DFW` Sendmail's read `aliases-listpointers`. For each mailing list `list` in on the `SFO` server, the `listpointers` file contains

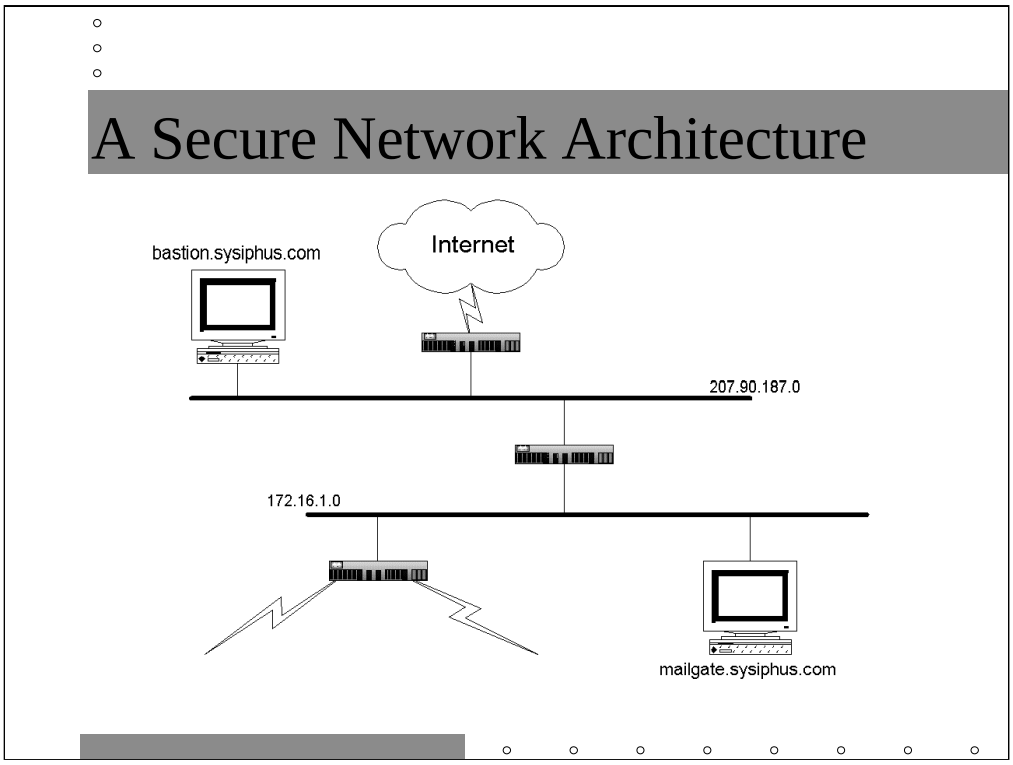
```
list:     list@sfo.eng.sysiphus.com
```



So far we've been doing the DNS and mail setup for `sysiphus.com` and not worrying very much about security. Unfortunately, there's an awful lot to worry about. There are lots of network attacks that malicious outsiders can mount against your mail and DNS servers, and more are being discovered every day.

In this section, we're going to talk about a *split-horizon DNS architecture*, which involves effectively cutting your internal network off from the rest of the Internet. Inside your organization can have a rich, open network environment, but the outside world sees only the barest amount of DNS information required for your network to function.

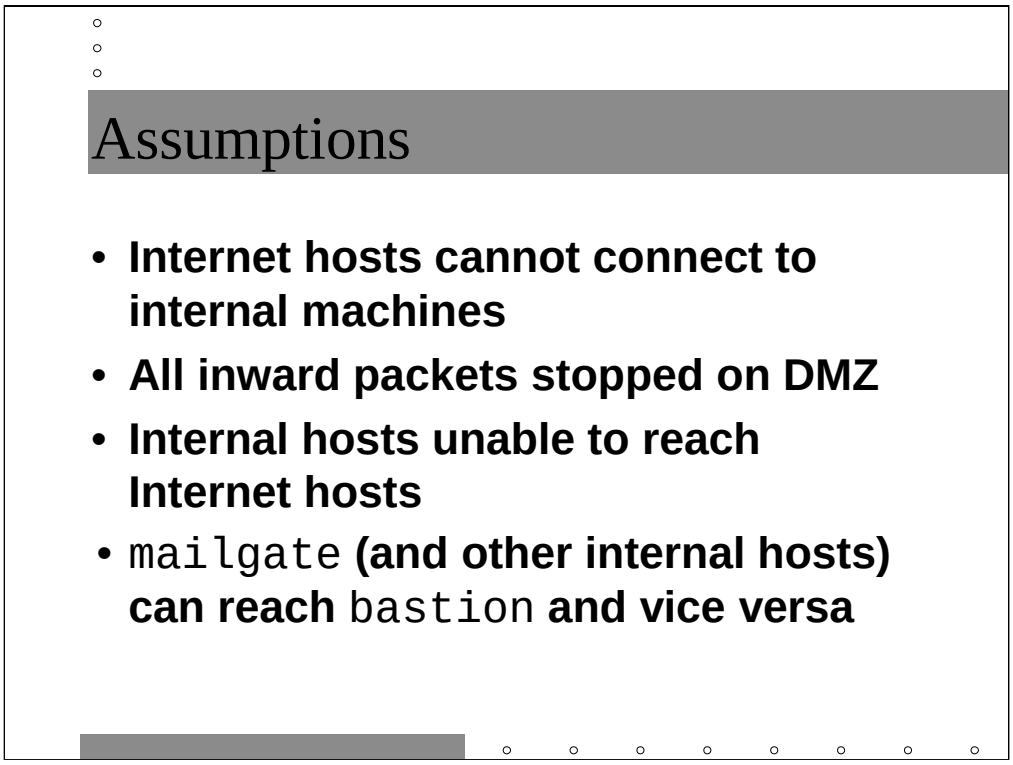
Once we get split-horizon DNS set up, we're going to set up an external mail relay for pushing mail into and out of our hypothetical company.



This is the same network diagram we've been looking at for Sysiphus Laborers, Inc., except that we've added a new network between the Internet and the rest of the company. Here we're assuming that the routers between the Internet and our new network, and between that network and the rest of the company are acting as firewalls.

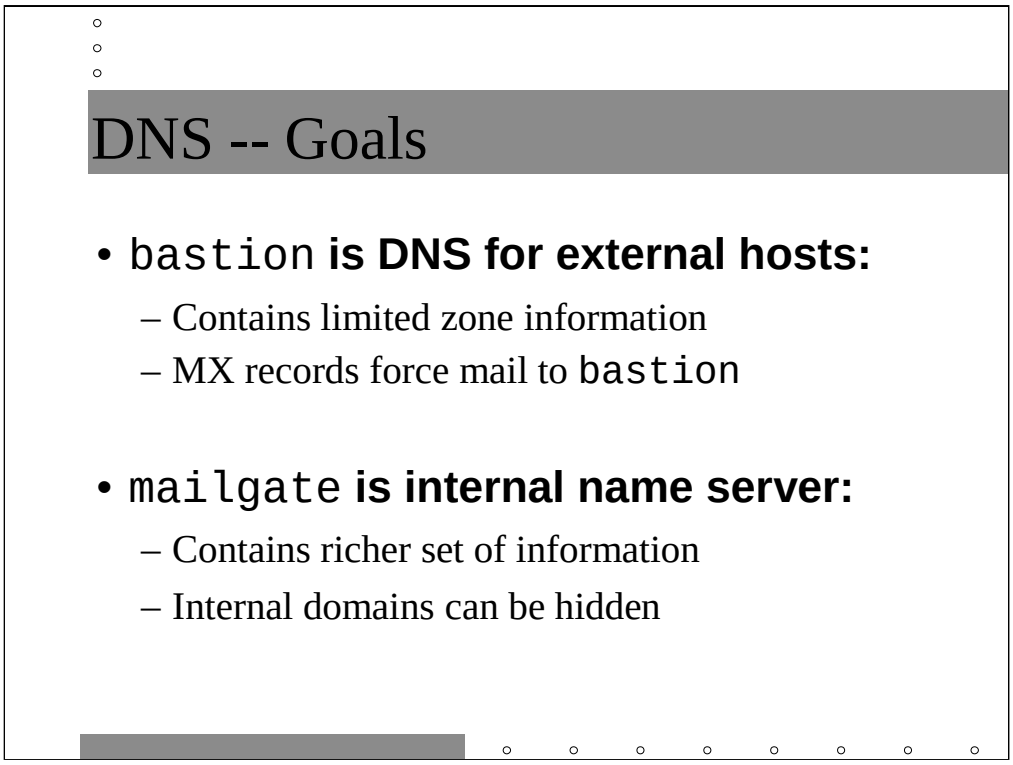
The function of the `bastion` host is to provide outside organizations with the minimal amount of DNS information they need to interact with `sysiphus.com`. By hiding information about our internal network, we make it harder for outsiders to attack us. `bastion` is also going to be the mail relay into and out of the company.

The internal DNS configuration is going to remain substantially the same, with some minor tweaks. Mail within the company will still move around as before, but mail heading out from the company needs to get relayed through the `bastion` host.



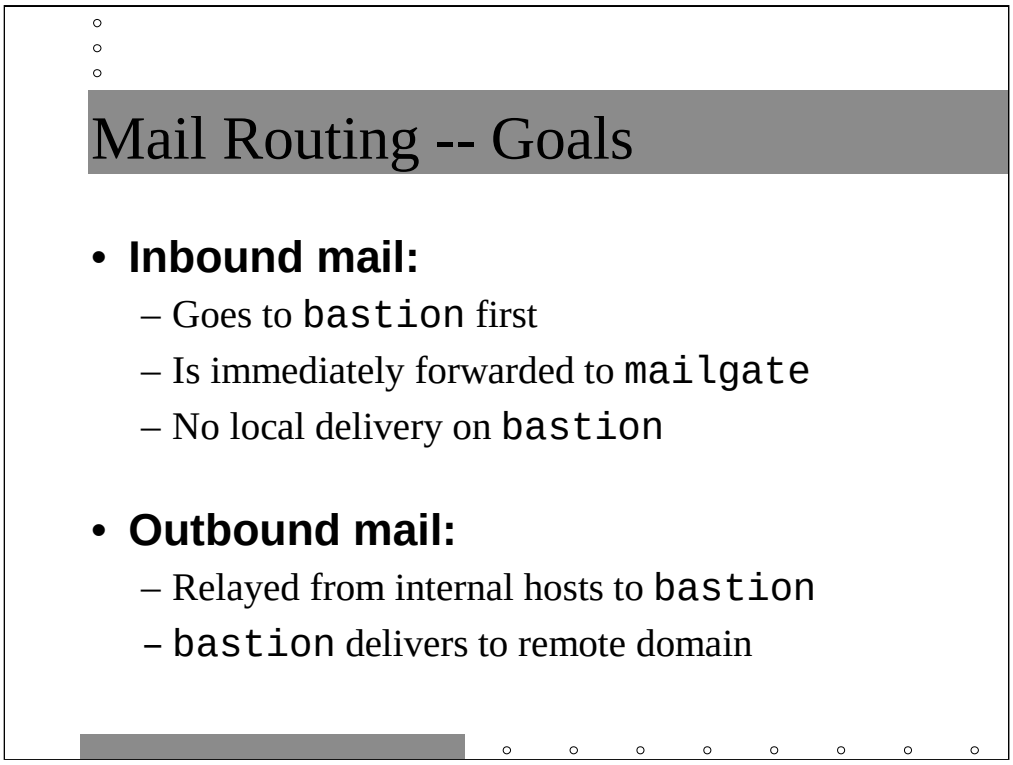
Throughout our examples in this tutorial, we've been using network `172.16.0.0` for host addresses. This network is part of three network ranges defined in RFC1918. RFC1918 defines networks that will never be given out to any company connected to the Internet and will never be allowed to route across the Internet. These networks are meant to be used internally by companies who cannot get enough "real" address space for all of their hosts.

In particular, this means that none of the hosts on the inside network of Sysiphus Laborers can connect directly to the Internet. On the plus side (from a security standpoint, anyway) this means that it's more difficult for Internet hosts to reach them. In particular, internal hosts can only talk to `bastion` and the `bastion` host must act as a DNS and mail proxy for all of the internal hosts. Other types of network connections like Web and FTP connections must also be proxied by some host on our new external network.



The `bastion` host needs an `SOA` record and appropriate `NS` records for your external version of `sysiphus.com`. It needs `A` and `PTR` records for the "public" servers on our new external network. This would be the `bastion` host and probably some other hosts to be your Web and FTP servers. Finally, we need to set up `MX` records to that all mail from the outside world that's sent to users under `sysiphus.com` gets sent to the `bastion` host first.

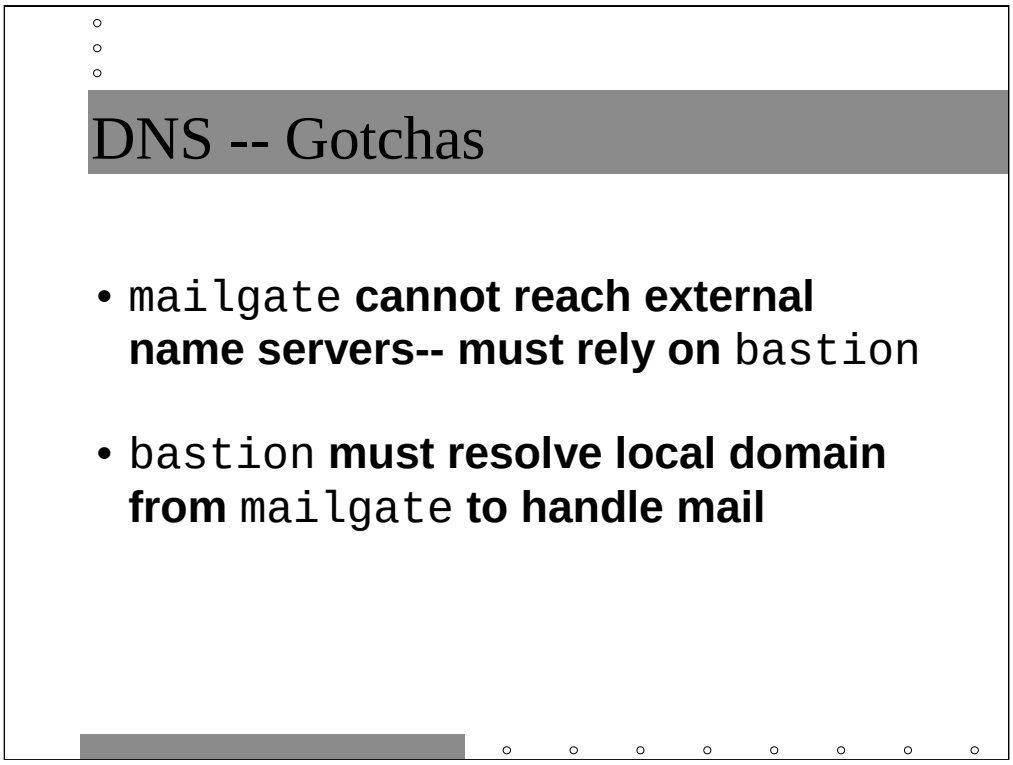
The internal DNS servers will keep the same information we've already configured into them. As we're going to see in a moment, the external DNS database on the `bastion` host doesn't even need configuration information for all of the subdomains we've set up internally. We can hide them completely from the outside world.



The external mail gateway will be used as a mail proxy for both inbound and outbound mail. Mail coming in from outside our organization has to stop on the `bastion` host and then be forwarded into our internal network. No local delivery will ever take place on the `bastion` host itself, thwarting many popular Sendmail-based attacks.

Since we are assuming that no internal hosts can directly reach hosts on the Internet, all outbound mail has to find its way to our `bastion` host, which in turn will deliver the mail to its final destination.

Given Sendmail's history of security problems, you may feel the risk of running Sendmail on your external mail server is too great. Previous versions of this course have recommended using SMAP from TIS as a replacement for the Sendmail daemon, but the SMAP code is somewhat buggy and no longer actively supported by TIS. You may wish to investigate QMail or Postfix as more secure options for Sendmail. On the other hand, the Sendmail sources have probably received much more scrutiny than the QMail or Postfix sources, and perhaps may actually be more secure than the newer alternatives.



This is a very restrictive environment to work in (there's always a security vs. ease-of-use tradeoff). 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` won't have enough DNS information locally to properly deliver inbound mail. It needs to be able to query the name server on `mailgate` 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 resolve DNS information from a completely different machine.

Bastion -- named.conf

```
options {
        directory "/etc/namedb";
        version "like nothing you have ever seen";
        allow-transfer { 207.90.181.1; 207.90.181.2; };
        allow-recursion { 172.16/16; 207.90.187/24; };
};

zone "sysiphus.com" {
        type master;
        file "ext-sysiphus.hosts";
};

zone "187.90.207.in-addr.arpa" {
        type master;
        file "ext-sysiphus.rev";
};

zone "." { type hint; file "named.ca"; };
```

This is the `named.conf` for the bastion host. Other than some new options (see below) it's not too different from our original `named.conf` for `mailgate`.

It is possible for outsiders to query your running name server and find out what version of BIND you are running using the following command:

`dig @<`*remote nameserver*`> version.bind txt chaos`

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.

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 since it is possible to do zone transfers from slave name servers, you should make sure that `allow-transfer` is configured correctly on all of your external secondary name servers as well (which may involve explaining `allow-transfer` to whoever is hosting your secondary name servers) . You should also configure your firewall to block zone transfers from the outside world as an extra layer of security (more on this in the last section of this course).

(continues on next slide…)

Bastion -- `named.conf` (cont.)

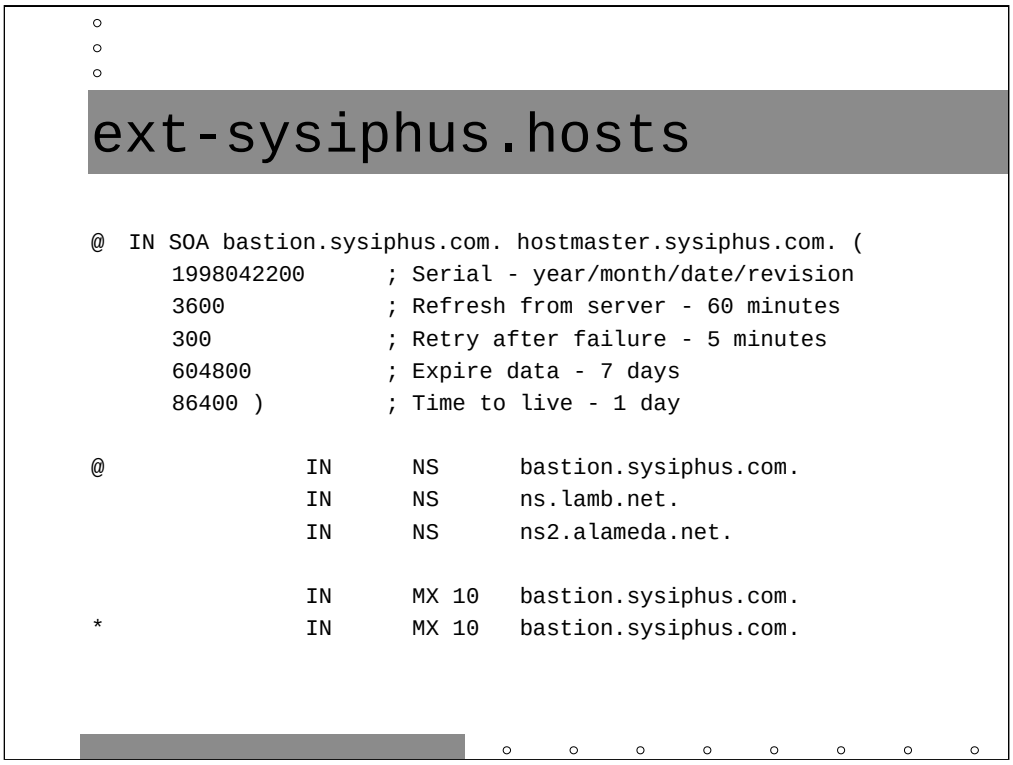
```
options {
        directory "/etc/namedb";
        version "like nothing you have ever seen";
        allow-transfer { 207.90.181.1; 207.90.181.2; };
        allow-recursion { 172.16/16; 207.90.187/24; };
};

zone "sysiphus.com" {
        type master;
        file "ext-sysiphus.hosts";
};

zone "187.90.207.in-addr.arpa" {
        type master;
        file "ext-sysiphus.rev";
};

zone "." { type hint; file "named.ca"; };
```

(continued from previous slide)

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.

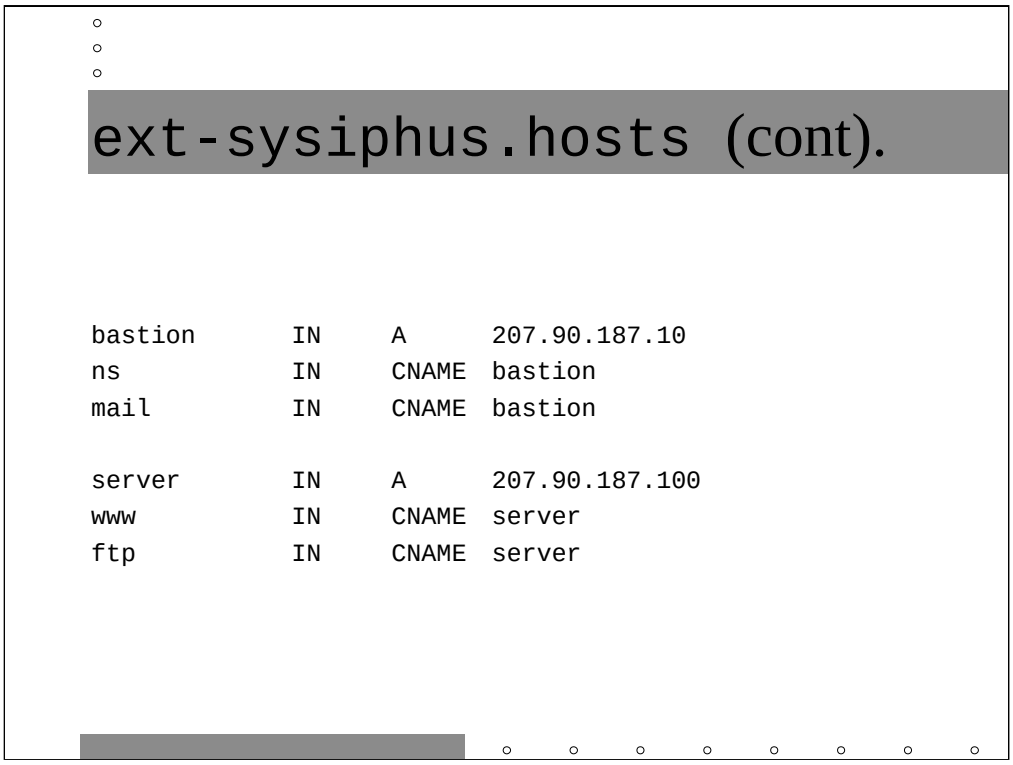
Note that all of these new options are most appropriate for your external name servers. You can apply them to your internal name servers as well, but you may find this more of an administrative hassle than anything. Aside from having to maintain the lists of IP addresses in the `allow-transfer` and `allow-recursion` 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.



First, there's a standard `SOA` record.

The `NS` records on bastion list our external secondaries. You will have to remove these external secondaries from the internal zone files on `mailgate`.

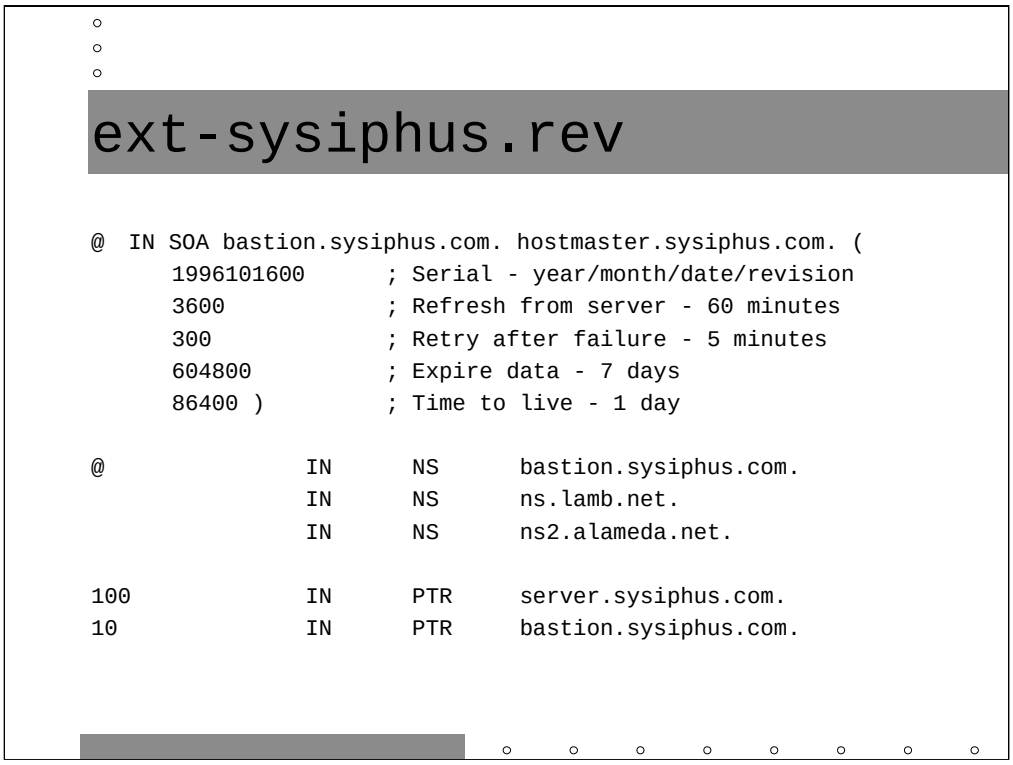
Next we have an `MX` record for `sysiphus.com`-- all mail for `user@sysiphus.com` is sent to `bastion`. Next we have a new sort of `MX` record, the *wildcard `MX` record*. The wildcard record says send all mail for subdomains of `sysiphus.com` (including multiple levels of subdomaining, such as `user@sfo.eng.sysiphus.com`) to `bastion` as well. Wildcard `MX` records are useful in split-horizon DNS configurations and dangerous otherwise.



Here are some standard forward address records that might appear in your external DNS information. We have two `A` records-- one for the `bastion` and another for an external server that's going to be the machine to serve Web pages and FTP sessions to the outside world.

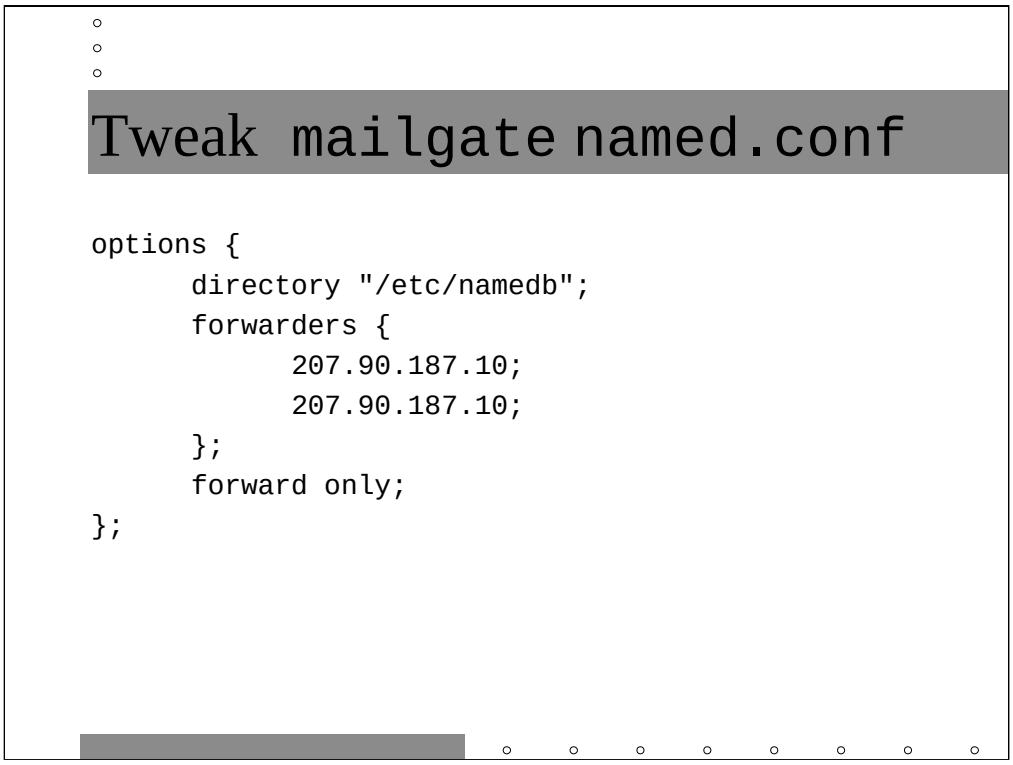
Notice this new type of DNS record. `CNAME` (Canonical NAME, again a really bad choice of name) are just aliases for hostnames. Here we're saying that `www.sysiphus.com` and `ftp.sysiphus.com` should resolve to `server.sysiphus.com` at address `207.90.187.100`.

The aliases `ns` and `mail.sysiphus.com` don't have any real meaning as far as DNS or Sendmail, they're just helpful signposts for human administrators.



Here's a pretty standard `in-addr.arpa` zone file. We've seen lots of these already.

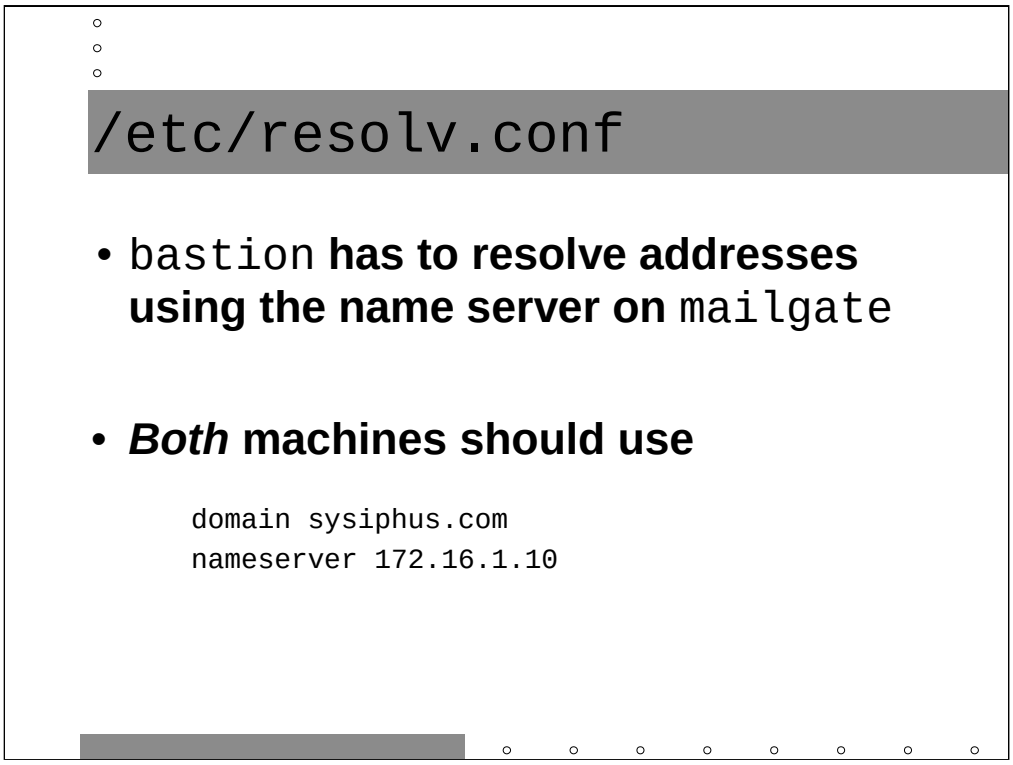
Note that we are returning the name of the host as the name defined in the `A` record for forward lookups, as opposed to any `CNAME`. Always do this, don't make yourself crazy.



Remember that we are assuming our internal name servers are 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 (this means that `mailgate` will be doing *recursive queries* via `bastion`-- c.f. the `allow-recursion` option we talked about earlier). In the event the query fails, the local server would normally attempt to contact a remote nameserver, but the `forward only` directive prevents this behavior. 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.

Note that you will have to add similar forwarding statements to all of the other internal DNS servers. You could use either the IP address for `bastion` or `mailgate`'s IP address in the `forwarders` directive.

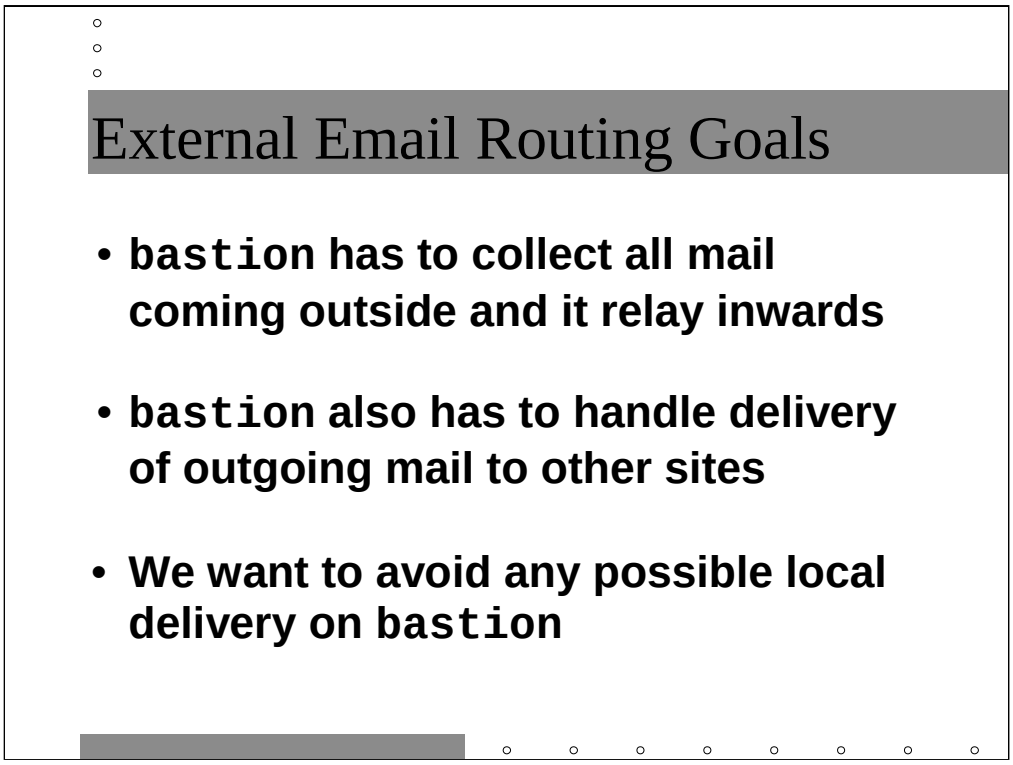
Obviously, there is some performance penalty for doing all this forwarding. Remember, though, DNS servers cache successful lookups, so you really only pay the performance penalty the first time you look up an external address.



bastion needs to get its DNS information from mailgate so that it can look up internal MX records to route email properly for inside users. If it were to obey its own DNS information, you would have a mail loop since bastion has wildcard MX records pointing back to itself.

One of Sendmail's security features is that it will only deliver email based on information from *authoritative name servers* for a given domain. Since bastion is using mailgate as a resolver for internal zone information, mailgate must be an authoritative name server for all internal domains. An *authoritative name server* is simply any machine which appears as one of the NS records for the given zone-- this is why we took pains to make mailgate be at least a secondary server for all of our internal domains.

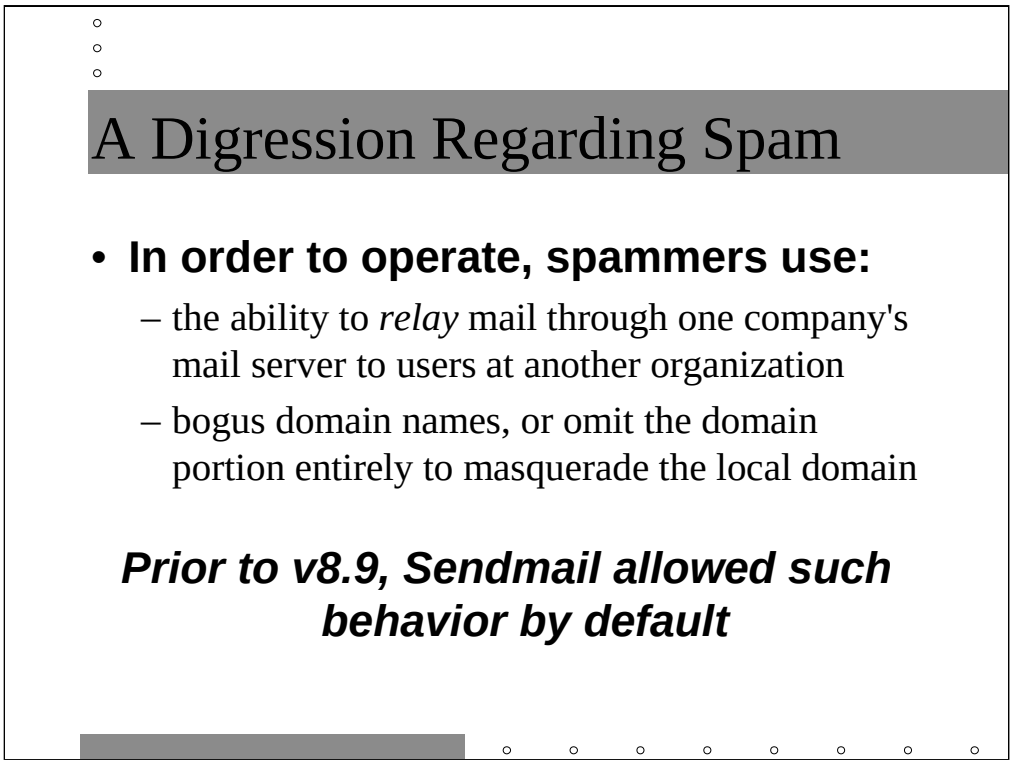
Note that there is no contradiction in having a machine running a name server that it doesn't resolve hostnames with. Remember that the resolver and the name server are completely separate. in.named is just another server process (like an FTP or Web server) that your host might be running.



Remember that due to our wildcard MX record, `bastion` will be the target for all mail coming into our company from the outside world. However, user mailboxes reside on the various internal company mail servers, so `bastion` should simply relay this incoming email to those internal servers for final delivery. Not doing delivery to local mailboxes on the `bastion` host has the added benefit of avoiding address-based buffer overflows, attacks involving insecure aliases, and other security problems with the local delivery mailer.

Aside from being the inbound channel for email, `bastion` is also the outbound gateway for mail originating in our company. Based on our assumption that internal hosts are unable to reach other machines on the Internet, `bastion` is the only machine in our hypothetical network configuration which is capable of forwarding email to other sites.

So, `bastion` needs rules to route email correctly into and out of our organization…



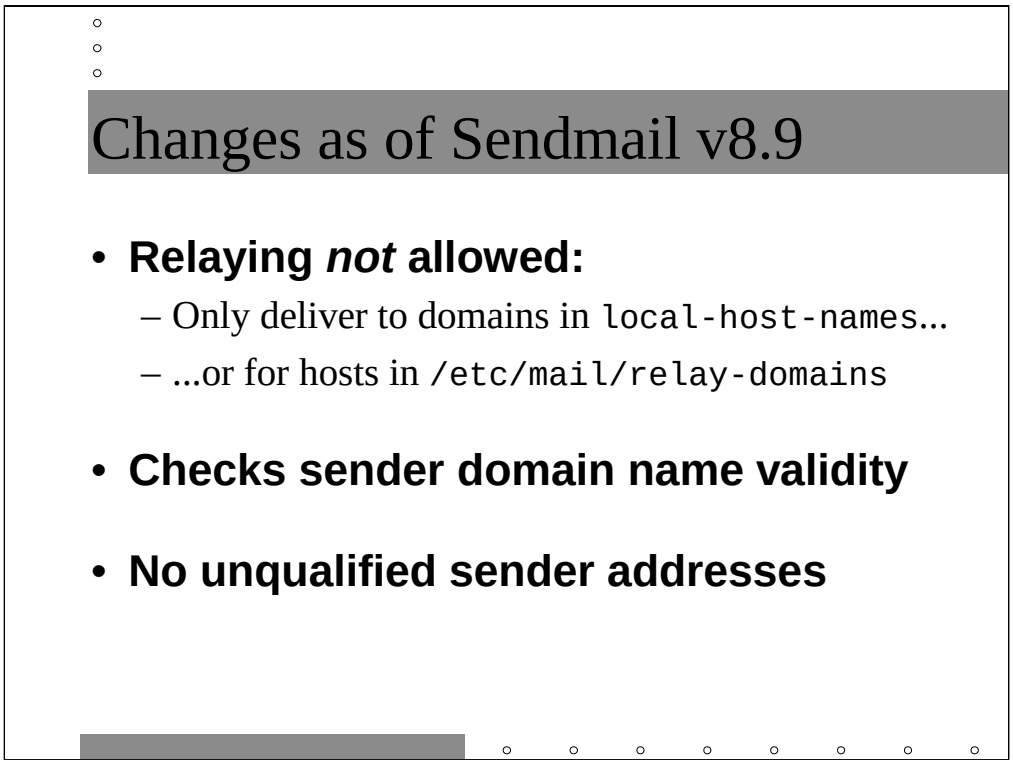
Until fairly recently, most mail servers on the Internet were willing to route mail from one external organization to users in another external organization. In the early days of the Internet, it was considered "neighborly" to help out other sites by delivering email for them when it was accidentally routed to you.

Unfortunately, this behavior is now being exploited by spammers. By using your servers as a relay, it appears to naïve users and administrators that your server was the originating point for the spam. Your mail servers may be attacked by outraged recipients and possibly taken off the air by their efforts.

Another favorite trick of spammers is to create the message using unqualified email addresses (no domain portion). The first mail server which transmits the mail tends to add its own local domain to any unqualified sender or recipient addresses-- again making the mail appear as if it were generated at your site.

Spammers may also use competely made-up domain names since they generally aren't interested in receiving replies to their spam. If they did, they'd shortly be knocked off the air.

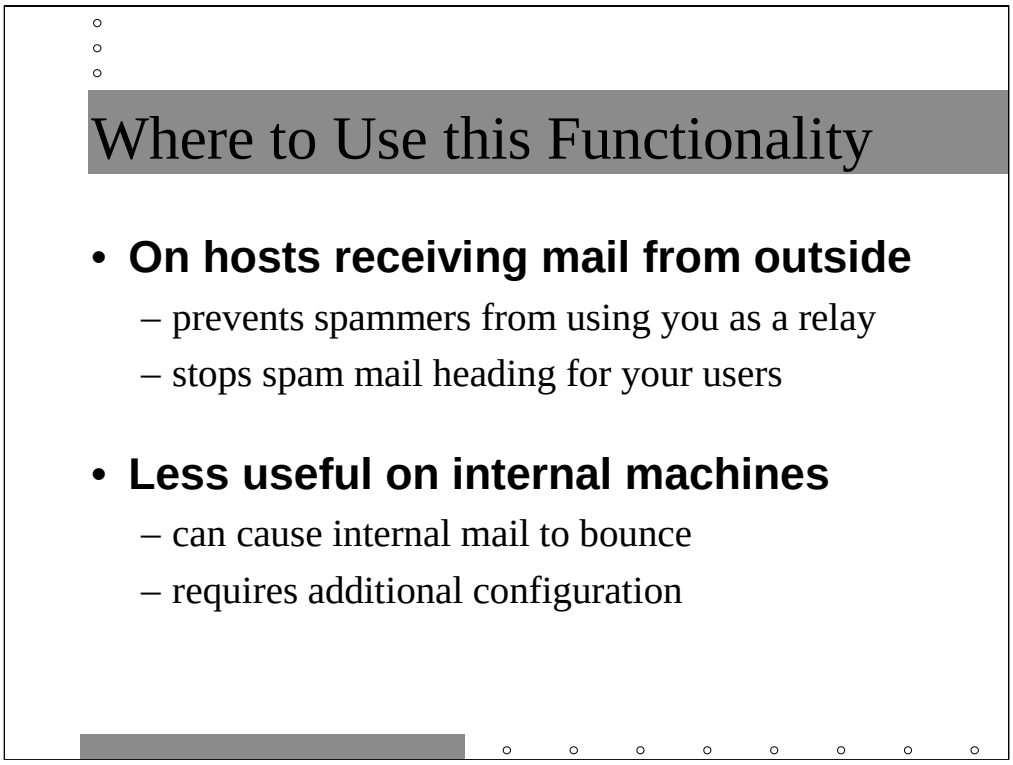
While earlier versions of Sendmail permitted this kind of behavior (unless the administrator went through some fairly complicated reconfiguration), things changed as of Sendmail v8.9…



Out of the box, Sendmail v8.9 and later do not accept mail destined for any domain not in `/etc/mail/local-host-names`. You may list machines which are allowed to send email to other domains in the (somewhat misnamed) `/etc/mail/relay-domains` file. In general you should list all of your internal IP addresses in this file. You can list partial IP addresses: for example, `172.16` in `relay-domains` would allow relaying for all hosts in the entire `172.16.0.0` network (a full Class B of addresses). As we have seen in previous slides, `FEATURE(promiscuous_relay)` will disable all anti-relaying checks.

Another new feature as of Sendmail v8.9 is automatic checking of the domain portion of the sender address. If Sendmail is unable to resolve an SOA record for the domain, mail from that domain is rejected. You can turn off this feature by enabling `FEATURE(accept_unresolvable_domains).`

Sendmail v8.9 and later also do not allow the sender address to be unqualified (contain no domain specifier). This check can be turned off with `FEATURE(accept_unqualified_senders).`



In general, you should enable these checks on any machine that accepts email from the outside world (in our example, this will be the `bastion` host). If the entire world were to take this step tomorrow, the spam problem would nearly vanish in a very short period of time.

On the internal network, however, all of these features don't have much use. You want to make it as easy as possible for internal users to send mail back and forth--even if their mail transfer agents are misconfigured in some way. Also, leaving these new features on means at least creating a `relay-domains` file on each mail server, which is just one more file to update when you renumber or add a network.

bastion -- relay.mc

```
# Sendmail configuration file for external relay hosts
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')

define(`LOCAL_SHELL_PATH', `/dev/null')
define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confPRIVACY_FLAGS',`noexpn,novrfy')
define(`confMIME_FORMAT_ERRORS',`False')
FEATURE(use_cw_file)

define(`MAIL_HUB', `mailgate.sysiphus.com')
MASQUERADE_AS(sysiphus.com)
MAILER(smtp)
```

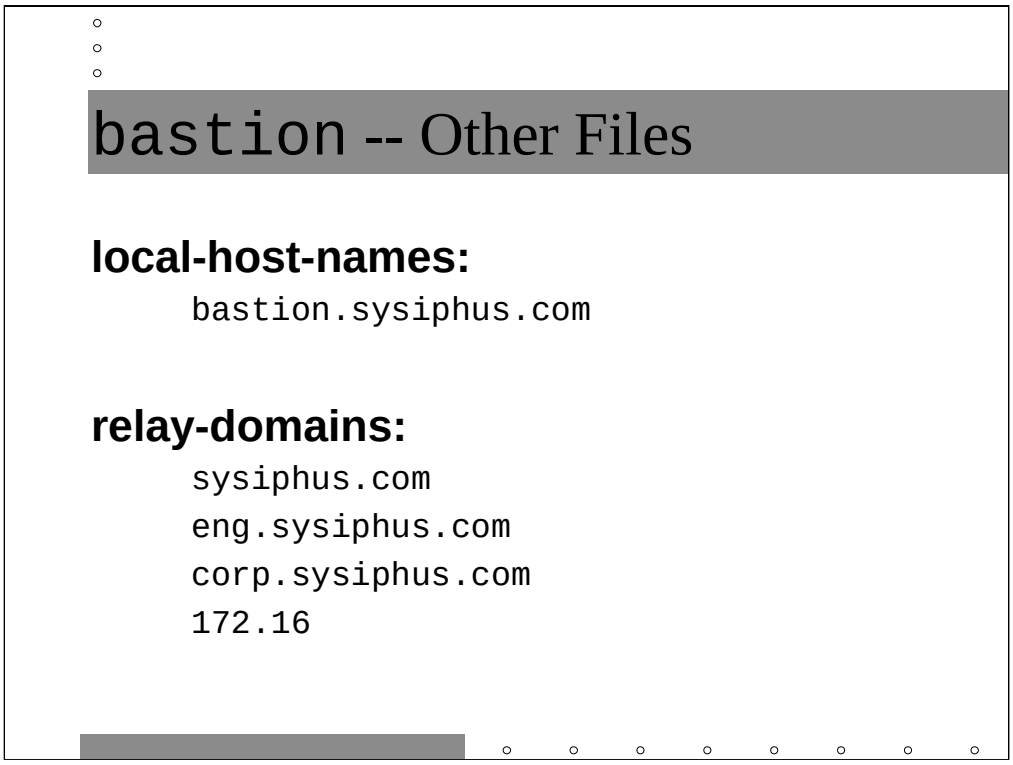
Changing `LOCAL_SHELL_PATH` to `/dev/null` prevents the local Sendmail daemon from executing any aliases which pipe mail to a program (a mechanism often exploited by system crackers). This should never be a factor anyway, since all locally delivered mail should be forwarded to `mailgate`.

The `PRIVACY_FLAGS` setting turns off the `EXPN` and `VRFY` commands and prevents outsiders from gaining information about users in your domain. The EXPN command will show all email addresses assigned to a given alias-- `bastion` shouldn't have any aliases configured on it, but turning off EXPN adds and extra layer of security. VRFY verifies a single email address, but only is useful if local delivery is going to be performed on this machine. Again, `bastion` should never be doing local delivery so VRFY is not that much of an issue.

`MAIL_HUB` causes all mail that would have been delivered locally on this machine to instead be forwarded to `mailgate` for delivery. Generally speaking, "local" email for this machine will probably only be generated by cron and other system programs-- nobody should be sending email to `bastion` from outside.

The `MASQUERADE_AS` directive only applies to mail generated on this machine.

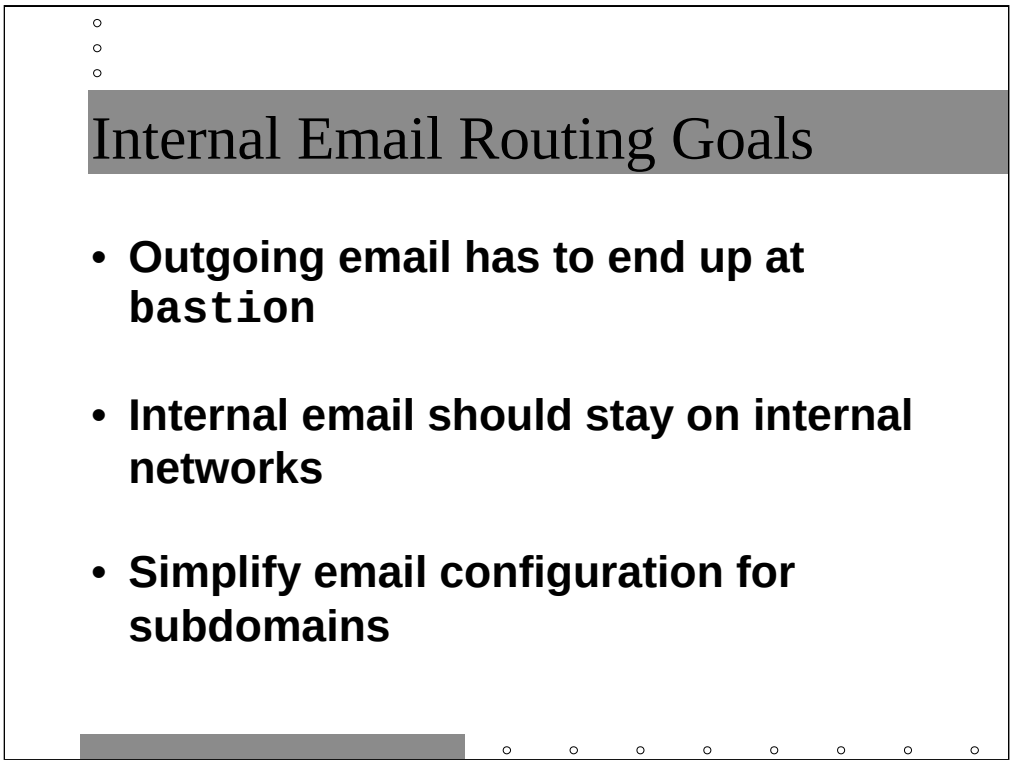
Note that this file *does not* contain the `FEATURE(promiscuous_relay)` and `FEATURE(accept_unqualified_senders)` tags to disable anti-spam checks. As we mentioned earlier external email gateways like `bastion` are the correct place to do spam filtering for your organization.



Unlike our previous examples, we did not enable `FEATURE(promiscuous_relay)` on `bastion`, so we need to make sure that the Sendmail anti-relaying checks don't interfere with normal delivery of `sysiphus.com` mail.

The only "local" address for the `bastion` host is the machine itself. Note that under normal operations nobody should be sending email to `user@bastion.sysiphus.com`. In any event we defined `MAIL_HUB` in bastion's `sendmail.cf` file (previous slide) so that local delivery won't be attempted on `bastion` even if mail shows up with this address.

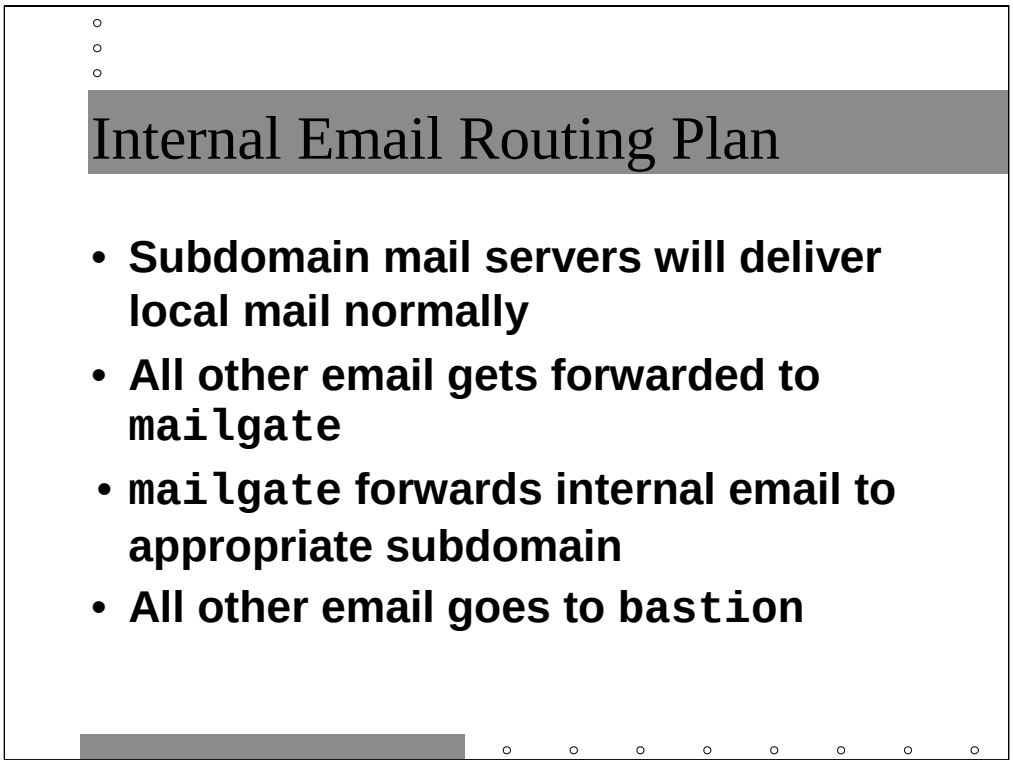
However, we do want to make sure that `bastion` passes along email for our internal domains, so we add these domains to the `relay-domains` file (you may also wish to add the `{sfo,bos,dfw}.eng.sysiphus.com` domain names just to be on the safe side). We also need to allow hosts on our internal networks to relay email out of the company, so we also add our internal address space to the `relay-domains` file.



We now have configured the `bastion` host to properly relay internal email to the appropriate internal mail server-- mostly by simply having `bastion` resolve internal zone information from `mailgate` which has the correct `MX` records for all internal zones. What do we do about internal email routing, though?

On the one hand, we'd like to make email routing as simple as possible for our subdomain administrators. Aside from making their lives easier, limiting the number of machines that have to be "smart" about internal email routing means that mishaps are less likely. However, if there are only a limited number of email routers for our company, then the loss of the mail routers can stop internal mail delivery.

We certainly would like to keep our internal email truly internal to the company. Mail from one employee to another should never touch our DMZ network or any other external network.



Our strategy is going to be to have all subdomain mail servers forward non-local mail to a single, central mail routing hub-- in our case `mailgate.sysiphus.com`. `mailgate` will be able to route mail between subdomains as well as forward all outgoing mail to `bastion`.

Obviously, `mailgate` now becomes a single point of failure. Normally we'd have two or more machines performing this routing function, listed at the same MX priority level. This allows the subdomain servers to automatically forward email to whichever machine happens to be up.

New Config for `mailhub.corp`

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')

define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')

FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)
FEATURE(use_cw_file)

define(`SMART_HOST', `mailgate.sysiphus.com')
MASQUERADE_AS(corp.sysiphus.com)
MAILER(smtp)
```

Here's a slightly modified version of the m4 macro file for `mailhub.corp.sysiphus.com`. `SMART_HOST` says to forward all email that isn't local to this machine to `mailgate` for further processing. This means that `mailhub.corp` will send all mail to `mailgate` that isn't specifically destined for `corp.sysiphus.com` (whether that's outgoing email or email for another subdomain like `eng.sysiphus.com`).

You would need to add a similar `SMART_HOST` line to the m4 macro files for the `eng.sysiphus.com` mail servers as well.

`SMART_HOST` is essentially the logical inverse of the `MAIL_HUB` directive we used on `bastion`.

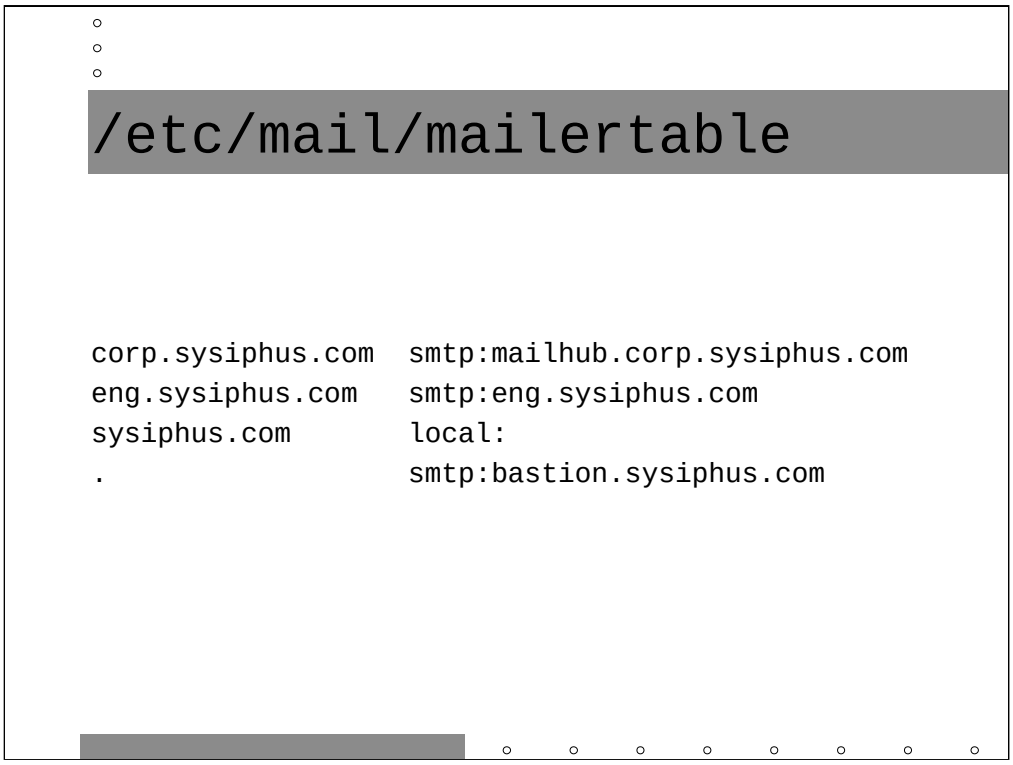
New Config for `mailgate`

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')
define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')

FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)
FEATURE(use_cw_file)

MASQUERADE_AS(sysiphus.com)
FEATURE(`mailertable', `dbm -o /etc/mail/mailertable')
MAILER(smtp)
```

Now we need a simple mechanism for configuring the more complex email routing environment on `mailgate`.

The `mailertable` feature is a mechanism to allow administrators to set up a simple email routing database to quickly add special routing instructions for different email domains without hacking the `sendmail.cf` file directly. In this case, Sendmail will do lookups in the NDBM files `/etc/mail/mailtertable.{dir,pag}` -- the `-o` flag means that these files are optional, so Sendmail will run without complaint even if there is no `mailertable` database on this machine.

A sample `mailertable` file for sysiphus.com is shown on the next slide...



First we create a text file called `/etc/mail/mailertable`. The file has two columns: the lefthand column gives a domain name and the righthand column specifies a mailer to use and a destination host.

In our case, mail for `user@corp.sysiphus.com` needs to go to `mailhub.corp` and mail for the Engineering subdomain needs to go to any one of the Engineering mail servers (recall that `eng.sysiphus.com` actually is an `MX` group which points to the three `mailhub.{sfo,bos,dfw}.eng` machines). Mail for `user@sysiphus.com` should simply be delivered locally on `mailgate`. All other email should be forwarded to the `bastion` host for further delivery.

How to Create NDBM Files

- **Sendmail sources include** `makemap`

```
% cd sendmail-8.9.x/makemap
% sh Build                          (lots of output)
% /bin/su
Password:
# chown root /etc/mail
# cp obj.SunOS.5.5.sun4/makemap /etc/mail
# cd /etc/mail
# vi mailertable                    (per previous slide)
# ./makemap dbm mailertable < mailertable
#
```

The Sendmail distribution includes the `makemap` program which will build a variety of different database file formats from a text file. There is a `Build` script for `makemap` as for the `sendmail` program. You may install the `makemap` binary anyplace on the system.

We are copying the `makemap` program into `/etc/mail`, which is normally owned by user `bin` on Solaris machines and some others. In order to copy the new binary into place and to be able to make the `*.{dir,pag}` files in this directory, we make the directory owned by `root`.

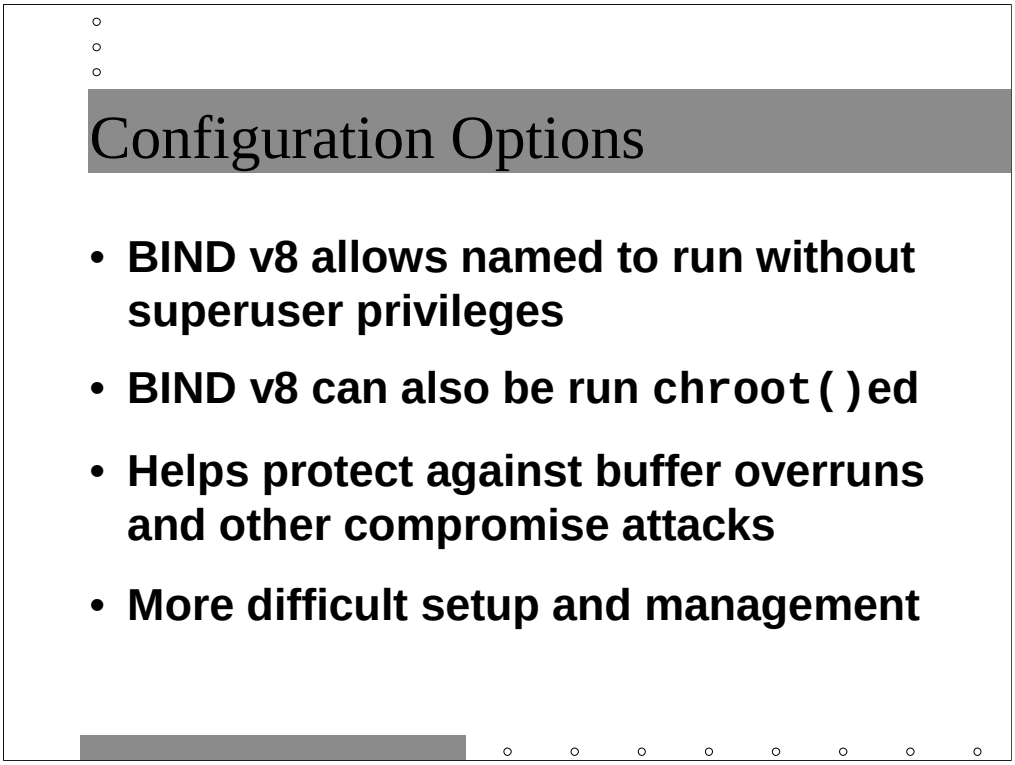
Having created the `makemap` binary, we type our `mailertable` information into `/etc/mail/mailertable` (remember, Sendmail is looking for `mailertable.{dir,pag}`).

The first argument to `makemap` is the database type and the second argument is the base filename to use when creating the `*.{dir,pag}` files. `makemap` reads in the source file on the standard input. Every time you update the source file you must re-run the `makemap` program (you might consider creating a `Makefile` which rebuilds the NDBM files whenever you type `make` in the `/etc/mail` directory)-- you may want to create a `Makefile` or shell script which does this task automatically.

Additional BIND Security

- ***Overview***
- ***Preparing the Directory***
- ***Starting the Name Server***

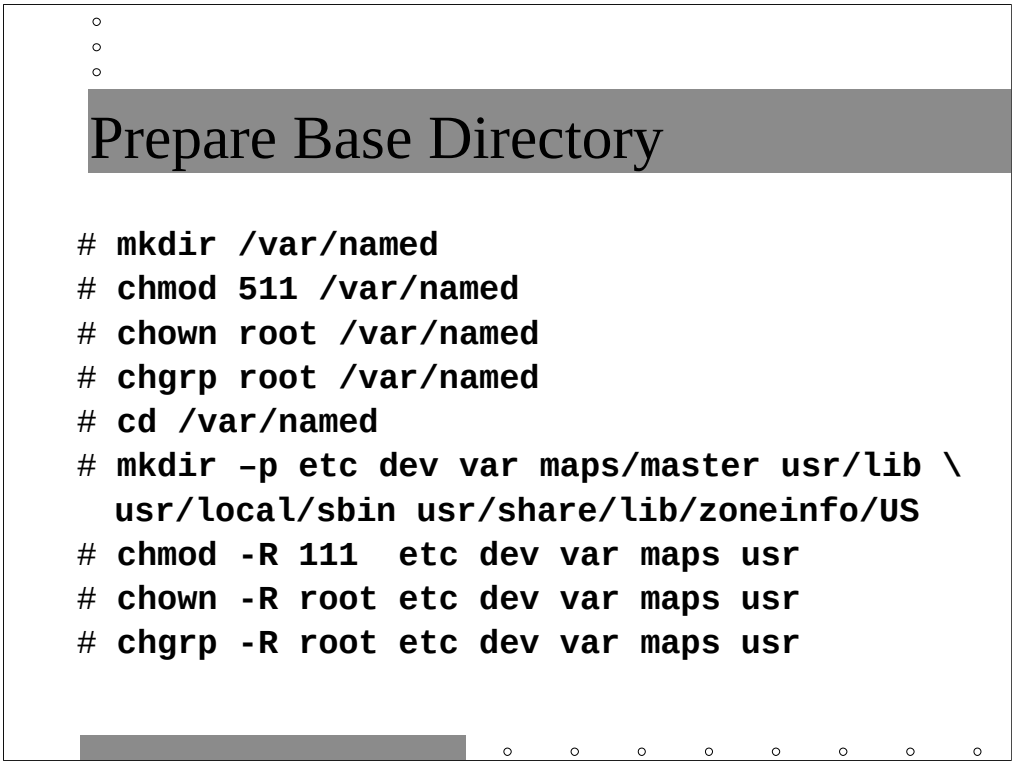
This section covers how to run BIND in a more secure fashion at the expense of additional administrative complexity. It is probably most appropriate to make this extra effort only on externally reachable name servers for your organization.



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 also allows 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 Solaris-specific configuration procedure. Information about Linux configuration is available in the *Securing Linux: Step-by-Step* guide published by the SANS Institute. Note also that this procedure assumes we have already set up an unprivileged `dns` user and group ID. You will need to be the super user to perform these configuration steps.

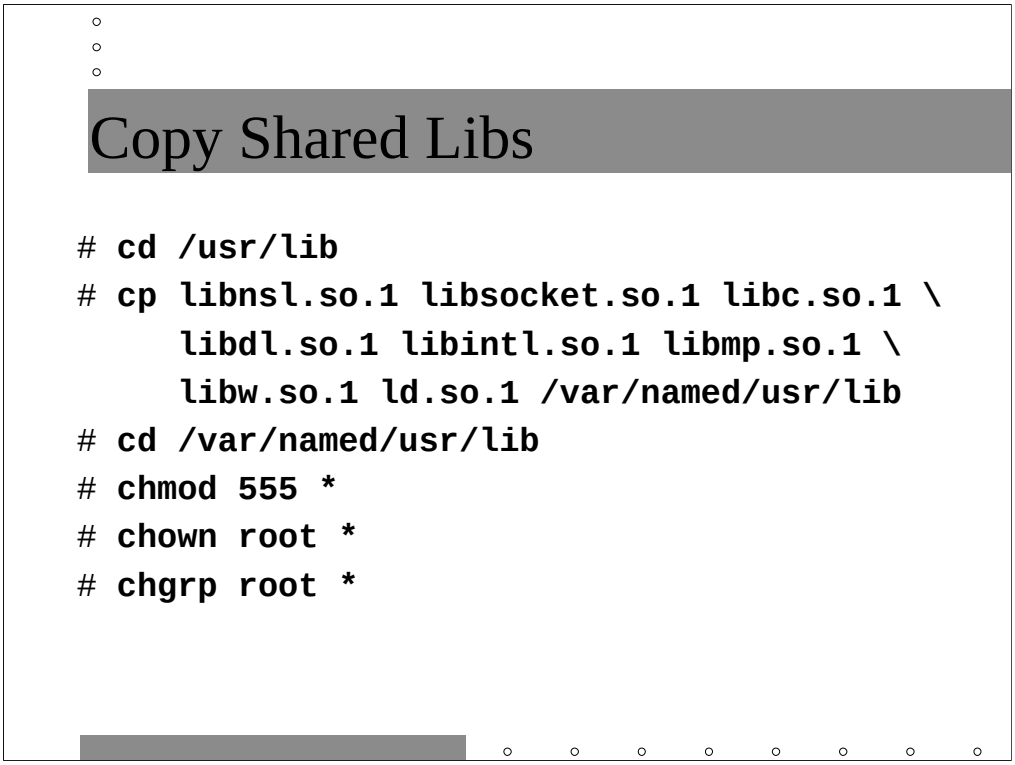
First we need to create the basic `chroot()`ed directory hierarchy we will be using for the name server. You can put this directory structure anywhere in your filesystem that you wish– in this case, we will be rooting our tree under `/var/named`.

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 `dns` user). Also, most directories should simply be mode `111` (only the execute bit set) so that the `dns` user can read files inside the directories, but not get directory listings.

Copy `named` & `named-xfer`

```
# cd /usr/local/sbin
# cp named-xfer /var/named/usr/local/sbin
# cd /var/named/usr/local/sbin
# chmod 111 named-xfer
# chown root named-xfer
# chgrp root named-xfer
```

Under normal operations, you name server needs a copy of `named-xfer`. Copy the binary from `/usr/local` (or wherever it exists on your system) into the appropriate location in the `chroot()`ed hierarchy.



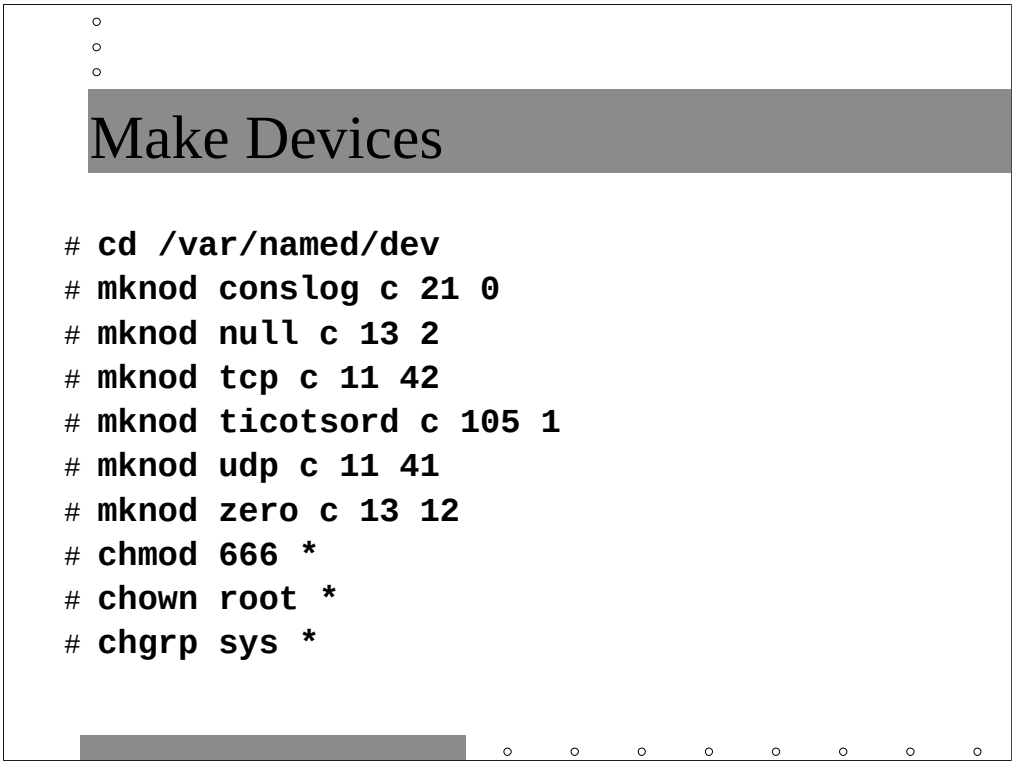
`named-xfer` is probably dependent on one or more shared libraries– you can determine exactly which libraries using the `ldd` command:

```
% cd /usr/local/sbin
% ldd named-xfer
named-xfer:
      libnsl.so.1 =>   /usr/lib/libnsl.so.1
      libsocket.so.1 => /usr/lib/libsocket.so.1
      libc.so.1 =>     /usr/lib/libc.so.1
      libdl.so.1 =>    /usr/lib/libdl.so.1
      libintl.so.1 =>  /usr/lib/libintl.so.1
      libmp.so.1 =>    /usr/lib/libmp.so.1
      libw.so.1 =>     /usr/lib/libw.so.1
```

Note that `ld.so.1` is also always required in addition to any shared libraries shown by `ldd`.

Note that since you have the BIND source code, you could attempt to compile a *statically-linked* version of `named-xfer` (one which is not dependent on any shared libraries). Note that some operating systems (notably Solaris) make this rather difficult to do, however. For more information on compiling statically-linked binaries under Solaris, see:

`http://www.deer-run.com/~hal/sol-static.txt`



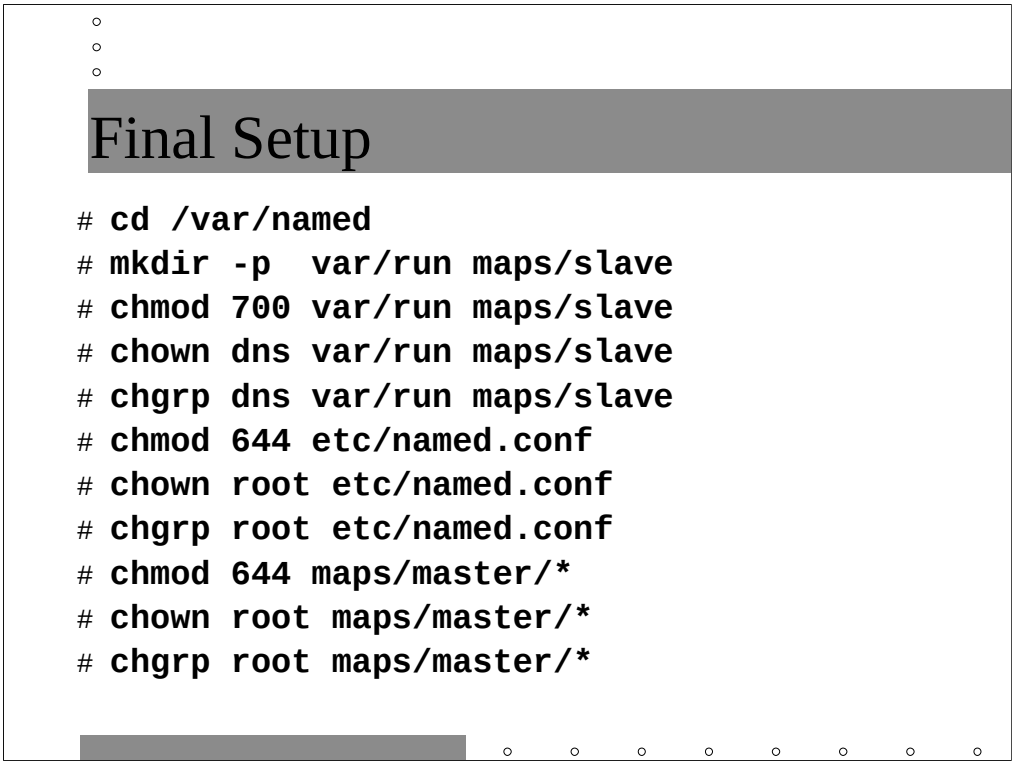
The name server also needs several device files to function properly. The arguments to `mknod` are a `c` indicating that these are character-special device files (don't worry about what this means) and the major and minor device numbers appropriate for the device. All of this information can be gathered by doing an `ls -l` on the corresponding system device and then copying the parameters in the `mknod` command.

Note that the file permissions on these devices is not a typo– all of the devices listed here are globally writeable.

Copy Misc Files

```
# cd /var/named/etc
# cp /etc/netconfig .
# chmod 444 netconfig
# chown root netconfig
# chgrp root netconfig
# cd ../usr/share/lib/zoneinfo/US
# cp /usr/share/lib/zoneinfo/US/Pacific .
# chmod 444 Pacific
# chown root Pacific
# chgrp root Pacific
```

The `netconfig` file is used by the name server to figure out which network device to choose for a given service/protocol. The file under `zoneinfo` contains information on the local time zone so that time stamps that `named` emits are expressed in local time. Obviously, you want to choose the appropriate time zone information file for your local site– it may be simpler just to copy the entire `zoneinfo` database rather than an individual file.



Assuming we have already set up a `dns` user and group ID, we want to create some directories under the `chroot()`ed hierarchy where the running name server can write files. In particular, `named` needs to write its PID and other debugging information into some directory– `/var/named/var/run` in our example– and `named-xfer` needs to be able to write the zone files it downloads into some directory– `/var/named/maps/slave` in our example.

We also need to make a copy of the `named.conf` file in 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 `chroot()`ed environment (see next slide).

We want to make sure that the primary zone files (stored in `/var/named/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 `dns` user.

Change `named.conf`

```
options {
    directory "/var/run";
    version "like nothing you have ever seen";
    allow-transfer { 207.90.181.1; 207.90.181.2; };
    allow-recursion { 172.16/16; 207.90.187/24; };
};

zone "." {
    type hint;
    file "/maps/master/named.ca";
};
[…]
```

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/named` directory– i.e., `/var/named/var/run` and `/var/named/maps/master/named.ca`.

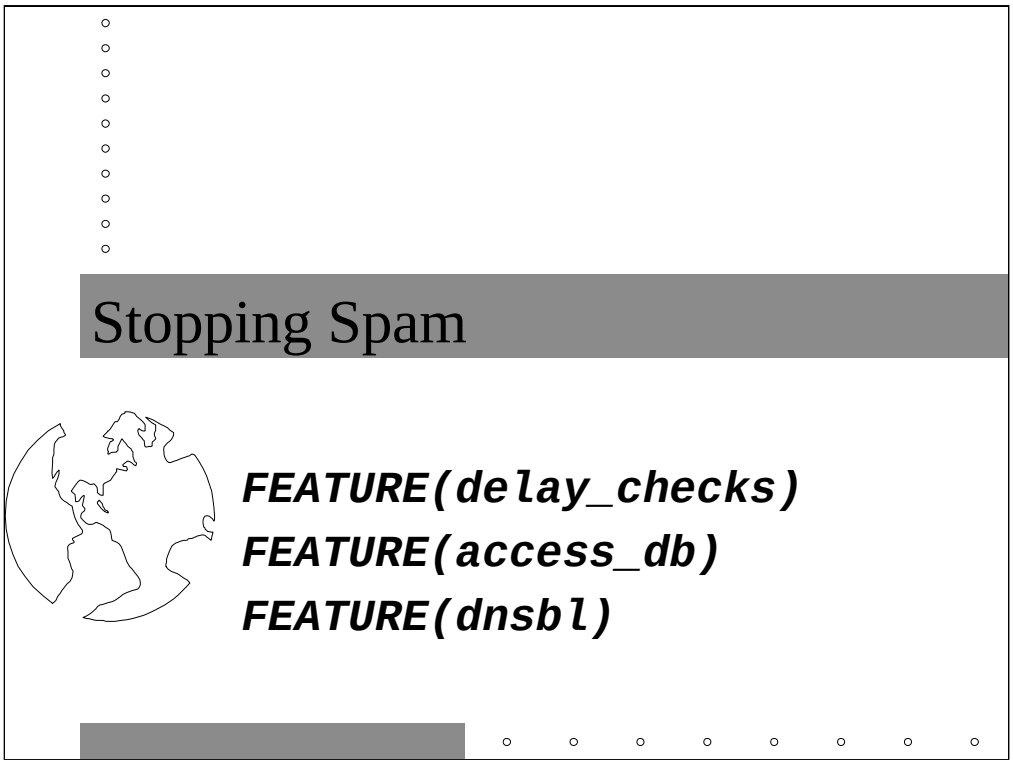
Starting the Name Server

```
# /usr/local/sbin/named \
        -u dns –g dns –t /var/named
```

Don’t forget to update your boot scripts!

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

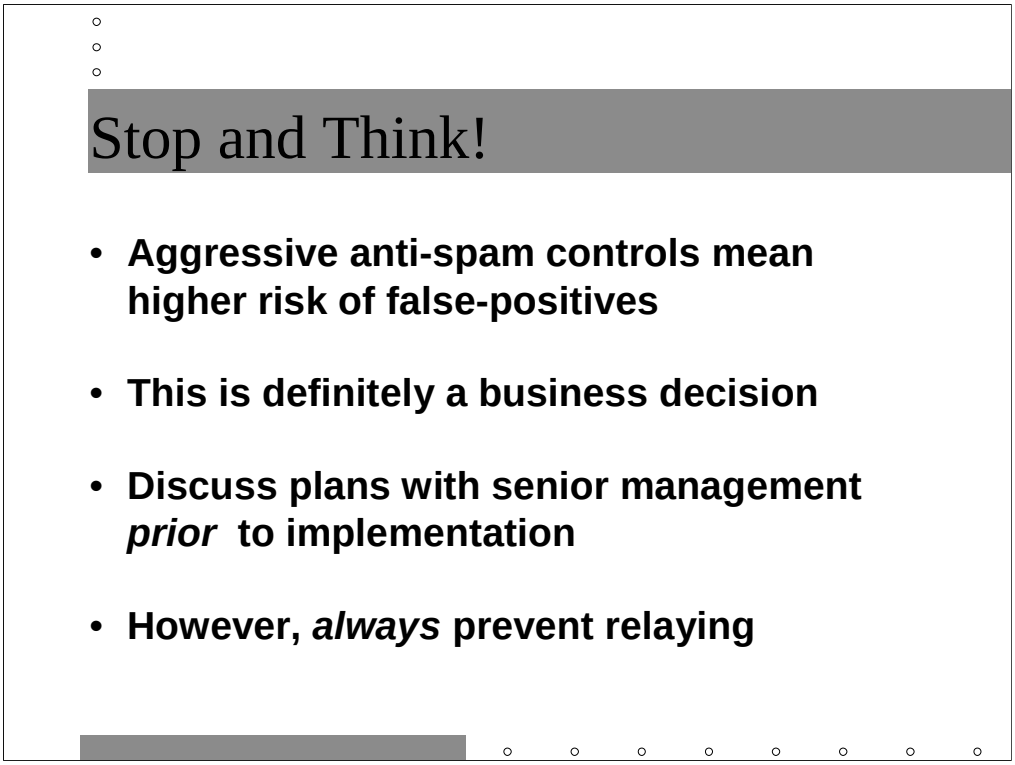
Note that you will have to make appropriate modifications to your boot scripts so that the `chroot()`ed name server is started at boot time as well.



This section contains information on more aggressive spam filtering techniques beyond the simple anti-relaying and sender domain checks that were instituted in Sendmail v8.9. As with those checks, however, the appropriate place to implement these additional features is on your external email servers that exchange email with other Internet sites.

For more information on Sendmail anti-spam functionality, see:

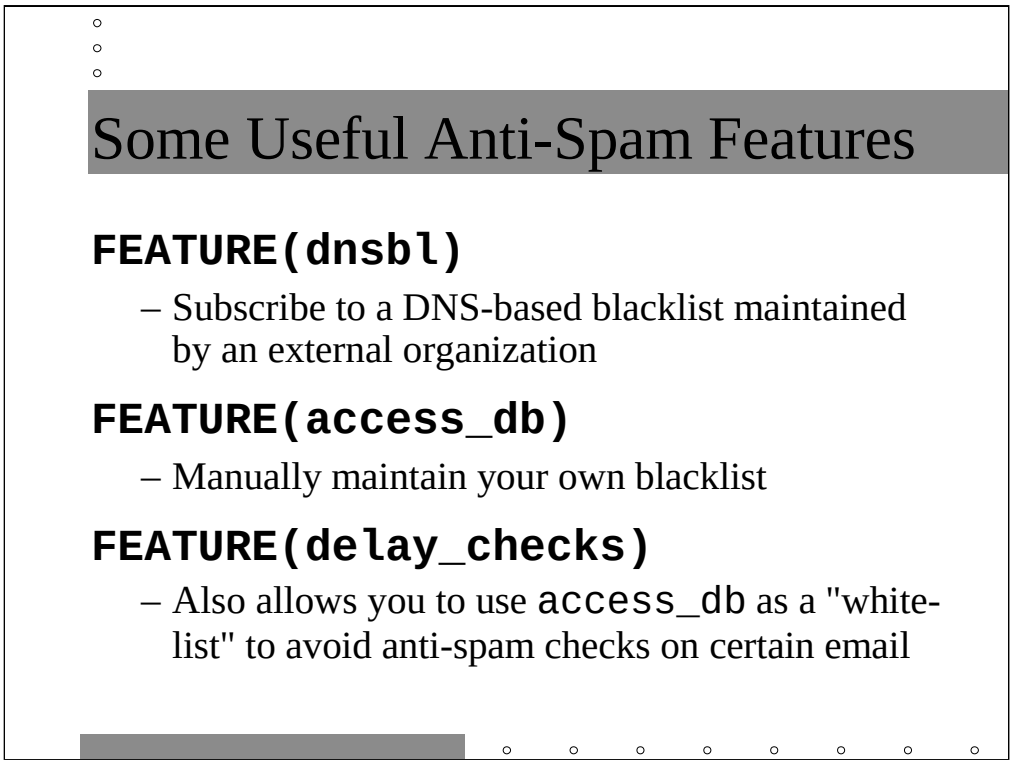
```
http://www.sendmail.org/m4/anti-spam.html
http://www.sendmail.org/tips/relaying.html
```



The more aggressive you are in fighting spam coming into your company, the greater your likelihood of a *false positive*– that is, rejecting legitimate email from the outside (possibly a customer or business partner). Fundamentally, your organization needs to make a business decision about their tolerance for this risk versus the cost to the organization of their users being peppered with spam mail (to say nothing of the disk space and bandwidth consumed).

Some companies which use email as a mechanism for communicating with customers may fear rejecting email from a customer who hasn't properly configured their mail server and is sending out email from a bogus domain or with unqualified addresses. Other organizations (particularly those that pay for email by the byte) are much less tolerant.

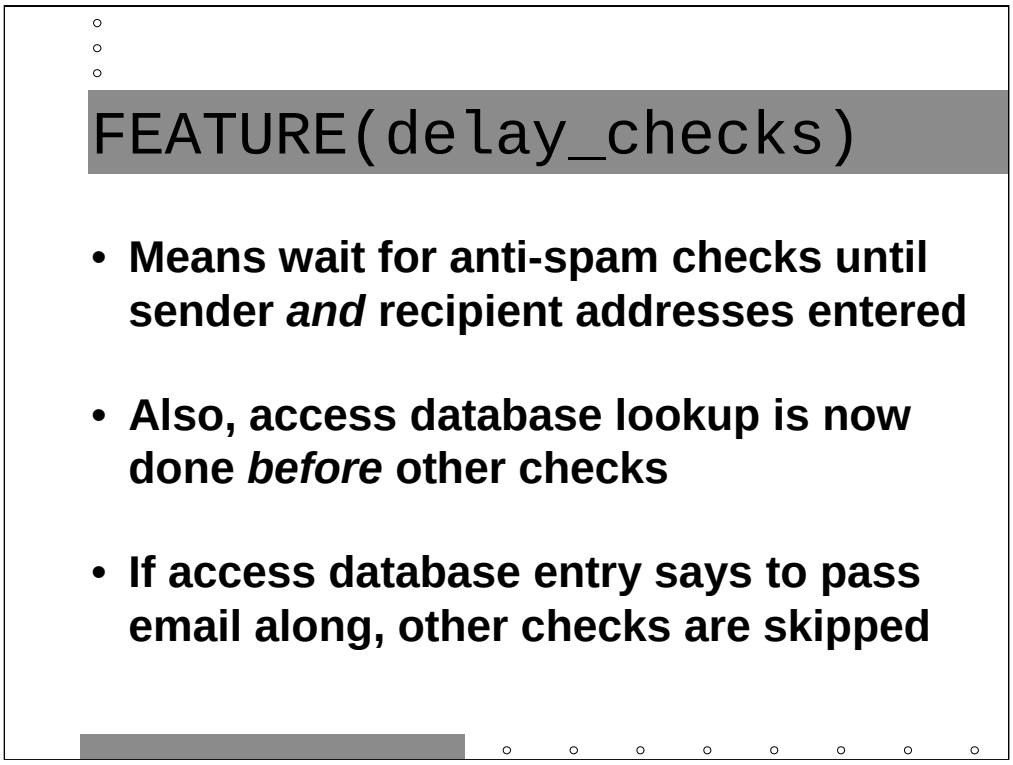
At a minimum, however, organizations should prevent relaying at all costs.



Various organizations now maintain automatically updated lists of known spammers and/or open relays on the Internet. Several of these lists are managed in special DNS zones so that Sendmail and other mailers can look up hosts in these blacklists and simply reject connections from these sites. `FEATURE(dnsbl)` is the mechanism for enabling this functionality in Sendmail.

The access database, enabled with `FEATURE(access_db)`, is a list of IP address ranges, domains, and/or email addresses which you want to reject email from. As we'll see, you have the option of silently discarding email or rejecting it with a (possibly customized) error message.

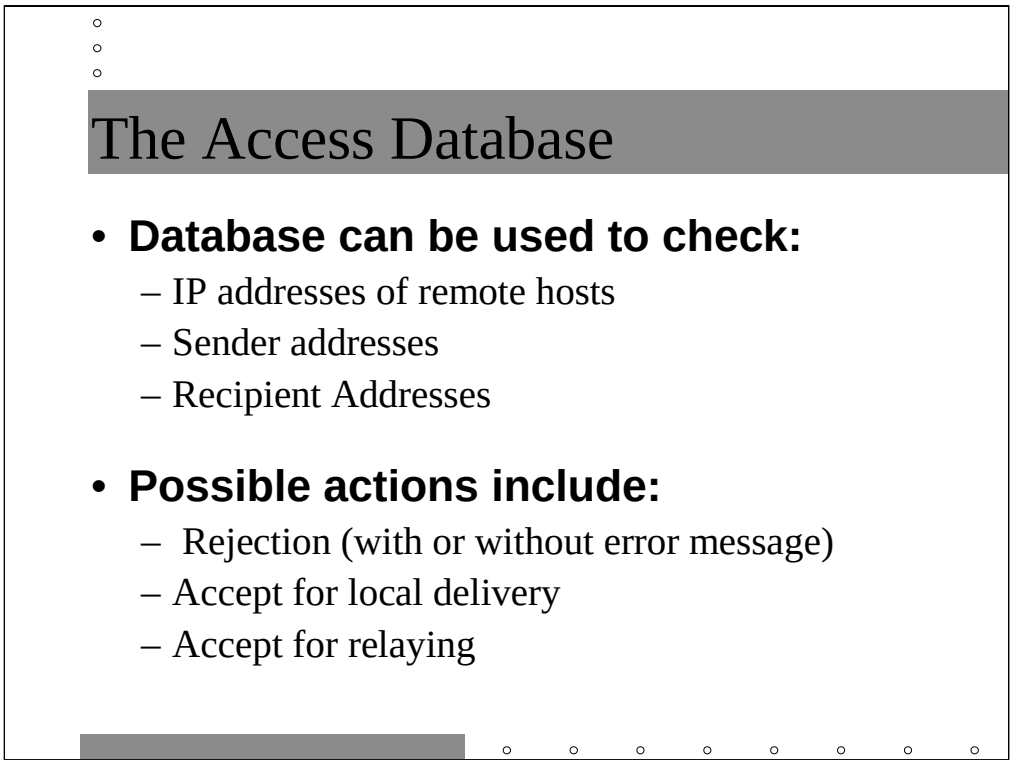
Actually, the access database allows administrators to list sites and/or individual email addresses that they *do* want to accept email from. Unfortunately, many of the anti-spam checks would normally happen *before* the access database is consulted. `FEATURE(delay_checks)` changes the order of processing on the anti-spam checks so that the access database can be used to both permit and deny email from specific sources.



By default, Sendmail's anti-spam checks are aggressive about rejecting email. For example, the IP address of the remote end of the connection is checked against DNS-based blackhole lists (the `dnsbl` feature) as soon as the connection is made. Similarly, email from certain sender addresses may be rejected immediately, before the recipient address is even entered.

Enabling `FEATURE(delay_checks)` causes Sendmail to wait until both the sender and the recipient address have been entered before doing any anti-spam checking. Also, lookups in the access database will be done before other checks– this allows the administrator to specify certain emails that should always be accepted, regardless of what any of the other anti-spam checks would normally do.

I use this feature to accept email from certain specific email addresses of friends and business associates who happen to be using ISPs whose mail servers are listed in one or more of the DNS-based blackhole lists.



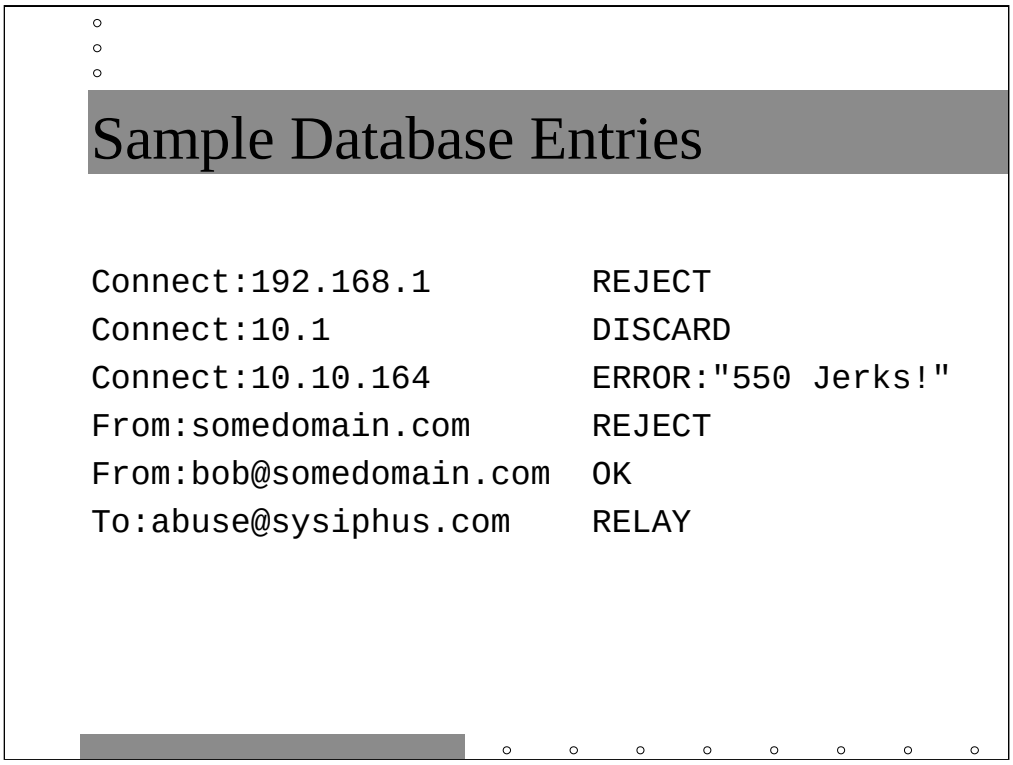
The macro for enabling the access database is similar to the mailertable feature we saw earlier:

FEATURE(`access_db', `dbm -o /etc/mail/access')

Again, the dbm argument means that the access database is going to be a DBM file (like the aliases and mailertable files), and the –o option means that Sendmail will run without complaining even if the database doesn't exist. The base name of the database files is /etc/mail/access, though the actual files will be named access.dir and access.pag.

The access database can key on the IP address of the remote end of the SMTP connection, the sender address (which you should remember might be forged), and/or the recipient address. Possible actions include rejecting the email, accepting it for local delivery, or allowing the email to be relayed elsewhere. If the email is rejected it can either be silently discarded, or an error message can be returned. In fact, the administrator can specify the error message to be returned if desired.

For some examples, turn to the next slide…



In the first line we've decided that network 192.168.1.0 is a haven for spammers and we won't accept any email from hosts in this domain.

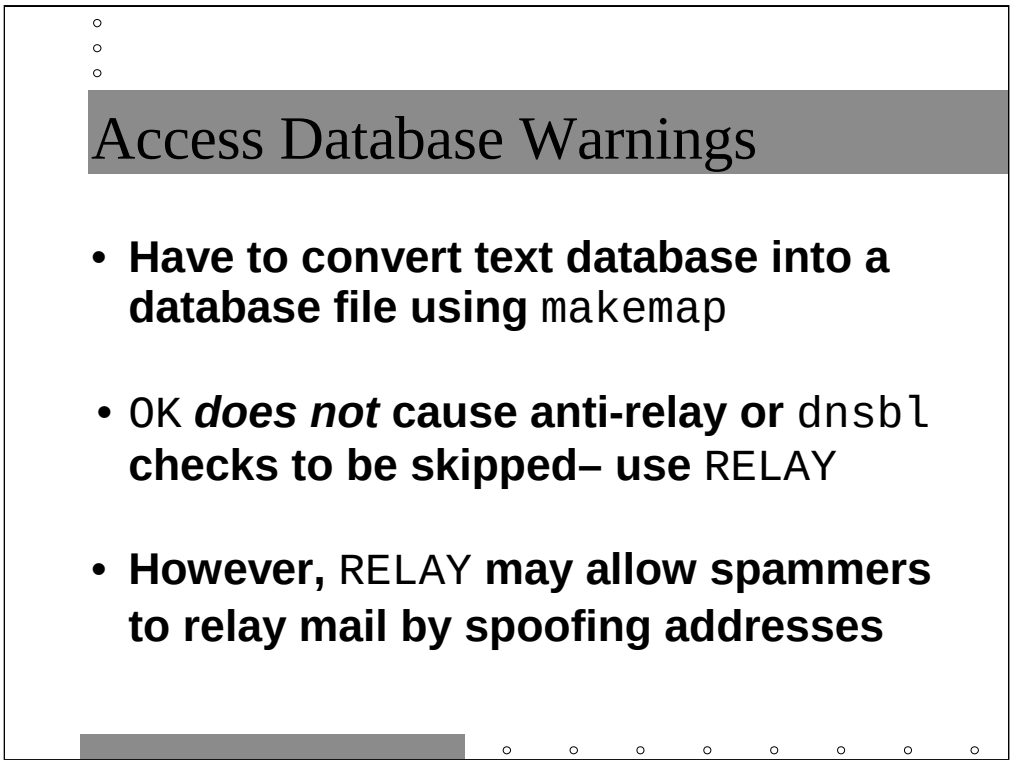
In the second line we've decided that we don't want to let on that we're not accepting email from network 10.1.0.0. The messages will be accepted by our server then thrown away.

The third line demonstrates how you can actually specify an individualized error message for sites that you particularly don't like for some reason. Actually, this feature can also be used to supply a different type of error message– for example one indicating that the recipient address doesn't exist, which might possibly cause the email address to be removed from the spammer's list of email addresses (in reality, the spammer rarely sees the bounce message, so this tactic really doesn't work).

The next two lines cause us to reject all email where the sender address is in `somedomain.com`, except when the email comes from our friend `bob@somedomain.com`. Note that the most specific match is preferred.

The last line says that we'll accept email from anywhere to our `abuse@sysiphus.com` address and relay this email inward to our internal mail servers.

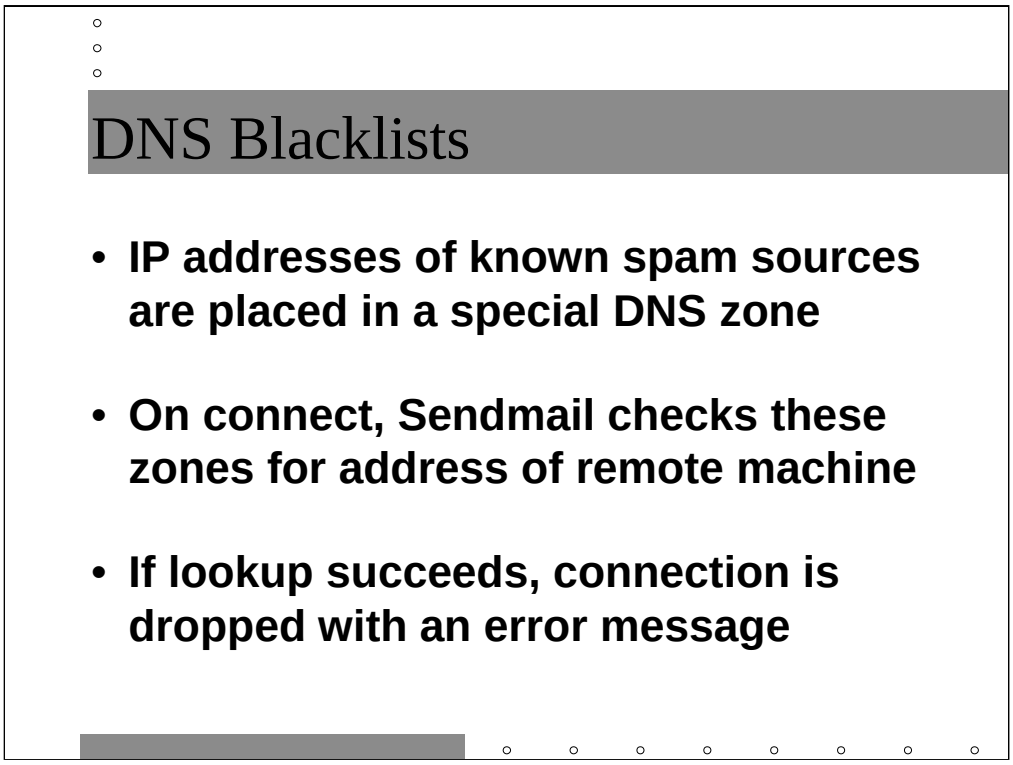
More on the distinction between `OK` and `RELAY` on the next slide…



First, remember that after creating the text file which contains your access database rules, you need to run the `makemap` program in order to convert this file into a DBM file for Sendmail. Check out the slide which describes building the `mailtertable` DBM file at the end of the *DNS and Sendmail vs. Firewalls* section earlier in the course.

Note that `OK` only means that the mail message is acceptable as far as local delivery on the machine with the access database. If the mail needs to be routed further into the organization, then the usual anti-relaying checks are done. The lookups in DNS-based blackhole lists (as enabled with the `dnsbl` feature) will also be performed. This means that you can't use `OK` to bypass these checks– you must use `RELAY` instead.

However, be aware that `RELAY` does not mean only "allow this email to be relayed inwards to my local mail servers"; it means *"allow this email to be relayed to any machine on the Internet"*. If you allow relaying based on the sender email address (`From:` on the lefthand side of the access database and `RELAY` on the righthand side), be aware that a spammer could forge the sender email address and use your server as a promiscuous relay to spam other sites.



Sendmail v8.9 included support for `FEATURE(rbl)` which allowed administrators to subscribe to the original DNS-based blackhole list run by the MAPS project (see `http://maps.vix.com/`). However, other organizations quickly began to use a similar strategy for maintaining blacklists and Sendmail v8.10 replaced `FEATURE(rbl)` with the more general `FEATURE(dnsbl)`.

Simply, `FEATURE(dnsbl)` allows the local administrator to specify a particular DNS domain. Whenever a new SMTP connection is made to the local server, Sendmail looks up the IP address of the remote end of the connection in the specified DNS domain. If there's a match, then the connection is rejected.

`FEATURE(dnsbl)` allows the administrator to specify the error message, so the site knows why they were rejected and what they can do to clean up the problem. It's important that the error message be descriptive in this fashion, because the goal of this exercise is to embarrass the remote site admins into closing open relays– when their local users start complaining about bounce messages, these problems tend to get resolved.

Using FEATURE(dnsbl)

```
FEATURE(dnsbl, `inputs.orbs.org',
   `Rejected- see http://www.orbs.org/blocked.html')
FEATURE(dnsbl, `outputs.orbs.org',
   `Rejected- see http://www.orbs.org/blocked.html')
FEATURE(dnsbl, `spamsource-netblocks.orbs.org',
   `Rejected- see http://www.orbs.org/blocked.html')
FEATURE(dnsbl, `spamsources.orbs.org',
   `Rejected- see http://www.orbs.org/blocked.html')

FEATURE(dnsbl, `relays.mail-abuse.org',
   `Rejected- see http://www.mail-abuse.org/rss/')
FEATURE(dnsbl, `blackholes.mail-abuse.org',
   `Rejected- see http://www.mail-abuse.org/rbl/')
FEATURE(dnsbl, `dialups.mail-abuse.org',
   `Rejected- see http://www.mail-abuse.org/dul/')
```

The two arguments after `dnsbl` are the domain that Sendmail should do lookups in, and the error message that should be returned in the bounced email if the lookup is successful. The URLs which appear in the error messages above are informational pages so that end-users can know why their email was rejected and hassle their local admins. Note that you may "subscribe" to many different blacklists– Sendmail will check each domain in the order specified. based on my experience, the above ordering is the most efficient– that is, the blacklists listed first on this page tend to reject the most email. Subscribing to the ORBS blacklists almost guarantees that you'll be getting some false-positives.

The two most popular sources of DNS blacklists are ORBS (`http://www.orbs.org/`) and the original MAPS project (`http://maps.vix.com/`). Each of these organizations actually maintains several different blacklists which can all be used simultaneously.

The ORBS "inputs" list is a list of known open relays, while "outputs" are hosts which are the endpoint of a series of relay hops which allow promiscuous relaying. "spamsources" are IP addresses which are known to directly generate spam and "spamsource-netblocks" are blocks of addresses which are known to harbor spammers or which provide other support resources for spammers (like spamming software).

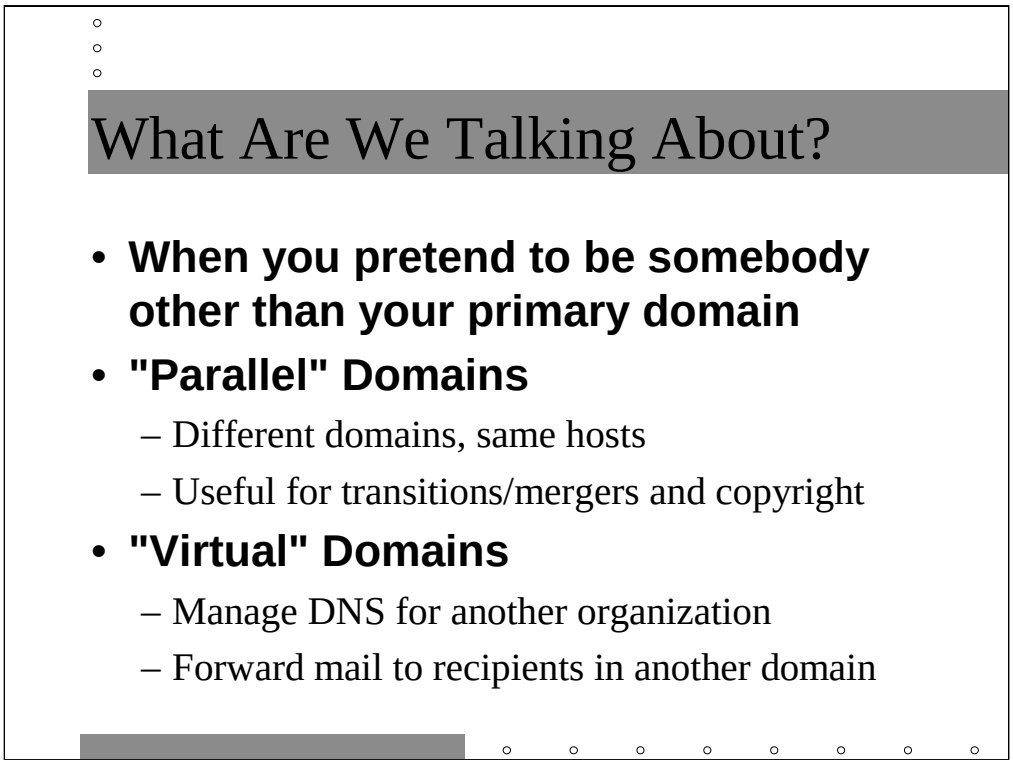
The MAPS "relays" domain is a list of open relays known to the MAPS project (note that the MAPS "relays" domain and the ORBS "inputs" domains do not always completely overlap). "blackholes" is a list of sites that the MAPS project has chosen to blackhole because they emit spam. "dialups" is a list of IP address ranges used by ISPs for dialup pools– hosts in these IP address ranges should be using their local ISP's mail servers for relaying mail, rather than sending email directly. Note that ISPs are encouraged to provide their dialup netblocks to the MAPS project so that they can be included in the "dialups" list.

Virtual Domains

- ***What? Why?***
- ***Parallel Domains***
- ***Virtual Domains***

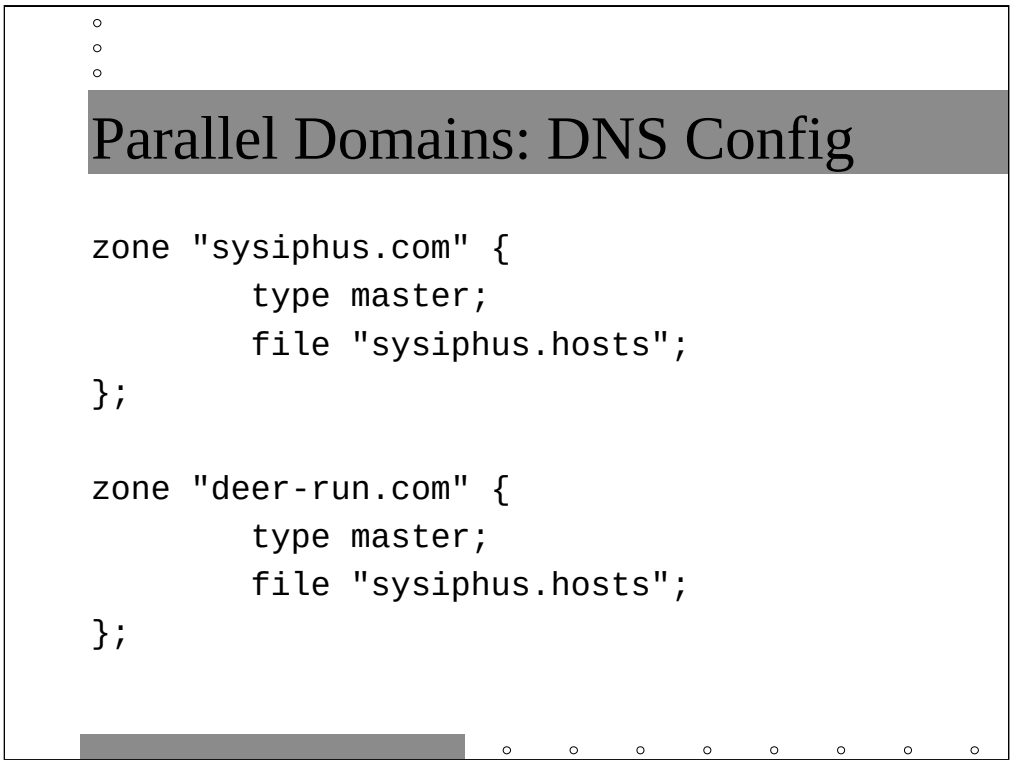
There used to be a time when sites only owned multiple domain names when they merged with another company or decided to change names for other reasons. The explosion of the Internet and vanity domains has meant that lots of people are hosting domains for other organizations.

Frankly, a lot of these hosting companies do poor job of pretending to be multiple domains. It's not much harder to do it right, as we'll see in this section.



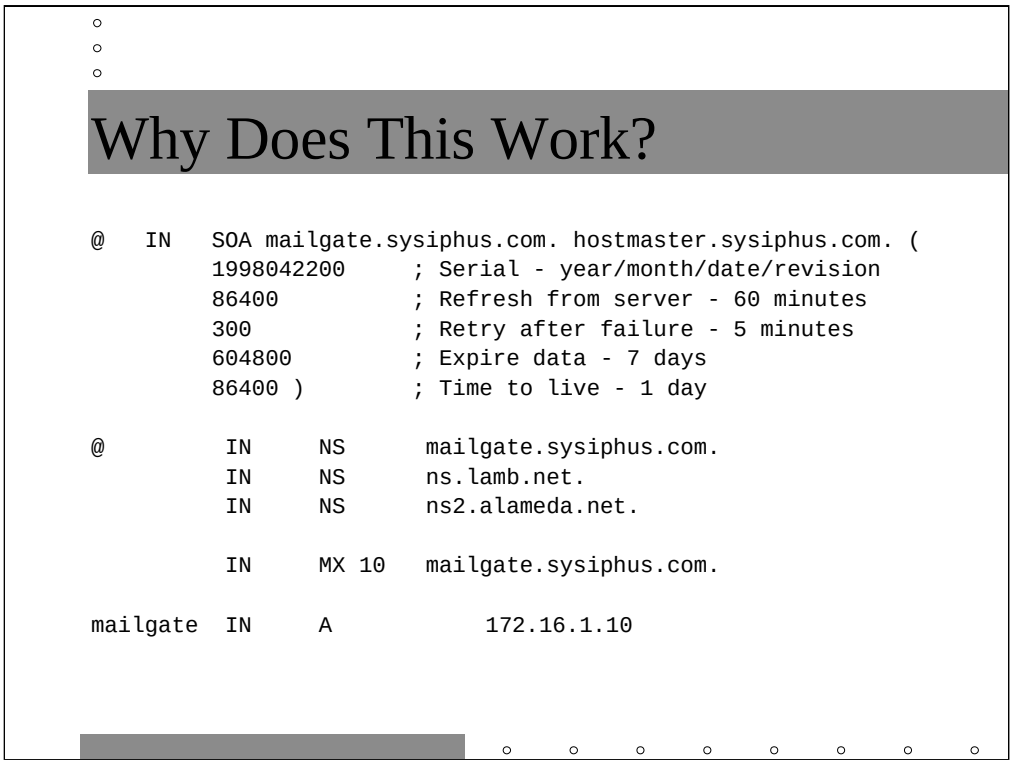
So, there are two types of domain masquerading. Running parallel domains means running two different domain names but using the same hostnames, zone contacts, and network architecture for both domains. You see this when two companies merge or a company changes names. For example, your author used to work at QMS (the laser printer people) which owned `imagen.com` (an acquisition), `qms.com`, and `aqm.com` (that's a long story, but it was QMS' NASDAQ stock symbol).

True virtual domains mean `www.virtual.com` is a completely different beast from `www.sysiphus.com`. Generally, one site will manage DNS and do Web hosting for multiple virtual domains and arrange for mail sent to users in that domain to get forwarded someplace else (maybe a mail account run by the hosting service, maybe another ISP).



Frankly, DNS for parallel domains is dead easy. You don't even have to create a new zone file, just use the one for the real domain. Don't think this works? Check out the next slide…

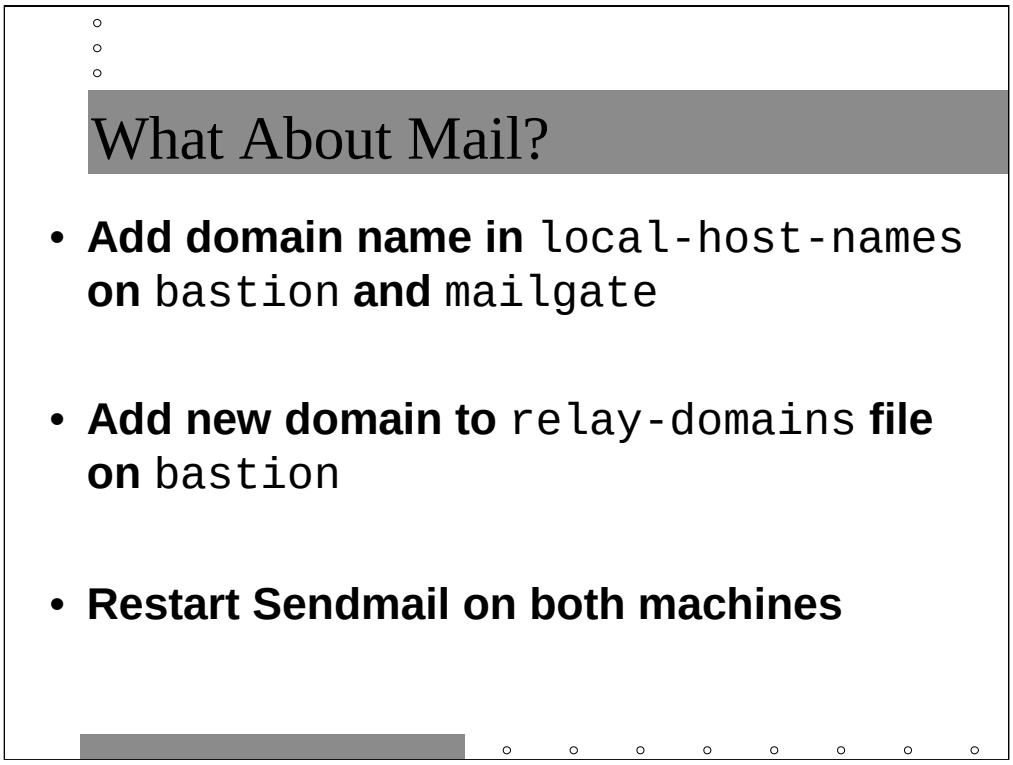
By the way, in reality `deer-run.com` is the author's real domain and `sysiphus.com` is the fake domain.



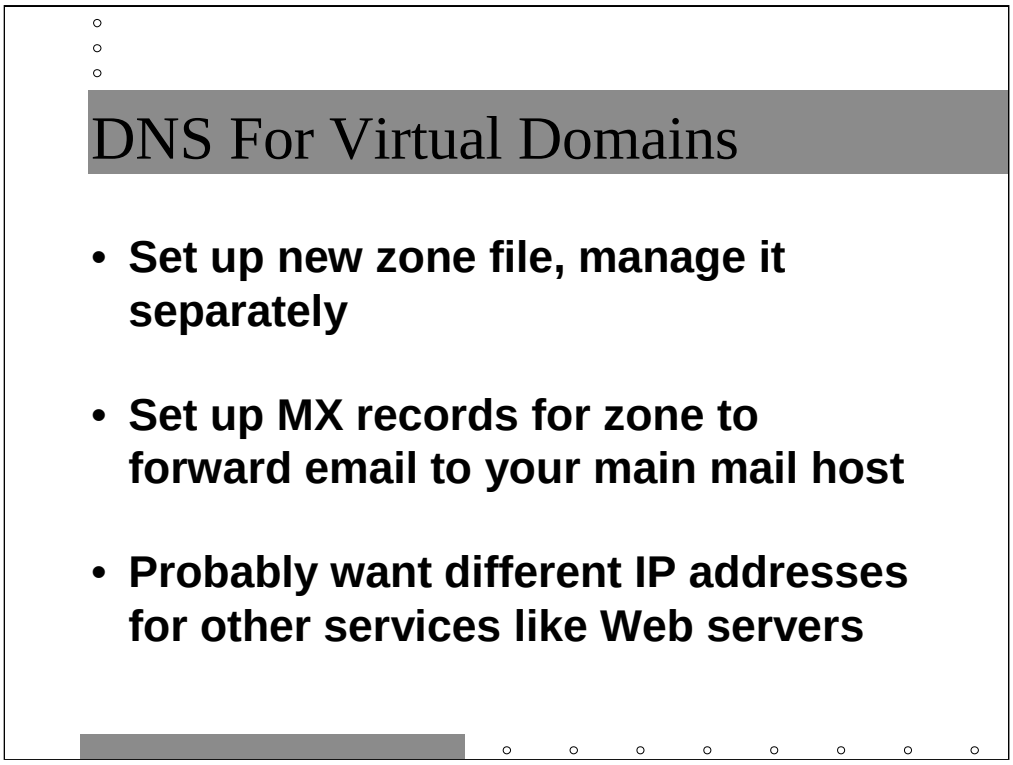
Remember the beginning of the course where we said that unqualified names were useful in the first column of DNS zone files and fully qualified names were useful in the last column? Parallel domains is the reason.

Remember that `@` expands to the domain given in the `named.conf` file, so we're good there. The `NS` and `MX` records for both `deer-run.com` and `sysiphus.com` end up being created properly-- both pointing at the real host `mailgate.sysiphus.com`.

The `A` record defines both `mailgate.deer-run.com` and `mailgate.sysiphus.com` with IP address `172.16.1.10`. The file for the `in-addr.arpa` domain has the `mailgate.sysiphus.com` name hard-coded in, but that's OK.



All you have to do to get mail working for parallel domains is to make sure the new domain is listed as local in the `local-host-names` files on `bastion`, `mailgate`, and any other internal machines that will be receiving mail for the domain. Also update `relay-domains` on `bastion` so mail from outside won't be rejected as a relay attempt.



For truly virtual domains, you'll need to set up a completely separate zone database for the new domain. It's likely you'll use the same `in-addr.arpa` zone. It's OK for `PTR` records in the same file to return hostnames in different domains-- your nameserver doesn't care.

We're going to be doing some interesting Sendmail hacks to pretend we're the mail server for another domain, so just point all of your virtual domain `MX` records to one central server (`bastion` and/or `mailgate` in our `sysiphus.com` example). You don't need to manage a separate mail server for each new domain.

Because we're maintaining a different zone file for each virtual zone, you can use different IP addresses for `www.virtual.com` and `www.sysiphus.com`. If you can't afford separate hardware, look at the `ifconfig` manual page on you system and learn about virtual interface configuration.

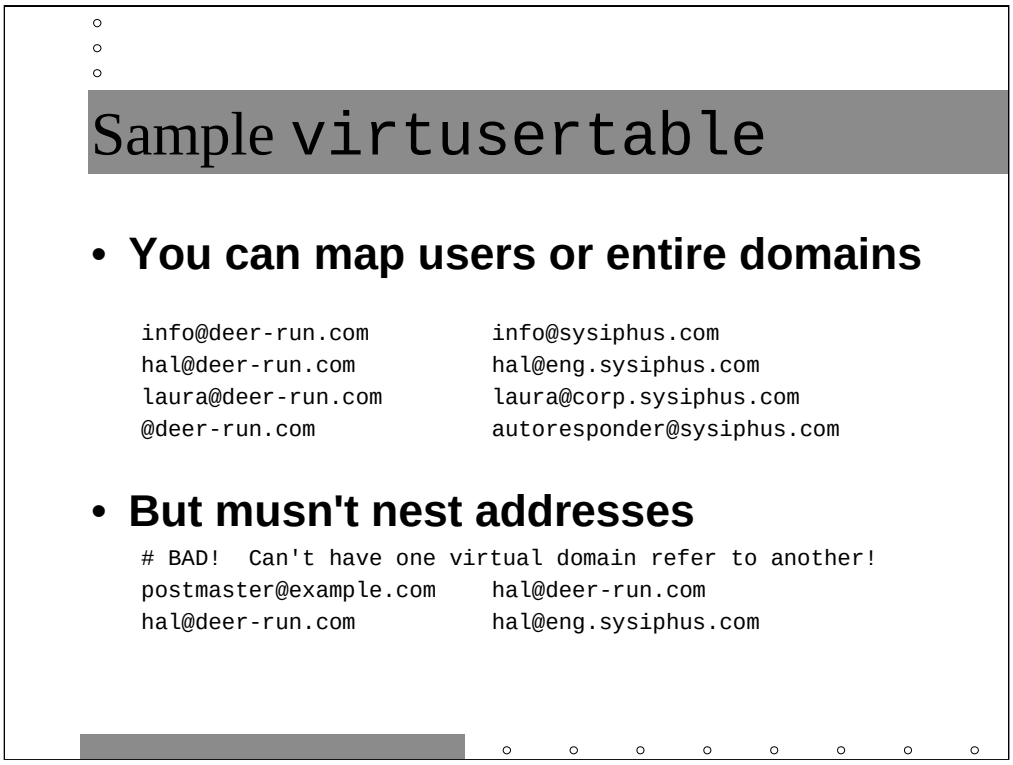
virtusertable Feature

```
include(`../m4/cf.m4')
OSTYPE(`solaris2')

define(`confMAX_HOP',`25')
define(`confSMTP_LOGIN_MSG', `$j mailer ready at $b')
define(`confMIME_FORMAT_ERRORS',`False')
FEATURE(promiscuous_relay)
FEATURE(accept_unqualified_senders)

FEATURE(use_cw_file)

MAILER(smtp)
MASQUERADE_AS(sysiphus.com)
FEATURE(`mailertable', `dbm -o /etc/mail/mailertable')
FEATURE(`virtusertable', `dbm -o /etc/mail/virtusertable')
```

Sendmail v8.8 added the `virtusertable` feature to allow admins to fake mail delivery for multiple virtual domains. The arguments in the second part of the `virtusertable` declaration describe how the `virtusertable` database is formatted.

In this case, the `virtusertable` will be an NDBM database file, just like the `mailertable` database we saw in the previous section. As with the `mailertable`, you will need to first create a text version of the `virtusertable` (see next slide) and then run `makemap` to create the NDBM file.

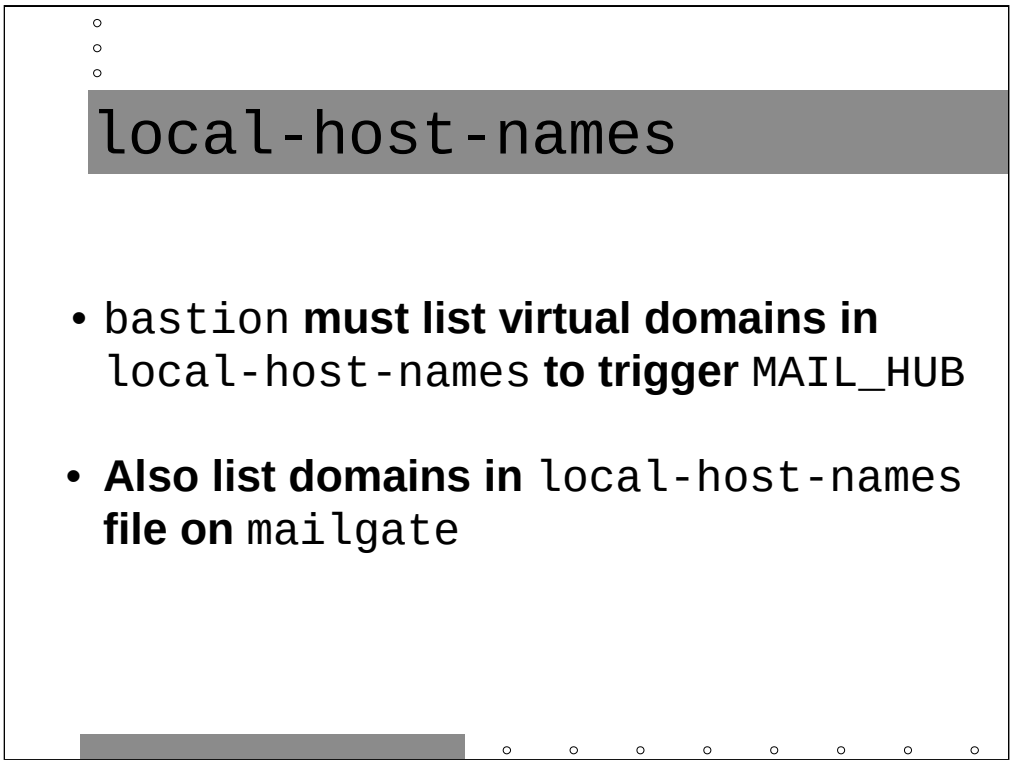
Note that we're assuming the `virtusertable` database is going to be running on `mailgate`. Theoretically, we could run the `virtusertable` database on the `bastion` host. However, aside from simplifying life by consolidating all of the "interesting" email routing tasks on single machine, we are also protecting our virtual domains by not running their email off of an external server.



In the first example we assign some addresses in the domain `deer-run.com` to real addresses in `sysiphus.com`. On the last line we send all other addresses in the domain to an `autoresponder` alias (which perhaps responds to the sender saying the message is invalid).

One thing `virtusertable` does not allow is having a virtual address which refers to another virtual address. You will generate an error message because Sendmail will not see the second definition. Since the same real address can map to multiple vitual addresses in the file, this is really not a problem. The correct way to write the two lines in the lower example is

```
# Correct.  Both lines use real email address.
postmaster@example.com     hal@eng.sysiphus.com
hal@deer-run.com           hal@eng.sysiphus.com
```



Assuming you're going to do the virtual domain faking on `mailgate`, you have to tell the `bastion` that the virtual domains are local so that it forwards the mail inwards-- since the virtual domains appear in `local-host-names`, the `MAIL_HUB` directive on the `bastion` will cause this email to be relayed directly to `mailgate`.

To activate the `virtusertable`, make sure all of your virtual domains are listed in the `local-host-names` file on `mailgate`. Like aliases, the `virtusertable` database is only consulted during a local delivery attempt.

Majordomo and Mailing Lists

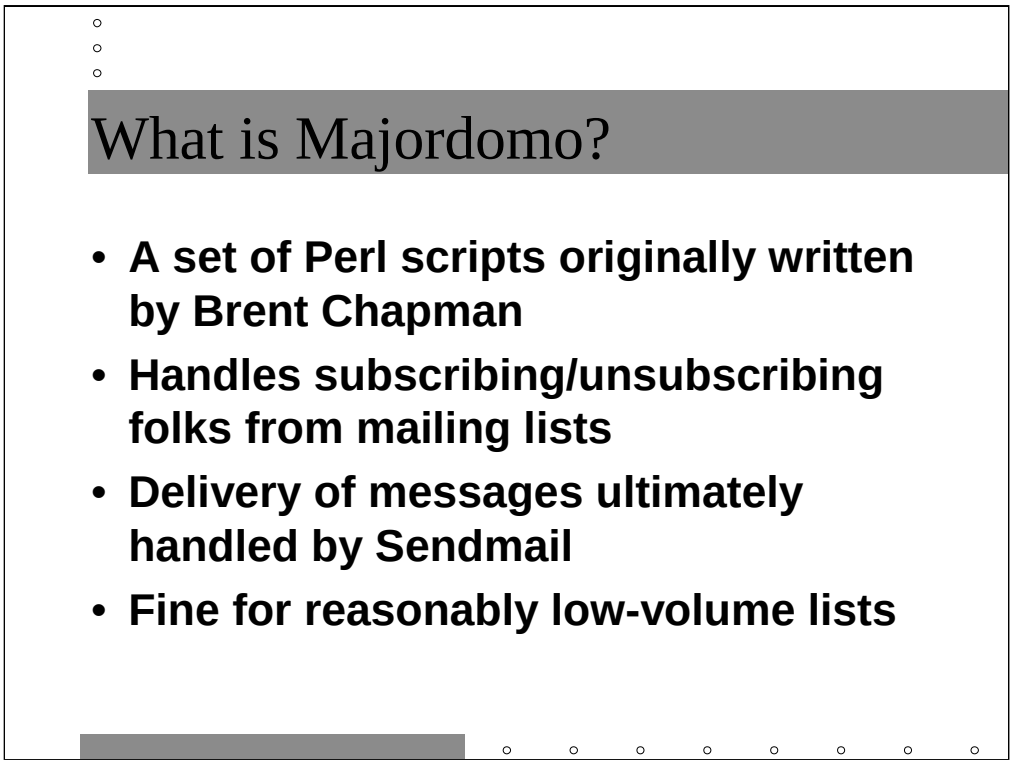
- ***About Majordomo***
- ***Building and Installing***
- ***Creating Mailing Lists***

In this section we're going to talk about how to use the freely available Majordomo software to create and manage mailing lists. Majordomo is pretty much the de facto choice for Internet mailing list management.

Mailing list are often better than creating an alias with a defined list of users. For one thing, users and the list manager can subscribe and unsubscribe without administrator intervention. You can control who is allowed to subscribe or send email to the list.

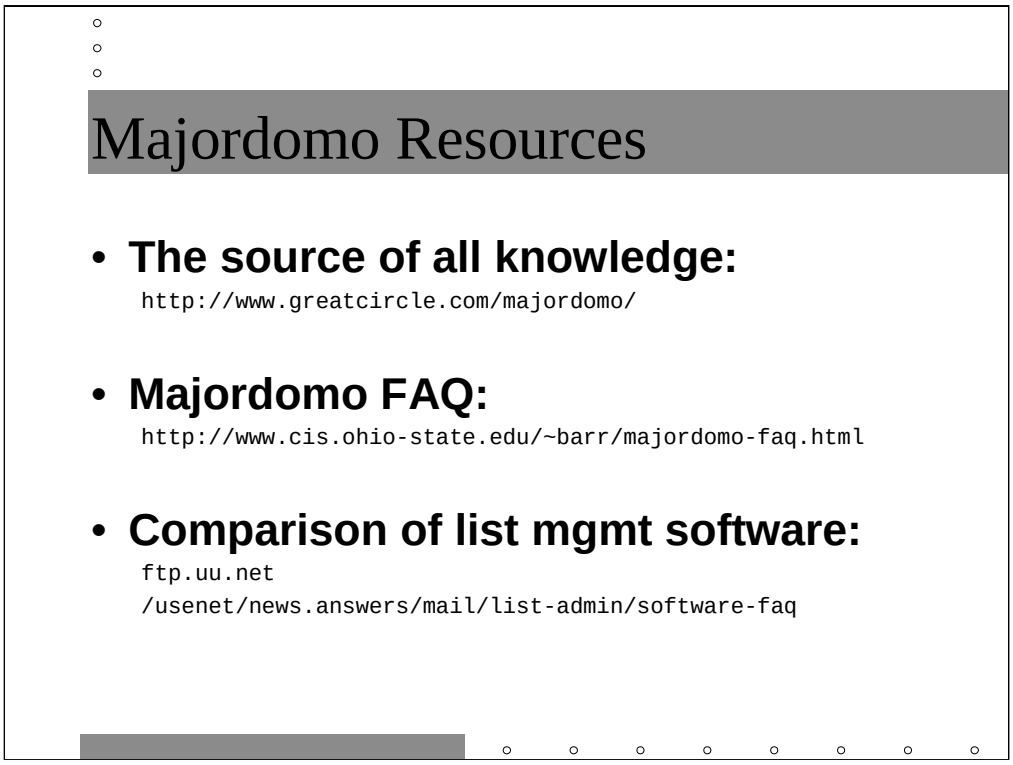
Majordomo also has built-in archiving and digest functionality. We won't discuss those features in this tutorial, however.

The downside is that Majordomo is a big Perl script. For very high volume lists or sites that run a lot of mailing lists, Majordomo may not be a good choice.



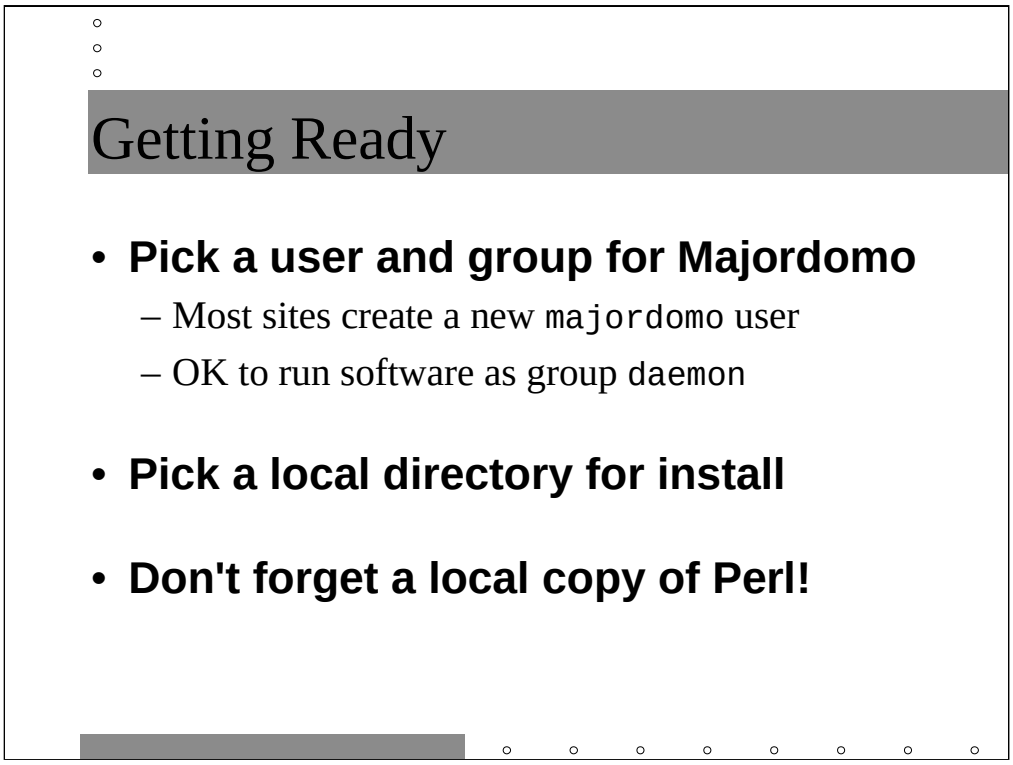
Majordomo was originally written by Brent Chapman. The current maintainer is Chan Wilson. Chan keeps threatening to do a major rewrite of the code but never seems to have time.

Majordomo's job is to subscribe and unsubcribe people from mailing lists and to resend mail that's sent to the list (so it can control who's allowed to post to the list, etc.). Majordomo hands everything off to Sendmail to actually get delivered, so every time a message comes into the mailing list, three processes are actually run to handle it: mail is received by Sendmail, the Majordomo Perl script is forked, Sendmail invoked again to send mail out. This can cause heavy loads and memory usage on machines doing mailing list expansion for busy lists. Many organizations run Majordomo on its own machine so the resources it consumes don't impact other services.



The Majordomo software is still kept at Great Circle Associates (the consulting company that Brent Chapman started). The FAQ is at Ohio State University and is maintained by Dave Barr. Read the FAQ!

There's an OK comparison of your options are far as list management software in one of the news.answers FAQs. I share most of the author's prejudices.



You need to do a little prep work before installing Majordomo. First pick an account for Majordomo to run as. `root` is *not* a good choice. Most sites create a special `majordomo` user with its own unique UID. This is a good choice.

You want to install Majordomo on a non-NFS-mounted directory so your mailing lists don't fail spectacularly when your NFS server dies. Since the Majordomo software is written in Perl, you also need a copy of Perl on your local drive.

Building Majordomo

- **Download software**

```
ftp.greatcircle.com
/pub/majordomo/majordomo.tar.gz
```

- **Unpack and build**

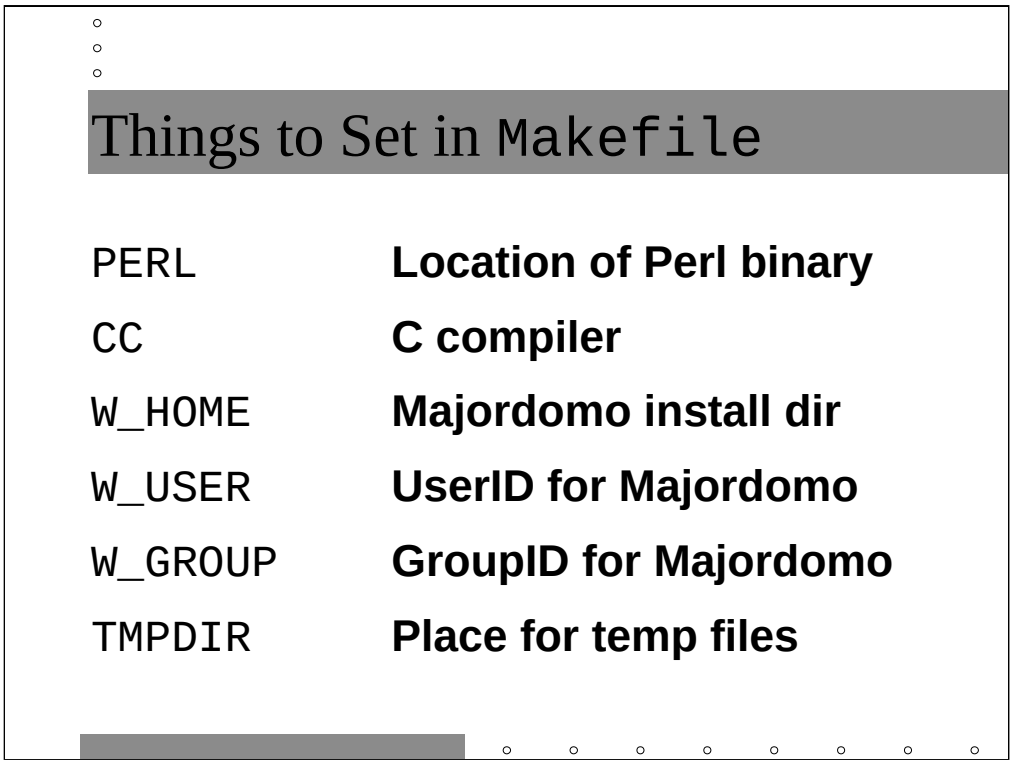
```
% zcat majordomo.tar.Z | tar xf -
% cd majordomo-1.94.4
% vi Makefile                    (see next slide)
% cp sample.cf majordomo.cf
% vi sample.cf                   (see notes)
% make wrapper
```

Now grab the software from Great Circle and unpack it. You need to edit the `Makefile` as discussed on the next slide.

You also need to copy the `sample.cf` file to `majordomo.cf` and then edit your `majordomo.cf` file. In this file you need to set

`$whereami`	`The local domain`
`$sendmail_command`	`/usr/lib/sendmail` (?)

`make wrapper` actually builds a suid binary which is used to safely invoke the Majordomo scripts as the `majordomo` user. Perl has historically had problems with its suid support, so the Majordomo developers figured they better write their own wrapper.

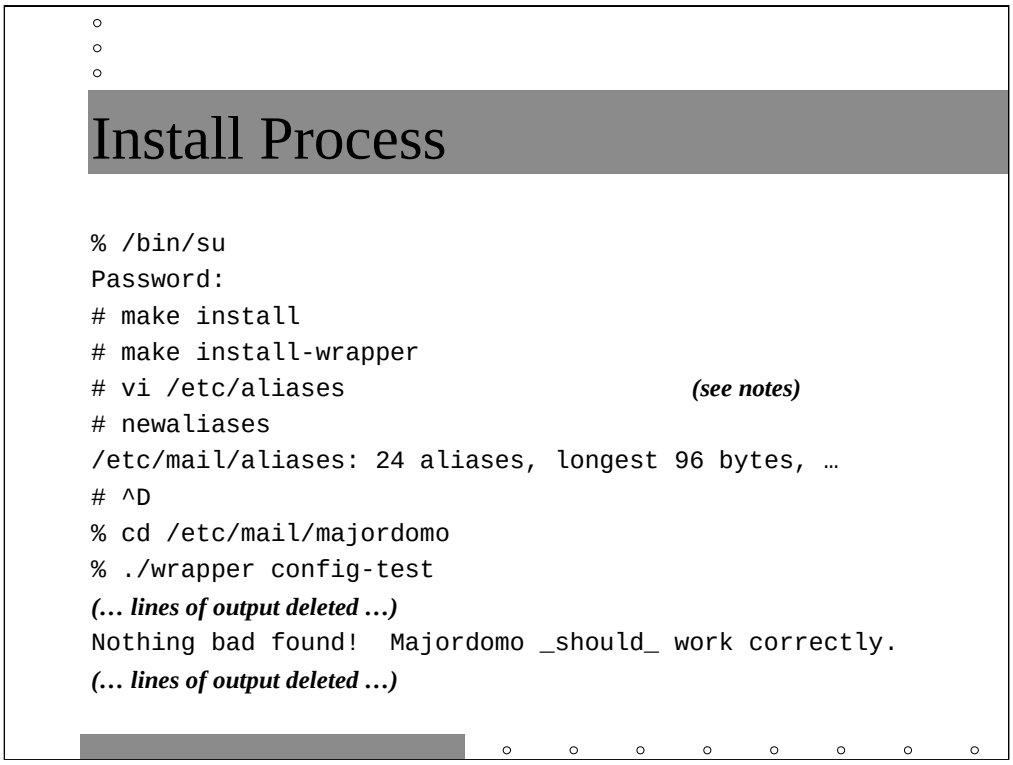


These are the variables you need to set in the Majordomo `Makefile`.

Again, your life will be better if your Perl binary is on a local drive and not an NFS directory.

I often install Majordomo in `/etc/mail/majordomo`, but you need a lot of space in your root filesystem for this (`/opt/Majordomo` and `/var/majordomo` are also good choices).

The default `TMPDIR` for Majordomo is `/var/tmp`-- you'll get better performance on Sun systems if you use `/tmp` which is a RAM disk.



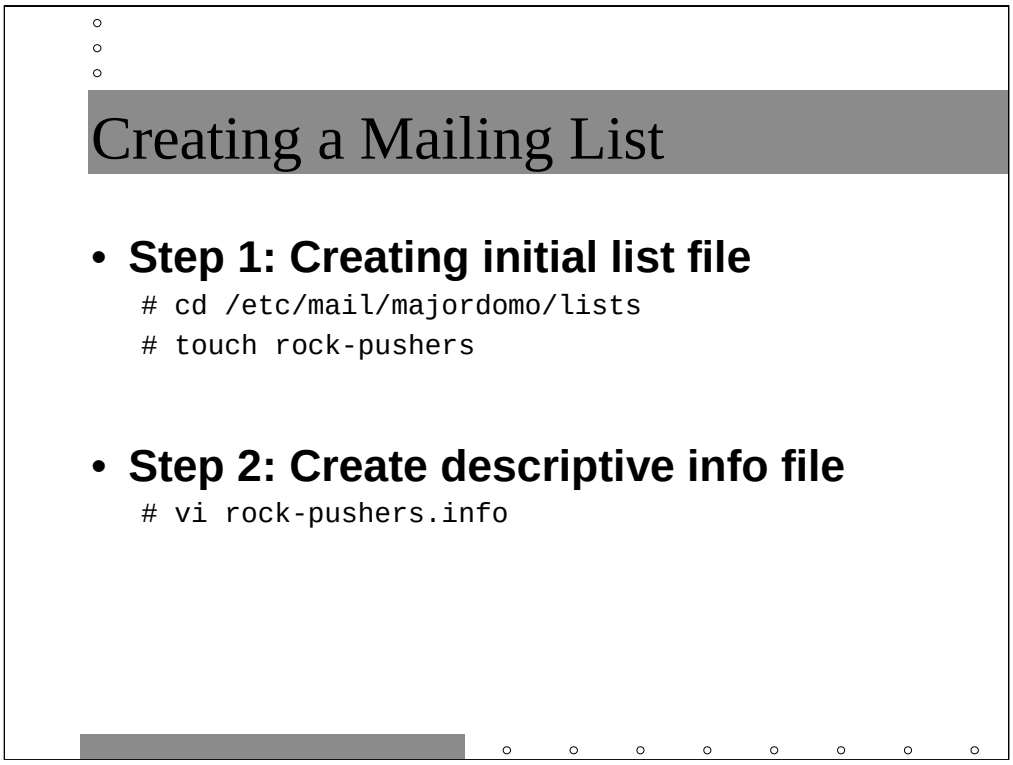
Once you've built the Majordomo wrapper script, you need to install the Majordomo scripts and the wrapper binary.

Edit your aliases file and add these aliases:

```
majordomo: \
  "|/etc/mail/majordomo/wrapper majordomo"
owner-majordomo: hal
majordomo-owner: hal
```

This way users can send mail to `majordomo@sysiphus.com` in order to subscribe or unsubscribe from mailing lists. Obviously the `owner` aliases should point to a real human being at your site. Don't forget to run `newaliases` when you're done!

Once you've completed the install process, run the `config-test` script via the Majordomo `wrapper`. Make sure you're *not* `root` when you do this or the results won't make sense. In all the output of the `config-test` script you're looking for the "`Nothing bad found!`" message. If you don't find this message then you have to read the rest of the output and figure out what you did wrong.



In example, we're creating the `rock-pushers@sysiphus.com` alias. Users will be able to send mail to the members of the list by sending mail to this address, and will be able to subscribe/unsubscribe from the list by sending mail to `rock-pushers-request@sysiphus.com`.

The first step is to cd into the …`/majordomo/lists` directory and create the initial list of recipients for the `rock-pushers` alias. This file can be empty or you can add some initial email addresses to the list right now.

Next you ought to create a file describing the list and its purpose, plus any rules or guidelines for the list and call that file `rock-pushers.info`. This `info` file will be sent to every user when they subscribe and can be requested from the Majordomo server by members of the lists (and outsiders if you permit it) at any time. The `info` file is optional.

Creating a Mailing List (cont.)

- **Step 3: Generate list config file**

```
# echo lists | mailx Majordomo@sysiphus.com
# vi rock-pushers.config
```

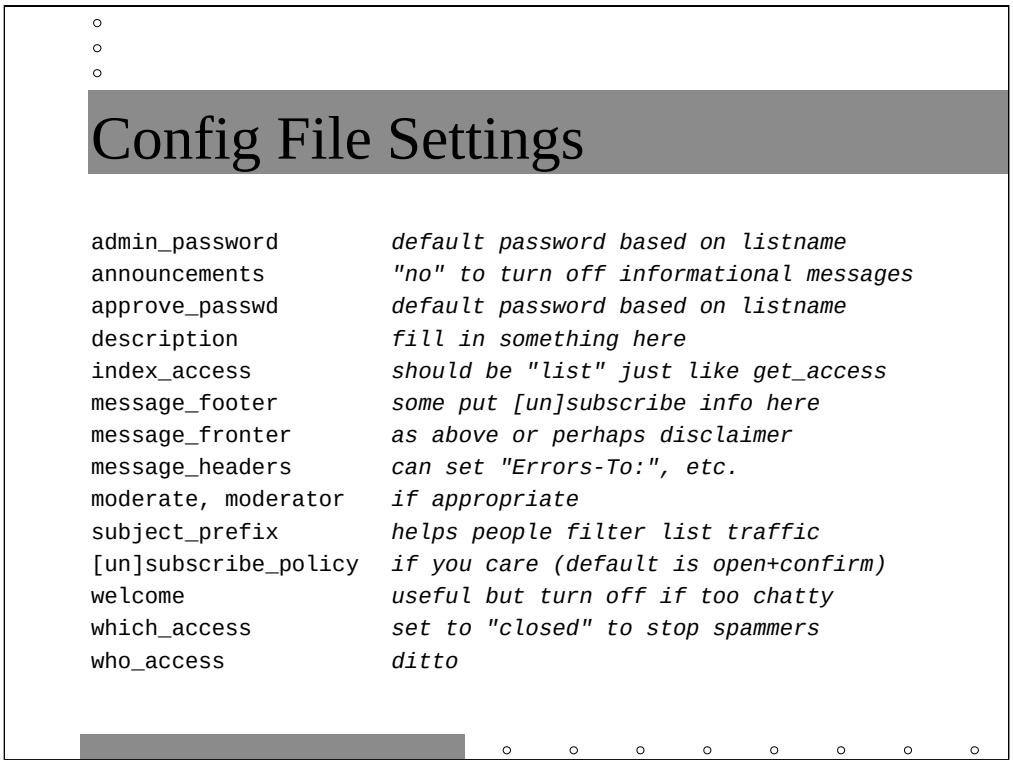
(see next slide)

- **Step 4: Set file ownerships**

```
# chmod 644 rock-pushers*
# chown majordomo rock-pushers*
# chgrp daemon rock-pushers*
```

Next you need to generate and tweak the `rock-pushers.config` file. The easiest way to generate the `config` file is to have Majordomo do it for you. Unfortunately, Majordomo makes some bad default choices, IMHO. You might want to keep a canonical `config` file around, copy it to the right file name, and tweak it for each new list. We'll talk more about settings in the `config` file on the next slide.

Before you leave the lists directory, be sure to verify that all files are owned by your `majordomo` user and group. The files should be writable by the `majordomo` user and world readable.



There are dozens of settings in the Majordomo list config files. These are just some of the more important ones.

The list passwords end up being a variant of the list name. You probably want to choose better passwords.

Some list managers like to append a disclaimer header/footer and/or instructions on how to unsubscribe from the list. You can also add a special string which is pre-pended to every message subject line to allow people to filter mail more easily.

The default subscribe/unsubscribe policy is `open+confirm` which means that anybody can subscribe to the list and when they do they have to send back a special confirmation message. This is to avoid people subscribing a person they don't like to every mailing list in the world as a denial of service attack. If you set the subscribe policy to `closed`, then the owner of the list has to approve the subscription.

The `which` command (find out which lists an address is subscribed to) and the `who` command (find out who's on a given list) are often used by spammers to help generate lists of addresses to pester. Shut off access to these commands-- even to list members because spammers can always subscribe to a list.

Creating a Mailing List: Aliases

```
rock-pushers: "|/etc/mail/majordomo/wrapper
  resend -l rock-pushers rock-pushers-outgoing"

rock-pushers-outgoing:
  :include:/etc/mail/majordomo/lists/rock-pushers

rock-pushers-request:
  "|/etc/mail/majordomo/wrapper majordomo
      -l rock-pushers"

rock-pushers-approval: hal
owner-rock-pushers: hal
rock-pushers-owner: hal
```

The first alias invokes the Majordomo `resend` script to forward a message to the recipients of a given list. The `resend` script enforces privacy features like only list members being allowed to send email to the list, as well as appending headers/footers, etc.

The `resend` script just fires the email to another alias which uses the standard alias `include` directive to pull in the contents of the list file corresponding to the `rock-pushers` list in the Majordomo directory. Spammers can completely circumvent the `resend` script and send mail directly to the `outgoing` list and there's nothing you can do about it. Furthermore, the `outgoing` alias appears in the headers of messages sent via the resend script, so the alias name is in no way hidden. A fix for this problem would involve hacking Majordomo.

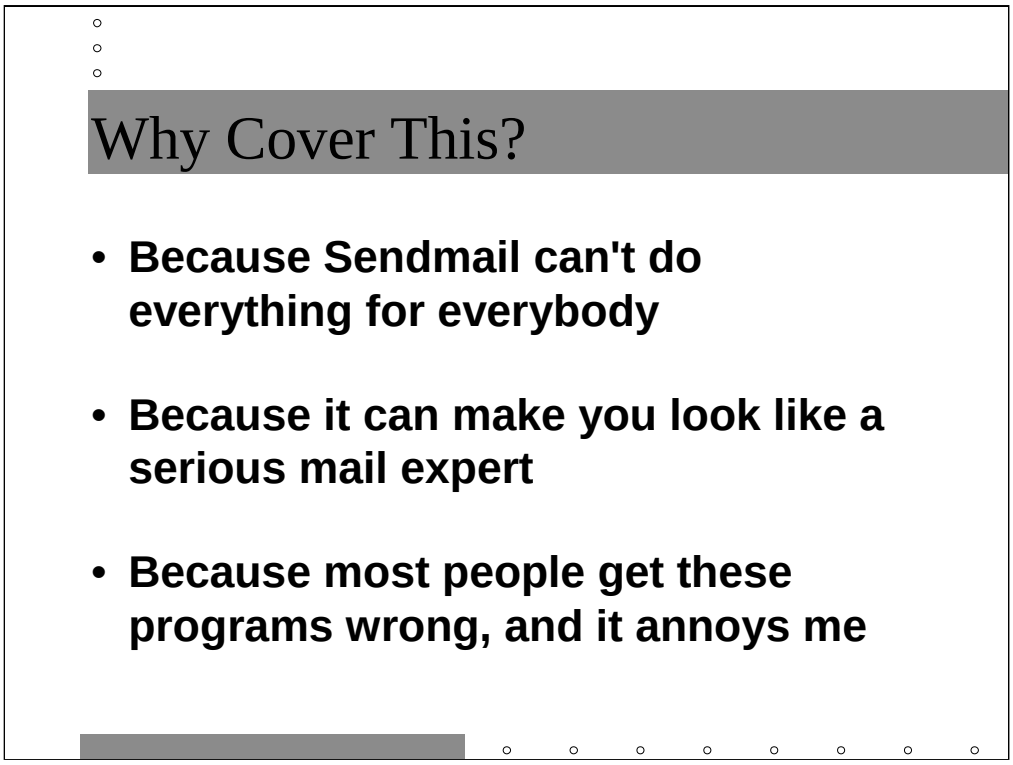
We also need to set up a `rock-pushers-request` alias to allow people to subscribe and unsubscribe from the list. Users can also do this for all lists via the `majordomo` alias. The `request` alias has become a standard for list management, however.

You need a real human owner of the list, and if you're moderating the list you need a real human at the `approval` alias. Note that you really only need the `owner-rock-pushers` alias (this is where Majordomo will direct list errors), but the `rock-pushers-owner` form is another one of those de facto standards.

Writing Your Own Mail Progs

- ***Why?***
- ***How They work***
- ***Advice and sample code***

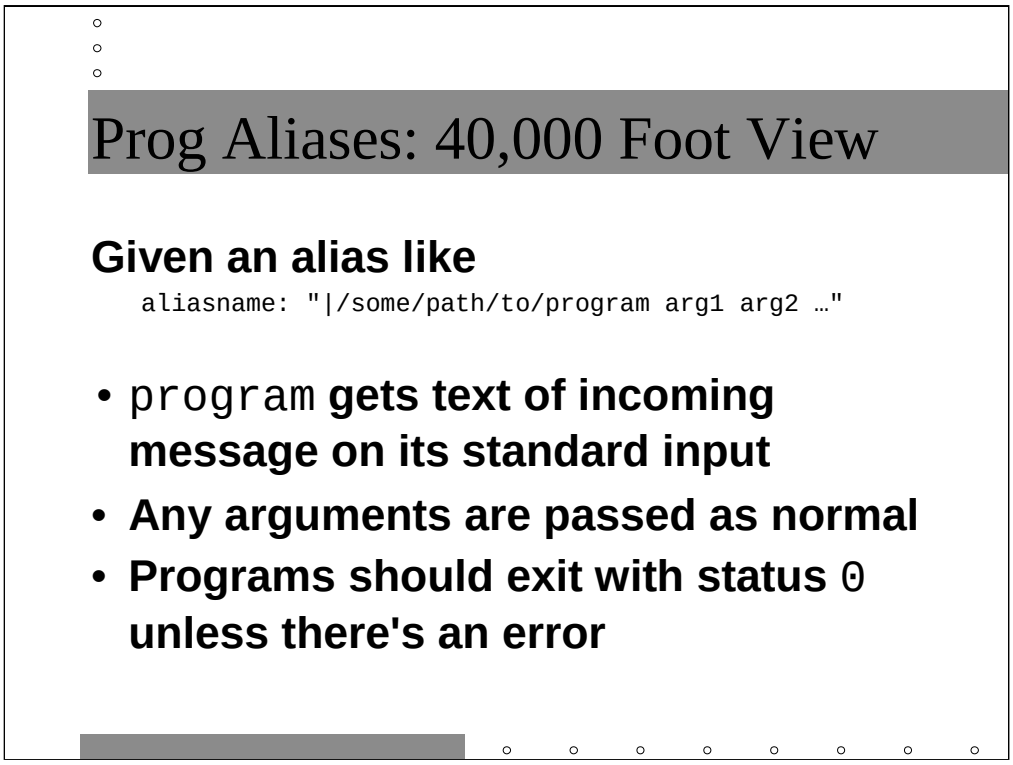
If you're going to really be a mail administrator you need to be able to write programs which handle mail. You'll often be called on to write autoresponders and even more complicated programs.



Sendmail is an extremely complicated program with many features, but it can't do everything. Writing programs to handle mail is a way of extending Sendmail's feature set.

Besides, if you write a cool hack to process mail people will write songs and epics to your fame and people of both sexes will throw themselves at your feet begging to have your love child. Or you might get a bonus, though this is less likely.

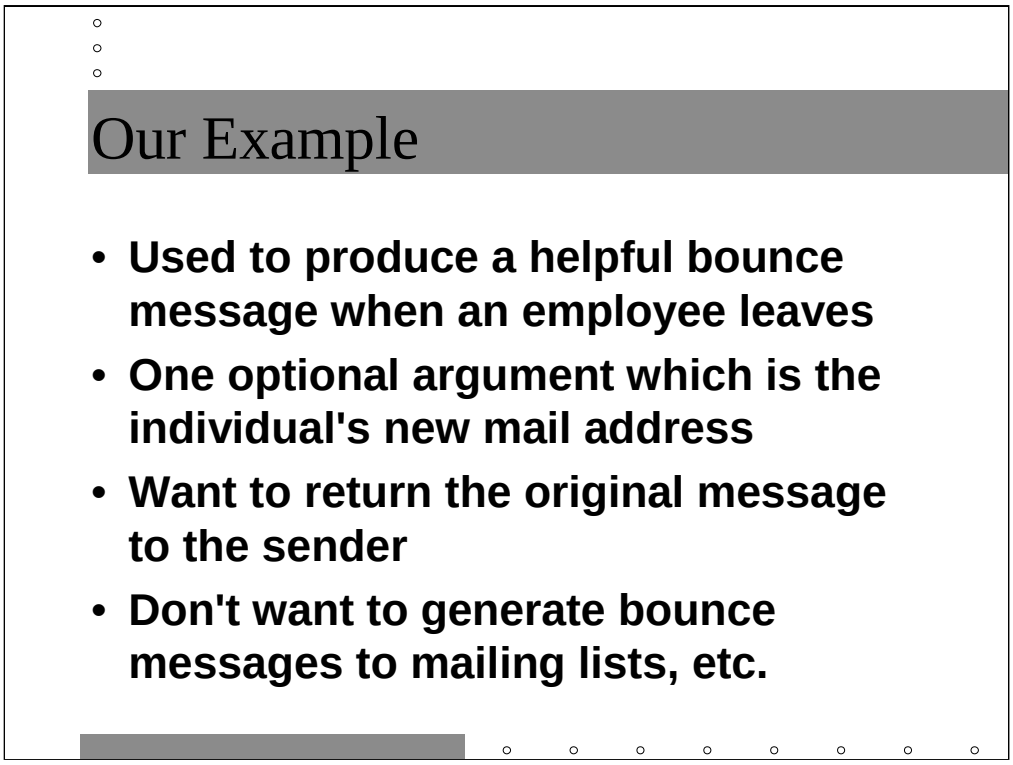
Frankly, most people do these sorts of programs badly and end up spamming mailing lists or getting into shouting matches with other autoresponders and causing denial of service attacks. Even worse, badly coded mail programs can allow attackers to get root access on your mail servers and/or destroy data.



Generally you trigger mail programs out of the aliases file. You can feed command line arguments to the program in the alias file but these arguments are hard-coded into each alias. The email message that triggers the alias is fed into the program on it's standard input file descriptor.

Your mail programs should exit with status `0` unless there's a problem. If you exit with an error message postmaster will get an email message like

```
Unknown mailer error: <error message>
```



Our example is going to be the standard "this employee no longer works here" auto-responder. You can invoke the alias with

```
john: \
   "|/etc/mail/natabot john@elsewhere.com"
mary: "|/etc/mail/natabot"
```

The script takes an optional argument which is the user's new email address.

We want to return the senders original message back to them and we want to make sure we don't spam mailing lists or respond to other auto-generated mail messages.

The complete source for the `natabot` program is included as the last of the Appendices to this tutorial.

Getting the Return Address

```
# Try to get return address from envelope From.
# Quit silently if message is from MAILER-DAEMON.
# Complain bitterly if we detect shell metachars.
#
$header[0] = <STDIN>;
($from) = $header[0] =~ /^From\s+(\S+)/;
exit(0) if (!length($from) ||
            $from eq '<>'  ||
            $from =~ /mailer-daemon/i);
die "Hostile address: $from\n"
        if ($from =~ /[\\|&;]/);
```

All SMTP-style email messages begin with a line that looks like

```
From <address> <date>
```

This is generally referred to as the *envelope From* header to distinguish it from the `From:` header that appears elsewhere in the message.

Lesson #1 is that the envelope From header is the only address header you should believe. Don't use any addresses from elsewhere in the message because they might be forgeries or unintentional lies.

Lesson #2 is that the envelope From can also be forged and people can do nasty, nasty things to you if you're not very careful (like embedding `rm -rf` commands in mail addresses). Note that we look for dangerous shell metacharacters before proceeding.

Lesson #3 is that you should never respond to messages from the user `MAILER-DAEMON` (aka user `<>`). These are always auto-generated messages.

Sucking in Mail Headers

```
# Scan through headers until blank line.
# If we find a Precedence: header, obey it.
#
while (<STDIN>) {
   last if (/^$/);
   if (($prec) = /^Precedence:\s+(\S+)/) {
      exit(0) if ($prec =~ /^(junk|list|bulk)$/);
   }
   push(@header, $_);
}
```

Lesson #4 is that the headers are separated from the body of the message by a single blank line (no whitespace, no nothing). You stop processing the headers when you hit this line and treat the rest of the message as an opaque blob of data.

Lesson #5 is that some folks put in a special `Precedence`: header and set the value of the header to `bulk` or `junk` or `list` meaning this message was auto-generated and should not be responded to by auto-responders. While the `Precedence:` header is not an official standard, you may as well obey it if somebody goes to the trouble of adding one.

Lesson #6 is that header lines look like

```
<header>:<whitespace><value>
```

or, in other words:

```
Precedence: bulk
```

`<value>` may be made up of several words separated by whitespace.

To be completely accurate, headers may continue on multiple lines as long as the continuation lines begin with whitespace. You see this most commonly in `Received-by:` headers

```
Received: (from smap@localhost)
    by doe.deer-run.com (8.8.7/8.8.7) id RAA03062
    for <hal@deer-run.com>; Mon, 26 Jan 1998...
```

Starting Sendmail Safely

```
# Be paranoid of the contents of $from and start
# Sendmail without invoking the shell.  Make sure
# we send our message from MAILER-DAEMON to avoid
# autoresponders at remote site.
#
$pid = open(MAIL, "|-");
die "fork() failed: $!\n" unless (defined($pid));
unless ($pid) {
    exec($sendmail, '-f', '<>', $from);
    die "exec() failed: $!\n";
}
```

Lesson #7 is that you can't be too careful with that potentially forged envelope From address. When you do the normal

`open(MAIL, "|/usr/lib/sendmail $from") || die…`

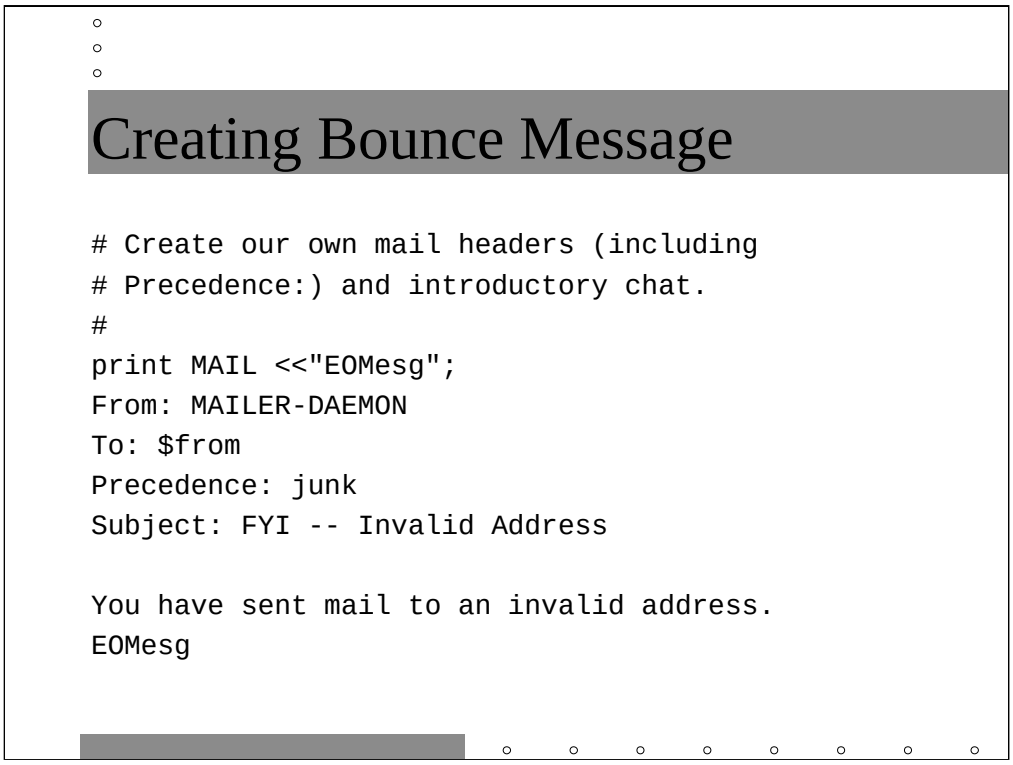
in Perl, Perl is actually invoking `/bin/sh` which then invokes Sendmail. If the From address has embedded shell commands in it, you're in for a world of hurt.

The above gibberish actually fires off a Sendmail process without invoking the shell. It's deep Perl magic, so don't freak if you don't understand what's going on.

The `open(MAIL, "|-")` line causes Perl to `fork()` (make a copy of the current process). The standard input of the new process (generally referred to as the *child*) is the other end of the `MAIL` file handle from the original process (the *parent*). The parent process gets back the process ID of the child while the child gets back `0` from the `open()`-- this is how the parent knows its the parent and the child knows its the child.

The child process then calls `exec()` to fire up Sendmail. `exec()` replaces the child Perl script with the Sendmail program but the Sendmail program inherits the standard input (and all other file descriptors) from the original child Perl script.

The parent just proceeds on as normal as if it had done the standard `open()` call.



Lesson #8 is how to create email messages on the fly.

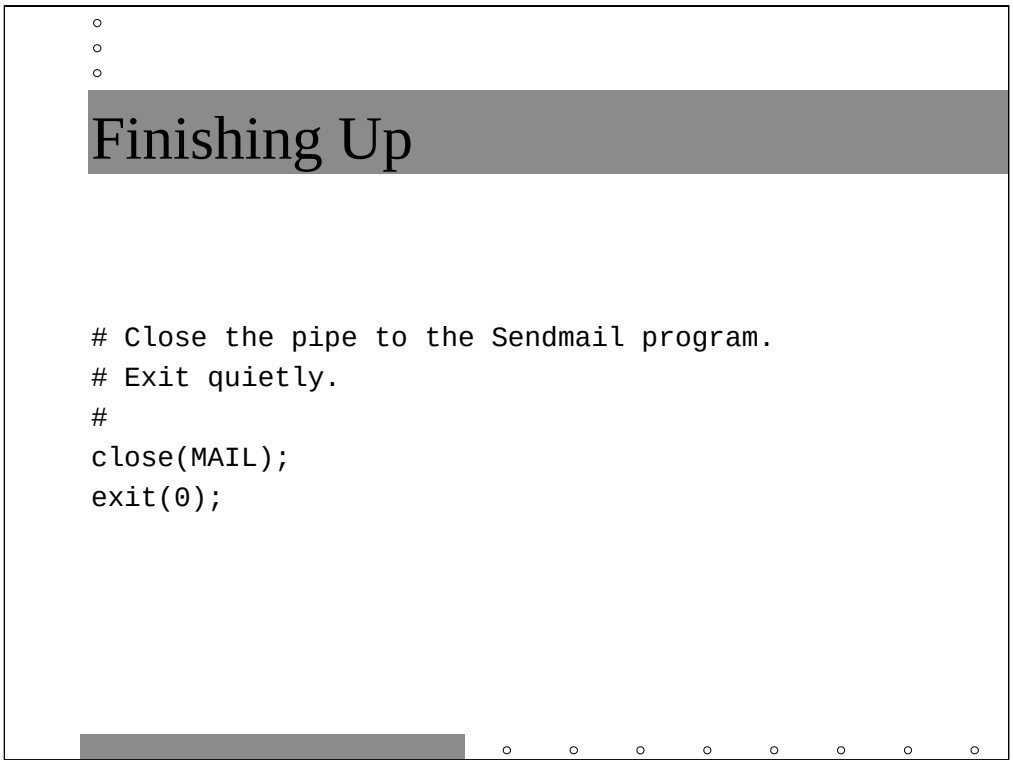
The parent process now starts composing a mail message and feeding it to the Sendmail process. We create a few headers (including our own `Precedence:` header) and then a blank line to separate the headers from the message body (that's *Lesson #4*).

Creating Bounce Message (cont.)

```
# Add new address if we have one.
# Include headers read so far and rest of msg.
#
print MAIL "Use new address: $ARGV[0]\n"
    if (length($ARGV[0]));
print MAIL "\nYour message returned below.\n\n";
print MAIL @header;
print MAIL "\n";
while (<STDIN>) { print MAIL; }
```

We add a line with the user's new address (if any). And then we include the original message (headers first, then the rest of the message body text) to send back to the sender of the original mail.

Lesson #9 is that it's always polite to return the original message sent to an auto-responder. Not everybody configures their mail program to keep a copy of all messages they send out (although they should!).



Finally we close the `MAIL` file descriptor and this causes the Sendmail program to fire off our response and exit. Then we exit ourselves with exit status `0` meaning that everything is OK.

Lesson #10 is always clean up after yourself.

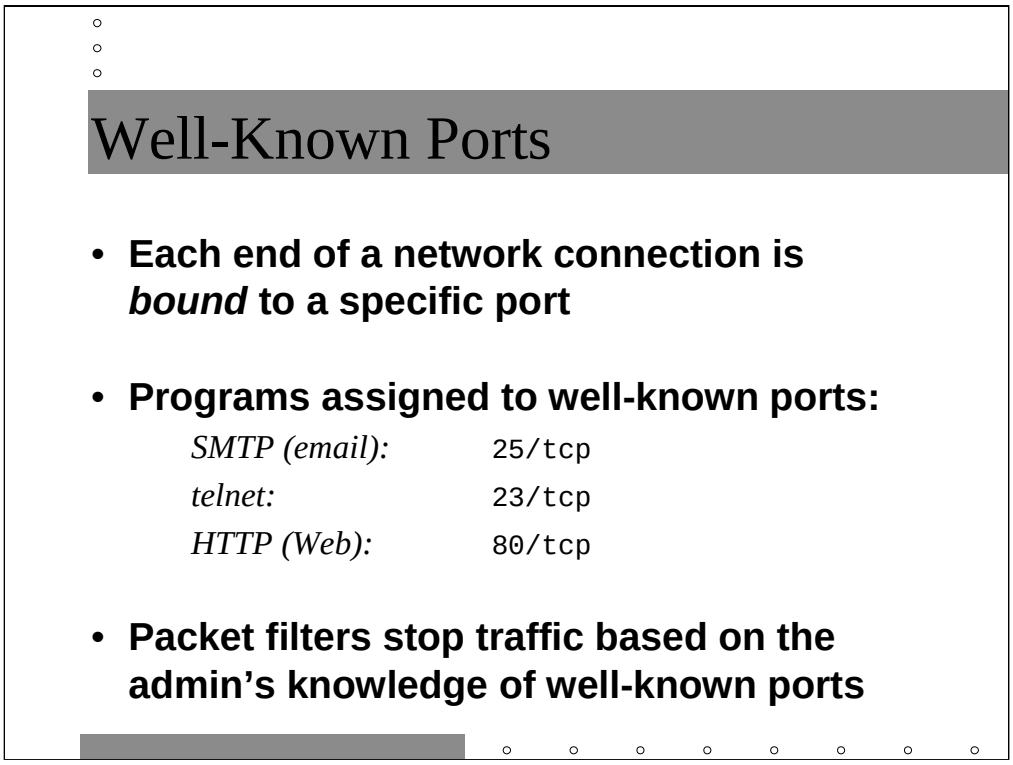
You knew there were going to be 10 lessons, didn't you?

Firewall Configuration

- ***Packet Filtering Basics***
- ***Cisco Access Lists***
- ***Examples!***

Earlier in the tutorial we talked about configuring split-horizon DNS to work in a firewalled environment, but we never really talked about how firewalls work and how this affects DNS and Sendmail.

We're going to talk about packet filtering, which is the old fashioned way of doing firewalls. Newer commercial firewall products still have a lot of features of old style packet filters.

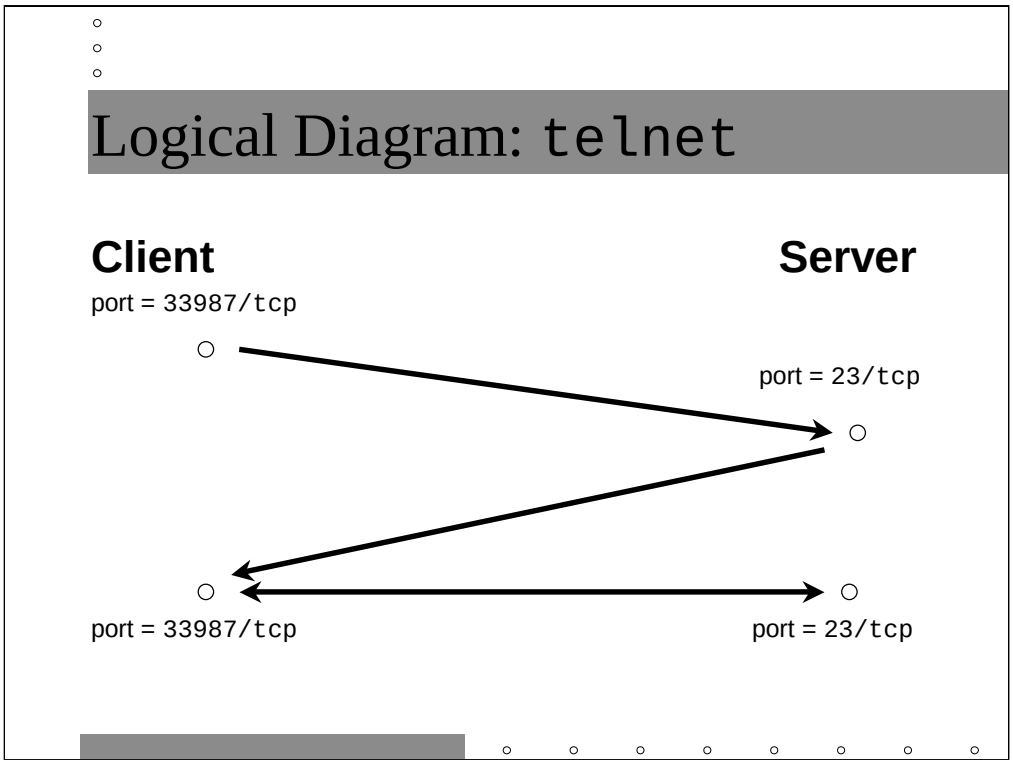


Two machines communicate across a network by directing packets of information at each other using unique IP addresses. However, any given machine may have dozens of simultaneous network connections happening at any given moment. A *network port* is a logical construct which allows a machine to keep multiple connections separate.

There are 16 bits of network port numbers-- TCP ports are distinct from UDP ports. Ports 0 through 1023 are reserved and not generally available to normal users. Furthermore, RFC1700 documents certain "well-known" ports that have been assigned to common network servers.

Well-known ports allow client software, e.g. the `telnet` program, to easily contact the appropriate server at the remote host. It would be impractical for a client to randomly probe the remote machine trying to find a server that will talk to it.

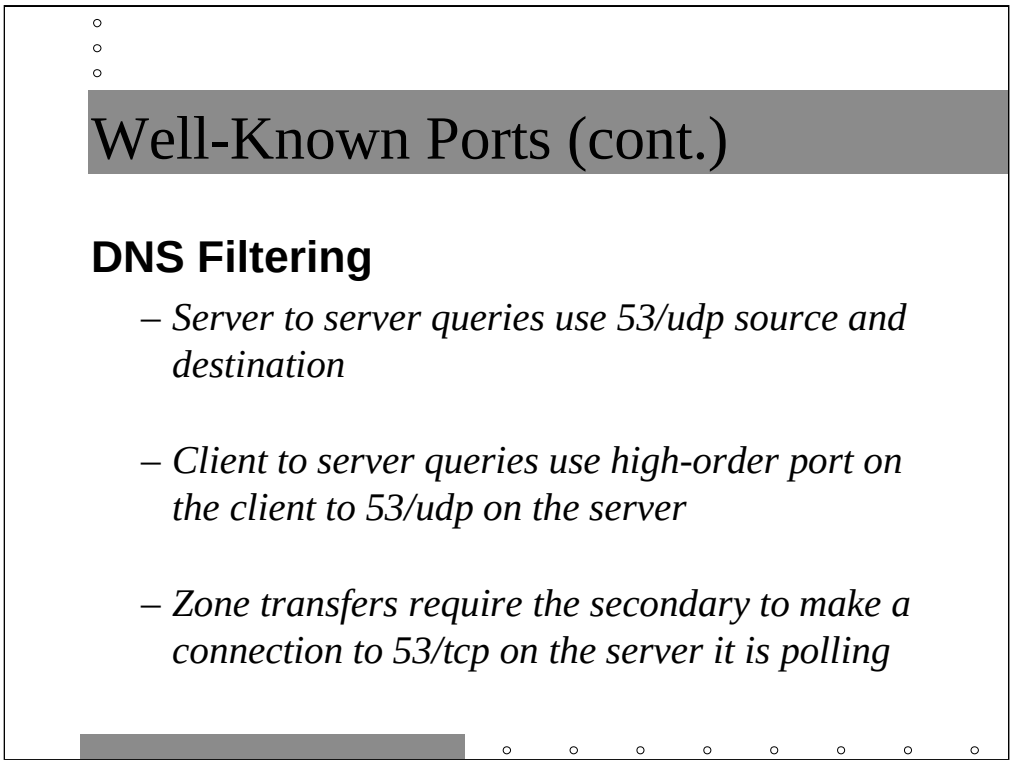
However, since the well-known ports constrain certain servers to listen on a fixed port number, an administrator can prevent access to a certain service by stopping packets targeted at a given network port. This is where packet filters come in.



In general, network clients grab a random unused port above 1023 (remember, ports lower than this are reserved). The client then constructs an initial packet with this random port number and its own IP address in the *source* portion of the packet. The *destination* is the IP address of the remote server and the well-known port appropriate for the given service. In this case, we're looking at a `telnet` client--the well-known port for this service is `23/tcp`.

The server sends back an acknowledgment packet using the source IP address and port from the initial packet. Remember in this case the server uses its own IP address and port `23/tcp` in as the *source address* and the IP address and random port selected by the client are in the *destination* fields.

The client now acknowledges the server's acknowledgment and the session proceeds.



When one DNS server talks to another, each one sources their packets with port `53/udp`*. There is a provision in the RFC to allow servers to use TCP instead of UDP to talk to each other, but you can operate fine without allowing this.

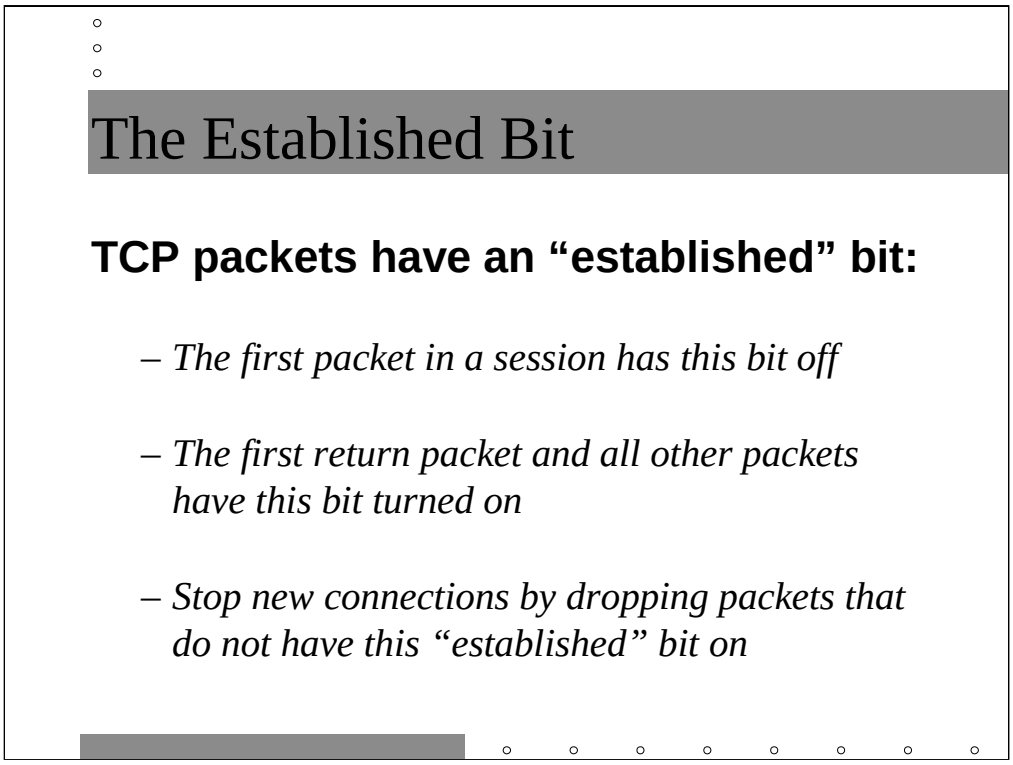
When your client asks for information from your local DNS server, the client uses a high-order UDP port and the server responds from `53/udp`. Note that your client doesn't have to talk to external servers in this way, and external clients don't have to talk to your server-- server to server queries suffice for regular DNS traffic.

For zone transfers to happen your remote secondaries have to be able to reach through your firewall and talk to your DNS servers on `53/tcp`. You should explicitly list the IP addresses of your remote secondaries and not allow just anybody to download your zone files, particularly if you don't do split-horizon DNS.

* One more word about server to server queries. BIND v8 has changed the default behavior for server to server queries: now you local nameserver picks a random port above 1023 for its connections. This breaks most existing firewalls. The

```
query-source address * port 53;
```

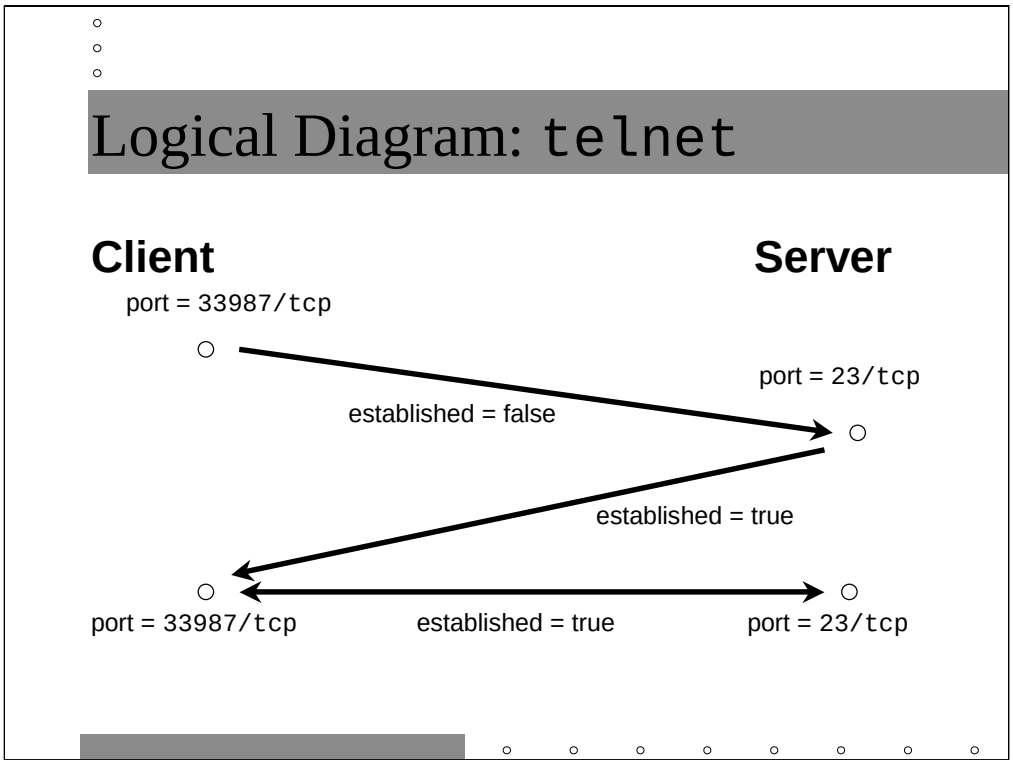
line in the `options` block of `named.conf` forces the old `53/udp` for source and destination behavior.



TCP connections also have a notion of a session that is “in progress” as opposed to the beginning of a new session. When a TCP client constructs the initial packet of a network connection, it sets a bit in the packet header to “false” indicating that no connection has been established yet. The first packet returned by the remote server has this bit set to “true” indicating that the packet is now part of a session in progress. All other packets sent by client and server also have this bit set to “true”.

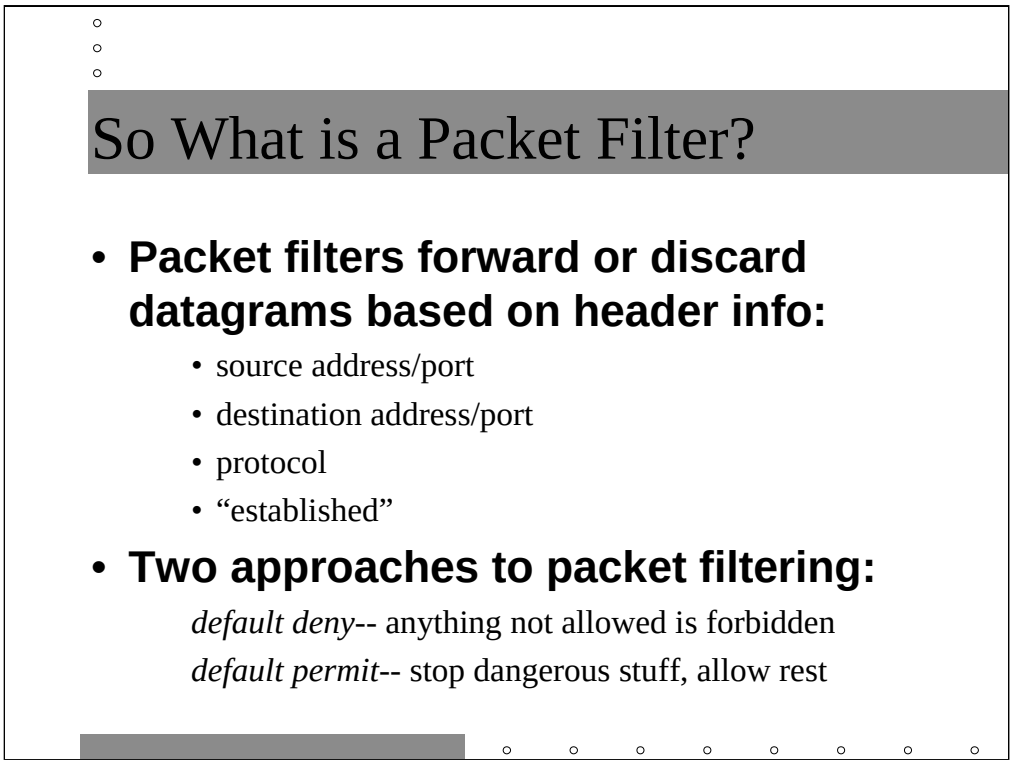
Thus, if you want to prevent outsiders from initiating connections, you need to stop packets that have this `established` bit set to false. The problem is that only TCP-based services use this bit. This is why UDP-based services are hard to filter and generally not allowed through firewalls-- there’s no way to tell whether this is a packet that’s part of a session started by a potential hacker on the outside or a legitimate user on the inside.

Note that the technical name for the `established` bit is the *SYN ACK bit* (sometimes just the *ACK bit*)-- the moniker “`established` bit” derives from the packet filtering syntax for Cisco routers as we will see later.



Here's our `telnet` example again. Note that the first packet from client to server has the established bit set to false. All subsequent packets have the bit set to true. If our packet filters can stop that first packet, then the client may never initiate a telnet session with the server.

Note that it is possible for an experienced network programmer to generate that same initial packet except that the established bit is set to "true". However, the server will have no record of an established session and will tear down the connection upon receiving the first packet from the client.



Classical packet filters only operate on information in the IP header of a packet. This limits their usefulness in certain circumstances. Things you can program into a packet filter include:

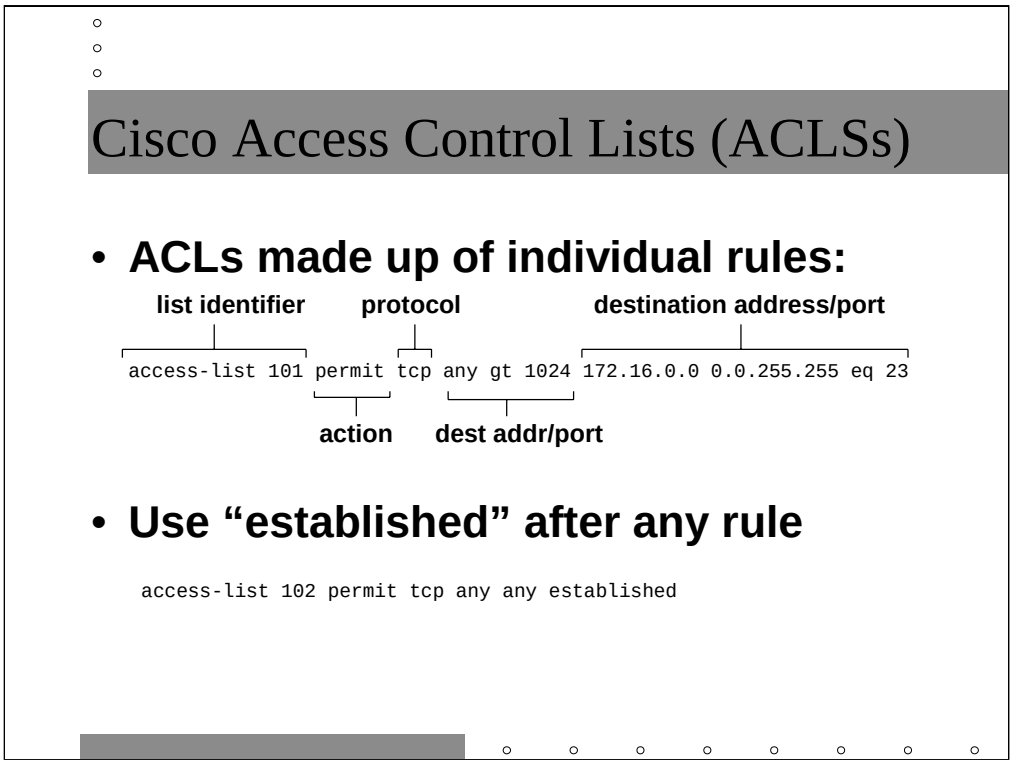
• Source and destination addresses. This includes both specific host addresses and network addresses using standard netmasking syntaxes. Most packet filters also allow you to specify "match all addresses".

• Source and destination ports, both specific ports and port ranges.

• The network protocol in use: TCP or UDP, ICMP, and special protocols like GRE. Again, most packet filters allow you to specify "match all protocols".

• The status of the established (ACK) bit

Philosophically, there are two approaches to packet filtering. In the *default deny* stance, the network administrator defines certain allowed services and then declares that all other services are not allowed. This is a more secure position to hold since new network protocols are being developed faster than any team of security experts is prepared to evaluate them. The alternative approach, *default permit*, has the network administrator disallowing obviously dangerous services (e.g. NFS and X Windows traffic) but by default allowing all other network traffic to proceed. This kind of filter can be used to stop obviously broken traffic from escaping from a testing lab and/or protect two sections of a large organization at the point where their networks connect.



Typical access lists can contain dozens or even hundreds of individual *access-list* statements. The largest access list your author has seen included over two thousand individual rules.

Each access list begins with an identifying number which groups individual rules into a single access list. Any number `100` or above may be used for Cisco extended access lists (lower numbers are used for basic access lists which have less functionality-- don’t bother).

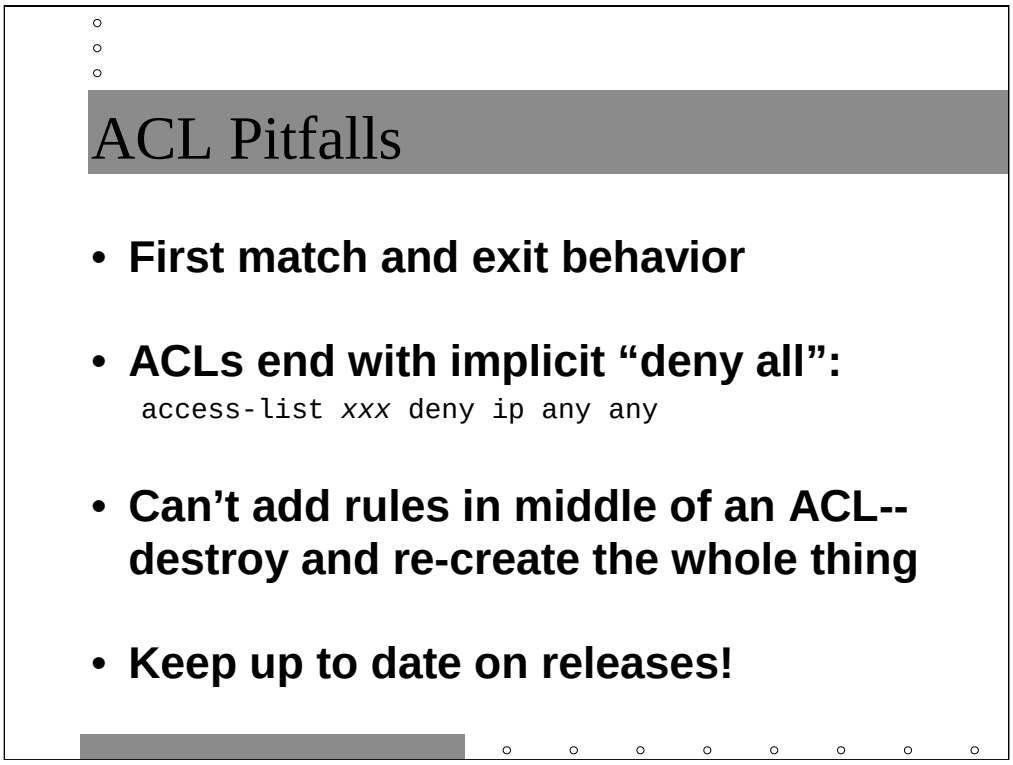
Next comes the action. `permit` means allow matching packets, `deny` means drop the offending packet and do not allow it to reach its destination.

Protocol can be `tcp`, `udp`, `icmp`, `gre`, etc. The protocol `ip` matches all protocols.

Next comes a source address followed by a port specifier and then a destination address and port specifier. Addresses can look like

`172.16.0.0 0.0.255.255`	*network address*
`host 172.16.1.1`	*individual host*
`any`	*match any addr*

Port specifiers are generally a comparison operator (e.g. `gt` for greater than, `lt` for less than, `eq` for equals) followed by a number. There is also a `range` operator for specifying an inclusive range of ports. The port specifier is always optional.



It is important to remember that the router stops processing access lists as soon as it finds a rule which matches. *Order is important!* Getting rules out of order can allow traffic that you thought was denied.

Cisco access lists are always in default deny mode. You can stop this behavior by putting

```
access-list xxx permit ip any any
```

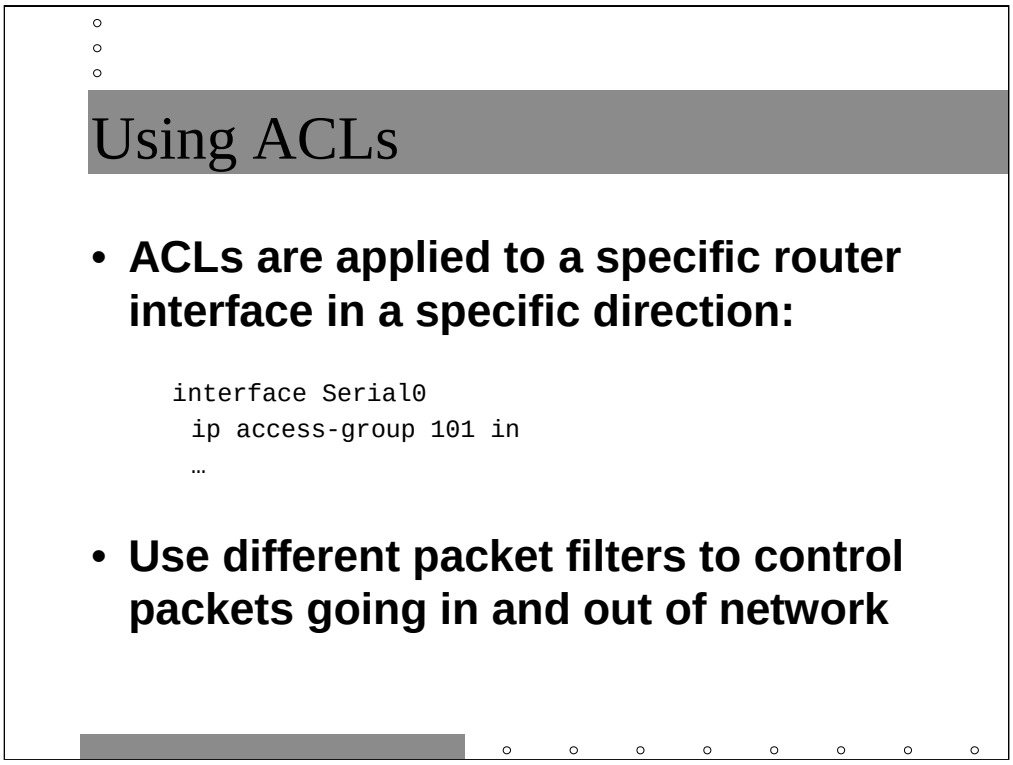
as the last explicit rule in any access list.

Cisco routers do not allow you to edit your access lists and insert rules in the middle. You must destroy and recreate a new access list to insert rules. I recommend putting each access list in a file with the following preamble:

```
no access-list 101
access-list 101 …
access-list 101…
```

You can then use `configure network` to read in this file as an atomic operation and update your ACL.

Over the years, many bugs have been reported against Cisco’s packet filtering functionality-- particularly related to handling of the established bit. Note that this is a good reason to trust Cisco’s packet filtering-- we think we’ve gotten all of the bugs out at this point. Still, make sure you keep up to date on the latest stable IOS release for your platform.

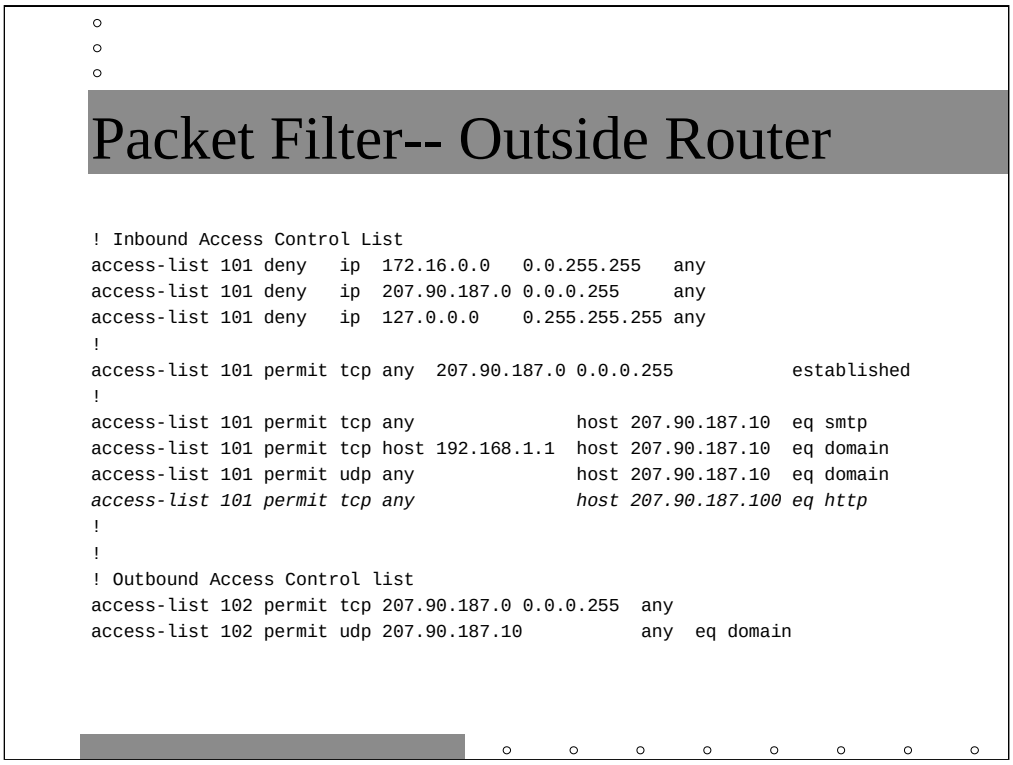


Once you define a complete access list, it must be assigned to a given interface. A given access list may be used by multiple interfaces and a given interface may use multiple access lists.

Note that access lists are applied to an interface in a given direction. If the direction is `in` then the access list is evaluated as packets are picked up off the wire, before making it into the router. If the direction is `out` then the access list is evaluated as the packets leave the router.

Most packet filtering routers use one access list to control packets coming from the "interior" network heading for the outside world and another to stop packets coming in from outside.

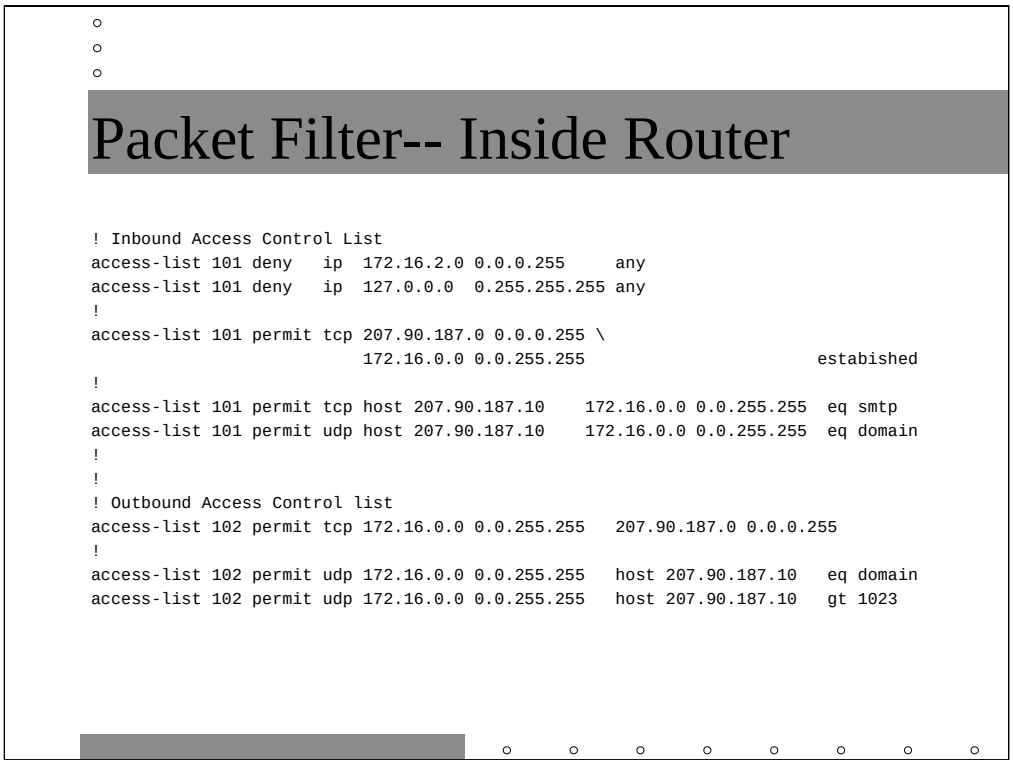
Note that inward packet filtering is a relatively new development (circa IOS version 10.0). In most cases, you can substitute an inbound filter on one interface with an outbound filter on another interface.



Now let's configure some packet filtering routers. The configuration above is for the Internet connected router on the "outside" of our DMZ. It's purpose is to allow outsiders to access the DNS and mail servers on the bastion and possibly other hosts such as your Web server. There are both "inbound" (which should literally be applied inbound on your ISP interface) and "outbound" (which can be applied "in" on the inside interface or "out" on the outside interface) packet filters.

The first two lines of the inbound filter stop spoofed packets (you should also apply a `no ip source-route` to the config for safety), including packets from the loopback network. You may also want to block all RFC1918 traffic at this point (instead of the 1918 network we're using for this example). The first next four lines permit access to DMZ services: email (SMTP), DNS zone transfers (from some external secondary at 192.168.1.1), DNS queries, and the Web (HTTP). The last two lines permit packets from `established` TCP sessions which happen on ports above 1023. Actually the two rules are redundant, not only because recent Cisco IOS versions allow you to combine the two rules, but also because people who try to initialize connections with packets marked as `established` should only get a reset from the remote server, but don't take chances.

The two lines of outbound filters permit outbound TCP connections (which also has the side-effect of letting your return packets from outside TCP connects from getting out) and allow server to server DNS queries only.

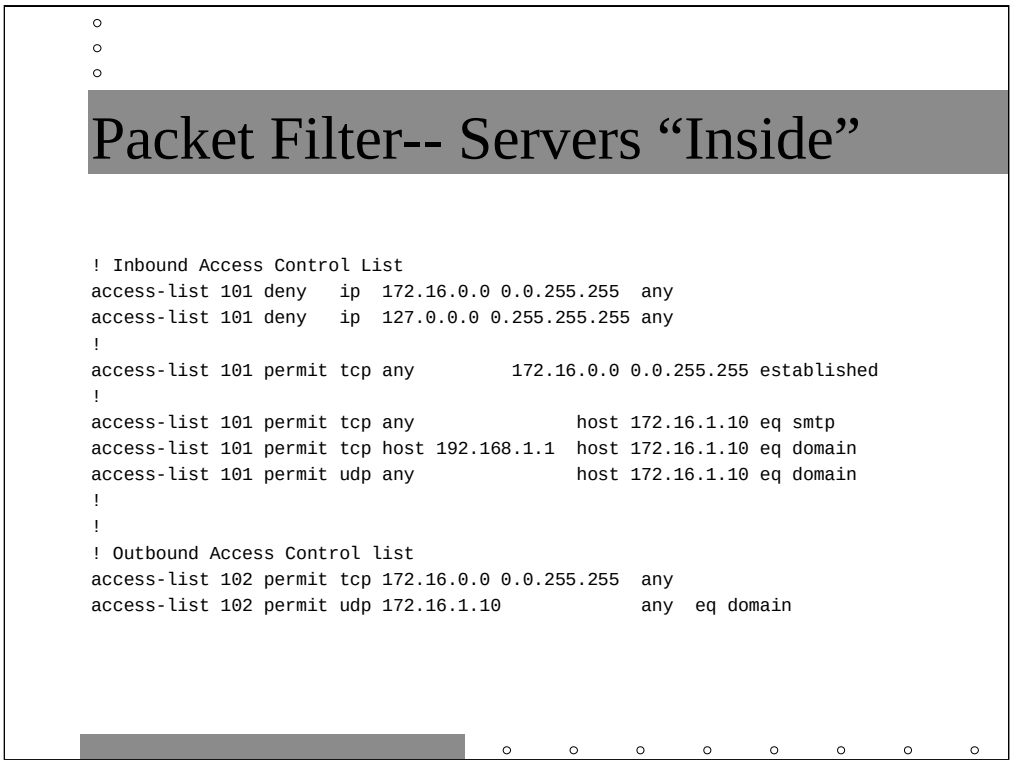


The packet filters for the inside router look fairly similar to the outside router filters.

Again the inbound filter first blocks spoofed packets. The filter also allows SMTP and DNS communications between the inside and outside routers. Finally packets from established sessions are permitted.

The outbound access list also permits traffic on `53/udp` between the inside and outside DNS servers, but also allows packets destined for UDP ports above 1023 to escape. This rule has to be there so that the resolver on the bastion host can make queries of the internal DNS server.

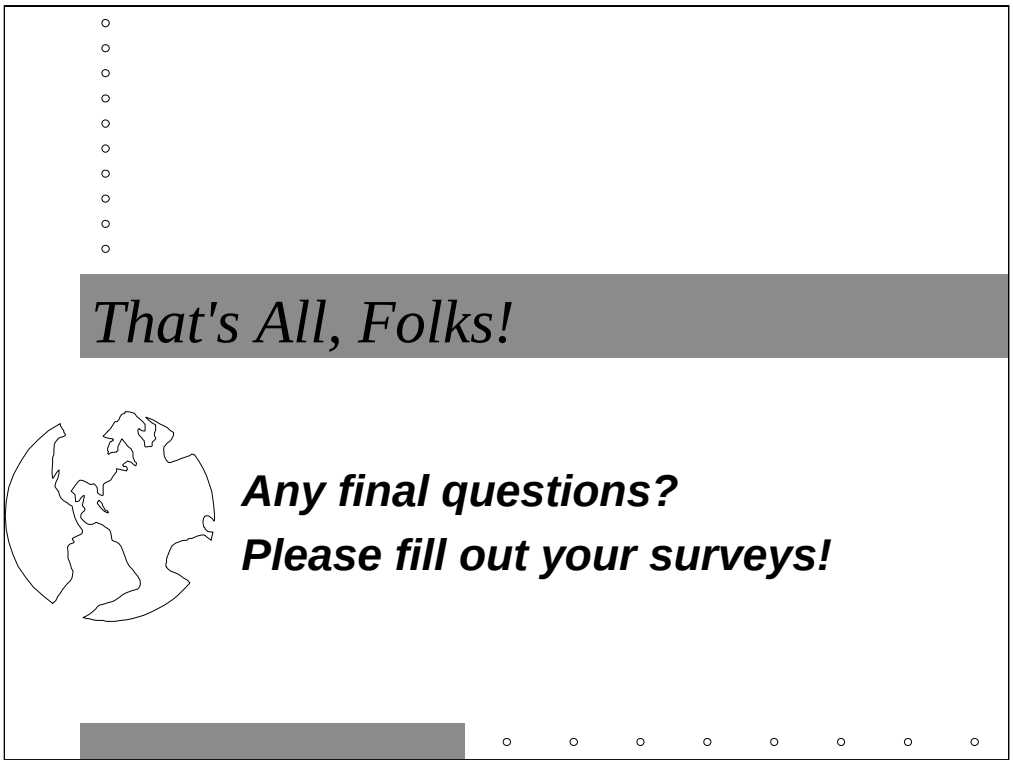
You will probably also need rules to allow your internal hosts to reach whatever proxy servers you're using on your DMZ to allow internal people to get out to the Web, etc.



This is an example of an access list for the case when you don’t have a DMZ network, just a single choke router between you and the Internet. Obviously, you would need real addresses inside instead of the RFC1918 addresses we're using here.

Again the inbound filter starts with anti-spoofing rules. Then we allow anybody to reach inward to our SMTP server (which should be running SMAP). The next lines permit zone transfers from a specific internal secondary and DNS server queries from external DNS servers. Finally we allow packets from established sessions.

Outbound we want to allow TCP packets for return connections and to allow insiders to use the Internet. We also allow server-to-server DNS queries to happen and allow our DNS responses to get out to the outside world.



This space intentionally left blank.