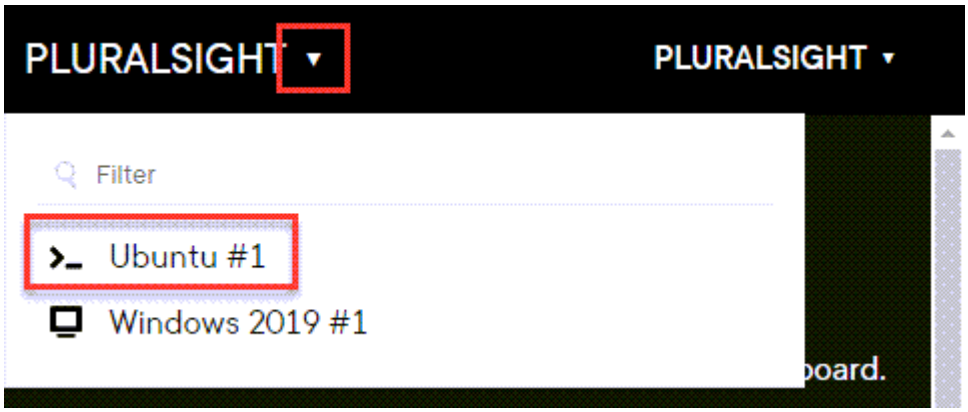
Inspecting TCP Flags and Ports

For this next challenge, you will be using the Ubuntu endpoint.

To switch:

- `CTRL` + `ALT` + `SHIFT` (Windows)
- `CTRL` + `CMD` + `SHIFT` (macOS)

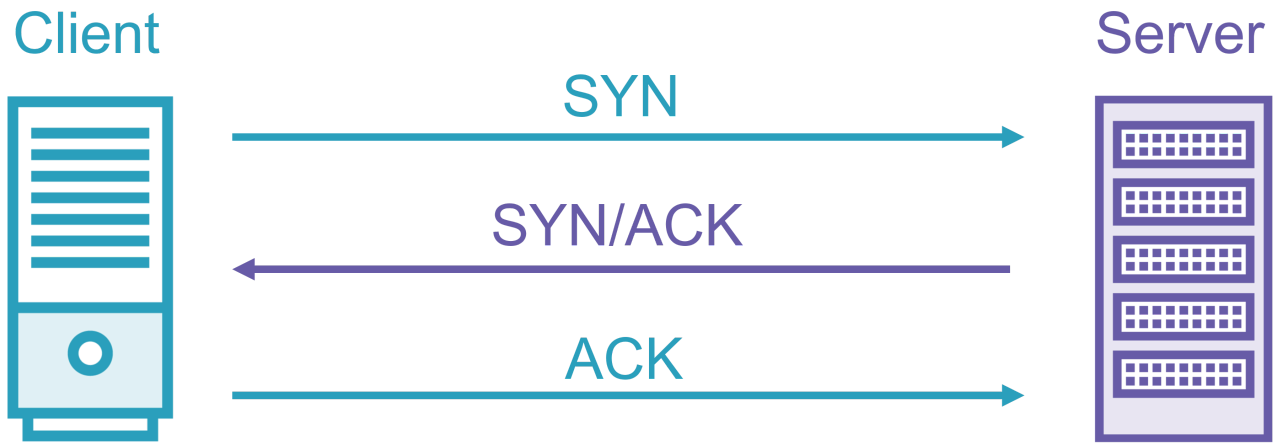


Important Note: We are building a custom environment just for you! This may take a few minutes depending on its complexity.

- In the terminal environment, typing status will return an overview on applications installing and/or lab files being downloaded. Once this returns "SYSTEM COMPLETE", your environment will be ready for full exploration!

In a typical TCP connection, before the application protocol begins communicating, a three-way handshake will take place to sync the packets and validate the port is open for communication. This handshake follows this typical flow:

- Client sends a TCP packet with the "SYN" flag set
- Server responds with the "SYN" and "ACK" flags set
- Client sends the final packet with the "ACK" flag set



Now the handshake is complete and protocol communication can begin! This process is repeated before every conversation.

Note: More information about the individual TCP flags can be found here: Transmission Control Protocol - Wikipedia

1. To begin, take a look at a normal HTTPS conversation.

```
tcpdump -nn -r simple-https.pcap | more
```

The TCP flags can be seen by looking inside the square brackets right after the word "Flags". Tcpdump uses a short-notation for referencing the TCP flags, and you can see the three-way handshake in those first three packets. In order, `[S]` for "SYN", `[S.]` for "SYN" and "ACK", and `[.]` for "ACK".

```
19:10:14.616601 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [S], seq 4192101519, win 29200, options [mss 1460,sackO
, length 0
19:10:14.647251 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [S.], seq 1778126322, ack 4192101520, win 64240, option
19:10:14.647336 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [.], ack 1, win 29200, length 0
19:10:14.647703 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 1:518, ack 1, win 29200, length 517
19:10:14.647959 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 518, win 64240, length 0
19:10:14.677822 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [P.], seq 1:157, ack 518, win 64240, length 156
19:10:14.677894 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [.], ack 157, win 30016, length 0
19:10:14.678305 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 518:569, ack 157, win 30016, length 51
19:10:14.678527 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 569, win 64240, length 0
19:10:14.678651 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 569:746, ack 157, win 30016, length 177
19:10:14.678746 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 746, win 64240, length 0
19:10:14.748978 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 746:1209, ack 157, win 30016, length 463
19:10:14.749237 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 1209, win 64240, length 0
19:10:14.778031 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [P.], seq 157:264, ack 1209, win 64240, length 107
19:10:14.778213 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 1209:1247, ack 264, win 30016, length 38
19:10:14.778346 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 1247, win 64240, length 0
19:10:14.813041 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 1247:1289, ack 264, win 30016, length 42
19:10:14.813228 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 1289, win 64240, length 0
19:10:14.841783 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [P.], seq 264:733, ack 1289, win 64240, length 469
19:10:14.842144 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 1289:1335, ack 733, win 31088, length 46
19:10:14.842327 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 1335, win 64240, length 0
19:10:14.877519 IP 192.168.218.131.33356 > 172.217.9.132.443: Flags [P.], seq 1335:1444, ack 733, win 31088, length 109
19:10:14.877660 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [.], ack 1444, win 64240, length 0
```

Note: Let's break down the `tcpdump` command to understand the additional options selected:

`-nn` = No IP or port resolution. For both operational security reasons and to ensure `tcpdump` isn't presenting you with modified information, these flags are important

`-r` = Allows you to "read-in" a PCAP instead of capturing packets off a live interface. This is followed by the name of the PCAP in your current directory.

2. You can determine how many successful connections took place by filtering on SYN/ACK packets

```
tcpdump -nn -r simple-https.pcap 'tcp[tcpflags] == (tcp-syn|tcp-ack)'
```

You will notice only one packets gets returned where the server, 172.217.9.132 responds back to the client on port 443 (the typical port for HTTPS).

```
reading from file simple-https.pcap, link-type EN10MB (Ethernet)
19:10:14.647251 IP 172.217.9.132.443 > 192.168.218.131.33356: Flags [S.], seq 17781
```

3. If the PCAP had more connections, a simpler way to view the output from multiple connections could be the following:

```
tcpdump -nn -r simple-https.pcap 'tcp[tcpflags] == (tcp-syn|tcp-ack)' | cut -f 3,7 -d " "
reading from file simple-https.pcap, link-type EN10MB (Ethernet)
172.217.9.132.443 [S.],
```

Note: The `cut` command allows you to filter command line output to show only your desired data.

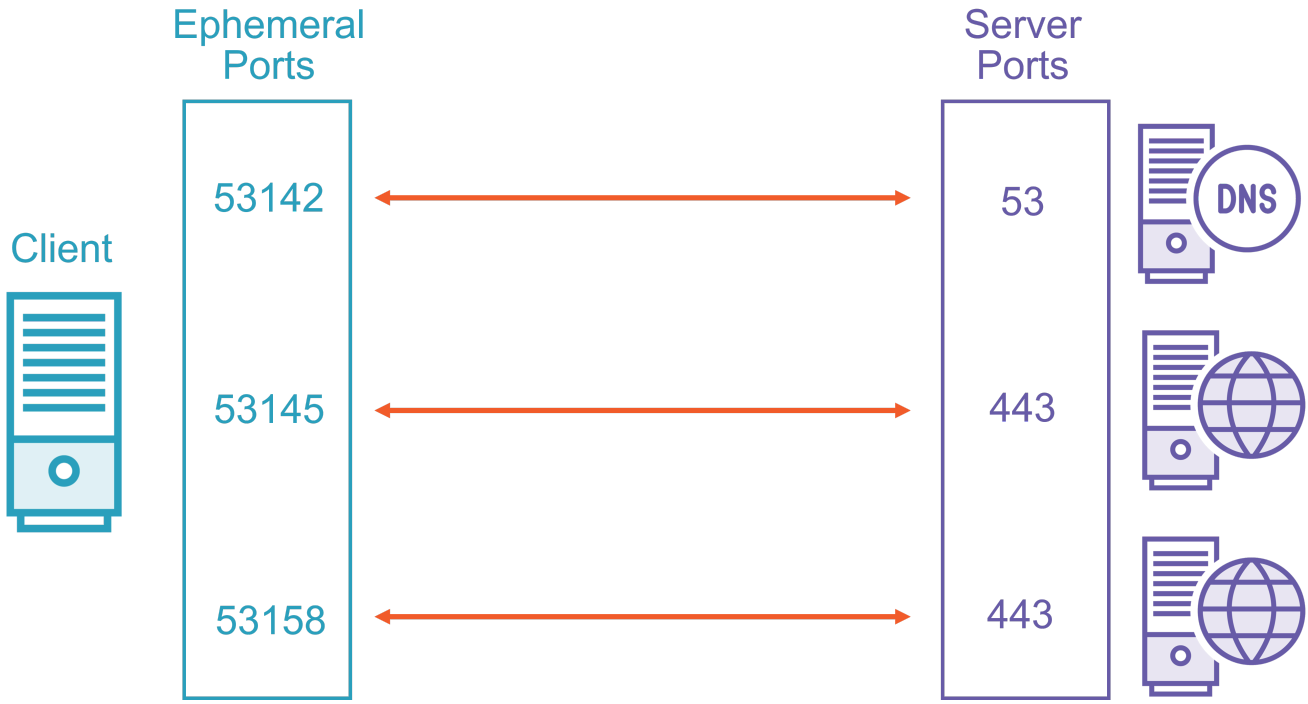
`-f` = Specifies the field (or column)

`-d` = Specifies the delimiter. In this case you are splitting by a space, so you put a space in between quotation marks, but this can be any character

---

At this point you may want to check and see if additional conversations over other ports were attempted. However, you'll first need to understand how ports work within TCP and UDP. Logical ports, or sockets, are the mechanism used to separate services and conversations between hosts. The client attempting to communicate will use a range of ports referred to as ephemeral ports. These are randomized high-value numbers and typically range between 49152 and 65535, although this can vary depending on configuration and operating system.

Ephemeral ports allow a client to communicate with multiple services at the same time without overlapping the conversations.



4. For this next session you will look at a larger PCAP to see multiple conversations. You can take a look at the ephemeral ports by filtering on the client host and looking at all the ports it is using:

```
tcpdump -nn -r normal-web-traffic.pcap 'src host 192.168.218.131' | cut -f 3 -d " "
```

```
192.168.218.131.33818
192.168.218.131.35110
192.168.218.131.51530
192.168.218.131.35374
192.168.218.131.55546
192.168.218.131.49026
192.168.218.131.49008
192.168.218.131.37668
192.168.218.131.57286
192.168.218.131.47972
192.168.218.131.47650
192.168.218.131.47972
192.168.218.131.47650
```

Note: `tcpdump` uses dotted notation to separate the IPs and ports. The first four octets (192.168.218.131) represent the IP and the final octet is the port (as seen highlighted).

5. Since that was a lot of data that scrolled through the command line, you can simplify the output by sorting and looking at only unique values:

```
tcpdump -nn -r normal-web-traffic.pcap 'src host 192.168.218.131' | cut -f 3 -d " " | sort | uniq -c | sort -n
```

```
 52 192.168.218.131.40646
 56 192.168.218.131.54118
 58 192.168.218.131.48392
 59 192.168.218.131.42310
 60 192.168.218.131.37056
 66 192.168.218.131.55070
 75 192.168.218.131.50418
 82 192.168.218.131.35684
 88 192.168.218.131.37746
 91 192.168.218.131.43898
103 192.168.218.131.51530
107 192.168.218.131.32876
119 192.168.218.131.35686
141 192.168.218.131.40670
262 192.168.218.131.33356
```

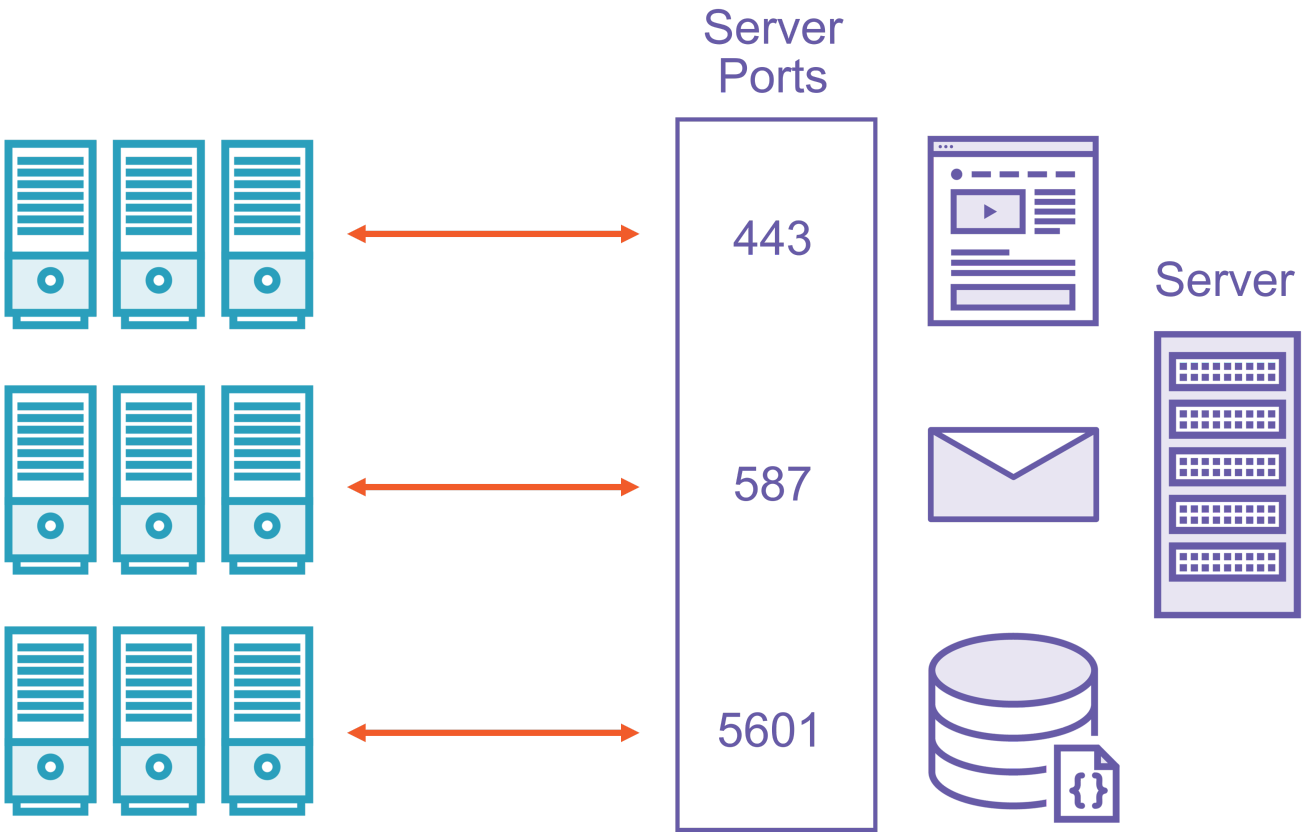
Note: The `sort` and `unig` commands allow you to filter command line output and return a cleanly sorted and unique list of results. You must sort the list first since the `unig` command follows a top-down approach and will display unexpected results on an un-sorted list.

`uniq -c` = provides a unique list of values and presents a count in the left-column on how many times that value was seen

`sort -n` = sorts a list numerically with the highest values at the bottom

Analysis: The output from this is expected for normal conversations. A client should have a long list of ephemeral ports with a relatively low count on total number of packets seen for each ephemeral port used. What you can deduce is the conversation using port "33356" had 262 packets sent from the client meaning this specific conversation lasted longer or had more data sent back and forth. If a specific port had a massive amount of packets sent using one specific ephemeral port, that could be an indicator of malicious activity like a beacon or a command and control (C2) channel.

In contrast, a server will host applications on various ports to deconflict multiple services. Many clients may connect to the same logical port depending on what service is being requested. For instance, a single server may host a public webpage, an email service, and a custom application. Since the server has a single IP address, the ports are used to differentiate between services.



6. To look at the various services our client is communicating with, you can use the same method as above while cutting out the responding IPs and ports.

```
tcpdump -nn -r normal-web-traffic.pcap 'src host 192.168.218.131' | cut -f 5
-d " " | sort | uniq -c | sort -n
```

```
118 23.54.150.212.443:
119 31.13.65.7.443:
120 35.190.62.200.443:
125 104.73.13.101.443:
148 213.208.154.14.443:
151 143.204.38.46.443:
155 143.204.29.46.443:
179 23.54.149.220.443:
201 23.54.149.134.443:
208 104.36.113.17.443:
293 172.217.9.132.443:
294 104.24.12.54.80:
648 192.168.218.2.53:
```

Analysis: You can see that 648 of the conversations were to a single host over port 53. This is DNS traffic to a server within the same subnet as the client IP and can be assumed to be the Active Directory server hosting DNS within the environment. The rest of the conversations take place over port 80 and 443, typical for normal web traffic.

Note: While there is a list of commonly associated ports with services (e.g. 443/HTTPS, 80/HTTP, 53/DNS, etc.), any port can be used to host any application. Considering security, an adversary may use a common port like 443 to hide malicious traffic amongst your normal traffic.

7. You can take this one step further by removing the IP addresses and getting a total count of ports seen:

```
tcpdump -nn -r normal-web-traffic.pcap 'src host 192.168.218.131' | cut -f 5 -d " " | cut -f 5 -d "." | sort | uniq -c | sort -n
```

```
   4 123:
 648 53:
1182 80:
6467 443:
```

Note: The additional `cut` command is added using a period as a delimiter to isolate the port numbers.

Analysis: You can see the same ports investigated before (443, 80, and 53), but now you can also see 4 packets sent to port 123. This is the typical port for

Network Time Protocol (NTP).

8. To validate any interesting connections, you can take the output from your total list of ports and create a new filter to check whether the connections are successful or not and if they are destined for a valid IP. For this example, use port 123:

```
tcpdump -nn -r normal-web-traffic.pcap 'host 192.168.218.131 and port 123'
19:11:04.361259 IP 192.168.218.131.48726 > 129.250.35.251.123: NTPv4, Client, length 48
19:11:04.392970 IP 129.250.35.251.123 > 192.168.218.131.48726: NTPv4, Server, length 48
19:11:07.415692 IP 192.168.218.131.50176 > 198.84.61.242.123: NTPv4, Client, length 48
19:11:07.484772 IP 198.84.61.242.123 > 192.168.218.131.50176: NTPv4, Server, length 48
19:11:07.616303 IP 192.168.218.131.55353 > 45.56.118.161.123: NTPv4, Client, length 48
19:11:07.670727 IP 45.56.118.161.123 > 192.168.218.131.55353: NTPv4, Server, length 48
19:11:07.912188 IP 192.168.218.131.45352 > 34.225.6.20.123: NTPv4, Client, length 48
19:11:07.979880 IP 34.225.6.20.123 > 192.168.218.131.45352: NTPv4, Server, length 48
```

Analysis: Here you can see the internal client successfully reaching out to four separate servers requesting NTP. Depending on your network policy and configuration this may be normal or anomalous. This emphasizes how important it is to understand what your clients should be reaching out to and what protocols they should be using. Additionally, you will notice NTP uses UDP for transport, so TCP flags will not be present, and you will have to rely upon looking at both sides of any UDP conversations to see whether data was successfully returned.

These methods may be used on any network capture to baseline conversations and get a basic understanding of what application protocols are being used within an environment. Understanding the data flow and transport protocol functionality is critical if you are expected to find anomalies amongst your normal traffic. In the next challenge, you will look at an example of malicious traffic that can be identified through the use of TCP flags.