

WU-FTPD and Apache Security Basics

*Hal Pomeranz
Deer Run Associates*

1


All material in this course Copyright © Hal Pomeranz and Deer Run Associates, 2000-2002. All rights reserved.

Hal Pomeranz * Founder/CEO * hal@deer-run.com


Deer Run Associates * PO Box 20370 * Oakland, CA 94620-0370

+1 510-339-7740 (voice) * +1 510-339-3941 (fax)

<http://www.deer-run.com/>



WU-FTPD



2

This section deals with security configuration for the Washington University FTP daemon (WU-FTPD), a complex FTP server with many useful features. Unfortunately, all of this complexity comes at a cost—configuring WU-FTPD can be a bit tricky, and the daemon has had a long history of security problems.

The Server They Love to Hack

- WashU FTP daemon (WU-FTPD) is still the de facto standard on the Internet
- This makes it a target for attackers:
 - Buffer overflows
 - First reported format string attack
 - Trojan WU-FTPD source code
- Keep up-to-date on releases!
- Configuration mistakes also a major issue

3

The Washington University FTP daemon (WU-FTPD) is so named because it was originally developed at Washington University in St. Louis. About a decade ago, the Washington University FTP server (`wuarchive.wustl.edu`) was one of the largest and most heavily used anonymous FTP servers in the country (if not the world) and WU-FTPD was developed to meet the special needs of this service. It was widely adopted by other large anonymous FTP sites and has essentially become the de facto standard anonymous FTP server for large archive sites (though this is changing lately). Development of WU-FTPD has since moved on from Washington University and the latest software and updates can now be found at <http://www.wu-ftp.org>.

Since WU-FTPD is so prevalent and since WU-FTPD servers distribute much of the free software on the Internet today, attackers are highly motivated to poke holes in WU-FTPD. After all, if an attacker can compromise a major FTP server, they can put Trojan horses in the software that is regularly downloaded from that server and infect a vast number of machines. The fact that WU-FTPD is a large, complicated piece of software means that it's very likely that there are still security holes in the code that we haven't discovered yet.

Many of the security issues in WU-FTPD over the years have been buffer overflows of one sort or another. The first published "format string" type attack was found in WU-FTPD (see CERT advisory CA-2000-13). There was even an incident (back when the source was still maintained at `wuarchive`) where attackers managed to plant a Trojan horse in the WU-FTPD sources at the primary archive site, polluting several versions of the software at a large number of archive sites (see CERT Advisory CA-1994-07). It's critical to keep up-to-date on WU-FTPD releases. As of this writing, the current version is 2.6.2, released on Nov 30, 2001.

Aside from bugs in the software itself, many times FTP sites are compromised because the FTP server is simply not properly configured. The problem of secure configuration of WU-FTPD servers is the topic of the rest of this section.

Why Keep Using WU-FTPD?

- Useful WU-FTPD features that make added complexity worthwhile:
 - Strict controls for anonymous uploads
 - Restricted "guest" user access
- Neither feature is needed if you only need to allow anonymous downloads
- For "download only" FTP servers, think about using an alternative FTP daemon

4

If WU-FTPD has had such a significant history of security problems, and if we anticipate more simply due to the sheer complexity of the software, wouldn't it just be better to use a different FTP server? WU-FTPD does have two features that are not widely available in other FTP daemons:

1) WU-FTPD gives administrators a number of different configuration commands for tightly controlling files uploaded by anonymous FTP users. These configuration options are very much required to allow safe anonymous uploads, and other FTP daemons generally don't have them.

2) WU-FTPD allows the administrator to set up special "guest" access for certain users. "Guest" access means that when the user FTPs into the system, they are essentially `chroot()`ed into their home directories. Again, other FTP servers generally don't have this feature.

On the other hand, if you never plan to allow anonymous uploads and aren't making use of the "guest" user feature, then there is no advantage to running WU-FTPD. Aside from the potential security problems, WU-FTPD has a bigger memory footprint than many other FTP daemons, which can be a problem on heavily used FTP servers. So, you're probably better off picking another FTP server if you are not planning to make use of WU-FTPD's most useful features.

Alternatives

vsftpd (vsftpd.beasts.org)

- Small, fast, secure FTPD

ProFTPD (www.proftpd.net)

- Very Apache-like configuration file

NcFTPD (www.ncftpd.com)

- Commercial software
- Restrict non-anonymous users to homedirs

5

WU-FTP's "competitors" are listed here.

- `vsftpd` ("very small FTP daemon") is becoming more and more popular these days. It's a very "bare bones" FTP server, but it's fast and reliable and designed with security in mind. `vsftpd`'s small memory footprint makes it popular with high-volume servers.
- ProFTPD is also popular, primarily because the configuration file for ProFTPD looks a lot like an Apache config file. So, if you need a small FTP area to go with your Apache Web server, using ProFTPD seems natural.
- NcFTPD is actually a commercial software product that also claims to be a small, fast, reliable, secure FTP daemon. NcFTPD is notable because it's the only other FTP server that seems to have something close to WU-FTP's "guest" user functionality.

Note that many sites are using the BSD-style FTP daemon that was audited by the OpenBSD team (<http://www.openbsd.org>). A Linux port is available from <http://www.owlriver.com/projects/ftpd-BSD/>, but the Web page notes that they've stopped supporting this software and are urging people to switch to `vsftpd`.

The Rest of this Section

- Setting up anonymous FTP
- Allowing safe anonymous uploads
- Configuring "guest" user access

6

The rest of this section is divided into three parts:

- The first section deals with basic anonymous FTP security configuration for a machine that is allowing *downloads only*. It's important to get the basic anonymous FTP configuration correct before you even consider allowing uploads to the server.
- The next section then introduces the extra configuration steps necessary to allow the server to accept anonymous uploads securely.
- The last section covers how to make use of the WU-FTPD "guest" user feature. WU-FTPD server can allow anonymous access, guest access, or both at the same time, so many of the configuration commands in the previous sections will apply here as well.

If You're Not Doing Anon FTP

- Disable the FTP daemon entirely and use `ssh/scp/sftp` instead
- *Remove* `ftp` user from `passwd` and `shadow` files
- Remove `ftp` user's home directory

7

Note that if the system is not being used for anonymous or "guest" FTP access, then it probably shouldn't be running FTP. SSH is a more secure file transfer mechanism (albeit slower due to the encryption overhead).

Disable the FTP daemon by removing or commenting out the appropriate line in `inetd.conf` (or setting `disable = yes` in the `xinetd` config file for the FTP server, usually `/etc/xinetd.d/*ftp*`). If the password file contains an `ftp` user, remove it since this user is only needed for anonymous FTP servers. The home directory of the `ftp` user is the location of the anonymous FTP directory on the system. If this directory exists, you can remove it as well.

Note that RedHat systems have an RPM called `anonftp` that may be installed on your system— removing this RPM automatically removes the anonymous FTP directory (`/var/ftp` usually).

Anonymous FTP

8

Secure anonymous FTP configuration is a combination of good directory setup and good server configuration. We'll tackle both of these issues in this section.

Creating the Anon FTP Dir

- Most OS vendors ship a pre-configured FTP dir or provide a script to make one
- Vendor config not always perfect:
 - `root:root` ownership on all files and dirs
 - All dirs except "pub" should be mode 111
 - All files mode 444, shared libs mode 555
 - Strip unneeded info from `passwd/group` files
 - "bin" dir should only contain `ls` program

9

These days, most vendors provide either a pre-configured FTP directory (ala RedHat's `anonftp` RPM) or at least provide a script for creating this directory. This is extremely useful, because it's always something of a hassle figuring out the vendor-specific device files and configuration files that are required in the anon FTP hierarchy. Usually you can find the vendor's script by looking at the `ftpd` manual page on the system.

Unfortunately, this vendor-provided material doesn't always give you an FTP directory that is as secure as it could be. It's worthwhile to check all of the permissions, etc. for files in the FTP directory before allowing anonymous access. Since the daemon will be running as user "ftp" after the user logs in for the anonymous FTP session, all files and directories in the anon FTP area should be owned by `root` to prevent the anonymous FTP user from tampering with them. Files should generally be mode 444 (except shared libraries which must be mode 555) and directories mode 111 (except the "pub" directory where download files are placed– this needs to be mode 555).

There only needs to be one program in the "bin" directory, namely the `ls` program. Some vendors like RedHat will put other programs like `tar`, `gzip`, and `compress` in the FTP "bin" directory for on-the-fly data conversions– probably not a good idea in general. If you cannot statically link the `ls` program, then you will need to use the `ldd` program to determine what shared libraries the `ls` binary needs, and copy these shared libs into the "lib" or "usr/lib" (depending on the OS) directory in the anonymous FTP area.

Most administrators will put stub `passwd` and `group` files into the "etc" directory so that the `ls` program can display human readable usernames and group names. It's a good idea to null out all data fields in these files except the username/groupname and the UID/GID, and also to delete any user accounts from the `passwd` file that won't be owning files in the FTP area. Just in case somebody does manage to download the `passwd` file out of the FTP area, you want them to get as little extra info as possible. *Never* put a `shadow` file in the anon FTP "etc" directory.

WU-FTP Config Files

- Usually located in `/etc` or `/etc/ftpd`:
 - `ftpusers` – users who are *not* allowed to FTP in
 - `ftpgroups`, `ftphosts` – other access controls
 - `ftpconversions` – "on-the-fly" `tar/gzip`
 - `ftppaccess` – the primary/critical config file
- For purposes of this discussion, we'll focus primarily on `ftppaccess` file configuration

10

Once you're satisfied with the configuration of the anonymous FTP directory, it's time to turn your attention to configuring WU-FTPD itself. The daemon will look for its configuration files in a particular directory chosen at compile time— usually `/etc` or `/etc/ftpd`.

Like almost all FTP daemons, WU-FTPD will look at the `ftpusers` file for a list of users who *are not* allowed to FTP into the system. "root" and the other non-user accounts on the system (like `bin`, `daemon`, `nobody`, etc.) should probably be listed here. Be careful not to put the "ftp" user in this file or anonymous FTP will not work! `ftpgroups` and `ftphosts` can control access by group membership or by IP address/network. However, WU-FTPD allows the administrator similar access control in the central `ftppaccess` config file, so there doesn't seem to be much point in using multiple separate files here.

`ftpconversions` allows the administrator to have the daemon automatically `tar/compress/gzip` certain data when it is downloaded by a user. This can sometimes be useful if your FTP directory contains a large source tree with lots of separate source files (rather than a big `*.tgz` file containing all the sources)— if a user requests part of the tree, the FTP server will just `tar` the directory structure up, `gzip` it, and fire it off to the user. Still, it seems like the FTP server automatically triggering commands in this fashion might open up a security hole. It's probably best to just `tar` and `gzip` your source files manually and make the `*.tgz` files available from your FTP site. Then you can remove the `ftpconversions` file entirely.

The real action happens in the `ftppaccess` file, and that's what we'll be looking at for the rest of this section.

Basic Server Config Items

```
hostname      ftp.sysiphus.com
email         ftpadmin@sysiphus.com

loginfails    3
passwd-check  rfc822    warn
defumask      022

timeout       RFC931    0
```

11

`hostname` is the name the server should use when printing banner messages, etc. If `ftp.yourdomain.com` is actually an alias pointing at a machine with a different "real" name, you probably will want to set this parameter to hide the actual machine name (also useful when `ftp.yourdomain.com` points at a cluster of machines behind a load-balancing system). `email` is the email address of the server admin— this value is also used in various informational messages.

`loginfails` is the number of times that normal or "guest" users may attempt to log in before the FTP connection is dropped by the server. `passwd-check`, on the other hand, applies to anonymous FTP users only. `passwd-check` sets how strict you want to be when checking the email address the anonymous user is supposed to enter as their password. "rfc822" here means that WU-FTPD will validate that the string entered looks like a "standard" email address as defined by this RFC (typically "user@some.dom"). "warn" means that the user gets a short warning message if what they entered doesn't look correct, but will be allowed access anyway. You could set this to "enforce" instead of "warn" to kick off people who don't enter legitimate looking email addresses, but you'll find that many people are simply ignorant of their actual email address and you'll probably end up blocking some Web browser based anonymous FTP sessions (which may not supply something that looks like an email address).

`defumask` is the default `umask` value to use when users upload files or create directories on the FTP server. Typically this is only used when you are allowing normal or "guest" FTP access. In the case of anonymous uploads, we will be explicitly setting the permissions on uploaded files for security reasons.

By default on each new connection WU-FTPD will attempt to connect back to the `ident` service running on the remote system. Most sites block `ident` access or simply don't run the service, so this is just a waste of time. Setting the `timeout` value for RFC931 type queries (RFC931 defines the `ident` protocol) to zero stops WU-FTPD from even trying to do an `ident` connection.

Basic Access Control

- "Standard" `ftppaccess` file allows regular, "guest", and anonymous access:

```
class all real,guest,anonymous *
```

- Allow anonymous access only:

```
class anon anonymous *
```

- Further restrict by IP address if desired:

```
class anon anonymous 192.168.0.0/16
```

12

The next step is to define what "classes" of FTP access are going to be allowed. The "standard" `ftppaccess` file that ships with WU-FTPD allows "real" users (normal logins by folks with accounts in the `passwd` file), "guest" access (normal access but user is `chroot()`ed to their homedir), and "anonymous" (anonymous FTP sessions in `chroot()`ed directory). If the machine is just going to be an anonymous FTP server and you don't want normal users FTPing into the system, then simply change the "class" line so that only "anonymous" access is allowed. Any class of access not explicitly listed in the `ftppaccess` file will be blocked.

The last field of the "class" directive is actually a list of address "globs" that say which IP addresses are allowed access for this particular class. Internet anonymous FTP servers will generally use "*" here since they want to allow access from any IP address. However, for an intranet FTP server, you may want to restrict access to only your local IP address ranges. You can either specify `<addr>:<mask>` (`192.168.0.0:255.255.0.0`) or `<addr>/<CIDR>` (`192.168.0.0/16`) and you can specify multiple different network "globs" separated by spaces. Host names and domain name globs ("`*.domain.com`") are allowed, but stick with IP addresses because DNS can be spoofed. You can also specify exceptions with "!", e.g. "`!10.1.1.0/24`" would mean "allow access from everywhere *except* network `10.1.1.0`".

Of course, this functionality is a direct overlap with the functionality provided by TCP Wrappers. Having this functionality in the FTP daemon, however, means less connection latency and less resource utilization— this is a good thing on heavily used FTP servers.

Class Limits

- Can also restrict access by day/time:

```
limit anon 25 Any /etc/msgs/toomany
limit anon 50 SaSu|Any2000-0600 /etc/msgs/toomany
```

- Useful for saving system resources and network bandwidth during regular hours
- See also `file-limit`, `data-limit`, `limit-time`, and `host-limit` options

13

Access for certain classes can be restricted by time of day and/or day of week. In the example on the slide, the "anon" class for anonymous access (which we defined on the previous slide) is allowed 25 simultaneous sessions any time and on any day of the week. Outside of "normal" hours— Saturdays, Sundays, and any day between 8pm and 6am local time— up to 50 users are allowed. If these thresholds are exceeded, the message in the file `/etc/messages/toomany` is displayed and the connection is dropped. Note that the administrator has to create the message file by hand, and the filename is relative to the directory the anonymous FTP server will `chroot()` to— so if your anon FTP directory is `/var/ftp`, you need to create `/var/ftp/etc/msgs/toomany`.

The `limit` control is sort of a gross restriction on the amount of simultaneous anonymous FTP traffic on your server. WU-FTP also has `file-limit` and `data-limit` options that can be used to control the number of files or bytes the user can transfer in a single session. `limit-time` controls the maximum duration of any single session.

`host-limit` limits the number of simultaneous connections from each unique IP address. This one can be tricky because many large sites now filter all of their Internet access through a network address translation device or a proxy server that makes all connections appear to come from a single IP address. Be careful using this option.

Basic Security Policy

<code>overwrite</code>	<code>no</code>	<code>anonymous</code>
<code>delete</code>	<code>no</code>	<code>anonymous</code>
<code>rename</code>	<code>no</code>	<code>anonymous</code>
<code>chmod</code>	<code>no</code>	<code>anonymous</code>
<code>umask</code>	<code>no</code>	<code>anonymous</code>
<code>noretrieve</code>	<code>/var/ftp</code>	
<code>allow-retrieve</code>	<code>/var/ftp/pub</code>	

14

After setting up the anonymous access class and any limits appropriate for anonymous access, the next step is to tightly restrict what the anonymous user can do. The permissions/ownerships on the anonymous FTP directory itself already tightly restrict what anonymous users can do, but reinforcing those restrictions in the `ftpdaccess` file is also a good idea. For one thing, restricting access via multiple mechanisms means that if you make a mistake in one area (like directory or file permissions), the access controls set elsewhere will still prevent malicious users from doing the wrong thing. Also, having these settings in the `ftpdaccess` file clearly documents to other administrators what the "correct" level of access is for anonymous users.

The first set of configuration directives shown on the slide prevents anonymous users from triggering various commands (note the settings shown here are the WU-FTPD defaults for anonymous users, but it's probably best to set them explicitly—again for "self documentation" purposes alone if for no other reason). Anonymous users are not allowed to `overwrite`, `delete`, or `rename` files in the anonymous FTP area—access that should also be blocked by the fact that files and directories in this area should only be writable by the `root` user. Anonymous users are also not allowed to `chmod` files or to change their default `umask` as set by the FTP server (c.f., the `defumask` command we saw earlier).

Anonymous users should only be allowed to download files from the "pub" directory where downloadable data is kept. You don't want anonymous users to be able to download files in the "bin", "lib", "etc" directories because this could give a potential attacker information about your system (the "ls" binary indicates the OS type and possibly version and patch level, the "passwd" file contains user info, etc.). Here we're using `noretrieve` to define a default policy that downloads are not allowed in the entire anonymous FTP area, and then using `allow-retrieve` to create a specific exception for the "pub" directory where the downloadable files reside. If you had multiple download directories under `/var/ftp`, then multiple `allow-retrieve` statements could be used.

Controlling File Names

- Prevent malicious users from exploiting strange file names:

```
path-filter anonymous /etc/messages/path
  ^[-A-Za-z0-9._[:space:]]*$
  ^\.  ^-  ^[:space:]
```

- First regexp is allowed file names, trailing patterns specify exceptions
- Becomes particularly important when we start allowing uploads

15

Note that the `path-filter` directive shown here must be written on a single long line in the `ftppaccess` file!

It is also a good idea to restrict what file names anonymous users are allowed to access. This becomes very important when we start allowing anonymous users to upload files, but it can even make a difference on a "downloads only" type FTP server. For example, some sites put special configuration information (typically used by system processes other than the FTP server) into "dot files" (files whose names begin with a period) in various directories under "pub" (this is *not* recommended as good practice, but it happens). It's likely that the system administrators wouldn't want anonymous users to be able to download these files.

The syntax of the `path-filter` directive is rather funky. The first argument is the access class the filter applies to (you could have one filter for anonymous users and a less restrictive one for real or guest users), and the next argument is the path name to a file containing a message that gets displayed to the user when the filtering rules are violated (again, this file must exist under the anonymous FTP directory). The first regular expression on the line says what file names are allowed— here we're saying that files names can contain only letters, numbers, dashes ("-"), periods, underscores ("_"), and spaces (if your FTP server has a lot of Windows or Mac users, allowing spaces in file names is usually expected). Any remaining regular expressions on the line are file names that are explicitly not allowed— here we're not allowing files to begin with a period, dash, or space.

The "[:space:]" syntax is necessary because the primitive WU-FTPD config file parsing code breaks up the arguments to the `path-filter` directive by whitespace (and doesn't understand quoting or backwhacking special characters). If you put a literal space into any of the regular expressions, then the config file parsing routine in WU-FTPD would treat the regular expression as two different regexps. Thanks to Lee Brotzman (leb@gms . com) for bringing up this problem and pointing out the solution.

Logging

- Can choose to log different sorts of info:
logfile /var/log/xferlog
#log commands anonymous
log transfers anonymous
log security anonymous
- Logging commands not that useful for anon FTP– we disabled most cmds earlier
- xferlog can still get huge quickly– rotate file on a regular basis

16

WU-FTPD allows for a fairly high level of control over what gets logged and where that logging happens. Logging commands logs all standard FTP commands triggered by members of the listed access class. Since we've already disabled most commands for the anonymous access class, logging commands may not be that useful. On the other hand, maybe you want to log these commands just in case somebody makes a mistake in the configuration file that ends up allowing anonymous users to use them. In the example above we're commenting out the "log commands anonymous" line so that these entries will not be logged.

Logging transfers causes WU-FTPD to log all uploads and downloads (file name, bytes transferred, duration, etc.)– these log entries also contain either the username (for real and guest access) or the email address (for anonymous access) of the user transferring the data. Logging security logs all violations of the security policy set up in the ftpaccess file (e.g., trying to access file names blocked by path-filter or trying to download files from a "noretrieve" directory).

By default, WU-FTPD logs all of this information to a file called /var/log/xferlog. This pathname can be changed with the logfile directive. WU-FTPD is also capable of logging this information via Syslog: "log syslog" says to log via Syslog only, "log syslog+xferlog" says to log both to Syslog and to the xferlog. Logging to Syslog is useful if you want to send your FTP server logs to some other central loghost for processing.

On a busy anonymous FTP server, xferlog grows extremely quickly. Be sure you have a cron job that rotates this log file on a regular basis.

Informational Messages

```
greeting  text    FTP server ready.
signoff   text    Adios!
stat      terse

message   /etc/msgs/welcome      login
message   .message             cwd=*

readme    README*              login
readme    README*              cwd=*
```

17

WU-FTPD also gives the administrator a lot of control over what messages are displayed to the remote user. `greeting` is the message that is displayed before the FTP login prompt— usually this message displays both the hostname and the FTP server version. You can hide this information by changing the greeting string as shown here. Note that you might want the message to be a warning banner like "Authorized uses only. All access may be monitored and reported." (this is particularly true on "intranet" FTP servers). `signoff` is what gets displayed after the user closes the FTP connection— the WU-FTPD default is to display the hostname and the amount of data transferred during the session, which may be more information than you want to give away. The `STAT` command in the FTP protocol allows the remote user to get information about the FTP server and the current session. "`stat terse`" causes WU-FTPD to report only the connection status (up or down)— "`stat full`" (the default) would report the server hostname, daemon version, and connection status, and "`stat brief`" only reports the hostname and connection status.

`message` and `readme` are used to display informational messages during the FTP session. `message` specifies a file whose contents are displayed when a particular condition occurs— like a welcome message at `login` that defines the rules for anonymous FTP access and how to get help (for anonymous access, this file would have to exist under the anonymous FTP directory). The second message directive says to display messages whenever the user changes directories— if the file named `.message` doesn't exist, then nothing is displayed. Files displayed via `message` may contain special escape sequences that will be automatically expanded by the FTP server— like `%E` which expands to the email address of the server admin that we set earlier with the `email` directive. For a complete list of escape sequences, check out the `ftppaccess` manual page.

The `readme` directive doesn't display the contents of the matching files. Instead, WU-FTPD prints an informational message that tells the remote user the last time the matching file was updated ("Please read the file `README`— it was last updated on ..."). The user is expected to do something like "`get README -`" to actually view the contents of the file.

Running the FTP Server

- Need an `ftp` user in `/etc/passwd`:

```
ftp:x:99:99:Anon FTP:/var/ftp:/dev/null
```

- Also need `inetd.conf` entry:

```
ftp      stream tcp nowait root
        /usr/local/bin/ftpd      ftpd -a -l
```

- Note that WU-FTPD is capable of running as a stand-alone daemon if desired

18

Once the anonymous FTP area is set up and the `ftpaccess` file fully configured, there are still a couple of steps necessary to actually allow anonymous access and start the FTP daemon itself. Generally it's a good idea to get all of the directory and `ftpaccess` security configuration done first *before* you actually enable anonymous access and start WU-FTPD.

To allow anonymous access, there must be a user called "ftp" in the system `passwd` and `shadow` files. Use a unique UID and GID for the `ftp` user that are not shared with any other user or process on the system. This user should have their password entry in the shadow file blocked (use "LOCK" or "*" or some other invalid password string) and should have an invalid shell field in `/etc/passwd` to prevent normal logins. Note that many FTP daemons require the invalid shell for the `ftp` user to be listed in `/etc/shells`, but this is *not* necessary for WU-FTPD. The home directory for the `ftp` user is where the FTP daemon with `chroot()` itself when a remote user logs in as `anonymous` or `ftp`.

You will probably need to replace the `inetd.conf` entry for the vendor's FTP daemon with a new entry that fires up WU-FTPD. The "-a" flag tells WU-FTPD to use the `ftpaccess` file (this is *not* the default for some reason) and "-l" causes each new FTP session to be logged via Syslog (similar to the logging done by TCP Wrappers). Note that we are not using TCP Wrappers for access control, since this can be handled by the daemon itself via the `ftpaccess` configuration. On the other hand, some sites prefer to do IP address based access controls via TCP Wrappers just so all of their access control policy is centralized in the `hosts.allow` and `hosts.deny` files. Of course, there's a performance hit for using TCP Wrappers.

For sites that are really concerned about performance, WU-FTPD can be run as a stand-alone daemon rather than via `inetd` or `xinetd`. For more information, see the "-S" (standalone) option in the `ftpd` manual page. There is also a "-r" flag that can force the daemon to `chroot()` itself as soon as it starts executing.

Anonymous Uploads

19

Once you've got the basic anonymous FTP server set up and working for downloads only, spend some time testing the configuration. Make sure that all of the various security settings like "noretrieve" and "path-filter" are working and that logging is happening like you expect. Once you're satisfied that the basic anonymous configuration is working properly you can go ahead and start configuring upload capability.

CAUTION: Anon Uploads

- Don't allow downloads of uploaded files
 - *You'll become a "warez" site*
- No executable or set-UID uploads
 - *Program could be triggered by different exploit*
- Restrict names of uploaded files
 - *File called "-r" is a problem when you "rm *"*
- Don't allow uploaded files to be overwritten
 - *Potentially serious confusion will result*

20

Anonymous uploads are a danger that really should be avoided if at all possible. Consider allowing "guest" access instead (more on this in the next section) because guest users at least have to enter a valid password and are each given their own private directory for accessing files.

If you do need to allow anonymous uploads, there are a few important rules to remember:

- Never allow uploaded files to be downloaded by anonymous users. If you permit downloads in the upload directory, then your upload area becomes a "warez" (pronounced "wares") site where anybody can distribute pirated software, pornography, or attack tools. Also you wouldn't want to allow data from one customer to be downloaded by other users.
- Always strip the execute and set-UID/set-GID bits on uploaded files. Suppose your Web server and FTP server were the same machine (a bad idea in general). A buffer overflow in the Web server only gives the attacker access with the privileges of the Web server user. But if the attacker were able to upload a set-UID `root` program via FTP and then execute it via the Web server buffer overflow, the attacker has `root` access on your system.
- Don't allow attackers to upload files with strange file names. Files with names like `-i` or file names that begin with spaces can be hard to remove, and truly malicious file names can be constructed that contain embedded shell commands that may be triggered by automated scripts.
- Also don't allow uploaded files to be overwritten by other users. You wouldn't want the crash dump from one customer to get whacked by another customer's data. You wouldn't want a malicious user to plant a Trojan horse copy of a file you're expecting from some other trusted source.

Safe Upload Configuration (1)

- Already getting some help in `ftppaccess`:

```
overwrite          no    anonymous
delete             no    anonymous
rename             no    anonymous
[...]
noretrieve         /var/ftp
allow-retrieve     /var/ftp/pub
[...]
path-filter anonymous /etc/msgs/path ...
```

- This is why it is critical to get anon FTP configuration right before doing uploads

21

The permissions we've already set up on the anonymous FTP directory, plus the configuration commands in the `ftppaccess` file are already helping to protect us from many of the dangers listed on the previous slide. Once again, having a "layered" security configuration with lots of redundancy helps prevent simple individual mistakes from becoming a big problem.

We want to prevent uploaded data from being downloaded. Well, the "noretrieve" directive prevents downloads by default in our anonymous FTP area. Restricting the file names of uploaded files is handled by the "path-filter" directive we set up earlier. The anonymous user is already not allowed to overwrite, delete, or rename files, so this should prevent uploaded files from being clobbered by other users.

The `ftppaccess` command which actually allows uploads gives us even more security and control, as we'll see on the next slide...

Safe Upload Configuration (2)

- Now add the `upload` directive:

```
upload /var/ftp * no
upload /var/ftp /incoming yes
root root 400 nodirs
```

- Uploaded files owned by `root`, mode 400:
 - Can't be read/downloaded by `ftp` user
 - Can't be overwritten by `ftp` user
 - Will never be executable or set-UID
- Configuration redundancy is a *Good Thing*

22

Note: the second `upload` directive shown above should appear as a single long line in the `ftppaccess` file.

The `upload` directive is used to control upload access in a directory. First specify the directory that the `upload` command applies to, then specify the file and subdirectory elements within that directory. Here we are first defining a default policy that uploads are not allowed at all in the anonymous FTP directory. Actually, it's probably a good idea to add this first line even in the `ftppaccess` file for "download only" FTP servers, just in case.

The next `upload` directive says that uploads are allowed in the `/incoming` directory under `/var/ftp`. "nodirs" means that files can be uploaded but anonymous users are not allowed to create their own subdirectories under the `/incoming` directory. When files are uploaded, the resulting file will be owned by `root`, group owner `root`, and have mode 400 (read-only by owner). Not only does this automatically strip off the execute and set-UID/set-GID bits, but it also accomplishes many of our other security goals. Since the uploaded files will only be readable by the `root` user, they can't be downloaded, overwritten, or deleted by anonymous users (user `ftp`). Once again, redundant security configuration is a good thing.

Of course, there needs to be some mechanism to allow authorized users to retrieve (the now `root` owned) uploaded files from your `/incoming` directory. You could give regular users on the system `sudo` access to copy files out of this directory, or simply have an automated job that copies files out of `/incoming` and into some publicly accessible area (possibly on another system inside of the corporate firewall). It's also a good idea to automatically delete uploaded files after some period of time (24 or 48 hours) just so the directory doesn't become too cluttered.

Now Create Upload Directory

- Upload area must be writable by `ftp` user—set "sticky" as an extra precaution:

```
# cd /var/ftp
# mkdir incoming
# chown ftp:root incoming
# chmod 1300 incoming
```

- Directory mode 1300 means no dir listings—possible problem for GUI-based FTP clients

23

Only after setting up the upload directives in the `ftppass` file should you create the actual `/incoming` directory. This directory needs to be writable by the `ftp` user so that anonymous FTP users can create new files in this directory.

The permissions used here are somewhat interesting. We're setting the "sticky bit" on the directory so that only the owner of the file can delete a file from the directory. Since all of the uploaded files will be owned by `root`, this is another layer of security to prevent anonymous users from deleting uploaded files and replacing them with different or malicious data.

We're also setting "write" and "execute" only permissions on the directory. This means that anonymous users can upload files but they will be unable to get a directory listing to see what other file names are there. File names in the upload directory might give away useful information (host names, customer or supplier names, etc.). Unfortunately, many GUI-based FTP clients expect to be able to get a directory listing in order to operate— you may have to relax the directory permissions to 1700 in order for these clients to operate.

Again, if you're really concerned about anonymous users being able to see file names in the upload directory, you should look into "guest" user access to give each user their own private directory space. "Guest" access is the topic of the next section...



Guest Users



24

Guest user access is a way of providing many individual users their own private FTP directory which they are unable to escape. Sort of a compromise between anonymous access and "real" user access. In fact, if you are allowing regular users to FTP into a system, you may want to set them all up as "guest" users instead so that they cannot escape their home directories.

What Are They?

- Guest users have usernames/passwords and homedirs just like regular users
- When guest users FTP in, they are `chroot ()`ed to their home directory
- Better than anon FTP because everybody gets their own private directory
 - Great for customer/partner access
 - Also used for updates to personal Web pages

25

Guest users have entries in the system `passwd` and `shadow` files and passwords just like regular users. The difference is that when a guest user logs in, they are locked into their home directory just as if it were an anonymous FTP dir.

The standard use for guest users is for business partner and vendor access. You don't want the data from your various business partners to intermingle, so each organization you deal with gets their own login and home directory.

Another growing use for guest user access is to allow remote users to update their own personal Web pages on some central Web server. The user gets locked up in their own personal section of the Web document root and can only update files in this area. Of course, the user pretty much has free reign within this area and could still upload poorly written CGI scripts or other executable content that could be used to compromise the system. The Web server administrator may wish to restrict users to only static HTML content in this sort of arrangement. More on this in the Apache section coming up next.

The Tricky Bits

- Guest users must have password entries, but you don't want to allow normal logins:
 - Give guest users invalid shell (`/noshell, ...`)
 - This shell must appear in `/etc/shells`!
- Would like to avoid creating `/bin, /lib, /etc, /dev` directories for each user...

26

Now guest users must have `passwd` and `shadow` file entries, but in most cases you only want your guest users to be able to log in via FTP (and not via SSH, telnet, etc.). The way to block normal login access is to give guest users an invalid shell in the last field of their `passwd` entry. Note, however, that this invalid shell must be listed in the system `/etc/shells` file or the FTP daemon will not allow the user to log in. Administrators may want to take advantage of the "noshell" program (distributed with the TITAN security package—<http://www.fish.com/titan/>)-- this program not only prevents logins, but also logs a line of data to Syslog if a regular login is attempted on one of these accounts.

If guest users were really `chroot()`ed to their home directory, then the administrator would have to create "bin", "lib", "etc", and other directories for each user just as if they were setting up an anonymous FTP area in each user's homedir. If the system has hundreds of guest users, this would become unmanageable. Luckily, WU-FTPD has a nifty trick for dealing with this problem, as we'll see shortly.

Initial Configuration

- Define basic guest access class:

```
class gst    guest    192.168.0.0/16
```

- Now add basic settings for guest users:

```
log commands      guest
log transfers     anonymous,guest
log security      anonymous,guest
```

- Any of the basic configuration commands can be applied to guest users

27

The first thing we need to do to allow guest users is define the guest access class. This is really no different from setting up the anonymous access class as we saw earlier. Don't forget that you can restrict access by IP address if you only want to allow guest user to have access from a limited range of network addresses.

All of the configuration directives that we applied to the anonymous class can also be applied to guest users. Certainly we want to log everything guest users do on the system (notice that we can turn on logging for multiple classes of users simultaneously). You might also want to define special `path-filter` rules for guest users— for example, you might want to allow guest users to upload file names beginning with dot so that they can update files like `.htaccess` in their Web trees. You can make guest user accounts "upload only" with the `noretrieve` directive or "download only" by blocking access with the `upload` directive. Pretty much every configuration directive we used for anonymous FTP can be applied to guest user access as well.

Note that an FTP server can allow guest access without allowing anonymous access (though this example is allowing both types of access simultaneously). It may be a good idea to separate guest user access from full anonymous access. Anonymous FTP servers are targets, and it may be better to move business partner data off to a less well advertised system.

Which Users are Guest Users?

- List accounts after `guestuser` directive:
`guestuser bob alice eve`
- Use "`guestuser *`" to simply force all users into guest access class
- `/etc/ftpusers` can selectively block users who shouldn't have FTP access (e.g. `root`)

28

The next step is to define which accounts in the `passwd` file should be treated as guest users. The `guestuser` directive allows the administrator to specify accounts by username. The administrator can also specify accounts by a range of UIDs: "`guestuser %10000-19999`" would mean that all users with UIDs between 10000 and 19999 would be treated as guest users. This configuration syntax is very useful on machines that have lots of guest users.

"`guestuser *`" says that *all* users in the `passwd` file will be treated as guest users and locked up in their home directories. If you have to allow FTP access to a system for normal users, this may be the safest way to do it (assuming the users don't need access to data outside of their own home directories).

Don't forget that even if all users on the system are being treated as guest users, you can still selectively block user FTP access with the `/etc/ftpusers` file. Again, this file lists users who *should not* be allowed to FTP into the system.

Directory Restriction

- Here comes the trick:

```
guest-root          /home
restricted-uid     *
```

- "guest-root" is where `chroot()` occurs
 - *This is the place to create /bin, /lib, /etc, ...*
- "restricted-uid" causes WU-FTPD to lock matching users into personal homedirs
 - *Not quite as secure as actual `chroot()`*

29

OK, so how do we avoid having to set up anonymous FTP like directories in each guest user's home directory?

Let's assume we are using "guestuser *" to make all users on the system guest users, and further assume that the system puts user home directories under /home. The trick is to tell WU-FTPD to `chroot()` itself to /home instead of each individual user's homedir— this is what "guest-root /home" does. Now we only need to create the "bin", "lib", etc. directories under /home and every guest user can share them.

But wait! Now guest users will be able to see each other's directories— we don't want that! "restricted-uid" tells the FTP server to enforce a restriction on all matching users that prevents them from escaping their individual home directories. Note that this restriction is enforced by the FTP daemon itself and not the system kernel (as would be the case after a `chroot()` call). A programming error in the FTP daemon (or a buffer overflow) could allow the user to escape their home directory and see other directories under /home, but at least get no further than that because of the `chroot()` to /home caused by the "guest-root" directive.

So the trade-off here is a potential loss of security (although nobody has reported holes in the "restricted-uid" code to date) versus the cost of having to maintain separate anonymous FTP style hierarchies in each guest user's homedir. For servers with large numbers of guest users, the convenience factor of only having to maintain one directory tree usually trumps the added security benefit of a strict `chroot()` for each individual user.

Special passwd Config

- Setup of guest `/bin`, `/lib`, `/etc` dirs essentially the same as for anon FTP
- Entries in guest area password file are slightly special, however:

Regular `/etc/passwd` entry:

```
bob:x:500:500:Bob Smith:/home/bob:/bin/bash
```

Special `/home/etc/passwd` entry:

```
bob::500:500::/bob:
```

30

Having configured `guest-root` and `restricted-uid` per the previous slide, we now need to create and populate the `/home/bin`, `/home/lib`, `/home/etc`, and other directories. If the machine is also an anonymous FTP server, often the easiest thing to do is to simply copy the directories from the anonymous FTP area into `/home`. Assuming your anonymous FTP area is `/var/ftp`, you could do something like

```
cd /var/ftp
tar cf - bin lib etc ... | (cd /home; tar xfp -)
```

If the machine does not already have an anonymous FTP area set up, you will need to create these directories by hand per the guidelines earlier in this section. Again, your vendor will typically provide some sort of script to help you create anonymous FTP directories—this script could also help you create the guest root directory.

For the `restricted-uid` directive to work properly, however, you will need to munge each user's `/etc/passwd` entry slightly before copying it into `/home/etc/passwd`. For one thing you should strip out all unnecessary information. The guest area `passwd` file only needs the user's username, UID, GID, and home directory fields—get rid of everything else. The other important change is to make the user's home directory entry relative to the `guest-root`: `/home/bob` becomes just `/bob` if the `guest-root` is `/home`. This `homedir` entry in the guest area `passwd` file tells the `restricted-uid` directive which directory each user should be locked up in.

Like anonymous FTP areas, the guest user "etc" directory should *never* have a shadow file installed in it.

Summary: WU-FTPD

- WU-FTPD features are nice when needed– use a simpler FTPD for "download only"
- History of security holes– stay up-to-date!
- Be very careful with anonymous uploads and employ layered configuration defense
- Guest user functionality great, but sometimes tricky to configure properly

31

WU-FTPD has a lot of other interesting configuration options that we haven't covered here. Check out the `ftppaccess` manual page for additional information. All of the WU-FTPD documentation, manual pages, and sample config files are available on-line via <http://www.wu-ftp.org> and <http://www.landfield.com/wu-ftp/>

WU-FTPD is complex software and security holes tend to be discovered with alarming regularity. Make sure you keep up-to-date on WU-FTPD releases. Also be sure to employ lots of redundancy in your WU-FTPD security policy by using good directory and file permissions, and taking advantage of all security options in `ftppaccess`– even when they overlap in functionality.

Guest user access is a useful feature unique to WU-FTPD. Again, you might even consider restricting regular users to guest-only access on your systems which still allow FTP access, as long as those users never need to access files outside of their personal home directories.

Apache Web Server

32

This section covers basic security configuration information for the Apache Web server (by far the most popular Web server on the Unix platform). This section does not attempt to address security issues with server-side application environments like Tomcat, Websphere, etc. Consult appropriate vendor documentation in these areas.

Biggest Security Problems

- Poorly written CGI scripts and other "server-side" executables
- Buffer overflows
- Poor configuration practices:
 - Running servers with privilege
 - Running "helper apps" with privilege
 - Overlapping Web root with other data/apps
 - Programs, libraries, config files in doc root

33

Before we start talking about how to secure Apache Web servers, it's useful to look at some of the most common ways Web servers are compromised.

One common issue is not a Web server security issue per se, but rather the fault of badly written (or in some cases maliciously written) code that is run by the Web server. While the latter part of this section will discuss some basic guidelines for writing secure CGI programs, complete coverage of secure server-side programming could take an entire day. The short answer is, disable server-side execution if you're not using it, and make sure to carefully review any server-side code you are running. Also take steps to protect the software that's being run by the server and keep it out of the normal document trees (more on this later).

The Apache Web server has had its share of buffer overflow problems. Keeping up-to-date on software releases is the primary defense here. Of course, you may be vulnerable to a previously undiscovered exploit in the current code, so some sites will run their Web servers in a `chroot ()` environment. However, `chroot ()`ing a complex Apache install with lots of CGI scripts and other server-side executables can be difficult.

Most of the other common Apache exploits come down to poor administration and configuration. There should never be a reason to run the Web server or any other apps related to the server (like CGIs, MySQL servers, etc.) with root privileges— all of these processes should be running as unique unprivileged UIDs. Another common mistake is to overlap the Web document root with the data area of another server— for example, sites that mix their Web docs with an anonymous FTP upload/download area. This opens a window for security problems in one service to impact the other, or possibly sets up a way to turn small vulnerabilities in each service into a massive problem for the machine as a whole. Another common mistake is to put executable code and/or server configuration files into the document root (some sites have accidentally put server password files and site certificates in vulnerable locations!). This may allow external attackers to view the source code for your apps (making it easier to spot vulnerabilities in the software).

apache.org Compromise

- Jan 2001 compromise of apache.org is clear example of these kinds of errors:
 - Web root and anon FTP area overlapped
 - Uploads to FTP area handled insecurely
 - Web server allowed server-side execution
 - MySQL server running as root
- Result was just a cute defacement, but could have been much worse...

34

The January 2001 break-in at `apache.org` is a good example of the kinds of configuration mistakes we were discussing on the previous slide:

- At the time of the incident, the `apache.org` server's Web root and FTP server area overlapped.
- Furthermore, the FTP area allowed files to be uploaded with insecure permissions.
- The attackers were able to upload a PHP document that allowed them to execute arbitrary commands via the Web server.
- The attackers also uploaded a network listener daemon, which they triggered via their PHP document that gave them a shell (as an unprivileged user) on the box.
- Once on the box, they investigated the system and found that the MySQL server was running as root and was accessible from their account. A username and password for accessing the server were found in one of the CGI scripts on the box.
- They were able to coerce the MySQL server into writing arbitrary files (though it wouldn't overwrite existing files). The attackers put a "create set-UID shell" script into `/root/.tcshrc` and waited for one of the admins to `su` to root. Game over.

The full description of the incident (from the perpetrators themselves) can be found at http://downloads.securityfocus.com/library/deface_apache.txt. Perhaps the most humorous part of the write-up is the comment from the attackers: "We didn't wanted [sic]to use any buffer overflow or some lame exploit". Nice to see they have some professional standards...

The Rest of This Section

- Configuring Apache
- Dealing with CGIs
- SSL Configuration

Covers both v1.3.x and v2.x servers!

35

The rest of this section is devoted to looking at the security-related issues around running an Apache Web server. This is not a complete course on how to administer Apache, just enough to allow you to talk to your server admins and content developers about the "right" way to run things.

Configuring Apache covers the basic Apache server configuration directives that can be used to improve security. *Dealing with CGIs* talks about the "right" way to set up CGI bin directories and gives some high-level guidance on doing the "right thing" when writing CGI programs. *SSL Configuration* looks at how to get SSL support into Apache and also covers topics like getting server certificates.

All of the configuration commands in this section apply equally well to both v1.3.x Apache servers as well as the new v2.x servers. Frankly, most of the changes in Apache 2.x have been internal– the configuration file syntax (for better or worse) hasn't changed much at all.

Configuring Apache

36

In this section we'll look at some basic "best practices" to use when configuring your Web server. Many of the topics covered here are also covered in the Apache "Security Tips" document available from

http://httpd.apache.org/docs/misc/security_tips.html

Three Config Files?!?

- Apache used to need three config files:
 - `httpd.conf` – basic configuration
 - `srm.conf` – doc trees and directory aliases
 - `access.conf` – access control
- Caused confusion– admins would fail to complete updates in all files
- Current standard is to now put everything in `httpd.conf`

37

In the old days Apache used three different configuration files– this is actually a hold-over from the original NCSA `httpd` that Apache was based on. The `httpd.conf` was used for basic server setup (what IP address and port to listen on, the number of child processes to fork, etc.). Document trees, CGI directories, et al were set up in the `srm.conf` file. Finally, the `access.conf` file was used for all of the access control configuration (IP address based access controls, password protected directories, etc.).

The problem with this scheme was that it almost encouraged configuration mistakes. For example, admins would change a document directory in `srm.conf`, but forget to update the appropriate entry in `access.conf`– suddenly a directory that had been password protected might be available to the world.

As of Apache 1.3.x, the standard is to now put all configuration directives in `httpd.conf` (though you can still use the old "three file" setup if you want). This makes for a more complicated configuration file, but in the long run seems to work better. The 1.3.x Apache distribution still ships with `access.conf` and `srm.conf`, but these files contain only comments and no configuration directives.

Basic Server Config

```
ServerName www.sysiphus.com
UseCanonicalName on
Listen 0.0.0.0:80          # default

User www                  # create special
Group www                 # ditto

ServerAdmin webmaster@sysiphus.com
ServerSignature off
ServerTokens Prod         # hides version
```

38

Setting the Web server's internal `ServerName` may not seem important from a security perspective, but using a single canonical name can be important with sites that are password protected. Basically this is because browsers cache authentication information by URL— if the Web server doesn't use a consistent canonical name, then you may end up prompting users several times for password access to the same information. The `UseCanonicalName` directive tells the Web server to always use the value of `ServerName` when constructing so-called "self-referential" URLs, so that naming will be consistent throughout your Web site.

We'll talk about the `Listen` directive on the next slide. `User` and `Group` determine what privileges the Web server should run with. Create a specific UID and GID for your Web server and *never* run Web servers as root.

`ServerAdmin` is the email address of the person who should be contacted in case of problems— this address appears in various error messages by default. `ServerSignature` controls whether or not a footer message (which happens to contain the Apache server version number) should be appended to server-generated error messages and other server generated docs. Generally we'd like to hide the server version by turning this off. Similarly, `ServerTokens` controls the format of the server header returned on HTTP requests. The most minimal setting is "Prod" which causes on the word "Apache" to be displayed (without version number, etc.)— many automated worms trigger on the Apache version number presented here, so you can slow down infection by hiding this information. If you want to hide that you're running "Apache", you'll actually have to hack the `httpd.h` file in the source distribution to change the values of the `SERVER_BASE*` variables and recompile. If you're going to this extreme, though, you'll also want to change the default error documents, etc. because the format of these messages gives away the fact that you're using Apache.

Quick Aside: Local Servers

- Developers may need local Web servers
- Security admins worry about potential security disaster with that many servers
- Consider having server bind to localhost:
`ServerName localhost`
`Listen 127.0.0.1:8080`
- Now only the local developer can see the server running on their machine

39

`Listen` sets the IP address and port number which the server is listening on. By default the Web server will try to listen on port 80 for all of the machine's IP addresses (represented as the special address `0.0.0.0`), but the admin may choose to bind the server to a specific address if desired. This could allow the admin to bind the server to a special "protected" interface on the system (and this is also used for setting up Web servers on "virtual" IP addresses for hosting multiple sites on the same physical machine). But there's also another use for the `Listen` directive.

In Web development shops, it's pretty common for every developer to have their own local Web server(s) running for development and testing. That's a lot of Web servers on a network— any one of which might be running some untested code which contains a security vulnerability. Not to mention the fact that the Web server software itself might have a buffer overflow or other remote exploit. Now the network used by your developers is probably protected by your corporate firewalls and security infrastructure. Still security admins may worry about having so many essentially "self-supported" Web servers running out there.

One option is to use the `Listen` directive to bind the server only to the "loopback" address on the system— address `127.0.0.1`. The developer using the system can still "see" their local Web server by pointing their browser at `http://localhost/`, but nobody outside of the system will be able to access the Web server. This seems like a reasonable compromise for everybody concerned.

Start at the Top

- Default policy should block all access:

```
<Directory "/">
  Options None
  Order allow,deny
  Deny from all
  Satisfy all
  AllowOverride None
</Directory>
```

- Override settings for specific dirs as needed

40

The next step in server configuration is to define access controls for the various directories served by the Apache software. "Best practice" is to first define a general "default deny" stance for the entire file system as we do here. Later we will configure access to specific directories that are actually part of the Web server's document trees.

If we first set a default policy of allowing no access and then have to explicitly enable access for Web server directories, this can help prevent "accidental" misconfigurations where material ends up getting into the Web server tree that doesn't belong there. This "default deny" policy also helps protect the server against attacks (like the recent "directory traversal" attack reported in Aug 2002) which allows external attackers to "escape" from the document trees and walk the file system of the local server.

So here we're setting options for the directory "/", or the root of the file system on downward. These options are put into a `<Directory...></Directory>` "container" which specifies the specific physical directory structure the options apply to. The alternative is to use `<Location...></Location>` which specifies a URL path rather than a physical directory. Frankly, `<Location...>` is confusing— best to stick with the literal path names on your system and use `<Directory...>`.

As for what all of these options mean, well we'll cover that in the next fifteen slides or so...

Setting Options Field

- Options can be any/all of the following:
 - Indexes
 - FollowSymlinks/SymlinksIfOwnerMatch
 - Includes/IncludesNOEXEC
 - ExecCGI
 - MultiViews
- "All" and "None" are also supported
- Some options have surprising results...

41

`Options` defines a list of the optional behaviors that are defined for the Web server on a given directory structure. Generally, the `Options` directive is written as a space separated list of the options you want turned on

```
Options Indexes MultiViews FollowSymlinks
```

Any option not specifically listed is disabled.

However, an alternate form is to list options with +/- to specifically enable/disable certain options:

```
Options +Indexes +MultiViews +FollowSymlinks -Includes -ExecCGI
```

It turns out the second form shown above can have some unexpected consequences when `Options` is set on both a parent directory and one of its sub-directories– when +/- is used for all options, then the *union* of the `Options` for parent and subdirectory is used. Best to use the first form.

"`Options All`" sets all options *except* `MultiViews` which must be set explicitly (this appears to be due to the "organic" nature of the growth of Apache– `MultiViews` is a relatively recent development). "`Options None`" disables all options (per our "default deny" stance on the previous slide).

Of all of the options shown here, only `MultiViews` has no real security implications (`MultiViews` enables "server negotiated" content– typically used to select multiple versions of the same Web document translated into different languages). We'll talk about `ExecCGI` later in the section when we talk about CGIs in general. But let's take a look at the other options right now...

Indexes Option

- Accessing a Web directory usually causes `index.html` file to be shown
- If `index.html` doesn't exist, Web server returns "404 Not Found"
- When `Indexes` is turned on, directory listing is displayed instead
- May not want this– can show "private" files, and give away docroot structure

42

Normally when you view a URL like `http://www.sans.org/info/`, the Web server displays a file called "`index.html`" in the "`info`" subdirectory of the Web server document root (actually, the file name doesn't have to be `index.html`– the file name can be set by the server administrator). If this file is not found, then by default the Web server will display the ever-popular "404 Not Found" error.

However, if `Indexes` is turned on for a directory, the Web server will instead show a directory listing of the other files and subdirectories in the given directory. Sometimes this is useful– particularly if you're using your Web server as a mechanism for sharing files. However, it can also be a problem– if somebody deletes the `index.html` file or forgets to create it in the first place, then you may be exposing files and directories that you didn't want to expose. It may also allow remote users to navigate into sections of your document tree that they shouldn't know about (though honestly if the information is that sensitive it either shouldn't be on the Web server in the first place or at least should be password protected).

Generally, it's best to disable `Indexes` unless you're sure you're going to need it. This has the best chance of preventing "accidental" errors which expose your doc root to scrutiny.

Symlink Options

- Basic idea is simple:
 - `FollowSymlinks`: Web server will follow symlinks– even out of normal doc tree
 - `SymlinksIfOwnerMatch`: only follow link if link owner and file owner are the same
- Disable `FollowSymlinks` to improve security, but has performance impact
- `SymlinksIfOwnerMatch` doubles performance impact

43

When `FollowSymlinks` is enabled, then the Web server will simply "open" any symbolic links in the doc root without question and display the contents of the file pointed to by the symbolic link. This means that content maintainers could accidentally create a symlink which points to a critical system configuration file and expose the contents of that file to the world. Now of course this file must still be readable by whatever user the Web server is running as, but this still may be something of a concern.

So, the first impulse is to turn `FollowSymlinks` off. However, doing this has an unexpected performance impact. The server must now check each "file" in the doc root before opening it, just to verify that the file is not in fact a symlink. This means an extra system call before each and every file opened by the Web server– which amounts to a lot of extra work on a busy server. For this reason, many sites choose to leave `FollowSymlinks` enabled. You can scan your document trees periodically for symlinks (perhaps even verifying that the links don't point out of the doc tree), or even run the Web server `chroot ()`ed if you're really concerned.

Another option is `SymlinksIfOwnerMatch`, which means only open the file pointed to by the symlink if the owner of the file and the owner of the symlink are the same UID. This was designed to prevent Web developers from accidentally pointing to root-owned system configuration files. Of course, this now means that the Web server potentially has to do *multiple* system calls on each file open– one to check for a symlink, and then additional calls to get the name of the file the link points to and the owner of that file.

Still, if your Web server is not heavily loaded, just don't enable `FollowSymlinks` at all.

Includes Option

- Allows "server parsed HTML" capability
- Has some real advantages:
 - Include standard headers/footers
 - Add limited dynamic content w/o using CGI
- Beware `#exec` option, however
- Consider using `IncludesNOEXEC`, which disables `#exec` command

44

`Includes` turns on "server parsed HTML" capability (usually indicated by files with the `.shtml` extension). This is a simple way to get bits of dynamic content into otherwise static HTML files without going to the trouble of writing a full CGI or PHP program. Another use for server-parsed HTML is to automatically include certain standard header/footer information in many different documents. For example, some sites include a copyright statement at the bottom of each page—updating the copyright date in a single file is much easier than having to modify all of the Web pages on your site.

However, server parsed HTML includes the `#exec` command which runs a specified command and substitutes the output of that command into the document (like the `date` command to display the current date/time on the page). This opens the door for a badly written document to execute a command which has unexpected consequences on the system. As an alternative, you can set `IncludesNOEXEC` which enables server parsed HTML, but disables `#exec`.

As an aside, we mentioned that server parsed HTML files are generally distinguished by the `.shtml` extension. This is accomplished by setting up a special "handler" for files with this extension:

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

Another option, however, is to set `XBitHack on`, which causes any HTML file that has the execute bit set to be treated as a server parsed HTML file. On the plus side, this means that outsiders won't know that the file is server parsed HTML because you no longer need the `.shtml` extension. On the minus side, it's easy to mess things up and forget to set the execute bit (or lose the bit when copying files from one directory to another) or to set this bit on files which shouldn't have it.

IP-Based Access Control

- Use "Allow from" and "Deny from" to control access from:
 - Hosts* – use host name or IP address
 - Networks* – use "net/mask" or CIDR notation
 - Domains* – use domain name
- Must be used with `Order` directive– this is the confusing part...

45

Now let's look at how to control access to Web directories based on the IP address of the remote machine.

Apache uses "Allow from" and "Deny from" to permit/deny access based on IP address. You can specify host names or IP addresses of individual machines, or even entire networks using either network/mask syntax (e.g. 192.168.1.0/255.255.255.0) or CIDR notation (192.168.1.0/24). You can even specify domain names. Note however that using host names and domain names depends on attackers not being able to spoof or corrupt the local DNS server– best to use IP addresses whenever possible.

By the way, you can also use the special keyword "all" ("Deny from all", "Allow from all") as a wildcard.

If "Allow from" and "Deny from" were the entire story, then things would be simple. However, there's an extra complication because of Apache's `Order` directive– more on this on the next slide...

The Order Directive

- Just remember that Order controls:
 - Eval order for Allow/Deny commands
 - Default behavior if no matching Allow/Deny
- "Order Allow,Deny"
 - Process Allow statements, then Deny rules
 - Default is *access denied*
- "Order Deny,Allow"
 - Do Deny statements first, then Allow rules
 - Access is *allowed* if no match

46

What's confusing about the Order directive is that it actually controls two essentially unrelated parameters. First, the Order directive controls the order in which Allow/Deny statements are processed– which is sort of intuitive given the name "Order". However, Order also defines a *default* policy for connections which don't match one of the explicit Allow/Deny statements given by the administrator– and, of course, the default policy is different depending on how the Order statement is written.

Basically the administrator has two options. "Order Allow,Deny" says to process the "Allow from" statements first, and then all of the "Deny from" statements. However, "Order Allow,Deny" also means that the default is to drop any connections that don't have an explicit matching "Allow from" (in other words, "Order Allow,Deny" is the standard *default deny* stance).

The alternative is "Order Deny,Allow". Here the "Deny from" statements are processed first, and then the "Allow from" statements. The default for "Order Deny,Allow" is to permit any connections that aren't explicitly blocked by "Deny from" (making this the *default permit* stance).

Confused yet? Let's try the example on the next slide...

An Example

- Suppose we had the following:

```
Allow from sysiphus.com
Deny from foo.sysiphus.com
Order Allow,Deny
```

- What happens with various connections?
- Now reverse the `Order` statement—what happens to the same connections?

47

Given the example shown on the slide, let's consider a few different scenarios:

- If a connection comes in from the host `foo.sysiphus.com` it will be denied— first the "Allow from" is consulted (access would be permitted because `foo.sysiphus.com` is part of `sysiphus.com`), and then the "Deny from" (which explicitly blocks `foo.sysiphus.com`).
- Connections from any other host in `sysiphus.com` would be allowed— these would match the first "Allow from" and be allowed, falling through the "Deny from" directive because they don't match.
- Connections from all other hosts would be blocked— these connections don't match either the "Allow from" or the "Deny from", so the implicit "deny everything" caused by "Order Allow, Deny" takes over and drops these connections.

OK, that was easy! Now suppose we changed the `Order` line above to "`Order Deny, Allow`" and left everything else the same:

- Now all of a sudden connections from `foo.sysiphus.com` are *allowed*— the "Deny from" is now consulted *first* (which would normally block the connection), but then the "Allow from" is checked (which allows connections from anything in `sysiphus.com`— including `foo.sysiphus.com` which otherwise would be blocked)
- Anything else in `sysiphus.com` is also allowed because of the "Allow from"
- Other hosts are also *allowed*— remember that our default policy is now "permit everything" because we now have "`Order Deny, Allow`"

Frankly, it's hard to imagine how the Apache folks could have messed this up any worse. They should have simply emulated the same sort of ACLs used by TCP Wrappers and have done with it, but they didn't.

Granting Access to Doc Root

```
DocumentRoot "/var/apache/docs"

<Directory "/var/apache/docs">
    Options Indexes SymLinksIfOwnerMatch \
        IncludesNOEXEC MultiViews
    Order deny,allow
    Allow from all
    AllowOverride None
</Directory>
```

48

OK, with all of that out of the way, let's look at how we might allow access to our Web server document tree. Remember that we implemented a "default deny" stance on the root file system, so we have to explicitly enable access to our DocumentRoot.

We do this by setting options on the appropriate directory in another `<Directory...></Directory>` container—these settings will override the settings for the "/" directory we set up earlier. First we set our `Options` directive to include the options we want on our doc root. Next we need to allow access from all hosts on the Internet with the "Order" and "Allow from" directives (more on `AllowOverride` later in this section).

Note that the "Allow from" here is actually redundant. Since we have "Order deny,allow" the default policy would be to allow all connections anyway. Most people prefer to have the explicit "Allow from all" anyway, just to make things more self-documenting. By the way the, "Order allow,deny" and "Deny from all" that we had set on the "/" directory several slides ago was also similarly redundant.

Password Access

- Directories can be password protected
- Web is stateless– username/password are resent for each page requested
- Standard is "Basic Auth"– passwords are sent in the clear
- More secure alternative is "Digest Auth"– most browsers don't support it

49

In addition to IP address based access controls, portions of your Web space can also be password protected. When entering a protected document space for the first time, the user is prompted for a username/password (this is generally referred to as "HTTP Basic Auth"). What the user doesn't see is that this same username/password is transmitted over and over again on each and every request for a document from that section of the Web tree– the browser caches the username/password and simply resends it without prompting the user (when the browser is killed and restarted, the user must re-enter the password). The bad news is that the username/password travels over the network in the clear and can be easily sniffed. So it's a good idea to combine HTTP Basic Auth with SSL encryption.

The alternative to Basic Auth is "Digest Auth"– Digest Auth transmits an MD5 hash of the user's password rather than the password itself. This prevents the password from being sniffed directly, but brute-force attacks are possible to recover the password from the MD5 hash. The other problem with Digest Auth is that very few Web browsers support it. MSIE 5.x and Opera 4.x support Digest Auth, but Netscape, Mozilla, et al do not. This means that Digest Auth isn't really generally useful.

To enable Digest Auth in Apache, you must configure the server with `--enable-digest` and recompile. Further information on configuring Digest Auth can be found at:

<http://httpd.apache.org/docs/howto/auth.html#digest>

Managing Password Files

- Format of Apache password files is:
username:password[:ignored]
- Don't use /etc/shadow file– users should have different Web passwords
- Admin can use htpasswd command to manage Apache password file
- Keep password files out of doc trees!!!

50

To password protect part of the Web tree, we first need a password file. The format of Apache password files is simple– just the username, a colon, and an encrypted password string. Anything after the encrypted password is ignored by Apache– so it's possible to simply use a copy of the system /etc/shadow file (you'd have to use a copy because /etc/shadow wouldn't normally be readable by the Web server user). This is a terrible idea for two reasons: (1) you'd have a copy of /etc/shadow on your system that was readable by a non-root user, and (2) users should use a different password for Web access and login access because Basic Auth insists on retransmitting the username/password all the time. It's also important to put the Apache password files someplace *outside* of the Web document tree (nothing is worse than finding your Web password file indexed on an Internet search engine). As we'll see, Apache allows the administrator to specify the location of the password file anywhere in the file system.

The htpasswd command provided with the Apache distribution allows the admin to add/remove users and change passwords. The syntax is "htpasswd [-ms] <pwdfile> <username>" where <pwdfile> is the path to the Apache password file, and <username> is the name of the user being added/changed. By default htpasswd uses old-style DES56 password encryption. The -m option tells htpasswd to use MD5 encryption, and -s uses SHA-1 (the same password file may have different passwords using all three types of encryption).

So here's how to add user "hal" to /var/apache/etc/passwd with MD5 password encryption (the same command is also used to change the password for user "hal"):

```
htpasswd -m /var/apache/etc/passwd hal
```

In this mode, the admin will be prompted for the password for the user's new password. htpasswd also supports "batch mode" where passwords can be entered on the command line (good for scripts):

```
htpasswd -mb /var/apache/etc/passwd hal NewP8ssw0rd
```

Of course this means that the user's password would be at least momentarily visible via ps.

Sample Directory Config

```
<Directory "/var/apache/docs/private">
  AuthType Basic
  AuthName "Private Info"
  AuthUserFile "/var/apache/etc/passwd"
  Require valid-user
</Directory>
```

- "Require" directive is *critical*
- Can optionally use Berkeley DB or DBM password files to improve performance

51

Here's how to actually configure password based access control on part of the doc tree. "AuthType Basic" says that we should use HTTP Basic Auth (as opposed to "AuthType Digest" for Digest Auth). "AuthName" is a string that is used as part of the username/password prompt by the browser— in this example, the prompt on the pop-up box would read something like "*Enter username/password for Private Info*". "AuthUserFile" is the location of the Apache password file— again, put this file someplace *outside* your document tree. Different directories can use different password files or all share the same file.

The Require directive says that the user must enter a valid username/password before being given access. If you forget to put the Require directive into the config file, then the username/password box will pop up but *any username and password will allow access!* The reason the Require directive exists at all is that it's also possible to say "Require hal sally bob", which would only grant access if the username were one of the three listed users (and obviously only if the correct password for the given account was entered as well).

One problem with standard flat text password files is that as the number of users grows, finding a particular user in the file takes longer and longer. And again remember that the username/password verification has to be done on each and every document access from the restricted area. For performance reasons, it's possible to create your password files as DBM or Berkeley DB style database files to enable faster lookups. You must reconfigure Apache with `--enable-auth_db` or `--enable-auth_dbm` flags and recompile before this is possible, however. Once the server has been properly built, simply use `AuthDBUserFile` instead of `AuthUserFile`. The Apache distribution comes with the `dbmmanage` program for managing user accounts in DB or DBM style database files. Again, for more info see

<http://httpd.apache.org/docs/howto/auth.html#database>

Access by Password *or* Address

```
<Directory "/var/apache/docs/locals-only">
  AuthType Basic
  AuthName "Local Info"
  AuthUserFile "/var/apache/etc/passwd"
  Require valid-user
  Order allow,deny
  Allow from 192.168.0.0/16
  Allow from 127.0.0.1
  Satisfy any
</Directory>
```

52

Now it's possible to discuss the "Satisfy" directive we saw in an earlier example. Satisfy simply controls the relationship between the various forms of access controls defined on a particular part of the doc tree. This includes IP based access controls and password based access controls. Basically the choice is either "Satisfy all", which means that both the IP based restrictions must be satisfied and a proper username/password entered, or "Satisfy any" which means that either the IP address must match or a correct username/password must be used.

This latter case is useful when you want to have a section of your Web tree that's only visible to "local" people. In the example on the slide, our IP based access controls allow access from the 192.168.0.0 network—since "Satisfy any" is defined, then as long as the connection comes from a host on this network, a password will not be required (in fact, the prompt will never even appear since the IP address of the source of the connection is checked first). Connections from other IP addresses will cause the password popup box to appear and force the user to enter a valid password.

What's AllowOverride Do?

- AllowOverride allows configuration commands to go in files in doc trees

PRO:

- Delegate administration to various groups
- Reconfigure server without restarting

CON:

- Security policy spread throughout doc tree
- Performance hit

53

The last configuration command to discuss is the AllowOverride option. Enabling AllowOverride allows the owner of the document tree to create a configuration file in the doc tree itself which overrides settings in the httpd.conf file (usually these files are named .htaccess, but this is customizable as we'll see).

On the plus side, this allows the Web server administrator to delegate administration to the owner of the document tree. For large Web environments, this may be the only manageable solution. Another advantage is that these configuration files are read every time a document is accessed from the given document tree, so any changes made to the configuration in these files takes effect immediately without having to restart the Web server (changes to httpd.conf require at least a HUP to the running daemon).

This is actually also a problem as well. Since the Web server has to look for these per-directory configuration files when AllowOverride is set, and since these files then need to be parsed by the Web server when found, this causes a performance hit for each and every file access. In fact, if AllowOverride were set on /var/apache/docs, and the file being accessed were in /var/apache/docs/local/config, the Web server would actually have to look for override files at each level of the tree from /var/apache/docs on downward. This is a real problem for deeply nested trees.

The other problem is that the security policy for the Web server is now spread throughout the document tree. This not only makes it difficult to know how the server or a particular part of the document tree is configured, it also makes it easier for somebody to make a mistake which opens up a vulnerability on the machine. And since the mistaken configuration directive is hidden away in some random file in the doc tree, it could go unfound for a long time.

AllowOverride Syntax

- AllowOverride can be "All" or "None", or any/all of the following:
 - Options – Change Options settings
 - Limit – Address-based access controls
 - AuthConfig – Password auth configuration
 - Indexes – Directory listing "look and feel"
 - FileInfo – Document and language control
- Allow setting of Indexes option with Options override, not Indexes!

54

The Web server administrator actually has some level of control over what configuration settings may be overridden in the per-directory config files. AllowOverride can be set to "None", which disables overrides entirely, or "All", which basically means that any configuration option can be overridden. However, the administrator may also choose to only allow overrides for certain groups of commands, as defined by the five options shown here.

"AllowOverride Options" means that the per-directory config files can override any of the Options set in `httpd.conf` for that directory (it also allows the `XBitHack` option for server parsed HTML to be turned on here as well). The `Limit` override allows the per-directory config file to set "Allow from", "Deny from", and "Order" to set IP based access controls. `AuthConfig` allows the per-directory config file to set all of the `Auth*` and `Require` directives used for password based access controls.

`Indexes` and `FileInfo` don't particularly have any security-related impact. "AllowOverride Indexes" just allows local control over how directory indexes appear, but doesn't allow the local config file to change the value of the `Indexes` option– the Web server administrator needs to set "AllowOverride Options" to allow this to happen.

Multiple override options may be specified with a single `AllowOverride` directive– simply enter a space-separated list of the desired overrides (example on the next slide).

Using AllowOverride

```
AccessFileName .htaccess

<Files ~ "^\.ht">
    Order allow,deny
</Files>

<Directory "/var/apache/docs/locals-only">
    AllowOverride Limit AuthConfig
</Directory>
```

55

`AccessFileName` defines the name of the override file recognized by the Web server. Again, the default is usually `.htaccess`, but you might want to choose a different standard for your Web servers to try and prevent external attackers from stealing these files out of your doc tree.

Whatever name you end up choosing, it's important to prevent these files from being displayed by your Web server. Here we're using the `<Files...></Files>` container to deny access to any file on our Web server named `.ht*` (actually "`<Files ~ regex>`" has been deprecated— you're supposed to use "`<FilesMatch regex>`" instead— but I still do it the old-fashioned way). Another example of this useful feature would be to block access to backup files created by common editors (like `*~` files created by Emacs, or `.bak` or `.sav` files):

```
<Files ~ "*(\~|\.bak|.sav|.orig)$">
    Order allow,deny
</Files>
```

Now we can set `AllowOverride` for our "locals only" document tree that we set up in the previous example. No further configuration would be required in `httpd.conf`— the contents of the `<Directory...></Directory>` container in our previous example would simply go into the `.htaccess` file in the `/var/apache/docs/locals-only` directory.

Dealing With CGIs

56

We mentioned earlier that insecure CGI programs were one of the common sources of compromise for Web servers. In this section we look at what CGIs are, how to enable CGI support safely in Apache, and give some pointers on safe CGI programming and administration.

Overview

- "Common Gateway Interface" defines how to write programs to generate HTML
- CGI programs can be written in any language– Perl is popular on Unix
- Programs execute with privileges of the Web server user
- Badly written CGI scripts the primary source of compromise for Web servers

57

The "Common Gateway Interface" defines a standard for programs which can be executed by the Web server and dynamically produce HTML content. Since the program being executed can do essentially anything to generate the HTML (connect to databases on other systems, execute shell commands, etc.) this is a very powerful mechanism. However, these CGI programs are running on the Web server with the privileges of the Web server user, so a badly written CGI script can cause a fair amount of trouble.

Bottom line is that any code that's going to be executed on the Web server needs to be carefully scrutinized for potential security problems.

Enabling CGI Access

- Restrict CGIs to a tightly controlled area:

```
ScriptAlias /cgi-bin/ "/var/apache/cgi-bin/"
<Directory "/var/apache/cgi-bin/">
    Options None
    AllowOverride None
    Order deny,allow
</Directory>
```

- Avoid using "Option ExecCGI" which scatters CGI scripts throughout doc trees

58

There are two ways to enable CGI scripts with Apache— a "right way" and the "wrong way". The "right way" is to restrict CGIs to one (or at worst a small number) of tightly controlled directories which are not part of the normal document tree. Only a few people should be authorized to add CGIs to this directory, and then only after close scrutiny of the program and some serious testing. CGI directories are created with the `ScriptAlias` directive— in the example on the slide we're saying that `http://www.ourdomain.com/cgi-bin/foo` means "run the program `foo` from the directory `/var/apache/cgi-bin`". Our "default deny" stance that we set up on the root of the file system means that we need to explicitly allow access to this CGI bin directory using the configuration commands in the `<Directory...></Directory>` container.

The alternative to setting up restricted CGI bin directories with `ScriptAlias` is to use the `ExecCGI` option on directories in the document root. Typically `ExecCGI` needs to be combined with an `AddHandler` directive so that the server can recognize CGI programs:

```
AddHandler cgi-script cgi pl sh
```

Now any file that ends with the extension `.cgi`, `.pl`, or `.sh` will be executed as a CGI script.

The problem with `ExecCGI` is that now anybody with access to the doc root can drop in a CGI script that could contain a vulnerability that compromises the system. If there are a lot of people providing content to the Web server, the situation can get out of control quickly. The Apache "Security Tips" document recommends using `ExecCGI` if and only if:

- You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.
- You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.
- You have no users, and nobody ever visits your server.

Hints for Safe CGIs

- Do not trust user input or environment variables– always validate before using
- "Hidden form fields" aren't hidden
- Always do bounds checking before copying inputs in memory

59

One of the most common uses for CGI scripts is processing form submissions containing user inputs. This is also the place where the most serious CGI security mistakes are made. The most important thing to remember is that that Web form can easily be modified by the user at the other end of the connection and the results that are posted back to the Web server can contain absolutely *anything* that user desires (it's easy enough to fake POST operations back to the Web server, but a more simple approach is to save the Web form as an HTML document on disk, make the appropriate changes, and then POST the data back from the hacked version of the form).

This means that validating the results of form input is critical. One common mistake is to use form input as arguments in shell commands in a CGI script. But a malicious user could have embedded shell commands (" ; rm -rf /*") in a form field. If your CGI script is written in the shell or Perl, this can cause major problems. Another trick is to embed HTML or Javascript code in form fields– if the form data is used to generate a Web page, again there could be problems. If possible, it's best to tightly define what should be present in a given input field: "phone numbers should contain digits, parens, + and –", "the right-hand side of an email address can only contain alphanumeric characters plus dot and dash", etc. This is better than trying to look for a set of "dangerous" characters because you may not be able to clearly define what's "dangerous" in all cases.

It's also important to do bounds checking on form input– particularly when you're using a language like C for producing CGI scripts (rather than Perl which dynamically allocates memory for data structures). Even though you've defined the form field to be only 40 characters long, a malicious user could send you 4K worth of data which includes a buffer overflow exploit.

Note also that those "hidden" form fields that programmers like to use for holding state data and other "secret" information are not hidden at all. "View...Source..." on any HTML form will clearly show all of these hidden fields and again they can be modified at will by a malicious user before the data gets posted back to the Web server.

Protect CGI Source Code!

- Develop programs somewhere other than CGI bin directory
- Be careful of automatic backup copies created by editors, etc.
- Put any libraries used by CGI scripts in separate directory outside of Web tree

60

When developing CGI programs, it's important to make sure that development is done outside of the CGI bin directory where the program will run in production—this is especially true if you're allowing CGI scripts in your doc trees with `ExecCGI`. For one thing, you don't want interim copies of CGI programs sitting around in directories where they might get executed. Also, the backup files left around by editors may end up being treated not as CGI scripts that should be executed, but rather as plain text files that should be displayed on a remote user's browser—giving away your CGI source code and potentially dangerous information like embedded passwords.

Also, there's no reason to put programming libraries used by CGI scripts (like shopping cart toolkits and other productivity enhancers) in the CGI bin directory. The programs can reference these libraries from anywhere in the file system, so put them in a special directory that's neither a CGI bin directory nor part of the document root. Again, these programming libraries can often contain embedded passwords for accessing databases et al and you don't want their contents mistakenly displayed to remote users.

Third-Party CGIs

- Lots of these out there, many are very poorly written and have security holes
- Some contain deliberate back-doors
- Check source code– particularly for calls to other programs and network connects
- Consider `chroot ()` and/or testing on isolated network at first

61

Speaking of CGI libraries, there are lots of programming libraries available for writing CGI scripts– some freeware and some commercial. In some sense, using third-party CGI tools from the 'net is even more dangerous than downloading and using third-party software packages like Emacs and SSH, because not only are you trusting the developer and the archive site for your own use, you're putting this code out there where anybody on the Internet can potentially execute it on your Web servers. There was even a popular shopping cart application which had a deliberate back door inserted by the author which allowed a remote user to run arbitrary commands on the Web server (the author claimed that the back door had been inserted "to help sites debug problems with their Web server configuration"– yeah, right).

If the library or application is distributed in source code form, it's important to carefully examine the code before use. In particular look for the common routines used to execute other programs– `exec ()`, `system ()`, `popen ()` (and `open ()`, ``...``, and `eval ()` in Perl and the shell). Also look out for code which makes connections across the network to other machines– `connect ()`, `bind ()`, `listen ()`, `accept ()`. For libraries and applications which are distributed in binary form, the best you can do is try running `strings` on the program and see what you turn up (you can also use `nm` on libraries and get a dump of the symbols defined in the library which may be of some help).

You may want to actually test the CGI applications on an isolated or "air-gapped" network at first. Put a sniffer on the network and watch for any suspicious network activity that may be occurring. Of course, this won't catch passive back-doors like our shopping cart example above.

SSL Configuration

62

Earlier we mentioned that it may be useful to combine SSL encryption with HTTP Basic Auth to protect passwords which are being used to access private Web sites. More commonly, though, SSL is used to protect sensitive data that is flowing between Web browser and Web server– credit card numbers, personal information, etc. This section looks at how to set up SSL support in Apache and also talks about how to get a server certificate for real Internet e-commerce.

Security Notice: As of Sept. 12, 2002 there is a worm running which is actively exploiting a buffer overflow problem in the ModSSL/OpenSSL source. As of this writing, the worm seems to be targeting Linux systems only, but it could mutate at some point in the future. The particular buffer overflow exploit has actually been known for months and is fixed in the latest releases of the various source distributions– please upgrade.

SSL vs. Apache 1.3.x

- Unpack latest OpenSSL sources, plus Apache and ModSSL sources in same dir
- ModSSL dist keyed to Apache version
- Configure and build OpenSSL
- Configure in ModSSL dir, build Apache:

```
# cd mod_ssl-2.8.10-1.3.26
# ./configure --with-ssl=../openssl-0.9.6g \
              --with-apache=../apache_1.3.26
# cd ../apache_1.3.26
# make
```

63

SSL support is not included in the Apache 1.3.x distribution by default. Not only do you need to get the OpenSSL libraries (<http://www.openssl.org>), but you also need the ModSSL hacks from <http://www.modssl.org/>. It's important to note that the ModSSL distribution is keyed to the particular version of Apache that you're using— if you're up to Apache 1.3.26, then you need to get the `modssl-*-1.3.26.tar.gz` file.

Unpack the OpenSSL, ModSSL, and Apache sources in the same directory so that the three source trees end up in the same directory at the same level of the file system. Now configure and build the OpenSSL distribution in the OpenSSL source directory (you don't actually have to install it for the Apache build to succeed).

The odd thing about ModSSL with Apache 1.3.x is that you run the standard `configure` command in the ModSSL directory, but then do the build in the Apache source directory (if you've previously configured and built Apache in your source directory without SSL support, make sure to do a "make distclean" in the Apache source directory before running `configure` in the ModSSL directory). What's going on here is that the `configure` in the ModSSL directory not only sets up the various `Makefiles` and include files for the Apache build, it also hacks the necessary SSL modules into the Apache sources. The arguments to the `configure` script in the ModSSL directory give the location of the OpenSSL distribution as well as the location of the Apache sources which will be modified. You can also specify other normal `configure` options like `--prefix` or `--enable-digest` and `--enable-auth_db`.

Hey, it's a huge hack, but it works...

SSL vs. Apache 2.x

- ModSSL built into distribution as of 2.x
- Must first build *and install* OpenSSL
- Compile in SSL support with appropriate options to configure script:

```
# cd httpd-2.0.40
# ./configure --prefix=/var/apache \
              --enable-ssl --with-ssl=/usr/local/ssl
# make
```

64

Life is better with Apache 2.x because ModSSL is actually included in the core Apache source distribution. You still need to get the latest OpenSSL libraries and install them someplace in the file system (this is different from the previous slide where Apache 1.3.x wanted the OpenSSL binaries to be located in the OpenSSL source directory).

Once OpenSSL is built and installed, just run configure as normal in the Apache source directory, but add the `--enable-ssl` and `--with-ssl=<dir>` flags (both are required although, frankly, `--with-ssl` should be enough).

Keys and Certificates

- SSL uses public/private key encryption
- Every SSL-ready server needs a key pair
- A *certificate* is a server's public key that has been signed by a Certificate Authority
- For e-commerce activity, your cert must be signed by a "recognized" CA

65

SSL is based on public/private key encryption (and thankfully RSA's patents on this technology have expired so there are no more licensing hassles). Each server that wants to support SSL communications needs a public/private key pair. Browsers expect that the server's public key has been signed by a recognized certificate authority (CA)– this signed public key is referred to as a *server certificate*.

For purely internal use, it's possible to set up your own CA or use dummy certificates that have been signed by a fake CA. This process is more fully documented on the Apache Web site:

http://httpd.apache.org/docs-2.0/ssl/ssl_faq.html#ToC29

Note also that the Apache 1.3.x distribution has an easy way to generate these dummy certs. Just run "make certificate" in the Apache 1.3.x source distribution and a helpful little shell script will walk you through the process.

When a Web browser accesses a site that has a dummy certificate, a little pop-up appears warning the user that the certificate is invalid and asking if they want to continue talking to the site. Most users click "yes" and accept the cert without thinking about it.

The reality, though, is that for public e-commerce type activity you really need to get a site certificate from one of the few CAs that are recognized by the popular Web browsers currently in use. This process is described more fully on the next slide...

Getting a Certificate

- Generate key (protect this file!):

```
openssl genrsa [-rand list:of:files] \  
-out server.key 1024
```

- Generate Certificate Signing Request:

```
openssl req -new -key server.key -out server.csr
```

- Transmit CSR (+ money) to CA to receive certificate (`server.crt`) file

66

The first step in getting a certificate is to generate a key pair for your Web server using the `openssl` command from the OpenSSL distribution. "`openssl genrsa`" generates a standard RSA type key which is used for standard Web browser SSL connections. The `-out` option specifies the file where the key is placed and 1024 is the key length. On systems which do not have a built-in `/dev/random` device, the `openssl` command needs some large amount of apparently random bits to generate the key— the `-rand` option can be used to specify a colon-separated list of files which can be used to generate pseudo random values. The usual tactic is to use log files, the system kernel, etc. but not that this approach doesn't really produce truly random values (log files generally use only alphanumeric characters, binaries contain lots of nulls, etc.).

The resulting `server.key` file contains the server's private key, so it's critical that you protect this file. Anybody who steals that key file could impersonate your server and/or act as a man-in-the-middle and compromise SSL sessions between your server and remote browsers. The `openssl` command gives you the option of encrypting this file with a password, but if you do that then your Web server will not be able to start automatically without this password being provided in some fashion (you could code it into the Web server boot script, but this doesn't seem much better than not encrypting the file at all). Since the Web server starts with root privileges (it needs to bind to port 80 after all), the `server.key` file should be mode 400 owned by root.

Once you've generated the server key, you need to use the `openssl` command again to generate the certificate signing request (CSR) that you will be transmitting to the certificate authority (a list of "recognized" CAs can be found at http://httpd.apache.org/docs-2.0/ssl/ssl_faq.html#ToC28). Along with the CSR file, you'll also be sending along a quantity of cash money and some proof of the legitimacy of your business. This magic certificate signing rain dance can actually take up to several weeks depending on the CA you choose, so factor that into your timelines. Eventually you'll get back a signed certificate (`.crt`) file which is what your Web server requires (along with the `server.key` file). Note that your certificate needs to be renewed every year (more money).

Configuring SSL

```
Listen 443
...
<VirtualHost _default_:443>
    ServerName www.sysiphus.com:443
    DocumentRoot "/var/apache/docs-private"
    ...
    SSLEngine on
    SSLCertificateFile /var/apache/ssl/server.crt
    SSLCertificateKeyFile /var/apache/ssl/server.key
    SSLCipherSuite ...
    ...
</VirtualHost>
```

67

Once you've got your `server.key` file and corresponding `server.crt` file, you need to enable SSL support in `httpd.conf` and tell the Web server where these files are installed (again, keep them *out* of the doc trees). The usual configuration (at least for sites with only one Web server) is to have an `httpd` which is listening on port 80 for unencrypted requests and on port 443, which is the standard port for SSL communications (after all, there's no point in shipping around image files via SSL because the encryption adds extra overhead and burns extra CPU cycles on the Web server). There's no problem with having multiple `Listen` directives in `httpd.conf`—the Web server will bind to all of the requested addresses/ports.

In this configuration, SSL support is generally enabled with a `VirtualHost` definition (`VirtualHost` is also used for supporting multiple different Web sites via a single Apache instance). Here we're defining a virtual host which is listening on port 443 on the default list of addresses that the server is also using for its port 80 bindings. The document root for this virtual server is different set of documents than those used for unencrypted traffic (it is permissible to use the same document root, but this seems fraught with peril). Note that the security configuration we set up elsewhere (IP based access controls, password protection, etc.) still applies to the `VirtualHost` directive, so you don't need to repeat all of that again inside of the `VirtualHost` declaration, but you will need to allow access to the new `docs-private` directory since our stance is still "default deny".

We do need to enable SSL (`SSLEngine On`) and tell the server where to find the `server.key` and `server.crt` files. Again the `server.key` file should be only readable by root, but the `server.crt` file needs to be world-readable. There are many, many other SSL options that need to be included, but you can just cop these directly from the sample files provided with the Apache distribution.

Starting Apache w/ SSL

- Server needs to be started with `-DSSL`:

```
/var/apache/bin/httpd -DSSL \  
-f /var/apache/conf/httpd.conf
```

- "`apachectl startssl`" accomplishes the same thing
- Don't forget to update your boot scripts!

68

Once you've got `httpd.conf` all set up, you must still start the Apache server with the `-DSSL` command line argument as shown on the slide. You can also use the `apachectl` program— "`apachectl startssl`"— which accomplishes the same thing. Don't forget to update your boot scripts so the Web server will be started correctly when the system reboots.

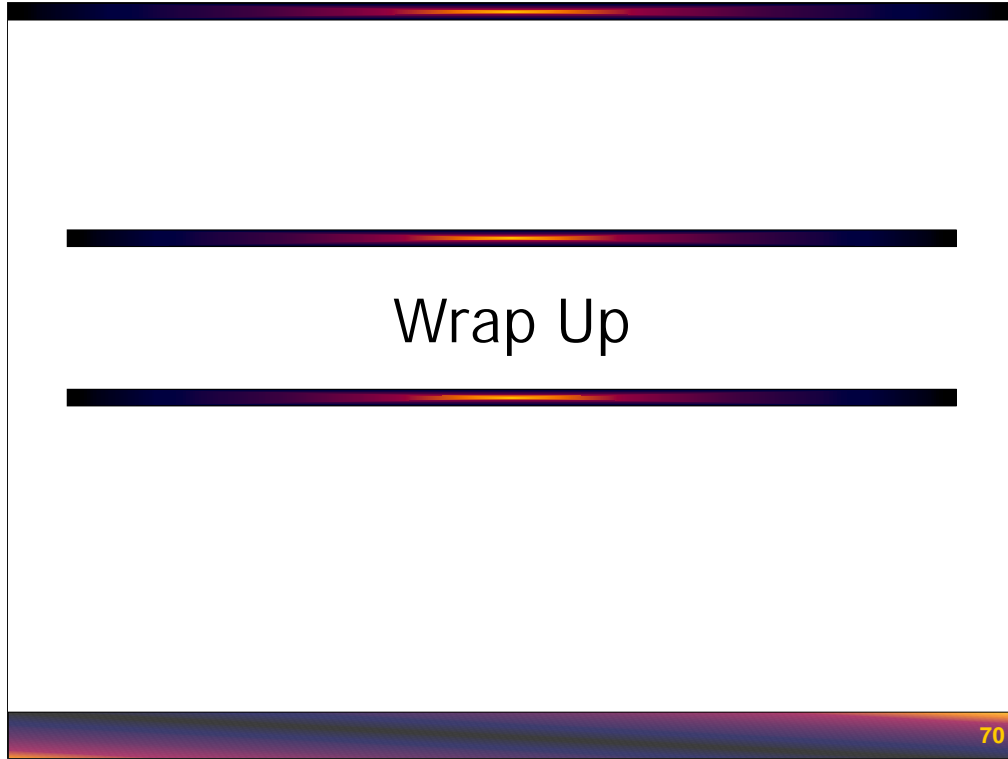
Again, it's sort of a pain that starting up SSL requires a special command line flag. Why can't the settings in `httpd.conf` be enough?

Conclusion

- The material presented here is enough to "get you going"
- Really only scratches the surface, though
- Lots of good docs at www.apache.org
- Keep up-to-date on new releases!

69

While this has been a quick introduction to the basic settings for Apache security, there's a lot of additional information available on general Web server configuration issues. Most of this documentation is linked off of www.apache.org, so take some time to peruse this site. It's also important to check this site regularly to make sure you're up-to-date on the latest Apache releases, because most new versions are driven by security-related bug fixes.



This space intentionally left blank.

That's It!

- Final Q&A?
- Please Fill Out Your Surveys!

This space intentionally left blank.