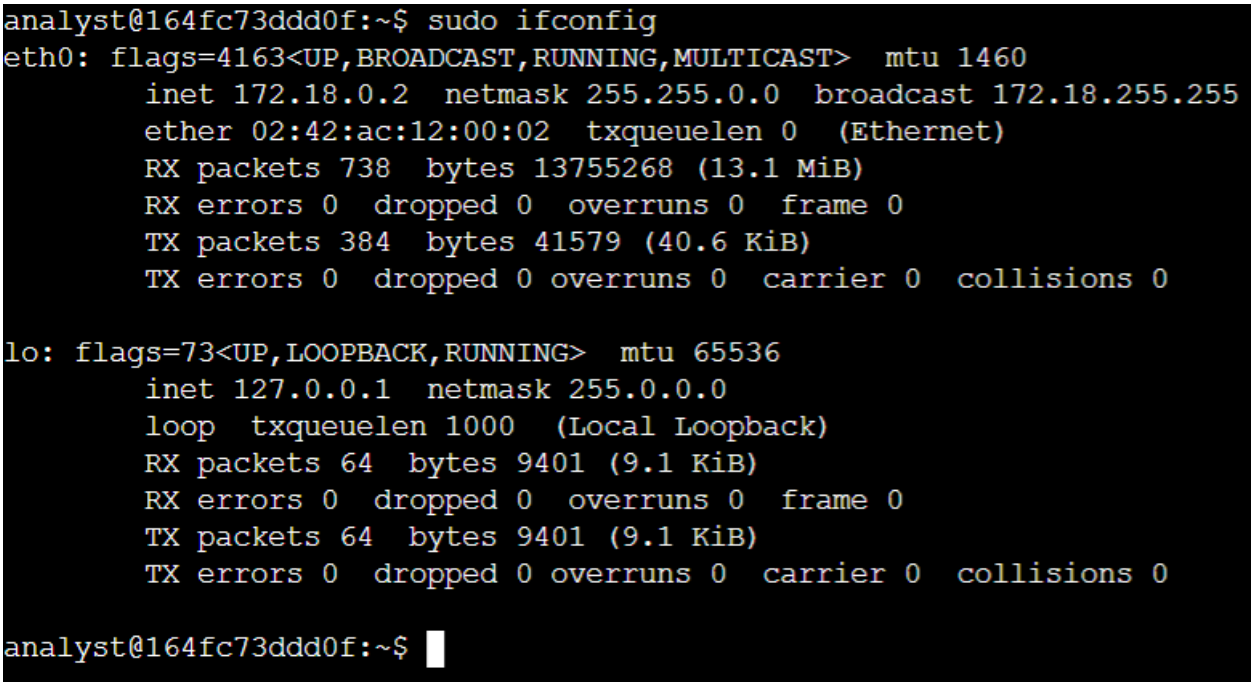
Task 1. Identify network interfaces

In this task, you must identify the network interfaces that can be used to capture network packet data.

1. Use `ifconfig` to identify the interfaces that are available:

```
sudo ifconfig
```

This command returns output similar to the following:



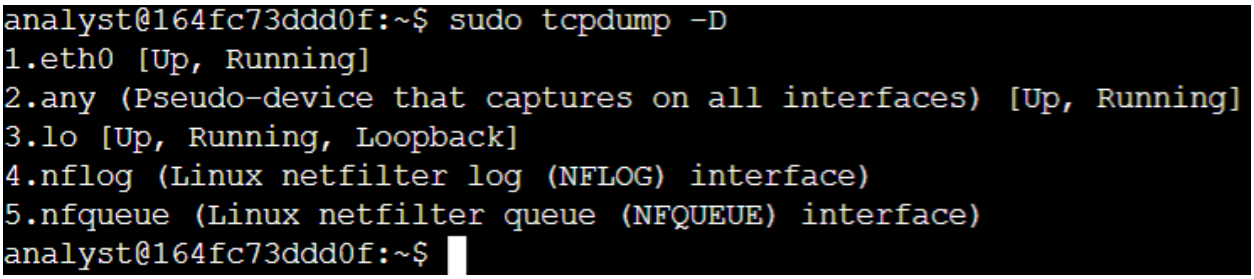
The Ethernet network interface is identified by the entry with the `eth` prefix.

So, in this lab, you'll use `eth0` as the interface that you will capture network packet data from in the following tasks.

2. Use `tcpdump` to identify the interface options available for packet capture:

```
sudo tcpdump -D
```

This command will also allow you to identify which network interfaces are available. This may be useful on systems that do not include the `ifconfig` command.



Task 2. Inspect the network traffic of a network interface with tcpdump

In this task, you must use `tcpdump` to filter live network packet traffic on an interface.

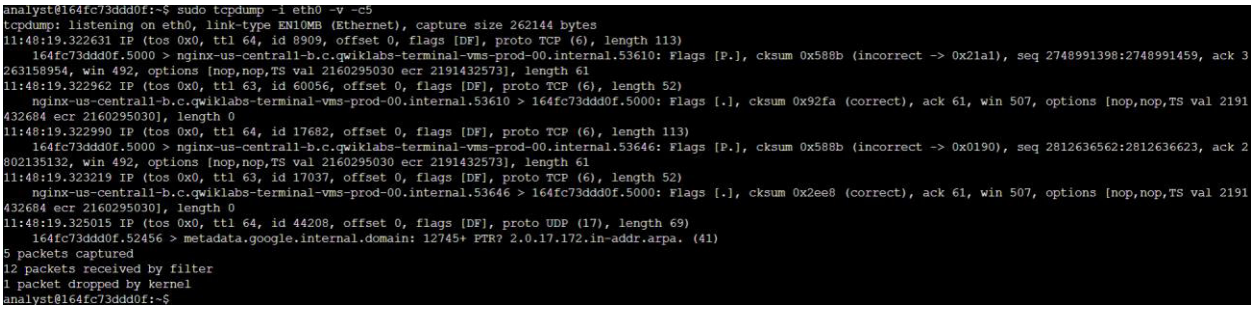
- Filter live network packet data from the `eth0` interface with `tcpdump`:

```
sudo tcpdump -i eth0 -v -c5
```

This command will run `tcpdump` with the following options:

- `-i eth0`: Capture data specifically from the `eth0` interface.
- `-v`: Display detailed packet data.
- `-c5`: Capture 5 packets of data.

Now, let's take a detailed look at the packet information that this command has returned.



Exploring network packet details

In this example, you'll identify some of the properties that `tcpdump` outputs for the packet capture data you've just seen.

1. In the example data at the start of the packet output, `tcpdump` reported that it was listening on the `eth0` interface, and it provided information on the link type and the capture size in bytes:

```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

2. On the next line, the first field is the packet's timestamp, followed by the protocol type, IP:

```
11:48:19.322631 IP (tos 0x0, ttl 64, id 8909, offset 0, flags [DF], proto TCP (6), length 113)
```

3. The verbose option, `-v`, has provided more details about the IP packet fields, such as TOS, TTL, offset, flags, internal protocol type (in this case, TCP (6)), and the length of the outer IP packet in bytes:

```
11:48:19.322631 IP (tos 0x0, ttl 64, id 8909, offset 0, flags [DF], proto TCP (6), length 113)
```

4. In the next section, the data shows the systems that are communicating with each other:

```
164fc73ddd0f.5000 > nginx-us-central1-b.c.qwiklabs-terminal-vms-prod-00.internal.53610: Flags [P.], cksum 0x588b (incorrect -> 0x21a1), seq 2748991398:2748991459, ack 3
263158954, win 492, options [nop,nop,TS val 2160295030 ecr 2191432573], length 61
```

By default, `tcpdump` will convert IP addresses into names, as in the screenshot. The name of your Linux virtual machine, also included in the command prompt, appears here as the source for one packet and the destination for the second packet. In your live data, the name will be a different set of letters and numbers.

The direction of the arrow (>) indicates the direction of the traffic flow in this packet. Each system name includes a suffix with the port number (.5000 in the screenshot), which is used by the source and the destination systems for this packet.

5. The remaining data filters the header data for the inner TCP packet:

```
164fc73ddd0f.5000 > nginx-us-central1-b.c.qwiklabs-terminal-vms-prod-00.internal.53610: Flags [P.], cksum 0x588b (incorrect -> 0x21a1), seq 2748991398:2748991459, ack 3
263158954, win 492, options [nop,nop,TS val 2160295030 ecr 2191432573], length 61
```

The flags field identifies TCP flags. In this case, the P represents the push flag and the period indicates it's an ACK flag. This means the packet is pushing out data.

The next field is the TCP checksum value, which is used for detecting errors in the data.

This section also includes the sequence and acknowledgement numbers, the window size, and the length of the inner TCP packet in bytes.

Task 3. Capture network traffic with tcpdump

In this task, you will use `tcpdump` to save the captured network data to a packet capture file.

In the previous command, you used `tcpdump` to stream all network traffic. Here, you will use a filter and other `tcpdump` configuration options to save a small sample that contains only web (TCP port 80) network packet data.

1. Capture packet data into a file called `capture.pcap`:

```
sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap &
```

You must press the **ENTER** key to get your command prompt back after running this command.

This command will run `tcpdump` in the background with the following options:

- `-i eth0`: Capture data from the `eth0` interface.
- `-nn`: Do not attempt to resolve IP addresses or ports to names.This is best practice from a security perspective, as the lookup data may not be valid. It also prevents malicious actors from being alerted to an investigation.
- `-c9`: Capture 9 packets of data and then exit.
- `port 80`: Filter only port 80 traffic. This is the default HTTP port.
- `-w capture.pcap`: Save the captured data to the named file.
- `&`: This is an instruction to the Bash shell to run the command in the background.

This command runs in the background, but some output text will appear in your terminal. The text will not affect the commands when you follow the steps for the rest of the lab.

2. Use `curl` to generate some HTTP (port 80) traffic:

```
curl opensource.google.com
```

When the `curl` command is used like this to open a website, it generates some HTTP (TCP port 80) traffic that can be captured.

```
curl opensource.google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://opensource.google/">here</A>.
</BODY></HTML>
analyst@164fc73ddd0f:~$ 9 packets captured
10 packets received by filter
0 packets dropped by kernel
```

3. Verify that packet data has been captured:

```
ls -l capture.pcap
```

Note: *The "Done" in the output indicates that the packet was captured.*

```
ls -l capture.pcap
-rw-r--r-- 1 root root 1401 Jan 21 12:03 capture.pcap
[1]+  Done                    sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap
```

Task 4. Filter the captured packet data

In this task, use `tcpdump` to filter data from the packet capture file you saved previously.

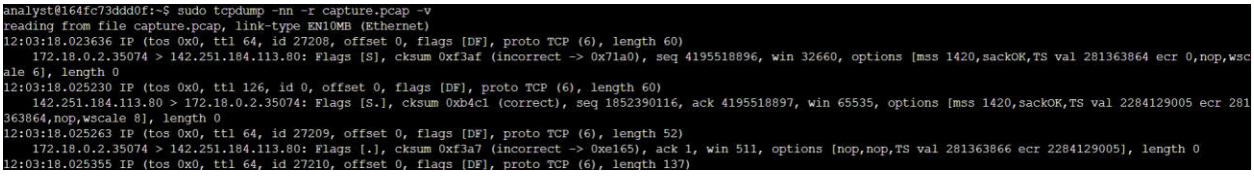
1. *Use the* `tcpdump` *command to filter the packet header data from the* `capture.pcap` *capture file:*

```
sudo tcpdump -nn -r capture.pcap -v
```

This command will run `tcpdump` *with the following options:*

- `-nn`*: Disable port and protocol name lookup.*
- `-r`*: Read capture data from the named file.*
- `-v`*: Display detailed packet data.*

You must specify the `-nn` *switch again here, as you want to make sure* `tcpdump` *does not perform name lookups of either IP addresses or ports, since this can alert threat actors.*



As in the previous example, you can see the IP packet information along with information about the data that the packet contains.

2. *Use the* `tcpdump` *command to filter the extended packet data from the* `capture.pcap` *capture file:*

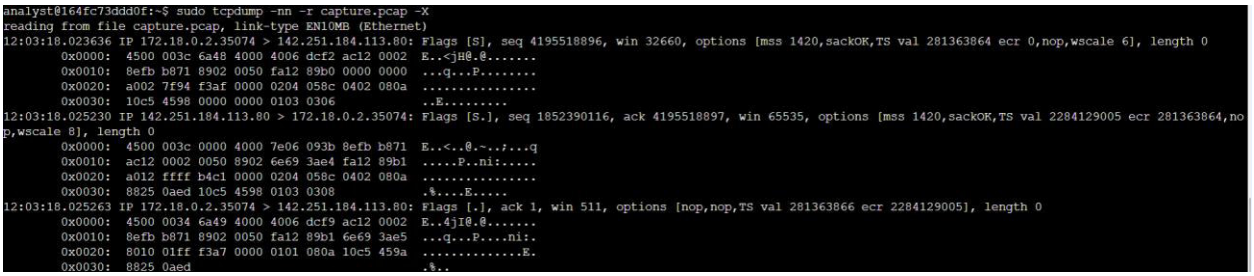
```
sudo tcpdump -nn -r capture.pcap -X
```

This command will run `tcpdump` *with the following options:*

- `-nn`*: Disable port and protocol name lookup.*

- *`-r`: Read capture data from the named file.*
- *`-X`: Display the hexadecimal and ASCII output format packet data. Security analysts can analyze hexadecimal and ASCII output to detect patterns or anomalies during malware analysis or forensic analysis.*

Note: *Hexadecimal, also known as hex or base 16, uses 16 symbols to represent values, including the digits 0-9 and letters A, B, C, D, E, and F. American Standard Code for Information Interchange (ASCII) is a character encoding standard that uses a set of characters to represent text in digital form.*



Conclusion

I have gained practical experience to enable you to

- *identify network interfaces,*
- *use the `tcpdump` command to capture network data for inspection,*
- *interpret the information that `tcpdump` outputs regarding a packet, and*
- *save and load packet data for later analysis.*