Information Security Management Handbook
Edited by Harold F. Tipton and Micki Krause
Boca Raton: CRC Press LLC, 2003

Controlling FTP: Providing Secured Data Transfers

Chris Hare, CISSP, CISA

Several scenarios exist that must be considered when looking for a solution:

- The user with a log-in account who requires FTP access to upload or download reports generated by an application. The user does not have access to a shell; rather, his default connection to the box will connect him directly to an application. He requires access to only his home directory to retrieve and delete files.
- The user who uses an application as his shell but does not require FTP access to the system.
- An application that automatically transfers data to a remote system for processing by a second application.

It is necessary to find an elegant solution to each of these problems before that solution can be considered viable by an organization.

Scenario A

A user named Bob accesses a UNIX system through an application that is a replacement for his normal UNIX log-in shell. Bob has no need for, and does not have, direct UNIX command-line access. While using the application, Bob creates reports or other output that he must upload or download for analysis or processing. The application saves this data in either Bob's home directory or a common directory for all application users.

Bob may or may not require the ability to put files onto the application server. The requirements break down as follows:

- Bob requires FTP access to the target server.
- Bob requires access to a restricted number of directories, possibly one or two.
- Bob may or may not require the ability to upload files to the server.

Scenario B

Other application users in the environment illustrated in Scenario A require no FTP access whatsoever. Therefore, it is necessary to prevent them from connecting to the application server using FTP.

Scenario C

The same application used by the users in Scenarios A and B regularly dumps data to move to another system. The use of hard-coded passwords in scripts is not advisable because the scripts must be readable for them to be executed properly. This may expose the passwords to unauthorized users and allow them to access the target system. Additionally, the use of hard-coded passwords makes it difficult to change the password on a regular basis because all scripts using this password must be changed.

A further requirement is to protect the data once stored on the remote system to limit the possibility of unauthorized access, retrieval, and modification of the data.

While there are a large number of options and directives for the `/etc/ftpaccess` file, the focus here is on those that provide secured access to meet the requirements in the scenarios described.

CONTROLLING FTP ACCESS

Advanced FTP servers such as wu-ftpd provide extensive controls for controlling FTP access to the target system. This access does not extend to the IP layer, as the typical FTP client does not offer encryption of the data stream. Rather, FTP relies on the properties inherent in the IP (Internet Protocol) to recover from malformed or lost packets in the data stream. This means one still has no control over the network component of the data transfer. This may allow for the exposure of the data if the network is compromised. However, that is outside the scope of the immediate discussion.

wu-ftpd uses two control files: `/etc/ftpusers` and `/etc/ftpaccess`. The `/etc/ftpusers` file is used to list the users who do **not** have FTP access rights on the remote system. For example, if the `/etc/ftpusers` file is empty, then all users, including root, have FTP rights on the system. This is not the desired operation typically, because access to system accounts such as root are to be controlled. Typically, the `/etc/ftpusers` file contains the following entries:

Exhibit 2-1. Denying FTP access.

```
C:\WINDOWS>ftp 192.168.0.2
Connected to 192.168.0.2.
220 poweredge.home.com FTP server (Version wu-2.6.1(1) Wed
Aug 9
05:54:50 EDT 20
00) ready.
User (192.168.0.2:(none)): root
331 Password required for root.
Password:
530 Login incorrect.
Login failed.
ftp>
```

- root
- bin
- daemon
- adm
- lp
- sync
- shutdown
- halt
- mail
- news
- uucp
- operator
- games
- nobody

When a user in this list, root for example, attempts to access the remote system using FTP, they are denied access because their account is listed in the `/etc/ftpusers` file. This is illustrated in Exhibit 2-1.

By adding additional users to this list, one can control who has FTP access to this server. This does, however, create an additional step in the creation of a user account, but it is a related process and could be added as a step in the script used to create a user. Should a user with FTP privileges no longer require this access, the user's name can be added to the `/etc/ftpusers` list at any time. Similarly, if a denied user requires this access in the future, that user can be removed from the list and FTP access restored.

Recall the requirements of Scenario B: the user has a log-in on the system to access his application but does not have FTP privileges. This scenario has been addressed through the use of `/etc/ftpusers`. The user

Exhibit 2-2. Sample `/etc/ftpaccess` file.

```
class all real,guest,anonymous *

email root@localhost

loginfails 5

readme     README*     login
readme     README*     cwd=*

message /var/ftp/welcome.msg   login
message .message               cwd=*

compress   yes   all
tar        yes   all
chmod      no    guest,anonymous
delete     no    guest,anonymous
overwrite  no    guest,anonymous
rename     no    guest,anonymous

log transfers anonymous,real inbound,outbound

shutdown /etc/shutmsg

passwd-check rfc822 warn
```

can still have UNIX shell access or access to a UNIX-based application through the normal UNIX log-in process. However, using `/etc/ftpusers` prevents access to the FTP server and eliminates the problem of unauthorized data movement to or from the FTP server. Most current FTP server implementations offer the `/etc/ftpusers` feature.

EXTENDING CONTROL

Scenarios A and C require additional configuration because reliance on the extended features of the wu-ftpd server is required. These control extensions are provided in the file `/etc/ftpaccess`. A sample `/etc/ftpaccess` file is shown in Exhibit 2-2. This is the default `/etc/ftpaccess` file distributed with wu-ftpd. Before one can proceed to the problem at hand, one must examine the statements in the `/etc/ftpaccess` file. Additional explanations for other statements not found in this example, but required for the completion of our scenarios, are also presented later in the chapter.

The `class` statement in `/etc/ftpaccess` defines a class of users, in the sample file a user class named `all`, with members of the class being `real`, `guest`, and `anonymous`. The syntax for the class definition is:

```
class <class> <typelist> <addrglob> [<addrglob> ...]
```

`Typelist` is one of `real`, `guest`, and `anonymous`. The `real` keyword matches users to their real user accounts. `Anonymous` matches users who are using anonymous FTP access, while `guest` matches guest account access. Each of these classes can be further defined using other options in this file. Finally, the `class` statement can also identify the list of allowable addresses, hosts, or domains that connections will be accepted from. There can be multiple `class` statements in the file; the first one matching the connection will be used.

Defining the hosts requires additional explanation. The host definition is a domain name, a numeric address, or the name of a file, beginning with a slash ('/') that specifies additional address definitions. Additionally, the address specification may also contain IP `address:netmask` or IP address/CIDR definition. (CIDR, or Classless Internet Domain Routing, uses a value after the IP address to indicate the number of bits used for the network. A Class C address would be written as 192.168.0/24, indicating 24 bits are used for the network.)

It is also possible to exclude users from a particular class using a '!' to negate the test. Care should be taken in using this feature. The results of each of the `class` statements are OR'd together with the others, so it is possible to exclude an allowed user in this manner. However, there are other mechanisms available to deny connections from specific hosts or domains. The primary purpose of the `class` statement is to assign connections from specific domains or types of users to a class. With this in mind, one can interpret the `class` statement in Exhibit 2-2, shown here as:

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

This statement defines a `class` named `all`, which includes user types `real`, `anonymous`, and `guest`. Connections from any host are applicable to this class.

The `email` clause specifies the e-mail address of the FTP archive maintainer. It is printed at various times by the FTP server.

The `message` clause defines a file to be displayed when the user logs in or when they change to a directory. The statement

```
message  /var/ftp/welcome.msg  login
```

causes wu-ftpd to display the contents of the file `/var/ftp/welcome.msg` when a user logs in to the FTP server. It is important for this file to be somewhere accessible to the FTP server so that anonymous users will also be greeted by the message.

NOTE: Some FTP clients have problems with multiline responses, which is how the file is displayed.

When accessing the test FTP server constructed for this chapter, the message file contains:

```
***** WARNING *****
This is a private FTP server. If you do not have an
 account,
you are not welcome here.
******************
It is currently %T local time in Ottawa, Canada.
You are %U@%R accessing %L.
for help, contact %E.
```

The %<char> strings are converted to the actual text when the message is displayed by the server. The result is:

```
331 Password required for chare.
Password:
230-***** WARNING *****
230-This is a private FTP server. If you do not have an
 account,
230-you are not welcome here.
230-******************
230-It is currently Sun Jan 28 18:28:01 2001 local time
 in Ottawa, Canada.
230-You are chare@chris accessing poweredge.home.com.
230-for help, contact root@localhost.
230-
230-
230 User chare logged in.
ftp>
```

The %<char> tags available for inclusion in the message file are listed in Exhibit 2-3.

It is allowable to define a class and attach a specific message to that class of users. For example:

```
class    real                    real        *
class    anon                    anonymous   *
message  /var/ftp/welcome.msg    login       real
```

Now, the message is only displayed when a real user logs in. It is not displayed for either anonymous or guest users. Through this definition, one can provide additional information using other tags listed in Exhibit 2-3. The ability to display class-specific message files can be extended on a

Exhibit 2-3. `%char` definitions.

Tag	Description
`%T`	Local time (form Thu Nov 15 17:12:42 1990)
`%F`	Free space in partition of CWD (kbytes)
`%C`	Current working directory
`%E`	The maintainer's e-mail address as defined in ftpaccess
`%R`	Remote host name
`%L`	Local host name
`%u`	Username as determined via RFC931 authentication
`%U`	Username given at log-in time
`%M`	Maximum allowed number of users in this class
`%N`	Current number of users in this class
`%B`	Absolute limit on disk blocks allocated
`%b`	Preferred limit on disk blocks
`%Q`	Current block count
`%I`	Maximum number of allocated inodes (+1)
`%i`	Preferred inode limit
`%q`	Current number of allocated inodes
`%H`	Time limit for excessive disk use
`%h`	Time limit for excessive files
`%xu`	Uploaded bytes
`%xd`	Downloaded bytes
`%xR`	Upload/download ratio (1:n)
`%xc`	Credit bytes
`%xT`	Time limit (minutes)
`%xE`	Elapsed time since log-in (minutes)
`%xL`	Time left
`%xU`	Upload limit
`%xD`	Download limit

user-by-user basis by creating a `class` for each user. This is important because individual limits can be defined for each user.

The message command can also be used to display information when a user enters a directory. For example, using the statement

```
message /var/ftp/etc/.message CWD=*
```

causes the FTP server to display the specified file when the user enters the directory. This is illustrated in Exhibit 2-4 for the anonymous user. The message itself is displayed only once to prevent annoying the user.

The `noretrieve` directive establishes specific files no user is permitted to retrieve through the FTP server. If the path specification for the file

Exhibit 2-4. Directory-specific messages.

```
User (192.168.0.2:(none)): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd etc
250-***** WARNING *****
250-There is no data of any interest in the /etc directory.
250-
250 CWD command successful.
ftp>
```

begins with a '/', then only those files are marked as nonretrievable. If the file specification does not include the leading '/', then any file with that name cannot be retrieved.

For example, there is a great deal of sensitivity with the password file on most UNIX systems, particularly if that system does not make use of a shadow file. Aside from the password file, there is a long list of other files that should not be retrievable from the system, even if their use is discouraged. The files that should be marked for nonretrieval are files containing the names:

- `passwd`
- `shadow`
- `.profile`
- `.netrc`
- `.rhosts`
- `.cshrc`
- `profile`
- `core`
- `.htaccess`
- `/etc`
- `/bin`
- `/sbin`

This is not a complete list, as the applications running on the system will likely contain other files that should be specifically identified.

Using the `noretrieve` directive follows the syntax:

```
noretrieve [absolute|relative] [class=<classname>] ...
[-] <file-name> <filename> ...
```

For example,

```
noretrieve passwd
```

prevents any user from downloading any file on the system named passwd.

When specifying files, it is also possible to name a directory. In this situation, all files in that directory are marked as nonretrievable. The option `absolute` or `relative` keywords identify if the file or directory is an absolute or relative path from the current environment. The default operation is to consider any file starting with a '/' as an absolute path. Using the optional `class` keyword on the `noretrieve` directive allows this restriction to apply to only certain users. If the `class` keyword is not used, the restriction is placed against all users on the FTP server.

Denying Connections

Connections can be denied based on the IP address or domain of the remote system. Connections can also be denied based on how the user enters his password at log-in.

NOTE: This password check applies only to anonymous FTP users. It has no effect on real users because they authenticate with their standard UNIX password.

The password-check directive informs the FTP server to conduct checks against the password entered. The syntax for the password-check directive is

```
passwd-check <none|trivial|rfc822> (<enforce|warn>)
```

It is not recommended to use `password-check` with the `none` argument because this disables analysis of the entered password and allows meaningless information to be entered. The `trivial` argument performs only checking to see if there is an '@' in the password. Using the argument is the recommended action and ensures the password is compliant with the RFC822 e-mail address standard.

If the password is not compliant with the `trivial` or `rfc822` options, the FTP server can take two actions. The `warn` argument instructs the server to warn the user that his password is not compliant but still allows access. If the `enforce` argument is used, the user is warned and the connection terminated if a noncomplaint password is entered.

Use of the `deny` clause is an effective method of preventing access from specific systems or domains. When a user attempts to connect from the specified system or domain, the message contained in the specified file is displayed. The syntax for the `deny` clause is:

```
deny <addrglob> <message_file>
```

The file location must begin with a slash ('/'). The same rules described in the class section apply to the `addrglob` definition for the `deny` command. In addition, the use of the keyword `!nameservd` is allowed to deny connections from sites without a working nameserver.

Consider adding a `deny` clause to this file; for example, adding `deny!nameservd /var/ftp/.deny` to `/etc/ftpaccess`. When testing the `deny` clause, the denied connection receives the message contained in the file. Using the `!nameservd` definition means that any host not found in a reverse DNS query to get a host name from an IP address is denied access.

```
Connected to 192.168.0.2.
220 poweredge.home.com FTP server (Version wu-2.6.1(1)
 Wed Aug 9 05:54:50 EDT 20
00) ready.
User (192.168.0.2:(none)): anonymous
331 Guest login ok, send your complete e-mail address as
 password.
Password:
530-**** ACCESS DENIED ****
530-
530-Access to this FTP server from your domain has been
 denied by the administrator.
530-
530 Login incorrect.
Login failed.
ftp>
```

The denial of the connection is based on where the connection is coming from, not the user who authenticated to the server.

Connection Management

With specific connections denied, this discussion must focus on how to control the connection when it is permitted. A number of options for the server allow this and establish restrictions from throughput to access to specific files or directories.

Preventing anonymous access to the FTP server is best accomplished by removing the `ftp` user from the `/etc/passwd` file. This instructs the FTP server to deny all anonymous connection requests.

The `guestgroup` and `guestuser` commands work in a similar fashion. In both cases, the session is set up exactly as with anonymous FTP. In other words, a `chroot()` is done and the user is no longer permitted to issue the `USER` and `PASS` commands. If using `guestgroup`, the `groupname` must be defined in the `/etc/group` file; or in the case of `guestuser`, a valid entry in `/etc/passwd`.

```
guestgroup <groupname> [<groupname> ...]
guestuser <username> [<username> ...]
realgroup <groupname> [<groupname> ...]
realuser <username> [<username> ...]
```

In both cases, the user's home directory must be correctly set up. This is accomplished by splitting the home directory entry into two components separated by the characters `'/./'`. The first component is the base directory for the FTP server and the second component is the directory the user is to be placed in. The user can enter the base FTP directory but cannot see any files above this in the file system because the FTP server establishes a restricted environment.

Consider the `/etc/passwd` entry:

```
systemx:<passwd>:503:503:FTP Only Access from
systemx:/var/ftp/./systemx:/etc/ftponly
```

When `systemx` successfully logs in, the FTP server will `chroot("/var/ftp")` and then `chdir("/systemx")`. The guest user will only be able to access the directory structure under `/var/ftp` (which will look and act as `/` to systemx), just as an anonymous FTP user would.

Either an actual name or numeric ID specifies the group name. To use a numeric group ID, place a `'%'` before the number. Ranges may be given and the use of an asterisk means all groups. `guestuser` works like `guestgroup` except uses the username (or numeric ID).

`realuser` and `realgroup` have the same syntax but reverse the effect of `guestuser` and `guestgroup`. They allow real user access when the remote user would otherwise be determined a guest. For example:

```
guestuser *
realuser chare
```

causes all nonanonymous users to be treated as `guest`, with the sole exception of user chare, who is permitted real user access. Bear in mind, however, that the use of `/etc/ftpusers` overrides this directive. If the user is listed in `/etc/ftpusers`, he is denied access to the FTP server.

It is also advisable to set timeouts for the FTP server to control the connection and terminate it appropriately. The timeout directives are listed in Exhibit 2-5. The `accept` timeout establishes how long the FTP server will wait for an incoming connection. The default is 120 seconds. The `connect` value establishes how long the FTP server will wait to establish an outgoing connection. The FTP server generally makes several attempts and will give up after the defined period if a successful connection cannot be established.

The data timeout determines how long the FTP server will wait for some activity on the data connection. This should be kept relatively long because the remote client may have a low-speed link and there may be a lot of data queued for transmission. The idle timer establishes how long the

Exhibit 2-5. Timeout directives.

Timeout Value	Default	Recommended
Timeout accept <seconds>	120	120
Timeout connect <seconds>	120	120
Timeout data <seconds>	1200	1200
Timeout idle <seconds>	900	900
Timeout maxidle <seconds>	7200	1200
Timeout RFC931 <seconds>	10	10

server will wait for the next command from the client. This can be overridden with the `-a` option to the server. Using the `access` clause overrides both the command line parameter if used and the default.

The user can also use the `SITE IDLE` command to establish a higher value for the idle timeout. The `maxidle` value establishes the maximum value that can be established by the FTP client. The default is 7200 seconds. Like the idle timeout, the default can be overridden using the `-A` command line option to the FTP server. Defining this parameter overrides the default and the command line. The last timeout value allows the maximum time for the RFC931 `ident/AUTH` conversation to occur. The information recorded from the RFC931 conversation is recorded in the system logs and used for any authentication requests.

Controlling File Permissions

File permissions in the UNIX environment are generally the only method available to control who has access to a specific file and what they are permitted to do with that file. It may be a requirement of a specific implementation to restrict the file permissions on the system to match the requirements for a specific class of users.

The `defumask` directive allows the administrator to define the umask, or default permissions, on a per-class or systemwide basis. Using the `defumask` command as

```
defumask 077
```

causes the server to remove all permissions except for the owner of the file. If running a general access FTP server, the use of a 077 umask may be extreme. However, umask should be at least 022 to prevent modification of the files by other than the owner.

By specifying a class of user following the umask, as in

```
defumask 077 real
```

all permissions are removed. Using these parameters prevents world writable files from being transferred to your FTP server. If required, it is possible

to set additional controls to allow or disallow the use of other commands on the FTP server to change file permissions or affect the files. By default, users are allowed to change file permissions and delete, rename, and overwrite files. They are also allowed to change the umask applied to files they upload. These commands allow or restrict users from performing these activities.

```
chmod <yes|no> <typelist>
delete <yes|no> <typelist>
overwrite <yes|no> <typelist>
rename <yes|no> <typelist>
umask <yes|no> <typelist>
```

To restrict all users from using these commands, apply the directives as:

```
chmod no all
delete no all
overwrite no all
rename no all
umask no all
```

Setting these directives means no one can execute commands on the FTP server that require these privileges. This means the FTP server and the files therein are under the full control of the administrator.

ADDITIONAL SECURITY FEATURES

There are a wealth of additional security features that should be considered when configuring the server. These control how much information users are shown when they log in about the server, and print banner messages, among other capabilities.

The `greeting` directive informs the FTP server to change the level of information printed when the user logs in. The default is `full`, which prints all information about the server. A `full` message is:

```
220 poweredge.home.com FTP server (Version wu-2.6.1(1)
Wed Aug 9 05:54:50 EDT 2000) ready.
```

A `brief` message on connection prints the server name as:

```
220 poweredge.home.com FTP server ready.
```

Finally, the `terse` message, which is the preferred choice, prints only:

```
220 FTP server ready.
```

The `full` greeting is the default unless the greeting directive is defined. This provides the most information about the FTP server. The `terse` greeting is the preferred choice because it provides no information about the server to allow an attacker to use that information for identifying potential attacks against the server.

The greeting is controlled with the directive:

```
greeting <full|brief|terse>
```

An additional safeguard is the banner directive using the format:

```
banner <path>
```

This causes the text contained in the named file to be presented when the users connect to the server prior to entering their username and password. The path of the file is relative from the real root directory, not from the anonymous FTP directory. If one has a corporate log-in banner that is displayed when connecting to a system using Telnet, it would also be available to use here to indicate that the FTP server is for authorized users only.

NOTE: Use of this command can completely prevent noncompliant FTP clients from establishing a connection. This is because not all clients can correctly handle multiline responses, which is how the banner is displayed.

```
Connected to 192.168.0.2.
220-************************************************************
220-*                                                          *
220-*                  * W A R N I N G *                       *
220-*                                                          *
220-* ACCESS TO THIS FTP SERVER IS FOR AUTHORIZED USERS ONLY.   *
220-* ALL ACCESS IS LOGGED AND MONITORED. IF YOU ARE NOT AN     *
220-* AUTHORIZED USER, OR DO NOT AGREE TO OUR MONITORING POLICY,*
220-* DISCONNECT NOW.                                          *
220-*                                                          *
220-* NO ABUSE OR UNAUTHORIZED ACCESS IS TOLERATED.            *
220-*                                                          *
220-************************************************************
220-
220 FTP server ready.
User (192.168.0.2:(none)):
```

At this point, one has controlled how the remote user gains access to the FTP server, and restricted the commands they can execute and the permissions assigned to their files. Additionally, certain steps have been taken to ensure they are aware that access to this FTP server is for authorized use only. However, one must also take steps to record the connections and transfers made by users to fully establish what is being done on the FTP server.

LOGGING CAPABILITIES

Recording information in the system logs is a requirement for proper monitoring of transfers and activities conducted on the FTP server. There are a number of commands that affect logging, and each is presented in this section. Normally, only connections to the FTP server are logged. However, using the `log commands` directive, each command executed by the

user can be captured. This may create a high level of output on a busy FTP server and may not be required. However, it may be advisable to capture traffic for anonymous and guest users specifically. The directive syntax is:

```
log commands <typelist>
```

As with other directives, it is known that `typelist` is a combination of `real`, `anonymous`, and `guest`. If the `real` keyword is used, logging is done for users accessing FTP using their real accounts. `Anonymous` logs all commands performed by anonymous users, while `guest` matches users identified using the `guestgroup` or `guestuser` directives.

Consider the line

```
log commands guest, anonymous
```

which results in all commands performed by anonymous and guest users being logged. This can be useful for later analysis to see if automated jobs are being properly performed and what files are uploaded or downloaded.

Like the `log commands` directive, `log transfers` performs a similar function, except that it records all file transfers for a given class of users. The directive is stated as:

```
log transfers <typelist> <directions>
```

The `directions` argument is inbound or outbound. Both arguments can be used to specify logging of transfers in both directions. For clarity, `inbound` are files transferred to the server, or uploads, and `outbound` are transfers from the server, or downloads. The `typelist` argument again consists of `real`, `anonymous`, and `guest`.

It is not only essential to log all of the authorized functions, but also to record the various commands and requests made by the user that are denied due to security requirements. For example, if there are restrictions placed on retrieving the `password` file, it is desirable to record the security events. This is accomplished for `real`, `anonymous`, and `guest` users using the `log security` directive, as in:

```
log security <typelist>
```

If `rename` is a restricted command on the FTP server, the `log security` directive results in the following entries

```
Feb 11 20:44:02 poweredge ftpd[23516]: RNFR dayo.wav
Feb 11 20:44:02 poweredge ftpd[23516]: RNTO day-o.wav
Feb 11 20:44:02 poweredge ftpd[23516]: systemx of
 localhost.home.com [127.0.0.1]
tried to rename /var/ftp/systemx/dayo.wav to /var/ftp/
 systemx/day-o.wav
```

This identifies the user who tried to rename the file, the host that the user connected from, and the original and desired filenames. With this information, the

system administrator or systems security personnel can investigate the situation.

Downloading information from the FTP server is controlled with the `noretrieve` clause in the `/etc/ftpaccess` file. It is also possible to limit uploads to specific directories. This may not be required, depending on the system configuration. A separate entry for each directory one wishes to allow uploads to is highly recommended. The syntax is:

```
upload [absolute|relative] [class=<classname>]... [-]
<root-dir> <dirglob> <yes|no> <owner> <group> <mode>
["dirs"|"nodirs"] [<d_mode>]
```

This looks overly complicated, but it is in fact relatively simple. Define a directory called `<dirglob>` that permits or denies uploads. Consider the following entry:

```
upload /var/ftp /incoming yes ftpadmin ftpadmin 0440
nodirs
```

This means that for a user with the home directory of `/var/ftp`, allow uploads to the incoming directory. Change the owner and group to be `ftpadmin` and change the permissions to `readonly`. Finally, do not allow the creation of directories. In this manner, users can be restricted to the directories to which they can upload files. Directory creation is allowed by default, so one must disable it if required.

For example, if one has a user on the system with the following password file entry:

```
chare:x:500:500:Chris Hare:/home/chare:/bin/bash
```

and if one wants to prevent the person with this `userid` from being able to upload files to his home directory, simply add the line:

```
upload /home/chare no
```

to the `/etc/ftpaccess` file. This prevents the user `chare` from being able to upload files to his home directory. However, bear in mind that this has little effect if this is a real user, because real users will be able to upload files to any directory they have write permission to. The `upload` clause is best used with anonymous and guest users.

Note: The wu-ftpd server denies anonymous uploads by default.

To see the full effect of the `upload` clause, one must combine its use with a guest account, as illustrated with the systemx account shown here:

```
systemx:x:503:503:FTP access from System X:/home/
systemx/./:/bin/false
```

Note in this password file entry the home directory path. This entry cannot be made when the user account is created. The `'/./'` is used by wu-ftpd to establish the `chroot` environment. In this case, the user is placed into his home directory, `/home/systemx`, which is then used as the base for his `chroot` file system. At this point, the guest user can see nothing on the system other than what is in his home directory.

Using the `upload` clause of

```
upload /home/chare yes
```

means the user can upload files to his home directory. When coupled with the `noretrieve` clause discussed earlier, it is possible to put a high degree of control around the user.

THE COMPLETE /etc/ftpaccess FILE

The discussion thus far has focused on a number of control directives available in the wu-ftpd FTP server. It is not necessary that these directives appear in any particular order. However, to further demonstrate the directives and relationships between those directives, the `/etc/ftpaccess` file is illustrated in Exhibit 2-6.

REVISITING THE SCENARIOS

Recall the scenarios from the beginning of this chapter. This section reviews each scenario and defines an example configuration to achieve it.

Scenario A

A user named Bob accesses a UNIX system through an application that is a replacement for his normal UNIX log-in shell. Bob has no need for, and does not have, direct UNIX command-line access. While using the application, Bob creates reports or other output that he must retrieve for analysis. The application saves this data in either Bob's home directory or a common directory for all application users.

Bob may or may not require the ability to put files onto the application server. The requirements break down as follows:

- Bob requires FTP access to the target server.
- Bob requires access to a restricted number of directories, possibly one or two.
- Bob may or may not require the ability to upload files to the server.

Bob requires the ability to log into the FTP and access several directories to retrieve files. The easiest way to do this is to deny retrieval for the entire system by adding a line to `/etc/ftpaccess` as

```
noretrieve /
```

Exhibit 2-6. The `/etc/ftpaccess` file.

```
#
# Define the user classes
#
class   all          real,guest     *
class   anonymous    anonymous      *
class   real         real           *
#
# Deny connections from systems with no reverse DNS
# deny !nameservd /var/ftp/.deny
#
# What is the email address of the server administrator.
# Make sure someone reads this from time to time.
email root@localhost
#
# How many login attempts can be made before logging an
# error message and terminating the connection?
#
loginfails 5
greeting terse

readme   README*   login
readme   README*   cwd=*
#
# Display the following message at login
#
message /var/ftp/welcome.msg               login
banner /var/ftp/warning.msg
#
# display the following message when entering the directory
#
message .message                            cwd=*

#
# ACCESS CONTROLS
#
# What is the default umask to apply if no other matching
# directive exists
#
defumask 022
chmod       no         guest,anonymous
delete      no         guest,anonymous
overwrite   no         guest,anonymous
rename      no         guest,anonymous
# remove all permissions except for the owner if the user
# is a member of the real class
#
defumask 077    real
guestuser       systemx
realuser        chare
#

#establish timeouts
#
```

Exhibit 2-6. The `/etc/ftpaccess` file (Continued).

```
timeout    accept 120
timeout    connect 120
timeout    data 1200
timeout    idle 900
timeout    maxidel 1200

#
# establish non-retrieval
#
# noretrieve passwd
# noretrieve shadow
# noretrieve .profile
# noretrieve .netrc
# noretrieve .rhosts
# noretrieve .cshrc
# noretrieve profile
# noretrieve core
# noretrieve .htaccess
# noretrieve /etc
# noretrieve /bin
# noretrieve /sbin
noretrieve /
allow-retrieve /tmp

upload /home/systemx / no

#
# Logging
#
log commands anonymous,guest,real
log transfers anonymous,guest,real inbound,outbound
log security anonymous,real,guest

compress   yes   all
tar        yes   all

shutdown /etc/shutmsg

passwd-check rfc822 warn
```

This marks every file and directory as nonretrievable. To allow Bob to get the files he needs, one must set those files or directories as such. This is done using the `allow-retrieve` directive. It has exactly the same syntax as the `noretrieve` directive, except that the file or directory is now retrievable. Assume that Bob needs to retrieve files from the `/tmp` directory. Allow this using the directive

```
allow-retrieve /tmp
```

When Bob connects to the FTP server and authenticates himself, he cannot get files from his home directory.

```
ftp> pwd
257 "/home/bob" is current directory.
ftp> get .xauth xauth
200 PORT command successful.
550 /home/chare/.xauth is marked unretrievable
```

However, Bob can retrieve files from the `/tmp` directory.

```
ftp> cd /tmp
250 CWD command successful.
ftp> pwd
257 "/tmp" is current directory.
ftp> get .X0-lock X0lock
200 PORT command successful.
150 Opening ASCII mode data connection for .X0-lock
 (11 bytes).
226 Transfer complete.
ftp: 12 bytes received in 0.00Seconds 12000.00Kbytes/sec.
ftp>
```

If Bob must be able to retrieve files from his home directory, an additional `allow-retrieve` directive is required:

```
class real real *
allow-retrieve /home/bob class=real
```

When Bob tries to retrieve a file from anywhere other than `/tmp` or his home directory, access is denied.

Additionally, it may be necessary to limit Bob's ability to upload files. If a user requires the ability to upload files, no additional configuration is required, as the default action for the FTP server is to allow uploads for real users. If one wants to prohibit uploads to Bob's home directory, use the upload directive:

```
upload /home/bob / no
```

This command allows uploads to the FTP server.

The objective of Scenario A has been achieved.

Scenario B

Other application users in the environment illustrated in Scenario A require no FTP access whatsoever. Therefore, it is necessary to prevent them from connecting to the application server using FTP.

This is done by adding those users to the /etc/ftpaccess file. Recall that this file lists a single user per line, which is checked. Additionally, it may be advisable to deny anonymous FTP access.

Scenario C

The same application used by the users in Scenarios A and B regularly dumps data to move to another system. The use of hard-coded passwords in scripts is not advisable because the scripts must be readable for them to be executed properly. This may expose the passwords to unauthorized users and allow them to access the target system. Additionally, the use of hard-coded passwords makes it difficult to change the password on a regular basis because all scripts using this password must be changed.

A further requirement is to protect the data once stored on the remote system to limit the possibility of unauthorized access, retrieval, and modification of the data.

Accomplishing this requires the creation of a guest user account on the system. This account will not support a log-in and will be restricted in its FTP abilities. For example, create a UNIX account on the FTP server using the source hostname, such as systemx. The password is established as a complex string but with the other compensating controls, the protection on the password itself does not need to be as stringent. Recall from an earlier discussion that the account resembles

```
systemx:x:503:503:FTP access from System X:/home/
   systemx/./:/bin/false
```

Also recall that the home directory establishes the real user home directory, and the ftp chroot directory. Using the upload command

```
upload /home/systemx / no
```

means that the systemx user cannot upload files to the home directory. However, this is not the desired function in this case. In this scenario, one wants to allow the remote system to transfer files to the FTP server. However, one does not want to allow for downloads from the FTP server. To do this, the command

```
noretrieve /
upload /home/systemx / yes
```

prevents downloads and allows uploads to the FTP server.

One can further restrict access by controlling the ability to rename, overwite, change permissions, and delete a file using the appropriate directives in the /etc/ftpaccess file:

```
chmod       no   guest,anonymous
delete      no   guest,anonymous
overwrite   no   guest,anonymous
rename      no   guest,anonymous
```

Because the user account has no interactive privileges on the system and has restricted privileges on the FTP server, there is little risk involved with using a hard-coded password. While using a hard-coded password is not considered advisable, there are sufficient controls in place to compensate for this. Consider the following controls protecting the access:

The user cannot retrieve files from the system.
The user can upload files.
The user cannot see what files are on the system and thus cannot determine the names of the files to block the system from putting the correct data on the server.
The user cannot change file permissions.
The user cannot delete files.
The user cannot overwrite existing files.
The user cannot rename files.
The user cannot establish an interactive session.
FTP access is logged.

With these compensating controls to address the final possibility of access to the system and the data using a password attack or by guessing the password, it will be sufficiently difficult to compromise the integrity of the data.

The requirements defined in the scenario have been fulfilled.

SUMMARY

This discussion has shown how one can control access to an FTP server and allow controlled access for downloads or uploads to permit the safe exchange of information for interactive and automated FTP sessions. The extended functionality offered by the wu-ftpd FTP server provides extensive access, and preventative and detective controls to limit who can access the FTP server, what they can do when they can connect, and the recording of their actions.

ABOUT THE AUTHOR

Chris Hare, CISSP, CISA, is an information security and control consultant with Nortel Networks in Dallas, Texas. A frequent speaker and author, his experience includes application design, quality assurance, systems administration and engineering, network analysis, and security consulting, operations, and architecture.