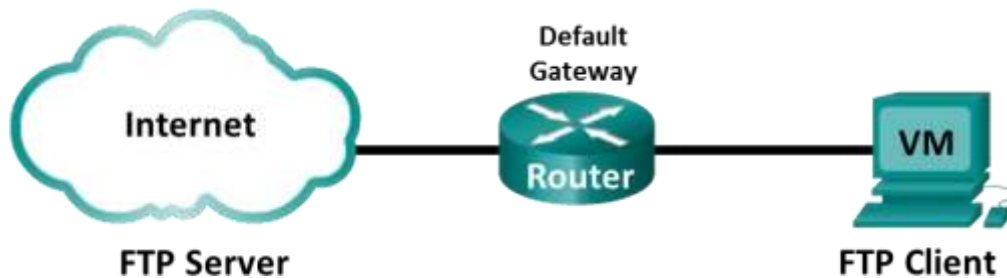


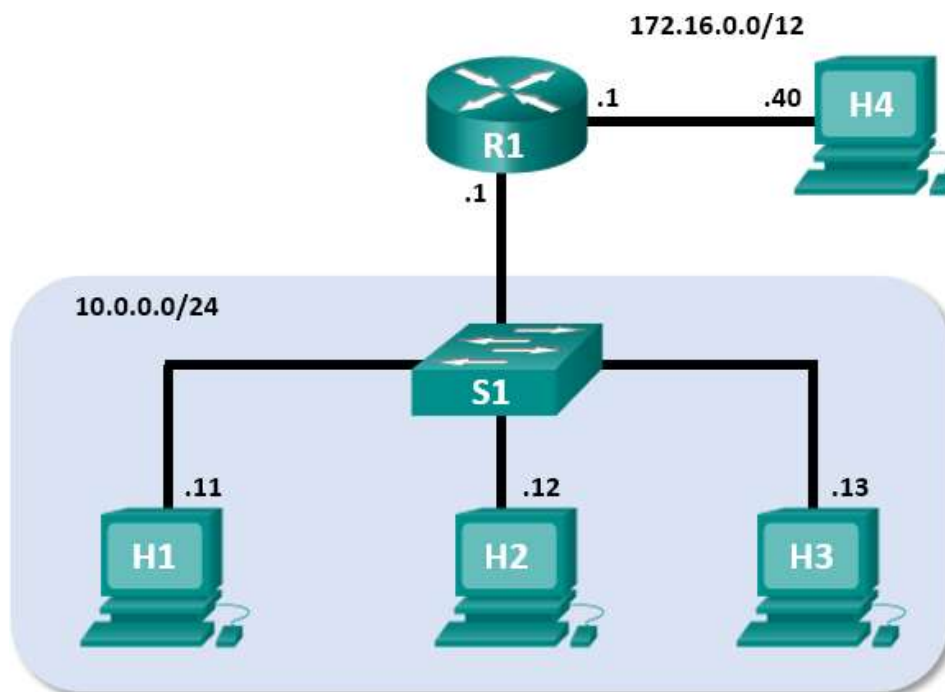
Lab - Using Wireshark to Examine TCP and UDP Captures

Topology - Part 1 (FTP)



Part 1 will highlight a TCP capture of an FTP session. This topology consists of the CyberOps Workstation VM with internet access.

Mininet Topology - Part 2 (TFTP)



Objectives

Part 1: Identify TCP Header Fields and Operation Using a Wireshark FTP Session Capture

Part 2: Identify UDP Header Fields and Operation Using a Wireshark TFTP Session Capture

Background / Scenario

Two protocols in the TCP/IP transport layer are TCP (defined in RFC 761) and UDP (defined in RFC 768). Both protocols support upper-layer protocol communication. For example, TCP is used to provide transport layer support for the HyperText Transfer Protocol (HTTP) and FTP protocols, among others. UDP provides transport layer support for the Domain Name System (DNS) and TFTP, among others.

In Part 1 of this lab, you will use the Wireshark open source tool to capture and analyze TCP protocol header fields for FTP file transfers between the host computer and an anonymous FTP server. The terminal command line is used to connect to an anonymous FTP server and download a file. In Part 2 of this lab, you will use Wireshark to capture and analyze UDP header fields for TFTP file transfers between two Mininet host computers.

Required Resources

- CyberOps Workstation VM
- Internet access

Instructions

Part 1: Identify TCP Header Fields and Operation Using a Wireshark FTP Session Capture

In Part 1, you use Wireshark to capture an FTP session and inspect TCP header fields.

Step 1: Start a Wireshark capture.

- Start and log into the CyberOps Workstation VM. Open a terminal window and start Wireshark. The ampersand (&) sends the process to the background and allows you to continue to work in the same terminal.
- Start a Wireshark capture for the **enp0s3** interface.
- Open another terminal window to access an external ftp site. Enter **ftp ftp.cdc.gov** at the prompt. Log into the FTP site for Centers for Disease Control and Prevention (CDC) with user **anonymous** and no password.

```
[analyst@secOps ~]$ wireshark &
```

```
[analyst@secOps ~]$ ftp ftp.cdc.gov
```

```
Connected to ftp.cdc.gov.
```

```
220 Microsoft FTP Service
```

```
Name (ftp.cdc.gov:analyst): anonymous
```

```
331 Anonymous access allowed, send identity (e-mail name) as password.
```

```
Password:
```

```
230 User logged in.
```

```
Remote system type is Windows_NT.
```

```
ftp>
```

```

Terminal - analyst@secOps:~
File Edit View Terminal Tabs Help
[analyst@secOps ~]$ ftp ftp.cdc.gov
Connected to ftp.cdc.gov.
220 Microsoft FTP Service
Name (ftp.cdc.gov:analyst): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp>

```

Step 2: Download the Readme file.

- a. Locate and download the Readme file by entering the **ls** command to list the files.

```

ftp> ls
200 PORT command successful.
125 Data connection already open; Transfer starting.
-rwxrwxrwx  1 owner  group           128 May  9  1995 .change.dir
-rwxrwxrwx  1 owner  group           107 May  9  1995 .message
drwxrwxrwx  1 owner  group              0 Feb  2  11:21 pub
-rwxrwxrwx  1 owner  group          1428 May 13  1999 Readme
-rwxrwxrwx  1 owner  group           383 May 13  1999 Siteinfo
-rwxrwxrwx  1 owner  group              0 May 17  2005 up.htm
drwxrwxrwx  1 owner  group              0 May 20  2010 w3c
-rwxrwxrwx  1 owner  group           202 Sep 22  1998 welcome.msg
226 Transfer complete.

```

Note: You may receive the following messages:

```

421 Service not available, remote server has closed connection
ftp: No control connection for command

501 Server cannot access argument
500 command not understood
ftp: bind: Address already in use

```

If this happens, then the FTP server is currently down. However, you can proceed with the rest of the lab analyzing those packets that you were able to capture and reading along for packets you did not capture. You can also return to the lab later to see if the FTP server is back up.

- b. Enter the command **get Readme** to download the file. When the download is complete, enter the command **quit** to exit. (**Note:** If you are unable to download the file, you can proceed with the rest of the lab.)

```

ftp> get Readme
200 PORT command successful.
125 Data connection already open; Transfer starting.
WARNING! 36 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Transfer complete.
1428 bytes received in 0.056 seconds (24.9 kbytes/s)

```

- c. After the transfer is complete, enter **quit** to exit ftp.

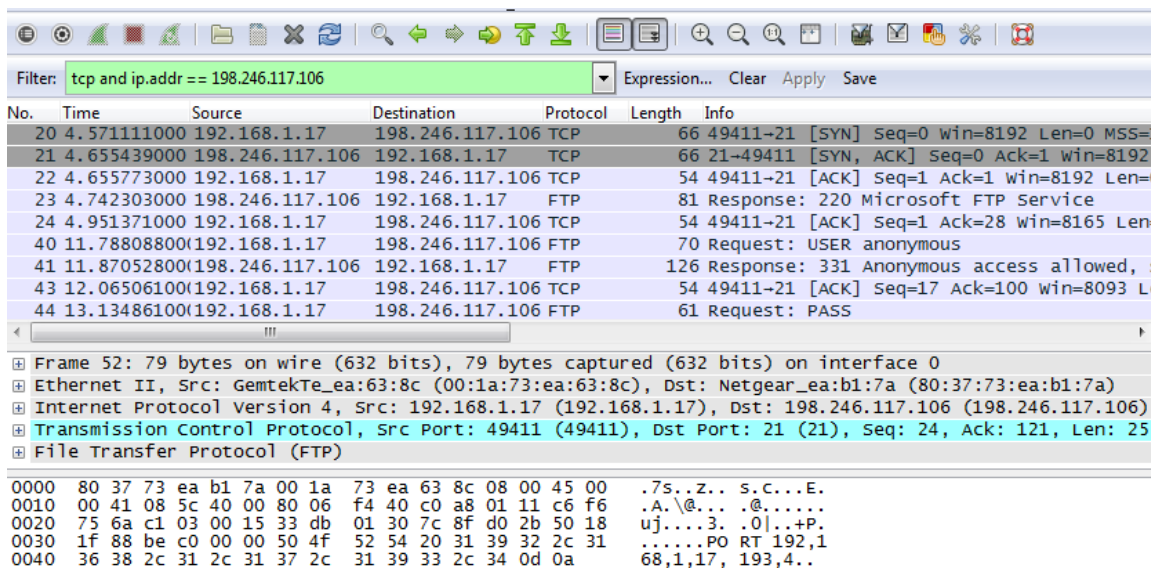
```
ftp> get Readme
200 PORT command successful.
125 Data connection already open; Transfer starting.
WARNING! 36 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Transfer complete.
1428 bytes received in 0.147 seconds (9.49 kbytes/s)
ftp>
ftp> quit
221 Goodbye.
```

Step 3: Stop the Wireshark capture.

Step 4: View the Wireshark main window.

Wireshark captured many packets during the FTP session to ftp.cdc.gov. To limit the amount of data for analysis, apply the filter **tcp and ip.addr == 198.246.117.106** and click **Apply**.

Note: The IP address, 198.246.117.106, is the address for ftp.cdc.gov at the time this lab was created. The IP address may be different for you. If so, look for the first TCP packet that started the 3-way handshake with ftp.cdc.gov. The destination IP address is the IP address you should use for your filter.



No.	Time	Source	Destination	Protocol	Length	Info
20	4.571111000	192.168.1.17	198.246.117.106	TCP	66	49411->21 [SYN] Seq=0 win=8192 Len=0 MSS=
21	4.655439000	198.246.117.106	192.168.1.17	TCP	66	21->49411 [SYN, ACK] Seq=0 Ack=1 win=8192
22	4.655773000	192.168.1.17	198.246.117.106	TCP	54	49411->21 [ACK] Seq=1 Ack=1 win=8192 Len=0
23	4.742303000	198.246.117.106	192.168.1.17	FTP	81	Response: 220 Microsoft FTP Service
24	4.951371000	192.168.1.17	198.246.117.106	TCP	54	49411->21 [ACK] Seq=1 Ack=28 win=8165 Len=0
40	11.788088000	192.168.1.17	198.246.117.106	FTP	70	Request: USER anonymous
41	11.870528000	198.246.117.106	192.168.1.17	FTP	126	Response: 331 Anonymous access allowed, s
43	12.065061000	192.168.1.17	198.246.117.106	TCP	54	49411->21 [ACK] Seq=17 Ack=100 win=8093 L
44	13.134861000	192.168.1.17	198.246.117.106	FTP	61	Request: PASS

Frame 52: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0	
Ethernet II	Src: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c), Dst: Netgear_ea:b1:7a (80:37:73:ea:b1:7a)
Internet Protocol Version 4	Src: 192.168.1.17 (192.168.1.17), Dst: 198.246.117.106 (198.246.117.106)
Transmission Control Protocol	Src Port: 49411 (49411), Dst Port: 21 (21), Seq: 24, Ack: 121, Len: 25
File Transfer Protocol (FTP)	

Offset	Hex	ASCII
0000	80 37 73 ea b1 7a 00 1a 73 ea 63 8c 08 00 45 00	.7s..z..S.C...E.
0010	00 41 08 5c 40 00 80 06 f4 40 c0 a8 01 11 c6 f6	.A.\@...@.....
0020	75 6a c1 03 00 15 33 db 01 30 7c 8f d0 2b 50 18	uj....3..0 ...P.
0030	1f 88 be c0 00 00 50 4f 52 54 20 31 39 32 2c 31PO RT 192,1
0040	36 38 2c 31 2c 31 37 2c 31 39 33 2c 34 0d 0a	68,1,17, 193,4..

Note: Your Wireshark interface may look slightly different than the above image.

Step 5: Analyze the TCP fields.

After the TCP filter has been applied, the first three packets (top section) display the sequence of [SYN], [SYN, ACK], and [ACK] which is the TCP three-way handshake.

No.	Time	Source	Destination	Protocol	Length	Info
20	4.571111000	192.168.1.17	198.246.117.106	TCP	66	49411->21 [SYN] Seq=0 win=8192 Len=0 MSS=
21	4.655439000	198.246.117.106	192.168.1.17	TCP	66	21->49411 [SYN, ACK] Seq=0 Ack=1 win=8192
22	4.655773000	192.168.1.17	198.246.117.106	TCP	54	49411->21 [ACK] Seq=1 Ack=1 win=8192 Len=0

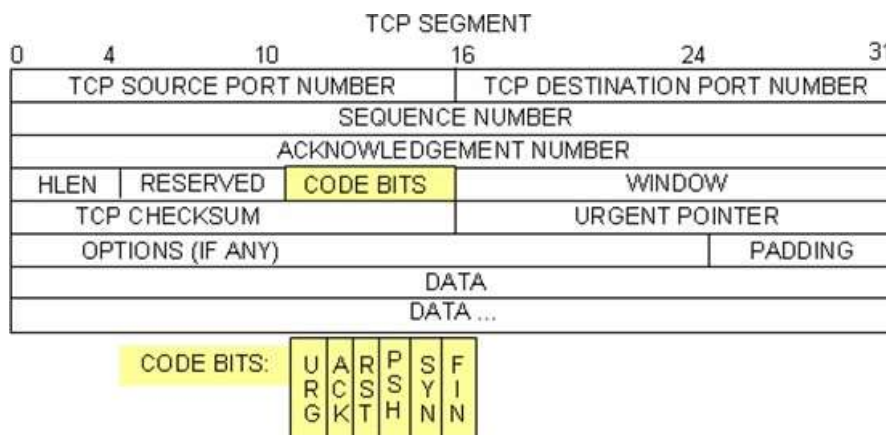
TCP is routinely used during a session to control datagram delivery, verify datagram arrival, and manage window size. For each data exchange between the FTP client and FTP server, a new TCP session is started. At the conclusion of the data transfer, the TCP session is closed. When the FTP session is finished, TCP performs an orderly shutdown and termination.

In Wireshark, detailed TCP information is available in the packet details pane (middle section). Highlight the first TCP datagram from the host computer, and expand portions of the TCP datagram, as shown below.

```

Frame 20: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c), Dst: Netgear_ea:b1:7a (80:37:73:ea:b1:7a)
Internet Protocol Version 4, Src: 192.168.1.17 (192.168.1.17), Dst: 198.246.117.106 (198.246.117.106)
Transmission Control Protocol, Src Port: 49411 (49411), Dst Port: 21 (21), Seq: 0, Len: 0
  Source Port: 49411 (49411)
  Destination Port: 21 (21)
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 0
  Header Length: 32 bytes
  ... 0000 0000 0010 = Flags: 0x002 (SYN)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion window Reduced (cwr): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...0 = Acknowledgment: Not set
    .... ....0 = Push: Not set
    .... ..0.. = Reset: Not set
    .... ....1. = Syn: Set
    .... ....0 = Fin: Not set
  window size value: 8192
  [calculated window size: 8192]
  Checksum: 0x5bba [validation disabled]
  Urgent pointer: 0
  Options: (12 bytes), Maximum segment size, No-Operation (NOP), window scale, No-Operation (NOP), No-
  
```

The expanded TCP datagram appears similar to the packet detail pane, as shown below.



The image above is a TCP datagram diagram. An explanation of each field is provided for reference:

- The **TCP source port number** belongs to the TCP session host that opened a connection. The value is normally a random value above 1,023.
- The **TCP destination port number** is used to identify the upper layer protocol or application on the remote site. The values in the range 0–1,023 represent the “well-known ports” and are associated with popular services and applications (as described in RFC 1700), such as Telnet, FTP, and HTTP. The combination of the source IP address, source port, destination IP address, and destination port uniquely identifies the session to the sender and receiver.

Note: In the Wireshark capture above, the destination port is 21, which is FTP. FTP servers listen on port 21 for FTP client connections.

- The **Sequence number** specifies the number of the last octet in a segment.
- The **Acknowledgment number** specifies the next octet expected by the receiver.
- The **Code bits** have a special meaning in session management and in the treatment of segments. Among interesting values are:
 - **ACK** — Acknowledgment of a segment receipt.
 - **SYN** — Synchronize, only set when a new TCP session is negotiated during the TCP three-way handshake.
 - **FIN** — Finish, the request to close the TCP session.
- The **Window size** is the value of the sliding window. It determines how many octets can be sent before waiting for an acknowledgment.
- The **Urgent pointer** is only used with an Urgent (URG) flag when the sender needs to send urgent data to the receiver.
- The **Options** has only one option currently, and it is defined as the maximum TCP segment size (optional value).

Using the Wireshark capture of the first TCP session startup (SYN bit set to 1), fill in information about the TCP header. Some fields may not apply to this packet.

From the VM to CDC server (only the SYN bit is set to 1):

Description	Wireshark Results
Source IP address	192.168.1.17
Destination IP address	198.246.117.106
Source port number	49411
Destination port number	21
Sequence number	0
Acknowledgment number	Not applicable for this capture.
Header length	32 bytes
Window size	8192

In the second Wireshark filtered capture, the CDC FTP server acknowledges the request from the VM. Note the values of the SYN and ACK bits.

```

+ Frame 21: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
+ Ethernet II, Src: Netgear_ea:b1:7a (80:37:73:ea:b1:7a), Dst: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c)
+ Internet Protocol Version 4, Src: 198.246.117.106 (198.246.117.106), Dst: 192.168.1.17 (192.168.1.17)
- Transmission Control Protocol, Src Port: 21 (21), Dst Port: 49411 (49411), Seq: 0, Ack: 1, Len: 0
  Source Port: 21 (21)
  Destination Port: 49411 (49411)
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header Length: 32 bytes
- .... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)
  000. .... = Reserved: Not set
  ...0 .... = Nonce: Not set
  .... 0... = Congestion window Reduced (CWR): Not set
  .... .0.. = ECN-Echo: Not set
  .... ..0. = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
+ .... .... ..1. = Syn: Set
  .... .... ...0 = Fin: Not set
  window size value: 8192
  [Calculated window size: 8192]
+ Checksum: 0x0ee7 [validation disabled]
  Urgent pointer: 0
+ Options: (12 bytes), Maximum segment size, No-operation (NOP), window scale, No-operation (NOP), No
+ [SEQ/ACK analysis]
  
```

Fill in the following information regarding the SYN-ACK message.

Description	Wireshark Results
Source IP address	198.246.117.106
Destination IP address	192.168.1.17
Source port number	21
Destination port number	49411
Sequence number	0
Acknowledgment number	1
Header length	32 bytes
Window size	8192

In the final stage of the negotiation to establish communications, the VM sends an acknowledgment message to the server. Notice that only the ACK bit is set to 1, and the Sequence number has been incremented to 1.

```

+ Frame 22: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
+ Ethernet II, Src: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c), Dst: Netgear_ea:b1:7a (80:37:73:ea:b1:7a)
+ Internet Protocol version 4, Src: 192.168.1.17 (192.168.1.17), Dst: 198.246.117.106 (198.246.117.106)
- Transmission Control Protocol, Src Port: 49411 (49411), Dst Port: 21 (21), Seq: 1, Ack: 1, Len: 0
  Source Port: 49411 (49411)
  Destination Port: 21 (21)
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header Length: 20 bytes
  - ... 0000 0001 0000 = Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion Window Reduced (CWR): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
  window size value: 8192
  [Calculated window size: 8192]
  [window size scaling factor: 1]
  + Checksum: 0x4f6a [validation disabled]
  Urgent pointer: 0
  + [SEQ/ACK analysis]
  
```

Fill in the following information regarding the ACK message.

Description	Wireshark Results
Source IP address	192.168.1.17
Destination IP address	198.246.117.106
Source port number	49411
Destination port number	21
Sequence number	1
Acknowledgment number	1
Header length	20
Window size	8192

How many other TCP datagrams contained a SYN bit?

One. The first packet sent by the host when the TCP session begins.

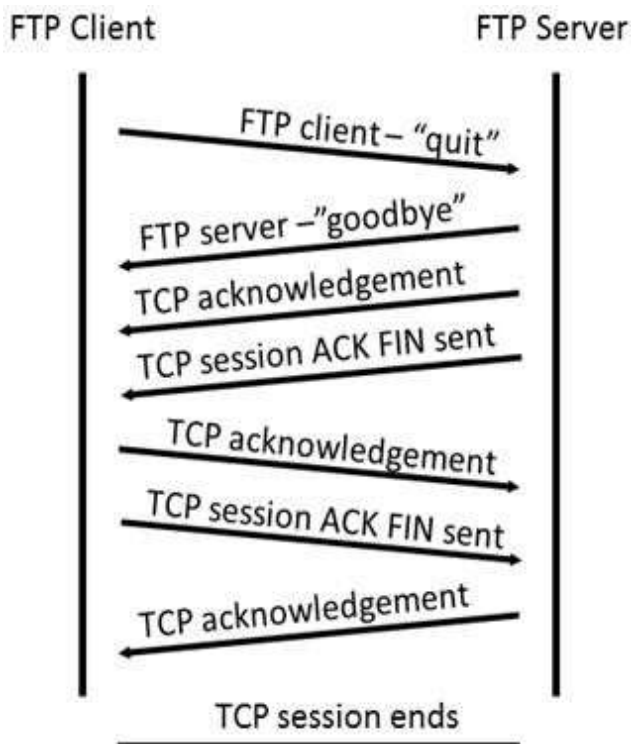
After a TCP session is established, FTP traffic can occur between the PC and FTP server. The FTP client and server communicate with each other, unaware that TCP has control and management over the session.

When the FTP server sends a *Response: 220* to the FTP client, the TCP session on the FTP client sends an acknowledgment to the TCP session on the server. This sequence is visible in the Wireshark capture below.

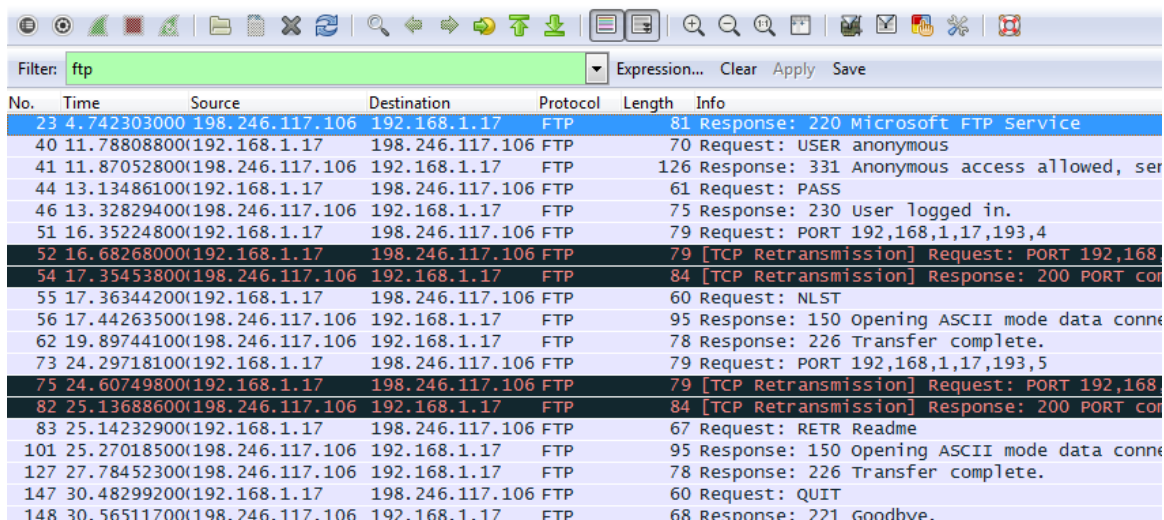
23	4.742303000	198.246.117.106	192.168.1.17	FTP	81	Response: 220 Microsoft FTP Service
24	4.951371000	192.168.1.17	198.246.117.106	TCP	54	49411→21 [ACK] Seq=1 Ack=28 win=8165 Len=
40	11.788088000	192.168.1.17	198.246.117.106	FTP	70	Request: USER anonymous
41	11.870528000	198.246.117.106	192.168.1.17	FTP	126	Response: 331 Anonymous access allowed, :

Frame 23: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
 Ethernet II, Src: Netgear_ea:b1:7a (80:37:73:ea:b1:7a), Dst: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c)
 Internet Protocol Version 4, Src: 198.246.117.106 (198.246.117.106), Dst: 192.168.1.17 (192.168.1.17)
 Transmission Control Protocol, Src Port: 21 (21), Dst Port: 49411 (49411), Seq: 1, Ack: 1, Len: 27
 File Transfer Protocol (FTP)
 220 Microsoft FTP Service\r\n
 Response code: Service ready for new user (220)
 Response arg: Microsoft FTP Service

When the FTP session has finished, the FTP client sends a command to "quit". The FTP server acknowledges the FTP termination with a *Response: 221 Goodbye*. At this time, the FTP server TCP session sends a TCP datagram to the FTP client, announcing the termination of the TCP session. The FTP client TCP session acknowledges receipt of the termination datagram, then sends its own TCP session termination. When the originator of the TCP termination (the FTP server) receives a duplicate termination, an ACK datagram is sent to acknowledge the termination and the TCP session is closed. This sequence is visible in the diagram and capture below.



By applying an **ftp** filter, the entire sequence of the FTP traffic can be examined in Wireshark. Notice the sequence of the events during this FTP session. The username **anonymous** was used to retrieve the Readme file. After the file transfer completed, the user ended the FTP session.

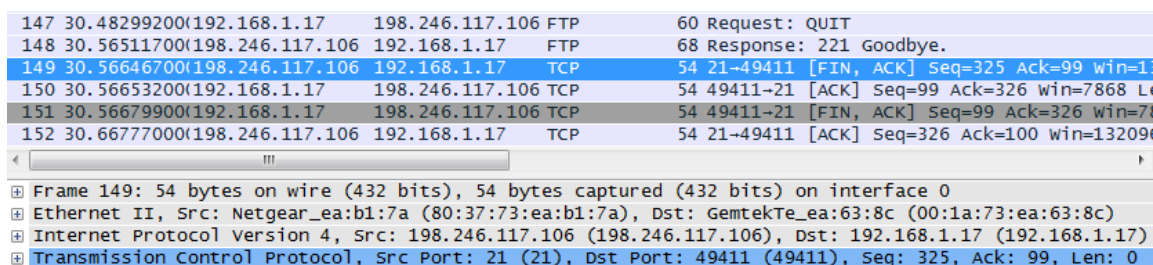


No.	Time	Source	Destination	Protocol	Length	Info
23	4.742303000	198.246.117.106	192.168.1.17	FTP	81	Response: 220 Microsoft FTP Service
40	11.788088000	198.246.117.106	192.168.1.17	FTP	70	Request: USER anonymous
41	11.870528000	198.246.117.106	192.168.1.17	FTP	126	Response: 331 Anonymous access allowed, send
44	13.134861000	198.246.117.106	192.168.1.17	FTP	61	Request: PASS
46	13.328294000	198.246.117.106	192.168.1.17	FTP	75	Response: 230 User logged in.
51	16.352248000	198.246.117.106	192.168.1.17	FTP	79	Request: PORT 192,168,1,17,193,4
52	16.682680000	198.246.117.106	192.168.1.17	FTP	79	[TCP Retransmission] Request: PORT 192,168
54	17.354538000	198.246.117.106	192.168.1.17	FTP	84	[TCP Retransmission] Response: 200 PORT co
55	17.363442000	198.246.117.106	192.168.1.17	FTP	60	Request: NLST
56	17.442635000	198.246.117.106	192.168.1.17	FTP	95	Response: 150 Opening ASCII mode data conn
62	19.897441000	198.246.117.106	192.168.1.17	FTP	78	Response: 226 Transfer complete.
73	24.297181000	198.246.117.106	192.168.1.17	FTP	79	Request: PORT 192,168,1,17,193,5
75	24.607498000	198.246.117.106	192.168.1.17	FTP	79	[TCP Retransmission] Request: PORT 192,168
82	25.136886000	198.246.117.106	192.168.1.17	FTP	84	[TCP Retransmission] Response: 200 PORT co
83	25.142329000	198.246.117.106	192.168.1.17	FTP	67	Request: RETR Readme
101	25.270185000	198.246.117.106	192.168.1.17	FTP	95	Response: 150 Opening ASCII mode data conn
127	27.784523000	198.246.117.106	192.168.1.17	FTP	78	Response: 226 Transfer complete.
147	30.482992000	198.246.117.106	192.168.1.17	FTP	60	Request: QUIT
148	30.565117000	198.246.117.106	192.168.1.17	FTP	68	Response: 221 Goodbye.

Apply the TCP filter again in Wireshark to examine the termination of the TCP session. Four packets are transmitted for the termination of the TCP session. Because TCP connection is full duplex, each direction must terminate independently. Examine the source and destination addresses.

In this example, the FTP server has no more data to send in the stream. It sends a segment with the FIN flag set in frame 149. The PC sends an ACK to acknowledge the receipt of the FIN to terminate the session from the server to the client in frame 150.

In frame 151, the PC sends a FIN to the FTP server to terminate the TCP session. The FTP server responds with an ACK to acknowledge the FIN from the PC in frame 152. Now the TCP session is terminated between the FTP server and PC.



No.	Time	Source	Destination	Protocol	Length	Info
147	30.482992000	198.246.117.106	192.168.1.17	FTP	60	Request: QUIT
148	30.565117000	198.246.117.106	192.168.1.17	FTP	68	Response: 221 Goodbye.
149	30.566467000	198.246.117.106	192.168.1.17	TCP	54	21->49411 [FIN, ACK] Seq=325 Ack=99 win=1
150	30.566532000	198.246.117.106	192.168.1.17	TCP	54	49411->21 [ACK] Seq=99 Ack=326 win=7868 L
151	30.566799000	198.246.117.106	192.168.1.17	TCP	54	49411->21 [FIN, ACK] Seq=99 Ack=326 win=7
152	30.667770000	198.246.117.106	192.168.1.17	TCP	54	21->49411 [ACK] Seq=326 Ack=100 win=13209

Frame 149: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
 Ethernet II, Src: Netgear_ea:b1:7a (80:37:73:ea:b1:7a), Dst: GemtekTe_ea:63:8c (00:1a:73:ea:63:8c)
 Internet Protocol Version 4, Src: 198.246.117.106 (198.246.117.106), Dst: 192.168.1.17 (192.168.1.17)
 Transmission Control Protocol, Src Port: 21 (21), Dst Port: 49411 (49411), Seq: 325, Ack: 99, Len: 0

Part 2: Identify UDP Header Fields and Operation Using a Wireshark TFTP Session Capture

In Part 2, you use Wireshark to capture a TFTP session and inspect the UDP header fields.

Step 1: Start Mininet and tftpd service.

- Start Mininet. Enter **cyberops** as the password when prompted.

```
[analyst@secOps ~]$ sudo lab.support.files/scripts/cyberops_topo.py
[sudo] password for analyst:
```

```

Terminal - analyst@secOps:~
File Edit View Terminal Tabs Help

[analyst@secOps ~]$ sudo lab.support.files/scripts/cyberops_topo.py
[sudo] password for analyst:

CyberOPS Topology:

      -----
      | R1 |-----| H4 |
      -----
      |
      |
      -----
    |-----| S1 |-----|
    |       |       |
    |       |       |
    -----
  | H1 |   | H2 |   | H3 |
  -----

*** Add links
*** Creating network
*** Adding hosts:
H1 H2 H3 H4 R1

```

- b. Start H1 and H2 at the **mininet>** prompt.

```

*** Starting CLI:
mininet> xterm H1 H2

```

- c. In the **H1** terminal window, start the tftpd server using the provided script.

```

[root@secOps analyst]# /home/analyst/lab.support.files/scripts/start_tftpd.sh
[root@secOps analyst]#

```

```

File Edit View Terminal [root@secOps analyst]# /home/analyst/lab.support.files/scripts/start_tftpd.sh
[root@secOps analyst]#

*** Add links
*** Creating network
*** Adding hosts:
H1 H2 H3 H4 R1
*** Adding switches:
s1
*** Adding links:
(H1, s1) (H2, s1) (H3, s1)
*** Configuring hosts
H1 H2 H3 H4 R1
*** Starting controller
*** Starting 1 switches
s1 ...
*** Routing Table on Router
Kernel IP routing table
Destination      Gateway
10.0.0.0          0.0.0.0
172.16.0.0        0.0.0.0

*** Starting CLI:
mininet> xterm H1 H2
mininet>
  
```

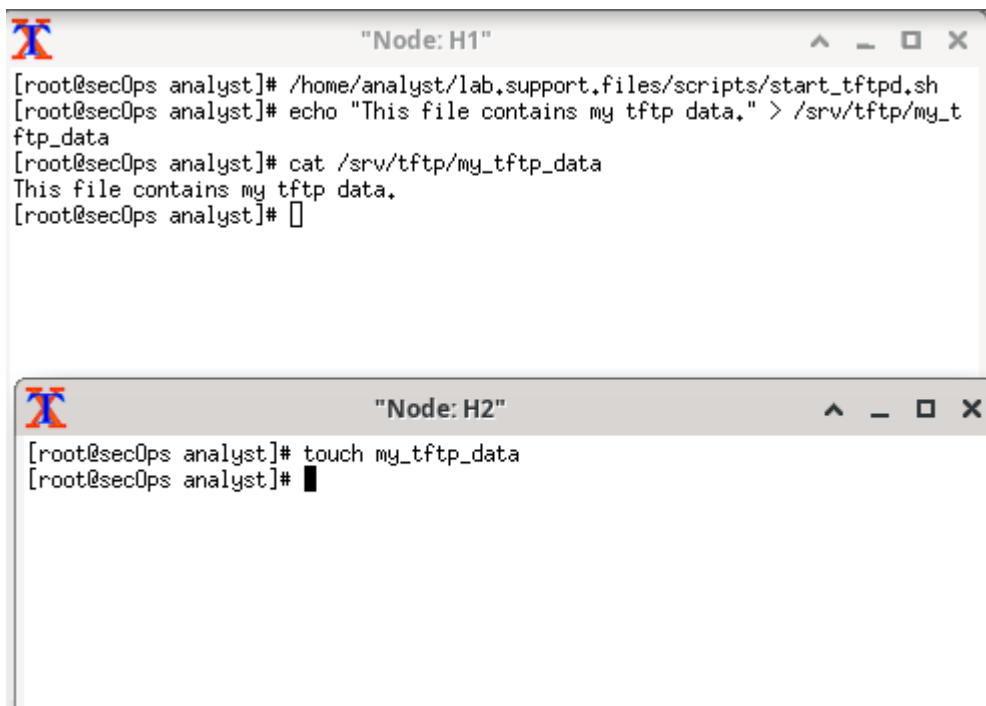
Step 2: Create a file for tftp transfer.

- Create a text file at the **H1** terminal prompt in the `/srv/tftp/` folder.


```
[root@secOps analyst]# echo "This file contains my tftp data." > /srv/tftp/my_tftp_data
```
- Verify that the file has been created with the desired data in the folder.


```
[root@secOps analyst]# cat /srv/tftp/my_tftp_data
This file contains my tftp data.
```
- Because of the security measure for this particular tftp server, the name of the receiving file needs to exist already. On **H2**, create a file named `my_tftp_data`.


```
[root@secOps analyst]# touch my_tftp_data
```



The image shows two terminal windows. The top window, titled "Node: H1", shows a user at the root@secOps analyst prompt running a script to start a TFTP server, creating a file named my_tftp_data with the content "This file contains my tftp data.", and then verifying the file's contents. The bottom window, titled "Node: H2", shows the same user running the touch command to create the my_tftp_data file on their local machine.

```

[Node: H1]
[root@secOps analyst]# /home/analyst/lab.support.files/scripts/start_tftpd.sh
[root@secOps analyst]# echo "This file contains my tftp data." > /srv/tftp/my_tftp_data
[root@secOps analyst]# cat /srv/tftp/my_tftp_data
This file contains my tftp data.
[root@secOps analyst]#

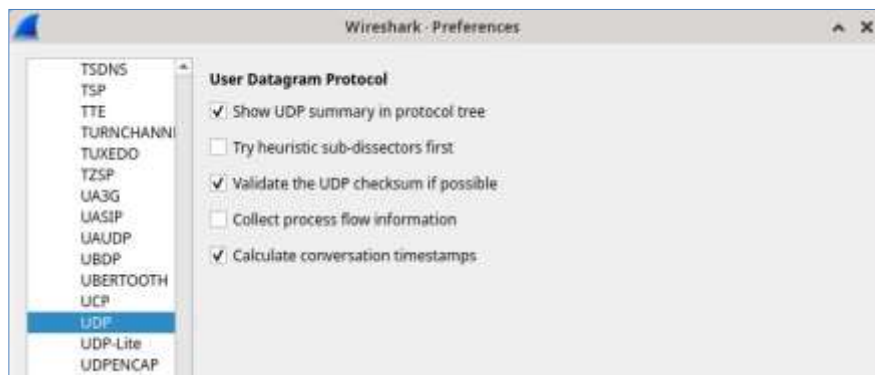
[Node: H2]
[root@secOps analyst]# touch my_tftp_data
[root@secOps analyst]#
  
```

Step 3: Capture a TFTP session in Wireshark

- a. Start Wireshark in H1.

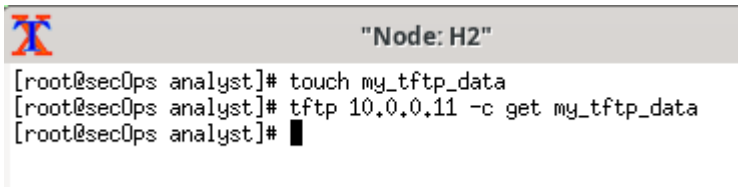
```
[root@secOps analyst]# wireshark &
```

- b. From the **Edit** menu, choose **Preferences** and click the arrow to expand **Protocols**. Scroll down and select **UDP**. Click the **Validate the UDP checksum if possible** check box and click **OK**.



- c. Start a Wireshark capture on the interface **H1-eth0**.
- d. Start a tftp session from **H2** to the tftp server on **H1** and get the file **my_tftp_data**.

```
[root@secOps analyst]# tftp 10.0.0.11 -c get my_tftp_data
```



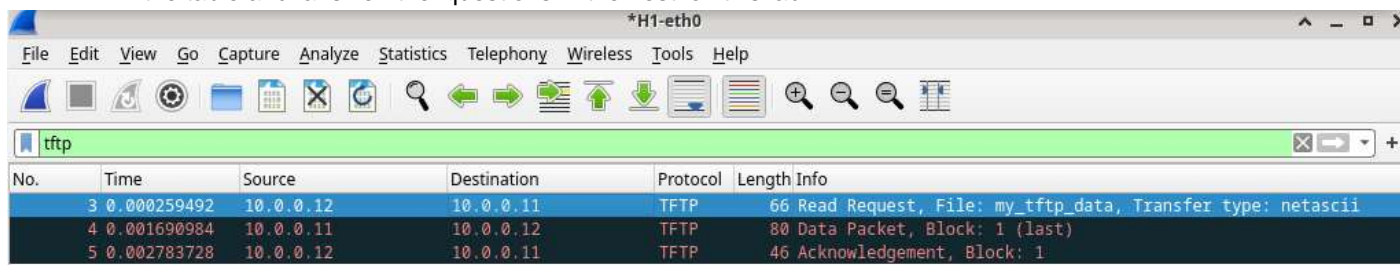
The image shows a terminal window titled "Node: H2". It shows the user running the touch command to create the file, then running the tftp command to retrieve the file from 10.0.0.11, and finally showing the prompt again.

```

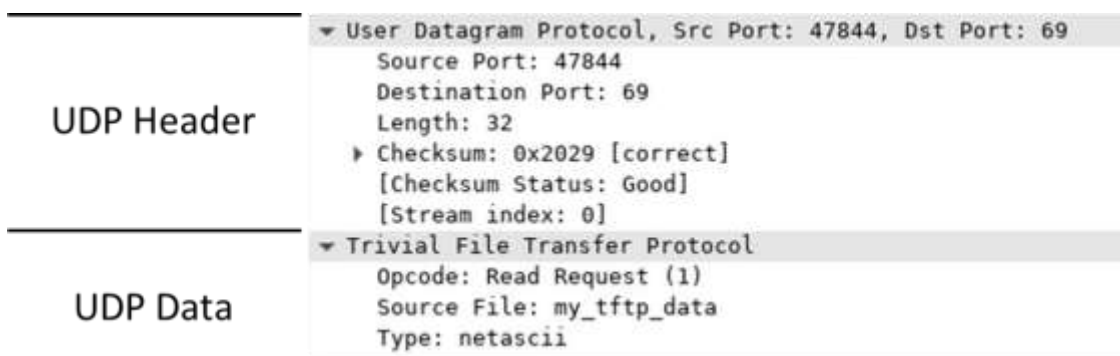
[Node: H2]
[root@secOps analyst]# touch my_tftp_data
[root@secOps analyst]# tftp 10.0.0.11 -c get my_tftp_data
[root@secOps analyst]#
  
```

Lab - Using Wireshark to Examine TCP and UDP Captures

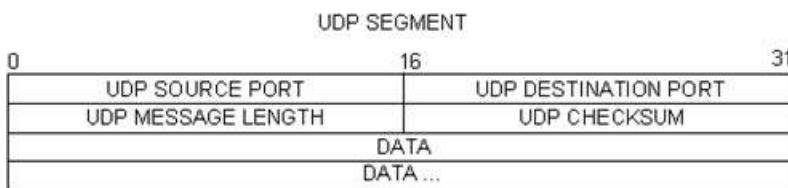
- e. Stop the Wireshark capture. Set the filter to **tftp** and click **Apply**. Use the three TFTP packets to fill in the table and answer the questions in the rest of this lab.



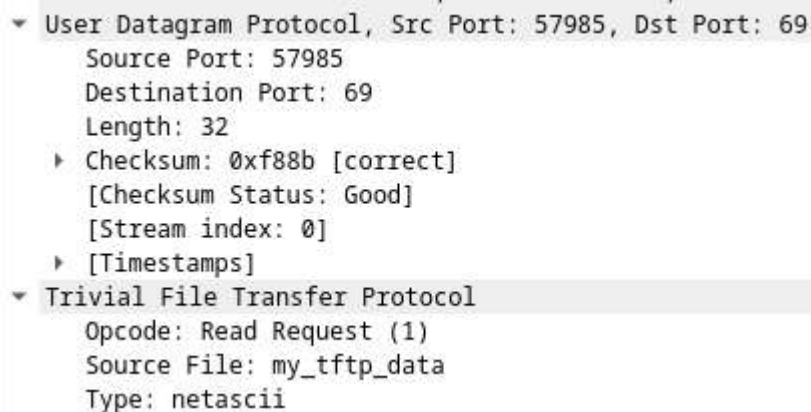
Detailed UDP information is available in the Wireshark packet details pane. Highlight the first UDP datagram from the host computer and move the mouse pointer to the packet details pane. It may be necessary to adjust the packet details pane and expand the UDP record by clicking the protocol expand box. The expanded UDP datagram should look similar to the diagram below.



The figure below is a UDP datagram diagram. Header information is sparse, compared to the TCP datagram. Similar to TCP, each UDP datagram is identified by the UDP source port and UDP destination port.



Using the Wireshark capture of the first UDP datagram, fill in information about the UDP header. The checksum value is a hexadecimal (base 16) value, denoted by the preceding 0x code:



Description	Wireshark Results
Source IP address	10.0.0.12
Destination IP address	10.0.0.11
Source port number	57985
Destination port number	69
UDP message length	32 bytes
UDP checksum	0xf88b [correct]

Examine the first frame returned from the tftpd server. Fill in the information about the UDP header:

- ▶ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 10.0.0.12
- ▼ User Datagram Protocol, Src Port: 48179, Dst Port: 57985
 - Source Port: 48179
 - Destination Port: 57985
 - Length: 46
 - ▶ Checksum: 0x1456 incorrect, should be 0x8bbd (maybe caused by "UDP checksum offload"?)
 - [Checksum Status: Bad]
 - [Stream index: 1]
 - ▶ [Timestamps]
- ▼ Trivial File Transfer Protocol
 - Opcode: Data Packet (3)
 - [Source File: my_tftp_data]
 - Block: 1
 - [Full Block Number: 1]
- ▶ Data (34 bytes)

Description	Wireshark Results
Source IP address	10.0.0.11
Destination IP address	10.0.0.12
Source port number	48179
Destination port number	57985
UDP message length	46 bytes
UDP checksum	0x1456 incorrect, should be 0x8bbd (maybe caused by "UDP checksum offload"?)

Notice that the return UDP datagram has a different UDP source port, but this source port is used for the remainder of the TFTP transfer. Because there is no reliable connection, only the original source port used to begin the TFTP session is used to maintain the TFTP transfer.

Also, notice that the UDP Checksum is incorrect. This is most likely caused by UDP checksum offload. You can learn more about why this happens by searching for "UDP checksum offload".

Step 4: Clean up

In this step, you will shut down and clean up Mininet.

- a. In the terminal that started Mininet, enter **quit** at the prompt.

```
mininet> quit
```

```
mininet> quit
*** Stopping 0 controllers

*** Stopping 2 terms
*** Stopping 5 links
.....
*** Stopping 1 switches
s1
*** Stopping 5 hosts
H1 H2 H3 H4 R1
*** Done
```

- b. At the prompt, enter **sudo mn -c** to clean up the processes started by Mininet.

```
[analyst@secOps ~]$ sudo mn -c
```

```
[analyst@secOps ~]$ sudo mn -c
[sudo] password for analyst:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```