

Studying Normal Traffic, Part Two: Studying FTP Traffic

Karen Kent Frederick 2001-02-21

Studying Normal Traffic, Part 2: Studying FTP Traffic

by *Karen Kent Frederick*

last updated Feb. 21, 2001

This is the second article in a three-part series devoted to studying normal traffic. As was explained in "[Studying Normal Traffic, Part One](#)", many intrusion detection analysts concentrate on identifying the characteristics of suspicious packets - illegal TCP flag combinations or reserved IP addresses, for example. However, it is also important to be familiar with what normal traffic looks like. A great way to do this is to generate some normal traffic, capture the packets and examine them. The first article in this series explained how to capture packets using WinDump and reviewed some simple examples of normal TCP/IP traffic. In this article, we will be examining FTP traffic, which, from a traffic flow standpoint, is more complicated than most other protocols. Note that in order to understand this material, you should already know the fundamentals of TCP/IP, and you should be familiar with the format of WinDump or tcpdump log files, as discussed in the first article of this series.

Standard FTP Sessions

FTP is a very interesting protocol to examine because of the way it establishes connections. In order to examine some normal FTP traffic, we want to initiate an FTP session between mypc.xx.yy.zz and a remote FTP server, in this case ftp.microsoft.com (207.46.133.140).

Initially, the FTP session is similar to the SSH session that was analyzed in [part one](#) of this series. Normally, the first step in initiating an FTP session is to make an ARP request for the MAC address of the default gateway. However, in the example that we are studying, the information is in mypc's ARP cache, so no ARP request is needed. As a result, the first information we see in the logs is the DNS query, asking for the IP address associated with ftp.microsoft.com, and the subsequent DNS response. Note that for these UDP packets, the client uses a high-numbered port and the server uses port 53, as we would expect for a DNS query.

```
13:50:46.490130 mypc.xx.yy.zz.3170 > dnsserver.xx.yy.zz.53: 1+ A? ftp.microsoft.com. (35)
13:50:46.500269 dnsserver.xx.yy.zz.53 > mypc.xx.yy.zz.3170: 1 q: ftp.microsoft.com. 1/4/4
ftp.microsoft.com. A 207.46.133.140 (215)
```

Now that mypc has the server's IP address, 207.46.133.140, it initiates the TCP three-way handshake from one of its high-numbered ports to port 21, the FTP port on ftp.microsoft.com:

```
13:50:46.564494 mypc.xx.yy.zz.3171 > 207.46.133.140.21: S 87666930:87666930(0) win 65535 (DF)
13:50:46.671920 207.46.133.140.21 > mypc.xx.yy.zz.3171: S 3058770520:3058770520(0) ack 87666931 win 17520 (DF)
13:50:46.672155 mypc.xx.yy.zz.3171 > 207.46.133.140.21: . 87666931:87666931(0) ack 3058770521 win 65535 (DF)
```

After the three-way handshake has been completed, the FTP client and server begin to exchange information. I am presented with a message saying that I am connected to ftp.microsoft.com, and I am asked to provide my user name. I provide the user name "anonymous" and am prompted to enter my e-mail address as the password, which I do. Eventually I am given an FTP prompt. This exchange between the client and server is a series of PUSH/ACK and ACK packets. Here is a sample of these packets. Note that they all use mypc's port 3171 and ftp.microsoft.com's port 21:

```
13:50:46.779273 207.46.133.140.21 > mypc.xx.yy.zz.3171: P 3058770521:3058770576(55)
ack 87666931 win 17520 (DF)
13:50:46.896881 mypc.xx.yy.zz.3171 > 207.46.133.140.21: . 87666931:87666931(0)
ack 3058770576 win 65480 (DF)
13:50:48.282313 mypc.xx.yy.zz.3171 > 207.46.133.140.21: P 87666931:87666947(16)
ack 3058770576 win 65480 (DF)
13:50:48.388962 207.46.133.140.21 > mypc.xx.yy.zz.3171: P 3058770576:3058770648(72)
ack 87666947 win 17504 (DF)
13:50:48.495939 mypc.xx.yy.zz.3171 > 207.46.133.140.21: . 87666947:87666947(0)
ack 3058770648 win 65408 (DF)
```

So far, the traffic pattern resembles that of an SSH session. However, FTP has a critical difference. It uses the initial connection as a control channel for the entire FTP session. This control channel transfers the client's commands to the server and sends the server's replies back to the client. Actual data, such as directory listings, file uploads and downloads, is never transferred over this connection. In this example, the control channel is the connection between the client's port 3171 and the FTP server's port 21. When a user downloads a file or asks for a directory listing, that data is actually sent through a separate TCP connection, which is created specifically for that listing and torn down as soon as the listing has been transferred.

When the user types "ls" in order to see a list of files, several steps occur. In order for the server to initiate a data connection to the client, it has to know which client port number to contact. In the first line of this log, the client is telling the server what IP address and port number it should connect to. The second line represents the server's response, telling the client that the IP address and port number are acceptable. The third line is the command from the FTP client software to the server that requests the file listing. Finally, in the fourth line, the server tells the client that it is initiating the data connection.

```
13:50:53.501031 mypc.xx.yy.zz.3171 > 207.46.133.140.21: P 87666966:87666994(28)
ack 3058770765 win 65291 (DF)
13:50:53.607175 207.46.133.140.21 > mypc.xx.yy.zz.3171: P 3058770765:3058770795(30)
ack 87666994 win 17457 (DF)
13:50:53.632708 mypc.xx.yy.zz.3171 > 207.46.133.140.21: P 87666994:87667000(6)
ack 3058770795 win 65261 (DF)
13:50:53.737852 207.46.133.140.21 > mypc.xx.yy.zz.3171: P 3058770795:3058770850(55)
ack 87667000 win 17451 (DF)
```

In order to transfer the data - in this case, the directory listing - from the server to the client, the server initiates a data connection from its port 20 to the port that was specified by the client, in this case 3172. We see the three-way handshake occur, then one PUSH/ACK packet is sent from the server to the client. This contains the file listing, which was small enough to fit into one packet. The log entry for the PUSH/ACK packet has (164) in it, which indicates that 164 bytes of data were transferred.

```

13:50:53.738024 207.46.133.140.20 > mypc.xx.yy.zz.3172: S 3061133541:3061133541(0)
win 16384 (DF)
13:50:53.738200 mypc.xx.yy.zz.3172 > 207.46.133.140.20: S 87674104:87674104(0)
ack 3061133542 win 65535 (DF)
13:50:53.844704 207.46.133.140.20 > mypc.xx.yy.zz.3172: . 3061133542:3061133542(0)
ack 87674105 win 17520 (DF)
13:50:53.850833 207.46.133.140.20 > mypc.xx.yy.zz.3172: P 3061133542:3061133706(164)
ack 87674105 win 17520 (DF)

```

At this point, there are two separate connections:

- a control channel between the client's port 3171 and the server's port 21; and,
- a data channel between the client's port 3172 and the server's port 20.

As soon as the server has completed the transfer of the directory listing, it initiates a graceful termination of the connection with a FIN/ACK packet. Note that the server is only terminating the data connection on its port 20, not the control channel on its port 21. While the data connection is being torn down, we see that traffic is still being passed on the control channel. The fifth entry below, which shows a transfer of 24 bytes over the control channel, represents the server sending a message to be displayed to the user. In this case, it says "226 Transfer complete", which indicates that the directory listing was transferred to the client successfully.

```

13:50:53.850981 207.46.133.140.20 > mypc.xx.yy.zz.3172: F 3061133706:3061133706(0)
ack 87674105 win 17520 (DF)
13:50:53.851068 mypc.xx.yy.zz.3172 > 207.46.133.140.20: . 87674105:87674105(0)
ack 3061133707 win 65371 (DF)
13:50:53.895937 mypc.xx.yy.zz.3171 > 207.46.133.140.21: . 87667000:87667000(0)
ack 3058770850 win 65206 (DF)
13:50:53.903415 mypc.xx.yy.zz.3172 > 207.46.133.140.20: F 87674105:87674105(0)
ack 3061133707 win 65371(DF)
13:50:54.002060 207.46.133.140.21 > mypc.xx.yy.zz.3171: P 3058770850:3058770874(24)
ack 87667000 win 17451 (DF)
13:50:54.009333 207.46.133.140.20 > mypc.xx.yy.zz.3172: . 3061133707:3061133707(0)
ack 87674106 win 17520 (DF)
13:50:54.196818 mypc.xx.yy.zz.3171 > 207.46.133.140.21: . 87667000:87667000(0)
ack 3058770874 win 65182 (DF)

```

If I retrieved a file from the server, we would see the same sequence of events. The server would initiate a new data connection from its port 20 to a different available high port on the client, and the three-way TCP handshake would occur. Then data would be transferred over that connection, which would be torn down as soon as the transfer was complete.

During an FTP session, several data connections may be made within seconds of each other. Each time a data connection is established, a different client port number is used. If you only look at a portion of the session, it may appear to be a portscan, particularly if the client allocates consecutive port numbers. You need to look at a larger amount of traffic to determine if this is a portscan or simply the establishment of more FTP data connections.

Passive FTP Sessions

FTP is rather unusual because the server initiates the data connections to the client, rather than the client initiating all connections with the server. FTP can cause problems with some firewalls and packet filters because its data connections are initiated by the server. To avoid this problem, users can utilize passive FTP instead of standard FTP.

Passive FTP begins with the same steps as FTP: the client initiates a control connection between one of its high ports and port 21 on the FTP server. However, when a data channel needs to be opened, the server sends one of its available high port numbers to the client. The client then initiates a connection between one of its high ports and the high port number that was sent by the FTP server. So when you are using passive FTP, port 20 is never used. Instead, all data is passed between two high ports.

Here's an example of how passive FTP works. This traffic was generated from an OpenBSD 2.8 machine and captured using tcpdump. By default, OpenBSD's FTP client uses passive FTP.

At first, passive FTP looks just like regular FTP. The client initiates a TCP connection from one of its high ports to port 21 on the FTP server. (Note that the log entries look slightly different from the previous trace because we are using OpenBSD instead of Windows 98 and tcpdump instead of windump. However, the traffic pattern is exactly the same.)

```
15:57:28.005993 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: S 157025335:157025335(0)
win 16384
15:57:28.099136 207.46.133.140.21 > bsdpc.xx.yy.zz.28348: S 1286994806:1286994806(0)
ack 157025336 win 17520 (DF)
15:57:28.099193 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: .
ack 1286994807 win 17376
15:57:28.193361 207.46.133.140.21 > bsdpc.xx.yy.zz.28348: P 1286994807:1286994862
(55)
ack 157025336 win 17520 (DF)
15:57:28.193413 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: . ack 1286994862 win 17376
```

(N.B.: Several additional PUSH/ACK and ACK packets have been omitted.)

Then, just like the previous trace, I type "ls" into my FTP client software to get a file listing. In the first packet below, my FTP client tells the server that it wants to use passive FTP. The second packet holds the server's response to that request, which is the server's IP address and the port number that it has allocated for the requested data connection. In the third packet, the client simply acknowledges that it has received the packet with the port information.

```
15:57:36.243722 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: P 157025383:157025389(6)
ack 1286995115 win 17376
15:57:36.342188 207.46.133.140.21 > bsdpc.xx.yy.zz.28348: P 1286995115:1286995166
(51)
ack 157025389 win 17467 (DF)
15:57:36.342213 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: .
ack 1286995166 win 17325
```

Now that the client knows what server port to use, it initiates a TCP connection to the server port from another of its high-numbered ports. In this case, the client uses port 24626, and the server uses port 3668. The three-way handshake is done, and we see information being transferred across the control channel, server port 21 and client

port 28348.

```
15:57:36.342370 bsdpc.xx.yy.zz.24626 > 207.46.133.140.3668: S 157871686:157871686(0)
win 16384
15:57:36.440039 207.46.133.140.3668 > bsdpc.xx.yy.zz.24626: S 1292391804:1292391804
(0)
ack 157871687 win 17520 (DF)
15:57:36.440076 bsdpc.xx.yy.zz.24626 > 207.46.133.140.3668: .
ack 1292391805 win 17376
15:57:36.440167 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: P 157025389:157025395(6)
ack 1286995166 win 17376
15:57:36.542608 207.46.133.140.21 > bsdpc.xx.yy.zz.28348: P 1286995166:1286995220
(54)
ack 157025395 win 17461 (DF)
15:57:36.542638 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: . ack 1286995220 win 17322
```

Now that the data connection has been established, the file listing can actually be transferred. If you look at the entries in this section of the log, which are in timestamp order, it looks like the server is tearing down the connection (in the first line) before the data is transferred (in the third line). However, if you look at the sequence numbers, you will realize that these packets were received out of order. The packet with 0 bytes of data was sent after the packet with 1167 bytes of data but it was received first. When you are analyzing log entries like these, it's helpful to have them in timestamp order, but you should focus on the sequence numbers.

```
15:57:36.547340 207.46.133.140.3668 > bsdpc.xx.yy.zz.24626: F 1292392972:1292392972
(0)
ack 157871687 win 17520 (DF)
15:57:36.547367 bsdpc.xx.yy.zz.24626 > 207.46.133.140.3668: .
ack 1292391805 win 17376
15:57:36.549363 207.46.133.140.3668 > bsdpc.xx.yy.zz.24626: P 1292391805:1292392972
(1167)
ack 157871687 win 17520 (DF)
15:57:36.549396 bsdpc.xx.yy.zz.24626 > 207.46.133.140.3668: .
ack 1292392973 win 16209
15:57:36.551374 bsdpc.xx.yy.zz.24626 > 207.46.133.140.3668: F 157871687:157871687(0)
ack 1292392973 win 17376
15:57:36.635184 207.46.133.140.21 > bsdpc.xx.yy.zz.28348: P 1286995220:1286995244
(24)
ack 157025395 win 17461 (DF)
15:57:36.635211 bsdpc.xx.yy.zz.28348 > 207.46.133.140.21: .
ack 1286995244 win 17376
15:57:36.647798 207.46.133.140.3668 > bsdpc.xx.yy.zz.24626: .
ack 157871688 win 17520 (DF)
```

Summary

We have closely examined the typical traffic flows of FTP. FTP uses two different types of connections: control and data. The control channel is used to send commands from the client to the server and to transmit replies to the commands from the server to the client. An FTP client initiates the control connection to an FTP server. In standard FTP, the server initiates all data connections to the client. In passive FTP, the client initiates all data connections to the server.

I encourage you to do your own packet captures of FTP sessions (with the proper authorization, of course!) The next article in this series will examine some additional characteristics of normal packets, such as TCP options.

To read **Studying Normal Traffic, Part Three: TCP Headers**, click [here](#).

Karen Kent Frederick is a senior security engineer for NFR Security. Karen has a B.S. in Computer Science and is completing her Master's thesis in intrusion detection through the University of Idaho's Engineering Outreach program. She holds several certifications, including Microsoft Certified Systems Engineer + Internet, Check Point Certified Security Administrator, and SANS GIAC Certified Intrusion Analyst. Karen is one of the authors and editors of "Intrusion Signatures and Analysis", a book on intrusion detection that was published in January 2001.

Relevant Links

[Studying Normal Traffic, Part One](#)

Karen Kent Frederick, SecurityFocus.com

[Abnormal Traffic](#)

Karen Kent Frederick, SecurityFocus.com

[RFC 959 - FTP](#)

Network Working Group

[tcpdump](#)

Tcpdump Group

[WinDump](#)

NetGroup

[Privacy Statement](#)

Copyright 2006, SecurityFocus