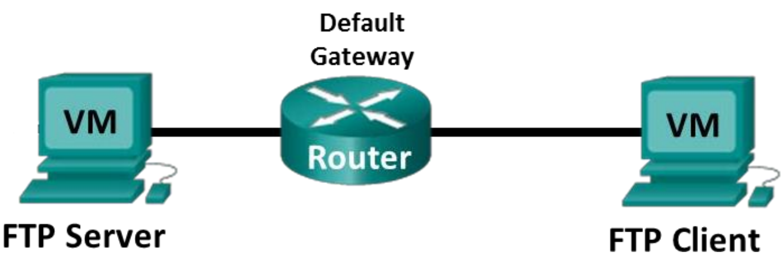
# Lab 4.6.4.3 - Using Wireshark to Examine TCP and UDP Captures

**This lab has been updated for use on NETLAB+.**
[www.netdevgroup.com](http://www.netdevgroup.com)

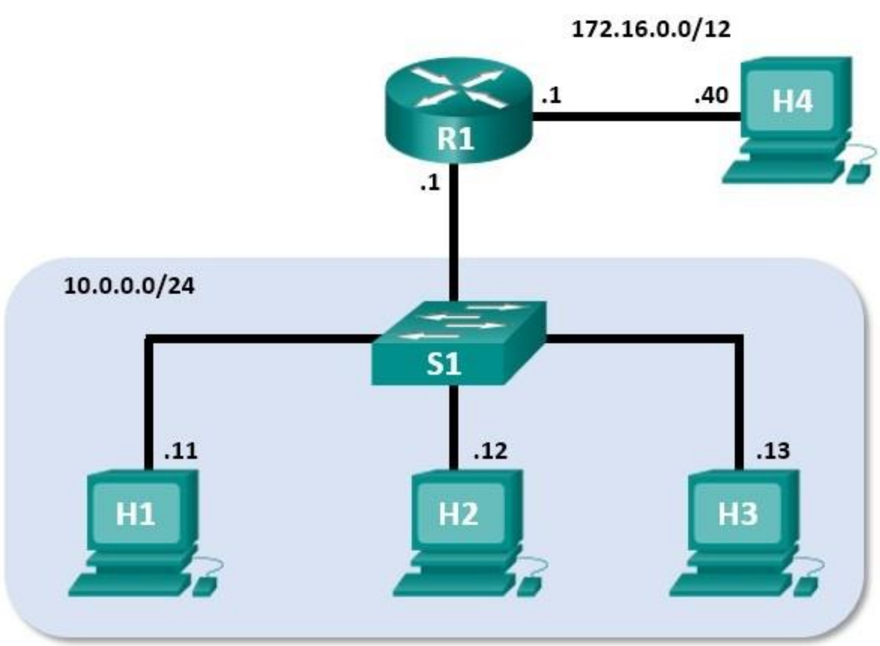
## Topology – Part 1 (FTP)




*Part 1* will highlight a *TCP* capture of an *FTP* session. This topology consists of the *CyberOps Workstation* VM with *Metasploitable 2*.

## Mininet Topology – Part 2 (TFTP)

*Part 2* will highlight a *UDP* capture of a *TFTP* session using the hosts in *Mininet*.



## 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.

# 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.

a. Start and log into the **CyberOps Workstation** VM. Open a terminal window and start **Wireshark**. Enter the password `cyberops` and click **OK** when prompted.

```
[analyst@secOps ~]$ sudo wireshark-gtk
```

b. Start a **Wireshark capture** for the **ens33** interface.

c. Open another terminal window to access an ftp site. Enter `ftp 209.165.200.235` at the prompt. Log into the *FTP* site for *Metasploitable 2* with user **msfadmin** and `msfadmin` as the password.

```
[analyst@secOps ~]$ ftp 209.165.200.235
Connected to 209.165.200.235.
220 (vsFTPd 2.3.4)
Name (209.165.200.235:analyst): msfadmin
331 Please specify the password.
Password: msfadmin
230 Login successful.
Remote system type is UNIX.
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. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--   1 1000     1000          514 Apr 04 17:48 Readme
drwxr-xr-x   6 1000     1000         4096 Apr 28  2010 vulnerable
226 Directory send OK.
```

b. Enter the command **get Readme** to download the file. When the download is complete, enter the command **quit** to exit.

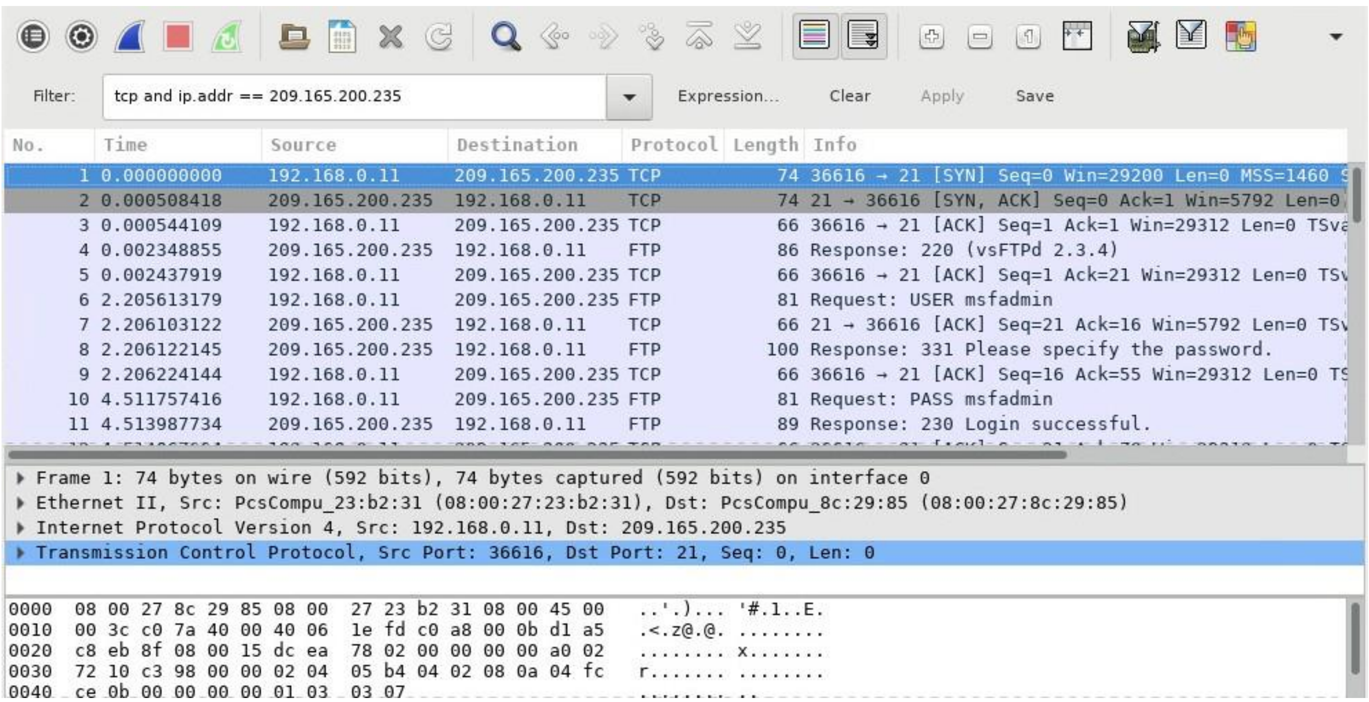
```
ftp> get Readme
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for Readme
(513 bytes).
226 Transfer complete.
513 bytes received in 0.056 seconds (24.9 kbytes/s)
```

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

### Step 3: Stop the Wireshark capture.

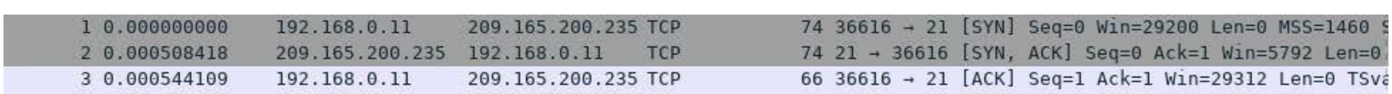
### Step 4: View the Wireshark main window.

*Wireshark* captured many packets during the *FTP* session to *209.165.200.235*. To limit the amount of data for analysis, apply the filter **tcp and ip.addr == 209.165.200.235** and click **Apply**.




### Step 5: Analyze the TCP fields.

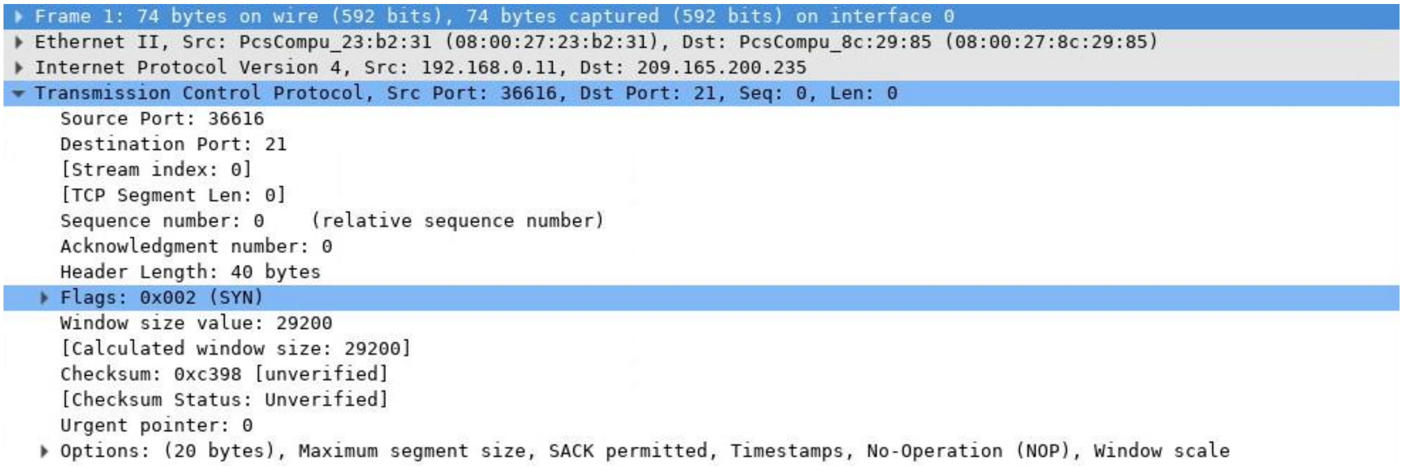
After the *TCP* filter has been applied, the first three packets (top section) display the the sequence of *[SYN], [SYN, ACK], and [ACK]* which is the *TCP* three-way handshake.

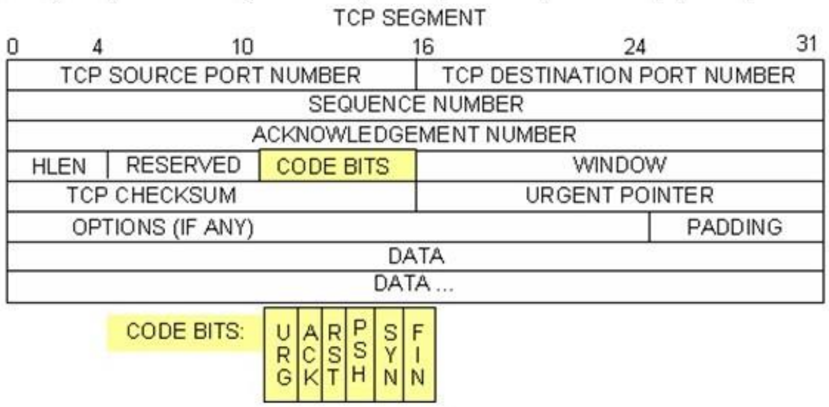



*TCP* is routinely used during a session to control datagram delivery, verify datagram arrival, and manage window size. For each data exchange between the *FTP* client and *FTP* server, a new *TCP* session is started. At the conclusion of the data transfer, the *TCP* session is closed. When the *FTP* 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 the *TCP* datagram. The expanded *TCP* datagram appears similar to the packet detail pane 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.

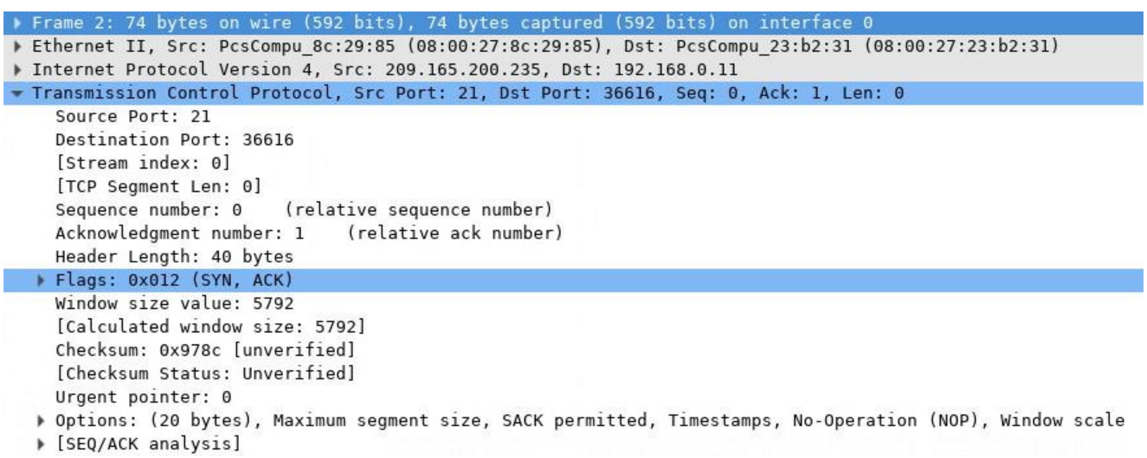
The *Options* has only one option currently, and it is defined as the maximum TCP segment size (optional value).

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 *Metasploitable* server (only the SYN bit is set to 1):

| | |
|---|---|
| Source IP address | 192.168.0.11 |
| Destination IP address | 209.165.200.235 |
| Source port number | 59452 |
| Destination port number | 21 |
| Sequence number | 0 |
| Acknowledgment number | 0 |
| Header length | 40 bytes |
| Window size | 22900 |

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



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

| | |
|---|---|
| Source IP address | 209.165.200.235 |
| Destination IP address | 192.168.0.11 |
| Source port number | 21 |
| Destination port number | 59452 |
| Sequence number | 0 |
| Acknowledgment number | 1 |
| Header length | 40 bytes |
| Window size | 5792 |

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 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_23:b2:31 (08:00:27:23:b2:31), Dst: PcsCompu_8c:29:85 (08:00:27:8c:29:85)
▸ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 209.165.200.235
▾ Transmission Control Protocol, Src Port: 36616, Dst Port: 21, Seq: 1, Ack: 1, Len: 0
    Source Port: 36616
    Destination Port: 21
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0xdc11 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
```

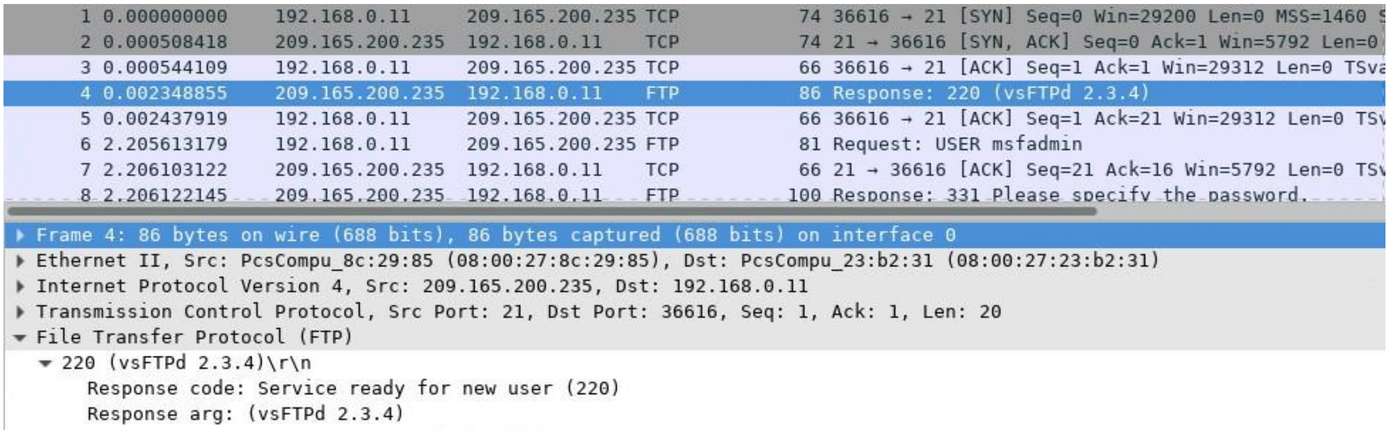
Fill in the following information regarding the ACK message.

| | |
|---|---|
| Source IP address | 209.165.200.235 |
| Destination IP address | 192.168.0.11 |
| Source port number | 20 |
| Destination port number | 40729 |
| Sequence number | 134 |
| Acknowledgment number | 2 |
| Header length | 32 bytes |
| Window size | 183 |

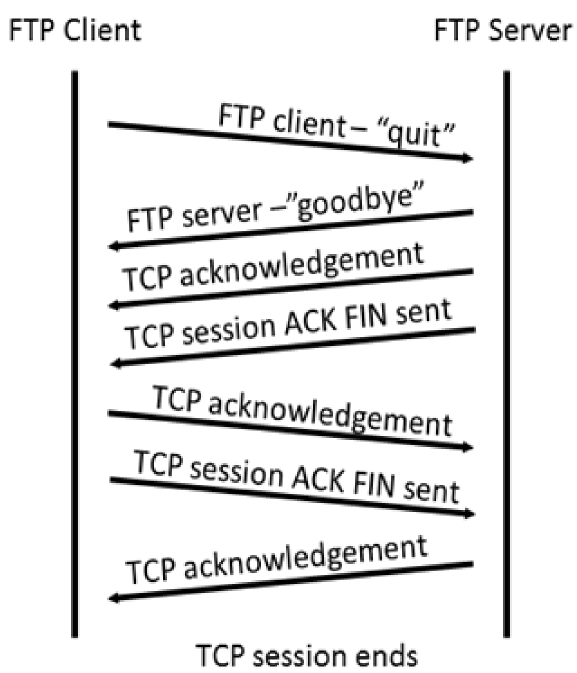
How many other *TCP* datagrams contained a *SYN* bit?
one

---

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.

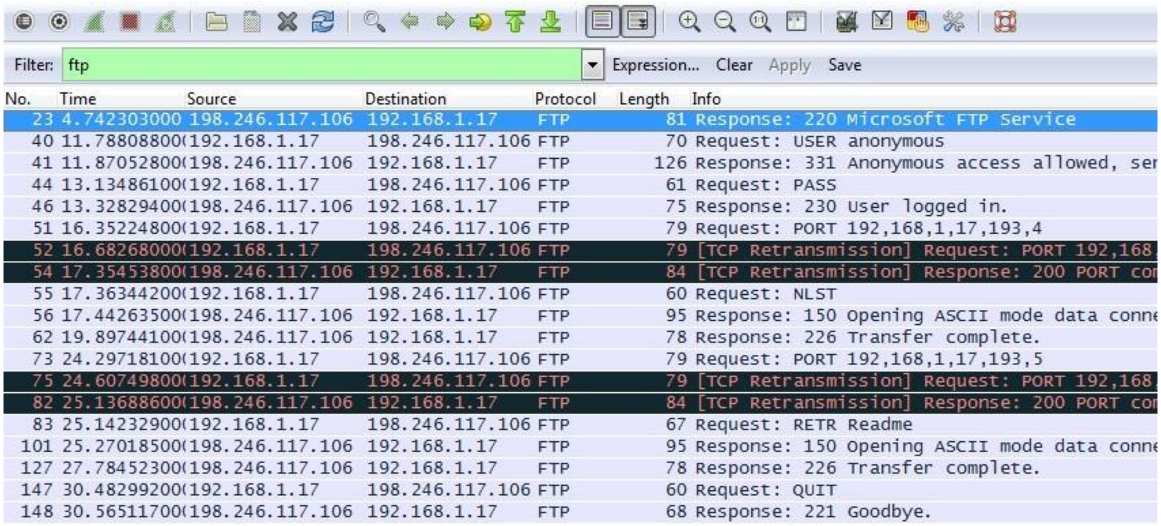



When the *FTP* session has finished, the *FTP* client sends a command to "quit". The *FTP* server acknowledges the *FTP* termination with a *Response: 221 Goodbye*. At this time, the *FTP* server *TCP* session sends a *TCP* datagram to the *FTP* client, announcing the termination of the *TCP* session. The *FTP* client *TCP* session acknowledges receipt of the termination datagram, then sends its own TCP session termination. W hen 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.



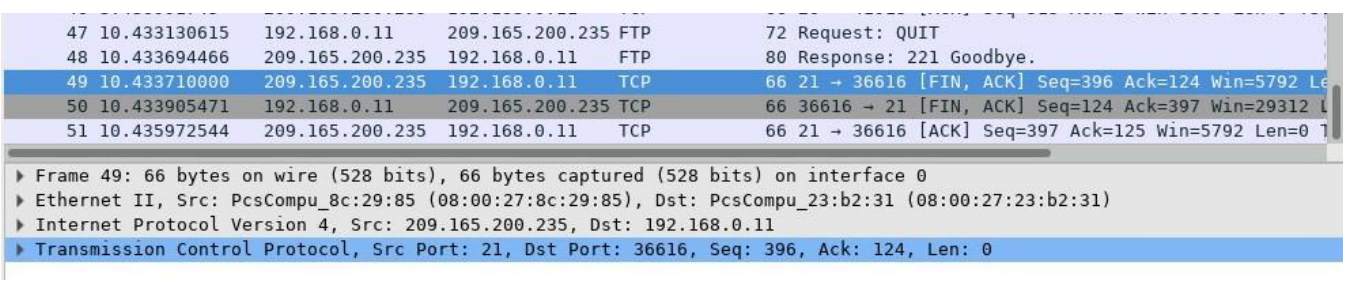
By applying an `ftp` filter, the entire sequence of the *FTP* traffic can be examined in *Wireshark*. Notice the sequence of the events during this *FTP* session. The username *msfadmin* was used to retrieve the *Readme* file. After the file transfer completed, the user ended the *FTP* session.




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 49. The PC sends an *FIN-ACK* to acknowledge the receipt of the FIN to terminate the session from the server to the client in frame 50.

In frame 51, the *FTP* server sends a *ACK* to the PC to acknowledge the termination of the *TCP* session. Now the *TCP* session is terminated between the *FTP* server and PC.




## 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.

a. In a terminal, start **Mininet**. Enter `cyberops` as the password when prompted.

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

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]#
```

## Step 2: Create a file for tftp transfer.

a. 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
```

b. 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.
```

c. 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**.

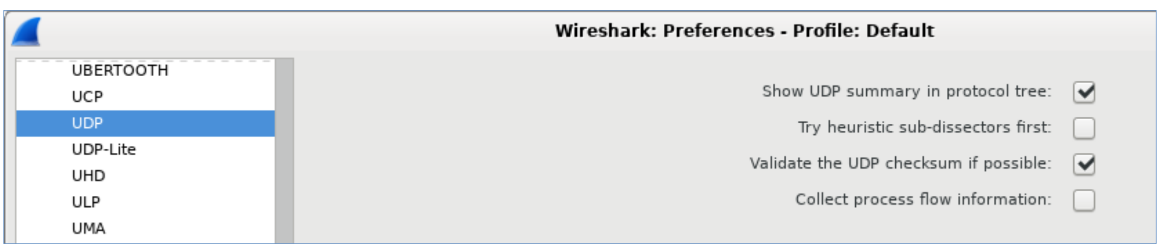
```
[root@secOps analyst]# touch my_tftp_data
```

## Step 3: Capture a TFTP session in Wireshark

a. Start Wireshark in **H1**.

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

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 **Apply**. Then click **OK**.

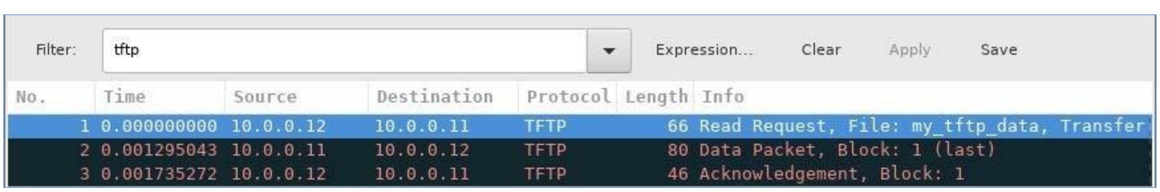



c. Start a *Wireshark* capture on the interface **H1-eth0**.

d. Start a tftp session from **H2** to the tftp server on **H1** and get the file **my_tftp_data**.

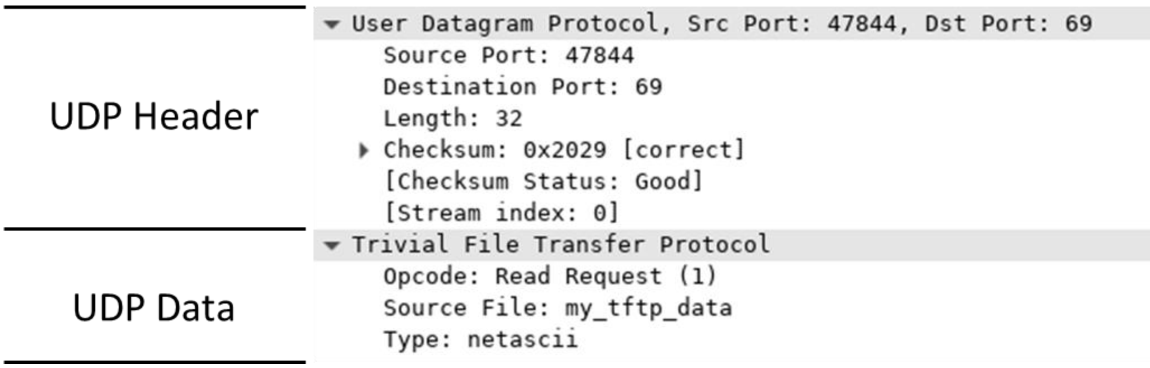
```
[root@secOps analyst]# tftp 10.0.0.11 -c get my_tftp_data
```

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.

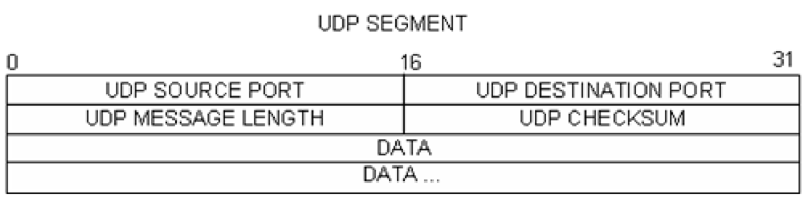


Detailed *UDP* information is available in the *Wireshark* packet details pane. Highlight the **first UDP datagram** from the host computer and move the mouse pointer to the packet details pane. It may be necessary to adjust the packet details pane and expand the UDP record by clicking the protocol expand 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:

| | |
|---|---|
| Source IP address | 10.0.0.12 |
| Destination IP address | 10.0.0.11 |
| Source port number | 34414 |
| Destination port number | 69 |
| UDP message length | 32 |
| UDP checksum | 0x549f |

How does *UDP* verify datagram integrity?

The checksum is sent in the UDP and datagram checksum is refigured upon the receipt when it is identical to the sent checksum then UDP is assumed to be complete.

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

| | |
|---|---|
| Source IP address | 10.0.0.11 |
| Destination IP address | 10.0.0.12 |
| Source port number | 40828 |
| Destination port number | 34414 |
| UDP message length | 46 |
| UDP checksum | 0x0488 |

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
```

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

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

## Reflection

This lab provided the opportunity to analyze *TCP* and *UDP* protocol operations from captured *FTP* and *TFTP* sessions. How does *TCP* manage communication differently than *UDP*?

Both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are transport layer protocols used for communication over the Internet. The TCP is reliable and its is full-duplex connection between two endpoints and it uses three-way handshake process this process are SYN, SYN-ACK, ACK and coming to UDP it is a simpler it is a faster protocol it don't give the guarantee delivery of packets.