Setting Up wuftpd for Non−Anonymous Accounts

By Glenn Fleishman

Table of Contents

Introduction

Tecleo aquí para una traducción española!

For some reason, it's extremely hard to find all the documentation you need to easily set up wuftpd to allow FTP to semi−secure areas of a Unix filesystem. It's relatively simple to setup anonymous FTP, where the user has little or no access. But there are many occasions where you might want to allow users to have access to, for instance, Web site directories without allowing them to get into higher levels.

There's only a few things you have to do, but failure to do any one of them results in frustration and failure.

I'm eager to keep improving this as a simplified account of how to set up an FTP server, so please write me with comments or improvements.

Install the Latest Version
Wuftpd (originally maintained by Washington University in St. Louis) is the standard FTP server used on Linux and most Unix boxes. It has great flexibility and configuration, but languished for lack of updates for quite a while, though sporadic attempts were made to maintain the last beta (from 1995 or so).

However, a new group is spearheading regular releases fixing security holes and bugs, and making wuftpd more compatible with modern operating system versions. Before proceeding below, get the latest "VR" release at ftp://ftp.vr.net/pub/wu−ftpd. Download the release named "...−vrXX.tar..." at the end.

A 2.5 release is expected soon, as is a site dedicated to wuftpd. The VR releases don't use the Red Hat "rpm" system, but are easy enough to make and install.

Setup an FTP user account for each user
This has to be separate from a regular user account with unlimited access, because of how the "chroot" environment works. Chroot makes it appear from the user's perspective as if the level of the filesystem you've palced them in is the top level of the file system. In /etc/password, you add a line like

```
frogstar:/ZFp0Rfh7B8PE:5035:2010::/usr/www/./frogstar/:/bin/noshell
```

The account is frogstar, but you'll notice the path to the home directory is a bit odd. The first part − /usr/www/ − indicates the filesystem that should be considered their new root. The dot divides that from the directory they should be automatically chdir (change directory'd) into, /frogstar/.

The /bin/noshell part disables their login as a regular user. Some Unix systems already have a null shell, so use that instead. Or, create a file in the location /bin/noshell

```
#!/bin/sh echo "You don't have login access"
```

Remember to add /bin/noshell to the /etc/shells file on a line by itself. If /etc/shells doesn't exist, create it, and add all your normal shells plus /bin/noshell.

User environment

Setup a chroot user environment for each location
This is a bit trickier. What you're essentially doing is creating a skeleton root filesystem with enough of the pieces necessary (libraries, password files, etc.) to allow Unix to do a `chroot` when the user logs in.

I find this process very weird, in some ways, as you have to create a number of odd files. First create all the directories. For the purposes of this example, let's assume your true root is "`/usr/www`". So first, `cd` to `/usr/www`. Then

---

$ mkdir bin $ mkdir dev $ mkdir etc $ mkdir usr $ mkdir usr/lib

---

Chmod them all to 0555:

$ chmod 0555 {bin,dev,etc,usr,usr/lib}

Next, you have to populate each of these areas. Copy `/bin/ls` and `/bin/more` to /usrw/ww/bin. `chmod` their permissions to 111. (You don't want users to be able to modify the binaries.)

Cd to `dev`. You'll need to create a `zero` file so that Unix can make zeroes. Don't ask me − read the `man` page. A simple command creates it:

$ mknod −m 666 zero c 1 5 $ chown root.mem zero

Next, figure out what libraries you need to copy to make `ls` and `more` work. You can do this by entering `ldd ls` and `ldd more`. Whatever libraries they use, copy them to `/usr/www/usr/lib`.

Finally, create the `group` and `passwd` files in `etc`. This should *not* be the same as your true ones. The passwd file in the `chroot` environment should have entries like:

---

root:*:0:0:Root:/:/bin/noshell frogstar:*:5035:2010::/frogstar/:/bin/noshell

---

You shouldn't include passwords − just use a * to prevent a password from being used. Users will be able to read this file, but it's really meaningless. It's just needed by the ftp daemon. The group file should correspond to your normal group file.

---

frogstars:*:2010:frogstar

If you use the same UID and GID (user and group ids), then `ls` will correctly display ownership. You want to create groups, even with a single member, to use the `guestgroup` directive in the `ftpaccess` file described below.

You should now be set to allow users to FTP into this environment. If you want to create totally protected environments, you need to go down a level for each Web directory and create all of these files − there may be an easier way, but I don't know of it.

The /etc/ftpaccess file

Setup the /etc/ftpaccess file
The `ftpaccess` file controls who gets into the FTP server and how. There are a couple of associated files (like `ftpusers`) that you can read `man` pages on to get more detail, but that documentation is actually quite clear, so I won't reiterate it here.

In `ftpaccess`, you're creating a plan for who can do what. With non−anonymous chroot access, you want to create a set of guestgroups, each of which corresponds directly to entries in the `/etc/group` file. You should repeat these entries in the chroot'd group file, too, with the same IDs.

Here are the lines you need to make sure are in the `ftpaccess` file to make this all work. Let's assume three groups: `frogstar`, `bilbo`, and `foobar`. I'm only including the parts that are specifically necessary to make non−anonymous access work; you can leave the rest of the model file the same, or read the `man` page to figure out how to further customize.

First, you set class to allow local and remote access from all addresses. You can tune this as needed, including removing anonymous remote access.

```
class local real,guest * class remote real,guest,anonymous *
```

Then you specify all of your guestgroups, one per line. The `/etc/group` file has entries for each of these groups, each of which has just one member: the singular form of the name. That is, the `bilbos` group comprises just `bilbo`.

```
guestgroup frogstars guestgroup bilbos guestgroup foobars
```

You probably should log all transfers for security purposes.

```
log commands real,anonymous,guest log transfers guest,anonymous,real inbound,outbound
```

If you don't specify the following directives, they default to yes for everybody. What you're doing here is giving permission for these guestgroups to delete, overwrite, and rename files, and you're allowing everybody but anonymous to chmod or use umask.

```
delete yes frogstars,bilbos,foobars overwrite yes frogstars,bilbos,foobars rename yes frogstars,bilbos,foobars chmod no anonymous umask no anonymous
```

You also have to explicitly allow upload permission by group and directory. This is a further security protection, so that even if your users can view the contents of other directories, they can't upload into them

even if they have write permission. (Study the `man` page for all the options on this line.)

---

upload /usr/www /frogstar yes root frogstars 0664 dirs upload /usr/www /bilbo yes root bilbos 0664 dirs upload /usr/www /foobar yes root foobars 0664 dirs

---

That's it! Enjoy! Write me with comments or improvements.