FTP by Example

by Bernd Eckenfels, ecki@lina.inka.de, 2000-12-06 for Freefire.org

This is a short Tutorial on how to speak the Internet File Transfer Protocol manually by hand. This will help you to better understand the known issues for setting up a firewall which is able to pass or block FTP. This document is not intended to be a complete reference to the FTP protocol. Actually it is not even trying to use formal language to describe the protocol.) But I hope it is useful to everybody who needs to understand how and why the FTP protocol works (for example to set up ip filter rules).

First, you need:

- A FTP Server you are allowed to connect to
- A program so you can type in commands and see the response on the screen or redirect the response to a file. Netcat, the TCP Swiss Army Knife (command is normally called `nc`) will be used here
- this ingredible Document
- RFC959, the specification for the FTP protocol
  I'm kidding, you do not need it to understand my short hands-on tutorial
- a lot of time

1. The Client is resolving the FTP-Server's IP-Address

The First Step of the FTP Connection is to establish a TCP Connection and to log in. Before this can happen the Client needs to know the IP-Address to connect to. Because usually you call the FTP client with a host name.

This is usually done by using the System's so called Resolver. The Resolver can use different Methods to get the IP-Address to a name. This includes lookup in local files (hosts, lmhosts, ...), it can include lookup in nameservers (DNS, Wins, NIS, LDAP) or in the cache or by Netbios Broadcasts (depeding on the Operating System, of course). The result is an IP-Address. We can simulate this by using `ping <hostname>` or you can just pass the hostname to netcat*, it will do the resolving itself.

2. The Client is building a TCP Connection

To build a TCP Connection you need the destination. The Destination consists of a Destination Address (which is the result of Step 1) and a destination port. The well-known (reserved) Port for FTP is 21/tcp. They way a client gets this number depends on the client. Normally it should do:

1. If a numerical port is specified on commandline, use it
2. If a symbolic port is specified, look it up in the services table and use it
3. If no port was given on the commandline (which is the normal situation) then use ftp/tcp and look it up in the services table
4. If no entry ftp/tcp is found use 21

A TCP Connection has also a local address and local port. Normally a FTP clients leaves the decision which source address and source port to use to the Operating System. Every well bahaving Operating System will use the IP-Address of the Network Interface which is used to reach the Destination (i.e. the Network Card the FTP Server is connected to or the PPP Interface which is used to access the Internet). The local port will picked randomly from the "high" ports. The high ports are the unpriveledged ports (1024-65535).

To simulate this, use netcat to connect to the Address of a FTP Server you are allowed to connect to:

```
ecki@calista:~> nc -vv ftp.eckenfels.net 21
ftp.eckenfels.net [212.227.90.117] 21 (ftp) open
220 FTP Server ready.
```

In this example **black** is the Input you should type at the commandline or into netcat, blue is the output of netcat (verbose output telling you details about the connection) and red is the output received from the FTP Server (via netcat).

The FTP Server is greeting you with a banner message. All Messages start with a numerical code (220) folowed by a blank and then a text message (FTP Server ready.). The actual message depends on the FTP Server, some of them advertise the program name and version.

3. The FTP Client is logging in

The FTP client can now log into the FTP server by giving username and password with two commands. For "normal" plain password authentication the commands are called **USER** followed by the Username and **PASS** followed by the password.

```
220 FTP Server ready.
USER ecki
```

```
331 Password required for ecki.
PASS youmustbekidding
230- Linux ftp.eckenfels.net Welcome!
230-
230 User ecki logged in.
```

In this example the User ecki is greeted by a short welcome message of the FTP server which consists of 3 lines. The - sign at the fist two lines is telling the FTP Client, that there are more lines following. Only the last line contains a blank in front of the Message (User ecki logged in.). If the password would be wrong, you will see a Message starting with the code 530 and a Message like Login incorrect..

4. The FTP Client is changing the Directory

An easy FTP command is the command to change the current working directory. The command is called **CWD**. It is the same as 'cd' under Windows/DOS or Unix. Note that the command is actually called CWD and CD wont work. This is interesting, since most command line FTP clients offer you the option to use 'cd'. This is a simple translation the FTP client is doing, we will see some more, watch out :). The command **PWD** will "Print Working Directory" and **CDUP** is moving up one directory in the file hierachy (it is the same as cd .. under Unix or Dos).

```
230 User ecki logged in.
CD /tmp
500 'CD /tmp': command not understood.
CWD /tmp
250 CWD command successful.
PWD
257 "/tmp" is current directory.
CDUP
250 CWD command successful.
PWD
257 "/" is current directory.
```

All those commands are nice and perhaps even useful, but now to the real stuff. Because as u might have noticed: no problems for Firewalls at all. Just a connection from a random port on the client to a defined port on the server.

5. The FTP Client wants to retrieve a directory listing

Now the Fun Part begins. If you want to see the content of a directory you might have a problem, because the command is not called 'dir' or 'ls' (as you might expect) but it is called **LIST**. But wait, thats not the only problem with listing a directory. The text, describing a directory is not tramsmitted with those funny messages starting with a 3 digit code, but it is transmitted in its own TCP connection. What?! - wait a moment - what do you mean? Well.. I mean, the Data for a list directory command (i.e. the directory listing) is treated like all data from/to FTP Servers in its own connection. The so called data connection (opposed to the control connection, which we where using all the time). The Idea behind this is described closer in the RFC, one of the main Reasons is, that you can actually keep connected to the FTP Server, even if a large data transfer fails for some reasons. Well, it caused a lot of headache, and there is realy no need for it, but anyway, see how it works in the next example.

The classical FTP protocol supports the **PORT** Command to set up the data-connection. It works like this:

1. The Client looks for a free Port on the local machine and installs a "Listener". I.e. it waits for incoming connections and expect them to contain the result of the LIST command.
2. The client then transmit the server the port number, so the server can actually connect to this port. In addition to that, the client also transmits the IP-Address the FTP server should connect to. This is another major annoyance of the protocol, see the sidebar "ftp-bounce" about the problems.
3. The server will acknowledge the port command
4. The client can now use a command which is retrieving data, in our case the LIST command.
5. The Server will connect to our local port and push the data into this connection. After the data is send by the Server it will give you a message in the control connection.

Now our little practical demonstration gets a bit complicated. You need to open a second command line window on your system and use netcat a second time. This time we will use netcat in the listening mode (-l) on a random free port. To get a random free port you just start with 1024 and try to use that port. If that port is used try 1025 (or any other random port) until you succeed to find a unused port. This will look like this:

```
ecki@calista:~> nc -vv -l -p 1024
retrying local 0.0.0.0:1024 : Address already in use
^C
 sent 0, rcvd 0
ecki@calista:~> nc -vv -l -p 1025
listening on [any] 1025 ...
```

First i tried to install netcat on port 1024. It failed because some other application was using this port. I canceled netcat with Control+C and used the next port. The verbose output of netcat was telling me, that this is successful. So now I am ready to tell the FTP server that I wait on port 1025 for incoming data. The PORT command has a wired Syntax to specify the IP-Address and the Port Number. you need to give the 4 Bytes of the IP-Address and the 2 Bytes of the Port comma separated. The integer 1025 is written as two bytes: 4 and 1. You can check this by multiplying the first byte with 256 and adding the second to it. For the IP, you just need to replace dots with commas and blanks: '10.0.0.1' -> '10, 0, 0, 1'

```
257 "/" is current directory.
PORT 10, 0, 0, 1, 4, 1 # 4*256+1=1025
200 PORT command successful.
LIST -l
150 Opening ASCII mode data connection for '/bin/ls'.
```

Watch your second window closely! You will actually get data (the listing of your FTP-Server's Root Directory):

```
listening on [any] 1025 ...
connect to [10.0.0.1] from ftp.eckenfels.net [212.227.90.117] 20
```

```
total 142
drwxr-xr-x    2 root     root        2048 Dec  5 14:11 bin
drwxr-xr-x    2 root     root        1024 Dec  4 11:12 boot
drwxrwxr-x    2 root     cdrom       1024 Apr 21  1999 cdrom
drwxr-xr-x    5 root     root       18432 Dec  5 14:23 dev
drwxr-xr-x  136 root     root        9216 Dec  6 04:32 etc
drwxrwxr-x    2 root     floppy      1024 Apr 21  1999 floppy
drwxr-xr-x   13 root     root        1024 Sep 24 21:10 home
drwxr-xr-x    2 root     root        1024 Apr 21  1999 initrd
drwxr-xr-x    6 root     root        5120 Dec  5 14:17 lib
drwxr-xr-x    2 root     root       12288 Dec 20  1998 lost+found
drwxr-xr-x    2 root     root        1024 Oct 26  1998 mnt
lrwxrwxrwx    1 root     root           9 Aug  6 08:59 opt
dr-xr-xr-x  139 root     root           0 Nov 20 07:15 proc
drwxr-xr-x   46 root     root        3072 Dec  4 11:14 root
drwxr-xr-x    2 root     root        4096 Dec  5 14:14 sbin
drwxrwxrwt    7 root     root       79872 Dec  6 09:47 tmp
drwxr-xr-x   16 root     root        1024 Dec  4 08:43 usr
drwxr-xr-x   20 root     root        2048 Oct 16 01:57 var
  sent 0, rcvd 1230
```

As you see in the output, my "manual" FTP-Client received a connection from the FTP server. The connection contains a Unix-Style directory listing (it actually depends on the FTP Server which options to **LIST** are allowed and how the Listing will look like). The last line is from netcat, telling you that the remote side has closed the connection and that 1230 bytes are received.

After the directory is received the FTP server will send you a message in the control connection, telling you, that the transfer suceeded.

```
150 Opening ASCII mode data connection for '/bin/ls'.
...2 seconds while data is transmitted...
226 Transfer complete.
```

Have you noticed? The incoming FTP-Data connection from the FTP server was originated from the FTP-Server's IP Address and from the source port 20. This is the ftp-data port. The RFC requires servers to use this port to send data. This is supposed to add some security (not everybody on the FTP Server can send you data, only the priveledged FTP Server). But actually this is a major complication for FTP servers, beause FTP servers actually need this additional privelege, which most of the time means for the FTP server it has to run as root. Since running as root is most of the time not desired for Daemons, and since the requiremnt for Source Port 20 has no additional reason other than the RFC959 was miss-designed (my personal opinion :) some FTP servers refuse to comply to the requirement from the RFC959 and will use a random high port as the source port. (DJB's leightweight and secure FTP Server publicfile* is not using ftp-data port).

6. The Client is retrieving a file

Okay, that was cool, now i have a directory Listing, but I also want to retrieve a File! Well.. that's now easy. Retrieving a file with the classic RFC mode (Port-Command) works the same as retrieving a directory. You only need to use a different command (**RETR** instead of LIST will RETRieve a File).

For the Purpose of completeness I will use a netcat command which can actually create a local file. Just open a new window an start netcat. We redirect the output of netcat to a local file:

```
ecki@calista:~> nc -vv -l -p 1025 > file.txt
listening on [any] 1025 ...
```

Now I tell the server about my prepared listening socket and retrieve a file. After a short moment the file is received and the FTP-Server will tell me about that in the Control connection:

```
PORT 10,0,0,1,4,1 # 4*256+1=1025
200 PORT command successful.
TYPE I
200 Type set to I.
RETR /home/ecki/whatsnew.html
150 Opening BINARY mode data connection for
                     'whatsnew.html' (2315 bytes).
...2 seconds to retrieve file...
226 Transfer complete.
```

Okay, as you might guess the **TYPE I** command is the one which will be send to the FTP Server if you tell your client to use binary mode (bin). The same is true for the RETR command, in the FTP client it is usually called GET. Now lets have a look at our other window (the data connection):

```
listening on [any] 1025 ...
connect to [10.0.0.1] from ftp.eckenfels.net [212.227.90.117] 20
 sent 0, rcvd 2315
ecki@calista:~> ls -l file.txt
-rw-rw-r--    1 ecki     ecki         2315 Dec  6 10:13 file.txt
```

Cool, isn't it? In case you need to do FTP and you do not have a FTP client handy (but a powerfull operation system like Linux with the cool network tool netcat) you now know, how to do FTP. Kidding :)

7. And now the Firewall Friendly passive mode

Actually the above steps should only show you, how the old traditional FTP protocol was defined. As you might guess sending your own IP Address and local port to the server and getting a connection from the Server back is a real problem for a Firewall, especially if it is a gateway which is doing masquerading or NAT (See RFC2993* on Architectural Implications of NAT). So, the solution of RFC1579 is to use the **PASV** command instead of PORT. The client tells the server that it wants to do a passive FTP transfer with PASV and the server responds with a message describing the Port where the client can connect to. (Again the encoding for the address and port number is a bit wired, it is even more wired because it is not standardised how the server will respond, so the client needs to 'search' the address in the line). So here is a typical PASV session for retrieving the output of **NLST** (you will recognize, this will also work for RETR). BTW: NLST works like LIST but with a more compressed output.

```
PASV
227 Entering Passive Mode (212,227,90,117,5,99) # (5*256+99=1379)
NLST
... waiting for the data connection ...
150 Opening ASCII mode data connection for 'file list'.
...2 seconds to retrieve data...
226 Transfer complete.
```

```
ecki@calista:~> nc -vv ftp.eckenfels.net 1379
ftp.eckenfels.net [212.227.90.117] 1379 (?) open
/lost+found
/usr
/var
/home
/boot
/lib
/bin
/etc
/sbin
/floppy
/dev
/cdrom
/initrd
/mnt
/proc
/root
/tmp
/opt
 sent 0, rcvd 2315
```

8. Storing a File to the Server (traditional PORT)

Okay, to put a file one the Server you just need to use the **STOR** command to tell the FTP server the filename you want to give the new file. If you use the PORT command then you will just need to wait for the server to acknowledge the PORT command and then you can send the file's content in a seconary data connection. See the followng transcripts on how to store a file on a FTP server:

```
ecki@calista:~> nc -vv -l -p 1025 < file.txt
listening on [any] 1025 ...
... waiting for the FTP server to connect ...
connect to [10.0.0.1] from ftp.eckenfels.net [212.227.90.117] 20
 sent 2315, rcvd 0
ecki@calista:~>
```

```
PORT 10, 0, 0, 1, 4, 1 # 4*256+1=1025
200 PORT command successful.
STOR /tmp/test.txt
... connecting to the client ...
150 FILE: /tmp/test.txt
... transmitting the file ...
226 Transfer complete.
```

In this example we use the feature of netcat to wait for an incoming connection and then send the data which is waiting to the server.

9. Storing a File to the Server (PASV)

After all those Examples I'm sure you will understand the last example. It is just important to note, that I issue the nc command in the secondary window after I send the STOR command to the Server. Note, it is quite interesting, that the Server will not acknowledge that it has received the command, and we do not need to wait for any acknowledgment, since the Server is already listening on the port (since we started the PASV command). Actually we can even connect to the server before we issue the STOR command. If we connect to the server the server will respond with 150 and if the connection is closed from our side the server asumes the file is transmitted completely and will issue a 226 line.

```
PASV
227 Entering Passive Mode (212,227,90,1,5,178) # 5*256+178=1458
STOR /tmp/test.txt
... waiting for connection from client ...
150 FILE: /tmp/test.txt
... transmitting the file ...
226 Transfer complete.
```

```
ecki@calista:~> nc -vv 212.227.90.1 1458 < file.txt
ftp.eckenfels.net [212.227.90.117] 1458 (?) open
... data is transmitted to the waiting server...
 sent 2315, rcvd 0
ecki@calista:~>
```

I hope you enjoyed that little hands-on tutorial on beeing a FTP Client. For futher and more complete information on the FTP protocl I can recommend you to visit Dan J. Bernsteins FTP Protocol specification or the original FTP RFC. Here is a short List:

1. The FTP RFC959, STD009*
2. Firewall Friendly FTP (PASV) RFC1579*
3. Requirements for Internet Hosts -- Application and Support RFC1123* (suggestion for PASV response format)
4. FTP Security Considerations RFC2577*
5. FTP Extensions for IPv6 and NATs RFC2428*
6. DJB's FTP practical Protocol Specification*
7. LIST parsing mess: DJB's solution ftpparse*
8. FTP Security Extensions (not covered in this article) in RFC2288

Fixed some typos in 2000-12-31
Added link to RFC2228 in 2001-09-10