Insider Tips: ProFTPD

Know-How Transfer

Transferring files is one of the major goals of free software – traditional FTP (File Transfer Protocol) tools do this job very efficiently. Admins wanting to set up their own FTP servers need a lot of background knowledge to avoid the typical pitfalls.

BY MARC ANDRÉ SELIG

Numerous protocols are available for distributing files across the Internet, but not all of them are useful in any given scenario. Popular peer-to-peer services, such as Gnutella, E-Donkey, and cohorts, were created to handle file sharing scenarios, where each participant needs to offer and receive files. Free software distribution is a different ball game. A small number of suppliers (the developers) need to cater for a large number of users. The protocols typically used to handle this situation are HTTP and FTP.

HTTP is the obvious choice for simple tasks. Thanks to Apache, a suitable server is easy to find, and almost any computer will have an easy-to-use client installed. Unfortunately, HTTP lacks a few features that users require to download large files across noisy lines.

The biggest complaint is the way HTTP handles broken connections. Although the HTTP 1.1 specification does envisage a resume function for interrupted transfers, most browsers cannot handle this, and thus lose a lot of useful bandwidth and time. Additionally, running a Web server is expensive in terms of resources, despite the availability of modern software.

Most of these issues do not affect FTP [1]. It is important to understand that FTP is not an extension or a revised version of HTTP, but an independent protocol, which is actually far older.

Installation

Many Web servers rely on an FTP daemon being available. If you are spoilt for choice of daemons, refer to the "Major FTP Daemons" box for an overview of the most widespread FTP servers for Linux. This article will be focussing on one of the most powerful, secure, and easy-to-learn daemons, ProFTPD.

Some current distributions take the headaches out of installing an FTP daemon, providing the required packages. If not, you can look forward to a few manual steps, starting with downloading the

Major FTP Daemons

Admins are spoilt for choice of FTP daemons. The following list introduces four established and well-known representatives of this server class:

BSD-FTPd: The granddaddy of many Linux FTP servers, BSD-FTPd also inspired many other development projects. The daemon provides more than adequate support for systems with low traffic volumes, and is included with many Linux distributions for this reason. Don't expect this daemon to handle more complex tasks or heavier loads.

wu-ftpd: The Washington University FTP daemon [2] is the implementation that most closely resembles BSD-FTPd of any major FTP daemons today. wu-ftpd provides a wide range of configuration options, and integrates seamlessly with Unix systems. The daemon has been affected by a series of vulnerabilities in the past few years, but by now, these should be a thing of the past.

NcFTPd: This commercial server [3] is available free of charge for home and educational use. Although the source code is proprietary, the daemon is still quite popular, as it installs automatically. Also, the server can handle multiple simultaneous sessions very quickly.

Pro FTPD: Just like NcFTPd, ProFTPD [4], which we will be looking at later in this article, can be run as a stand-alone server; this makes it more responsive than its competitors. When the server accepts an incoming connection, it does not need to launch a new process to handle the session. However, you can use *inetd* to launch the FTP daemon, if required. ProFTPD is Open Source, easy to configure, and can be hardened.

sources, and their digital signatures from [4]. Make sure that you check the signatures: Listing 1 shows the steps. First, run *md5sum* to verify the MD5 checksum. This ensures that the package is has not been damaged in transit, and avoids unexpected errors.

Testing the GPG signature is more important, however. In contrast to the MD5 checksum, this verifies the authenticity of the program.

You should note that the example does not provide any indication that the public key received from a key server actually belongs to the programmer of ProFTPd. Ideally, there would be a web of trust verifying the key. For the moment, you must accept this slight uncertainty.

You can then follow the normal steps to compile the sources and install the binaries on Linux. Listing 2 provides an overview of the steps. The configure options in line 3 are of interest in this case.

Listing 1: Checking Signatures

```
mas@ishi:/tmp> md5sum -c proftpd-1.2.9.tar.bz2.md5
proftpd-1.2.9.tar.bz2: OK
mas@ishi:/tmp> gpg proftpd-1.2.9.tar.bz2.asc
gpg: Signature made Fri Oct 31 09:39:42 2003 CET using DSA key ID
A511976A
gpg: Good signature from "TJ Saunders <tj@castaglia.org>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
owner.
Primary key fingerprint: 697E 684D 1668 D696 8428  405C B78E 893F A511
976A
mas@ishi:/tmp>
```

Users and Directories

A few more steps are required to tie up the installation. First check if you have an account for the daemon. ProFTPD typically uses three accounts, each of which is assigned to a specific group. Most of the code uses the *nobody* user account, and the *nogroup* group. The daemon requires root privileges for some tasks, launching the server being one of

The File Transfer Protocol

Browser users will not notice much difference between FTP and HTTP connections, although the underlying techniques are quite dissimilar.

FTP and User Names

Each FTP session starts off with a username and password-based login. As the protocol transmits these credentials in the clear, it makes sense to use different credentials for FTP than for your normal account. Having to assign an account for each visitor who wants to upload or download files, would cause a lot of administrative overhead. This has led to a convention honored by most FTP servers: a guest account called *anonymous* or *ftp*. FTP servers will typically accept any email address as a password for this account; the *username@* string is interpreted as meaning "username on this host".

Ports

Every network service uses one or multiple ports. For example, HTTP uses TCP port 80. The server keeps port 80 open and listens for incoming connections on that port. Clients can use any port on their local host to open up a connection to port 80 on the server. A lot of protocols use simple assignments like these.

This is not so straightforward in the case of FTP. As Figure 1 shows, the protocol distinguishes between the control connection (for commands) and data connections, which actually transfer files. The data connection is also responsible for directory lists and the like.

The control connection reflects the traditional client/server model and uses TCP port 21. The client decides how the data connection will be handled. It can request active or passive mode transfers. In active mode, the client will open an arbitrary local port and use the control channel to tell the server the port number. The server then uses port 20 to actively open a TCP connection to the port specified by the client. The connection is thus opened in the opposite direction to what one might expect.

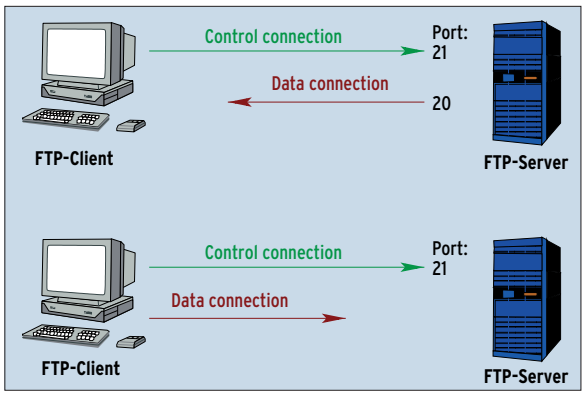

Figure 1: FTP uses a control connection and one, or multiple, data connections. In active mode (top), the server opens the data channel; in passive mode (bottom), the client opens the data channel.

Command line based FTP clients typically use active mode. Passive connections are the domain of Web browsers. Mozilla, Opera, or even Internet Explorer all ask the server to specify a port for the data connection. The server responds with the port number, and the client then opens up a second connection from an arbitrary local port to the port specified by the server. The control connection remains in place, no matter what status the data connection might have.

Watch Out for Firewall Rules

You need to be aware of this unusual architecture when installing an FTP server. Many distributions automatically set up IP filters that will block additional ports. Only the active mode data connection specified by the server will be permitted by default, but the data connection is useless if a control connection cannot be established. FTP admins might like to modify their firewall configurations, and specify suitable filter rules. Assuming a kernel 2.5, these will be as follows:

```
iptables -A INPUT  -p tcp ➥
--sport 1024:  --dport 21 ➥
-j ACCEPT
iptables -A INPUT  -m state ➥
--state ESTABLISHED,RELATED ➥
-j ACCEPT
iptables -A OUTPUT -m state ➥
--state ESTABLISHED,RELATED ➥
-j ACCEPT
```

Listing 2: Installing ProFTPD

```
$ bzip2 -cd proftpd-1.2.9.tar.bz2 | tar xf -
$ cd proftpd-1.2.9
$ ./configure --sysconfdir=/etc/proftpd --➥
localstatedir=/var/run
[...]
$ make
[...]
$ su
Password:
# make install
```

Listing 3: FTP Home Directory

```
# chmod 733 ~ftp/incoming
# ls -la ~ftp
total 20
drwxr-xr-x   4 root   root   4096 Jan 18 18:58 .
drwxr-xr-x   4 root   root   4096 Jan 18 18:55 ..
drwx-wx-wx   2 root   root   4096 Jan 18 18:55 incoming
drwxr-xr-x   2 root   root   4096 Jan 18 18:55 pub
-rw-r--r--   1 root   root     90 Jan 18 18:58 welcome.msg
#
```

them. Pro-FTP uses a guest account to handle typical anonymous logins. Install a user account for *ftp* with the *ftp* group to take care of this.

The anonymous user needs a home directory, and this is where to store files for public access. Note that the daemon should be able to read these files, although it will not be the owner of the files. ProFTPD can be configured to enforce write-protection, but it makes a lot of sense to observe security best practices in this case.

If needed, you can also create an upload directory. Most servers use a directory called */incoming*, or */pub/incoming*. Assign access permissions to allow FTP to write to this directory, but not read from it. This prevents malevolent hackers from misusing the directory as a temporary repository on the Web. After all, who knows what kind of files might otherwise end up on your server, or what legal repercussions this might have?

The final directory tree for *~ftp* should resemble the structure shown in Listing 3.

Configuring and Enabling the Server

Your next step is to check the default configuration file, */etc/proftpd/proftpd.conf*. ProFTPD 1.2.9 installs a sensible minimal version, unless you have a configuration file from a previous installation. The file has a similar syntax to that used by Apache, with the aim of being as intuitive as possible. The sample file assumes anonymous downloads only. If you want to allow uploading, you will need to add a few entries. The Linux Magazine FTP server has a sample *proftpd.conf* that supports both uploads and downloads [6].

Launching

There are two possible alternatives to launching your FTP server when booting your machine. You can either use a centralized approach with *inetd* or *xinetd*, or run the daemon as a stand-alone background process, allowing it to handle incoming connections independently. We have already looked at the first variant in a previous issue of Admin Workshop [7]. So let's look at the second one this time. Despite the resource overhead, this approach will mean a quicker response for users, which may be a benefit to your organization.

Linux systems typically use so-called init scripts to launch services (and other system components). These scripts evaluate the arguments passed to them: the *start* parameter will launch a service, and *stop* tells the service to quit. As the name suggests, *restart* will restart the service – this is often used to parse a modified configuration.

An example of a suitable init script is available from [6]. Copy the file to */etc/init.d/proftpd*, for example, and then create a symbolic link to the directory where the daemon resides. The commands in Listing 4 below, will do this on a typical Linux system. Depending on your distribution, you may need to replace */etc/init.d/rc3.d* with */etc/rc3.d* in line 4.

Listing 4: Start script

```
# cp proftpd /etc/init.d/
# chown root:root /etc/init.d/proftpd
# chmod 744 /etc/init.d/proftpd
# cd /etc/init.d/rc3.d
# ln -s /etc/init.d/proftpd S85proftpd
# cd ../rc5.d
# ln -s /etc/init.d/proftpd S85proftpd
```

Linux systems use a script to launch the individual services in the correct order. The order is defined by the file names. An *S* at the beginning of a line means "start the service" and is followed by a number (defining the order). The rest of the line specifies the name of the service to be launched.

Finished!

Ensure that you are root, and launch the server for the first time, by typing */etc/init.d/proftpd start*. After rebooting, the script will automatically launch the server.

Assuming that everything worked out, you should be able to access your own FTP server as *ftp://localhost*. Check the functionality on the local host, but also from other systems. This ensures that your firewall rules are appropriate. You can then go on to populate the home directory, *~ftp/pub*, with your public content.

Of course, a number of additional steps are recommended to harden critical systems. Take a look at [4] for details. ■

INFO

[1] RFC 959: *ftp://ftp.rfc-editor.org/in-notes/rfc959.txt*

[2] wu-ftpd: *http://www.wu-ftpd.org*

[3] NcFTPd: *http://www.ncftpd.com/ncftpd/*

[4] ProFTPD: *http://proftpd.linux.co.uk*

[5] ProFTPD sources: *ftp://ftp.proftpd.org/distrib/source/*

[6] Configuration file and init script: *http://www.linux-magazine.com/Magazine/Downloads/2004/41/Admin/*

[7] Marc André Selig: "Finger Pointing", Linux Magazine, Issue 36, December 2003, p56