TCPDump - capture your first packet

Scenario

You're a network analyst who needs to use tcpdump to capture and analyze live network traffic from a Linux virtual machine.
The lab starts with your analyst user account, already logged in to a Linux terminal.
Your Linux user's home directory contains a sample packet capture file that you will use at the end of the lab to answer a few questions about the network traffic that it contains.
Here's how you'll do this: First, you'll identify network interfaces to capture packet data. Second, you'll use tcpdump to filter live network traffic. Third, you'll capture network traffic using tcpdump. Finally, you'll filter the captured packet data.

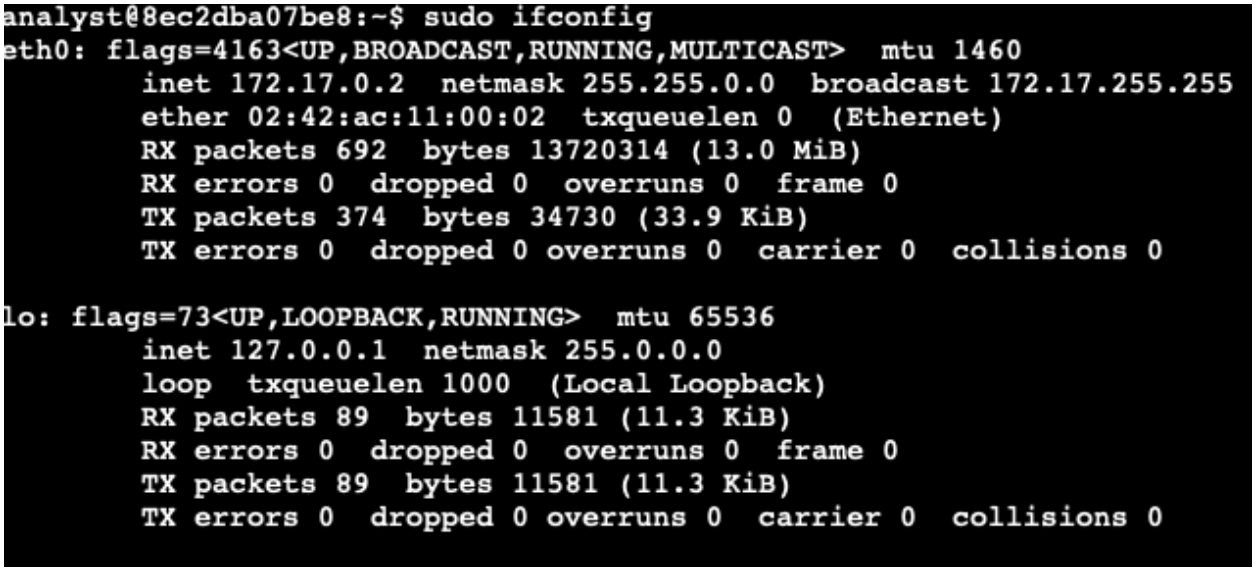
Task 1 - Identify Network Interfaces

In this task, I've identified the network interfaces that can be used to capture network packet data.

1. Use ifconfig to identify the available interfaces:

   Command: sudo ifconfig

   Output:



The entry identifies the Ethernet network interface with the eth prefix.

So, in this lab, I'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:

   Command: sudo tcpdump -D

   Output:

```
analyst@8ec2dba07be8:~$ sudo tcpdump -D
1.eth0 [Up, Running]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.nflog (Linux netfilter log (NFLOG) interface)
5.nfqueue (Linux netfilter queue (NFQUEUE) interface)
analyst@8ec2dba07be8:~$
```

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, I'll use `tcpdump` to filter live network packet traffic on an interface.

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

Command: 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.

Output:

```
analyst@8ec2dba07be8:~$ sudo tcpdump -D
1.eth0 [Up, Running]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.nflog (Linux netfilter log (NFLOG) interface)
5.nfqueue (Linux netfilter queue (NFQUEUE) interface)
analyst@8ec2dba07be8:~$ sudo tcpdump -i etho -v -c5
tcpdump: etho: No such device exists
(SIOCGIFHWADDR: No such device)
analyst@8ec2dba07be8:~$ sudo tcpdump -i eth0 -v -c5
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:18:00.855807 IP (tos 0x0, ttl 64, id 60387, offset 0, flags [DF], proto TCP (6), length 61)
    8ec2dba07be8.5000 > nginx-us-east1-b.c.qwiklabs-terminal-vms-prod-00.internal.51820: Flags [P.], cksu
m 0x5857 (incorrect -> 0x910e), seq 1471662570:1471662579, ack 3402523918, win 501, options [nop,nop,TS v
al 593580239 ecr 4183570531], length 9
15:18:00.856027 IP (tos 0x0, ttl 63, id 8383, offset 0, flags [DF], proto TCP (6), length 52)
    nginx-us-east1-b.c.qwiklabs-terminal-vms-prod-00.internal.51820 > 8ec2dba07be8.5000: Flags [.], cksum
 0x46fc (correct), ack 9, win 507, options [nop,nop,TS val 4183570580 ecr 593580239], length 0
15:18:00.856902 IP (tos 0x0, ttl 64, id 10318, offset 0, flags [DF], proto UDP (17), length 69)
    8ec2dba07be8.37413 > metadata.google.internal.domain: 8992+ PTR? 2.0.18.172.in-addr.arpa. (41)
15:18:00.861812 IP (tos 0x0, ttl 63, id 0, offset 0, flags [none], proto UDP (17), length 140)
    metadata.google.internal.domain > 8ec2dba07be8.37413: 8992 1/0/0 2.0.18.172.in-addr.arpa. PTR nginx-u
s-east1-b.c.qwiklabs-terminal-vms-prod-00.internal. (112)
15:18:00.862829 IP (tos 0x0, ttl 64, id 4780, offset 0, flags [DF], proto UDP (17), length 74)
    8ec2dba07be8.58240 > metadata.google.internal.domain: 22099+ PTR? 254.169.254.169.in-addr.arpa. (46)
5 packets captured
6 packets received by filter
0 packets dropped by kernel
```

Exploring network packet details

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

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.
4. In the next section, the data shows the systems that are communicating with each other. 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. 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 acknowledgment numbers, the window size, and the length of the inner TCP packet in bytes.

Task 3 - Capture the network traffic using tcpdump

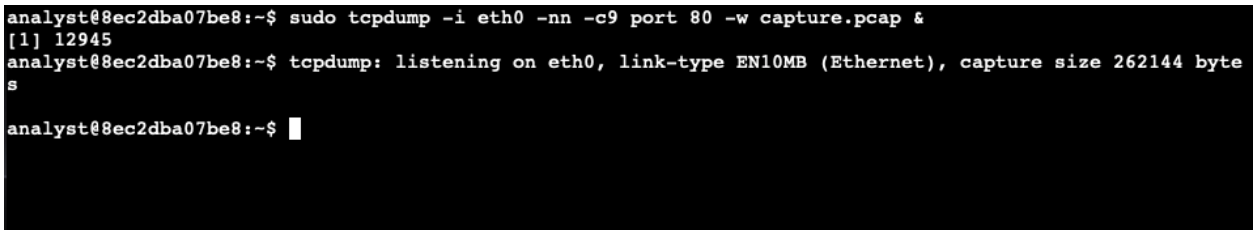
I'll use `tcpdump` to save the captured network data to a packet capture file.

In the previous command, I used `tcpdump` to stream all network traffic. Here, I 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`

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

Output:



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

   Command: curl opensource.google.com

Output:

```
analyst@8ec2dba07be8:~$ 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@8ec2dba07be8:~$ 9 packets captured
10 packets received by filter
0 packets dropped by kernel
```

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

3. Verify that packet data has been captured.

   Command: ls -l capture.pcap

   Output:

```
analyst@8ec2dba07be8:~$ ls -l capture.pcap
-rw-r--r-- 1 root root 1455 Mar 26 15:51 capture.pcap
analyst@8ec2dba07be8:~$
```

Task 4 - Filter the captured packet data

Here, we'll 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

   Command: 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.

Output:

```
analyst@7c356ee8d3cb:~$ sudo tcpdump -nn -r capture.pcap -v
reading from file capture.pcap, link-type EN10MB (Ethernet)
16:07:08.666963 IP (tos 0x0, ttl 64, id 33372, offset 0, flags [DF], proto TCP (6), length 60)
    172.17.0.2.53360 > 173.194.213.138.80: Flags [S], cksum 0x2f8f (incorrect -> 0xfe4b), seq 3701989374,
 win 65320, options [mss 1420,sackOK,TS val 490762411 ecr 0,nop,wscale 7], length 0
16:07:08.667720 IP (tos 0x0, ttl 126, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    173.194.213.138.80 > 172.17.0.2.53360: Flags [S.], cksum 0xe0a5 (correct), seq 540800173, ack 3701989
375, win 65535, options [mss 1420,sackOK,TS val 961793664 ecr 490762411,nop,wscale 8], length 0
16:07:08.667761 IP (tos 0x0, ttl 64, id 33373, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.2.53360 > 173.194.213.138.80: Flags [.], cksum 0x2f87 (incorrect -> 0x0d4c), ack 1, win 511,
 options [nop,nop,TS val 490762411 ecr 961793664], length 0
16:07:08.667822 IP (tos 0x0, ttl 64, id 33374, offset 0, flags [DF], proto TCP (6), length 137)
    172.17.0.2.53360 > 173.194.213.138.80: Flags [P.], cksum 0x2fdc (incorrect -> 0x7bfe), seq 1:86, ack
1, win 511, options [nop,nop,TS val 490762412 ecr 961793664], length 85: HTTP, length: 85
        GET / HTTP/1.1
        Host: opensource.google.com
        User-Agent: curl/7.64.0
        Accept: */*

16:07:08.667995 IP (tos 0x0, ttl 126, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    173.194.213.138.80 > 172.17.0.2.53360: Flags [.], cksum 0x0df5 (correct), ack 86, win 256, options [n
op,nop,TS val 961793664 ecr 490762412], length 0
16:07:08.670260 IP (tos 0x0, ttl 126, id 0, offset 0, flags [DF], proto TCP (6), length 645)
    173.194.213.138.80 > 172.17.0.2.53360: Flags [P.], cksum 0x1916 (correct), seq 1:594, ack 86, win 256
, options [nop,nop,TS val 961793667 ecr 490762412], length 593: HTTP, length: 593
        HTTP/1.1 301 Moved Permanently
        Location: https://opensource.google/
        Cross-Origin-Resource-Policy: cross-origin
        X-Content-Type-Options: nosniff
        Server: sffe
        Content-Length: 223
        X-XSS-Protection: 0
        Date: Tue, 26 Mar 2024 15:47:30 GMT
        Expires: Tue, 26 Mar 2024 16:17:30 GMT
        Cache-Control: public, max-age=1800
        Content-Type: text/html; charset=UTF-8
        Age: 1178

        <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>
16:07:08.670265 IP (tos 0x0, ttl 64, id 33375, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.2.53360 > 173.194.213.138.80: Flags [.], cksum 0x2f87 (incorrect -> 0x0aa4), ack 594, win 50
7, options [nop,nop,TS val 490762414 ecr 961793667], length 0
16:07:08.671598 IP (tos 0x0, ttl 64, id 33376, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.2.53360 > 173.194.213.138.80: Flags [F.], cksum 0x2f87 (incorrect -> 0x0aa2), seq 86, ack 59
4, win 507, options [nop,nop,TS val 490762415 ecr 961793667], length 0
16:07:08.672033 IP (tos 0x0, ttl 126, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    173.194.213.138.80 > 172.17.0.2.53360: Flags [F.], cksum 0x0b9b (correct), seq 594, ack 87, win 256,
options [nop,nop,TS val 961793668 ecr 490762415], length 0
```

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

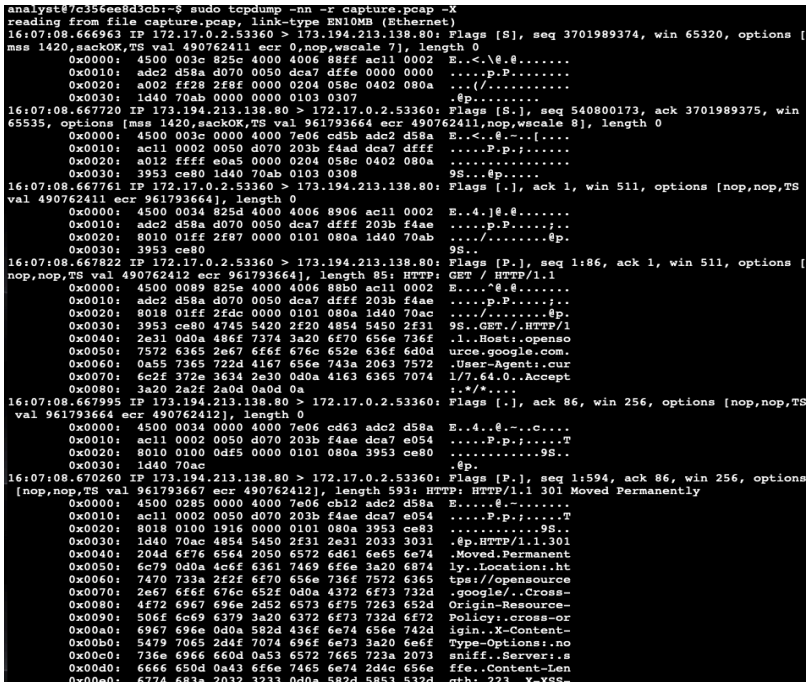
2. Use the `tcpdump` command to filter the extended packet data from the `capture.pcap` capture file.

   Command: 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.

   Output:



Conclusion

Gained practical experience in:

- 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.