Capturing and Filtering Network Traffic with Tcpdump

Tcpdump is a network packet analyzer that utilizes a command line interface. In this project tcpdump is run inside of a Linux virtual machine to demonstrate how it can capture and filter web traffic.

Identifying network interfaces

The first thing to do in this project is to identify the network interfaces that could be used to capture packet data. We do this by using the “sudo ifconfig” command.

```
analyst@50d02b144737:~$ sudo ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1460
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 1423  bytes 13798222 (13.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 914  bytes 75509 (73.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 66  bytes 9409 (9.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 66  bytes 9409 (9.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

analyst@50d02b144737:~$
```

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

So here we will use *eth0* as the interface that you will capture network packet data from.

On systems where the “ifconfig” command is not available, “sudo tcpdump -D” could be used.

```
analyst@50d02b144737:~$ 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@50d02b144737:~$
```

Inspecting the network traffic of a network interface

To filter live packet data from the *eth0* interface, we begin by typing in the following command: “sudo tcpdump -i eth0 -v -c5”.

```
analyst@50d02b144737:~$ sudo tcpdump -i eth0 -v -c5
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:56:58.693267 IP (tos 0x0, ttl 64, id 40728, offset 0, flags [DF], proto TCP (6), length 113)
    50d02b144737.5000 > nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662: Flags [P.], cks
um 0x588e (incorrect -> 0x2d62), seq 1320386951:1320387012, ack 2211840941, win 491, options [nop,nop,TS
 val 1282563350 ecr 720789744], length 61
14:56:58.693620 IP (tos 0x0, ttl 63, id 35003, offset 0, flags [DF], proto TCP (6), length 52)
    nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662 > 50d02b144737.5000: Flags [.], cksu
m 0xd865 (correct), ack 61, win 507, options [nop,nop,TS val 720789873 ecr 1282563350], length 0
14:56:58.704247 IP (tos 0x0, ttl 64, id 40729, offset 0, flags [DF], proto TCP (6), length 147)
    50d02b144737.5000 > nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662: Flags [P.], cks
um 0x58b0 (incorrect -> 0x1a16), seq 61:156, ack 1, win 491, options [nop,nop,TS val 1282563361 ecr 7207
89873], length 95
14:56:58.704615 IP (tos 0x0, ttl 63, id 35004, offset 0, flags [DF], proto TCP (6), length 52)
    nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662 > 50d02b144737.5000: Flags [.], cksu
m 0xd7f0 (correct), ack 156, win 507, options [nop,nop,TS val 720789884 ecr 1282563361], length 0
14:56:58.727423 IP (tos 0x0, ttl 64, id 29781, offset 0, flags [DF], proto UDP (17), length 69)
    50d02b144737.55818 > metadata.google.internal.domain: 59430+ PTR? 2.0.21.172.in-addr.arpa. (41)
5 packets captured
8 packets received by filter
0 packets dropped by kernel
```

Let's break down each individual part of the command:

-i eth0: Capture data specifically from the eth0 interface.

-v: Display detailed packet data (verbose).

-c5: Capture 5 packets of data.

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
```

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

```
14:56:58.693267 IP
```

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:

```
(tos 0x0, ttl 64, id 40728, offset 0, flags [DF], proto TCP (6), length 113)
```

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

```
    50d02b144737.5000 > nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662:
```

Tcpdump will convert IP addresses into names, as can be seen in the screenshot. The name of the Linux virtual machine used in the project, also included in the command prompt, appears here as the source for one packet and the destination for the second packet. The direction of the arrow (>) indicates the direction of the traffic flow in each packet. Each system name includes a suffix with the port number (.5000 in this case), which is used by the source and the destination systems for this packet.

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

```
    50d02b144737.5000 > nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.41662: Flags [P.], cks
um 0x588e (incorrect -> 0x2d62), seq 1320386951:1320387012, ack 2211840941, win 491, options [nop,nop,TS
 val 1282563350 ecr 720789744], 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 acknowledgment numbers, the window size, and the length of the inner TCP packet in bytes.

Capture network traffic with tcpdump

Now, let's save the captured network data to a packet capture file. To do this, we will use the command "sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap &"

```
analyst@f9bd8cf17bf5:~$ sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap &
[1] 12869
analyst@f9bd8cf17bf5:~$ tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
es
```

*Note: This part of the project was run on a separate VM, so the name of the machine is different from the previous section.

This command will run tcpdump in the background with the following configuration:

-i eth0: Capture data from the eth0 interface.

-nn: Do not attempt to resolve IP addresses or ports to names. This is considered 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 (default HTTP) traffic. We want to save a small sample that contains only web (TCP port 80) network packet data.

-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.

To generate some HTTP traffic that can be captured, let's run the *curl* command to open the website *opensource.google.com*

```
analyst@f9bd8cf17bf5:~$ 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@f9bd8cf17bf5:~$ 9 packets captured
10 packets received by filter
0 packets dropped by kernel
```

Now, to verify that the data was captured and written onto the *.pcap* file, the command "ls -l capture.pcap" is used.

```
ls -l capture.pcap
-rw-r--r-- 1 root root 1401 Oct 17 15:40 capture.pcap
[1]+  Done                    sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap
```

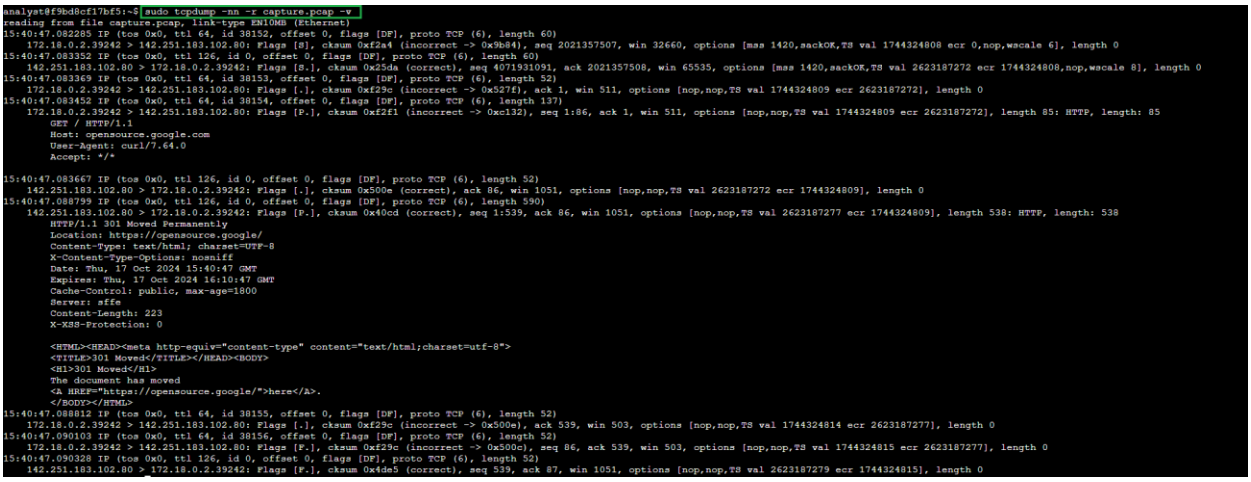
Filtering the captured packet data

To filter the data that was collected, there are several commands that could be used.

For example, “sudo tcpdump -nn -r capture.pcap -v”. This command will run tcpdump with the following options:

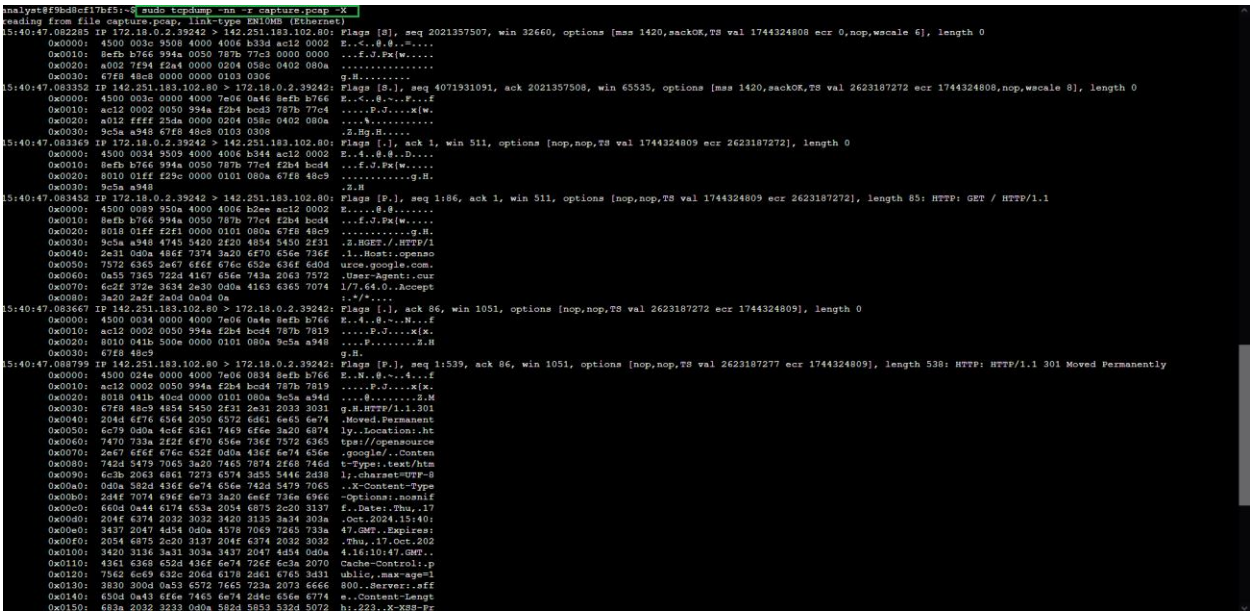
- -nn: Disable port and protocol name lookup.
- -r: Read capture data from the named file.
- -v: Display detailed packet data.

The -nn switch is specified again here, as one would want to make sure tcpdump does not perform name lookups of either IP addresses or ports, since this can alert threat actors.



As before, we can see IP packet information along with information about the data that the packet contains.

To filter extended packet data prom the *.pcap* file, the command “sudo tcpdump -nn -r capture.pcap -X” can be used.



The difference here is the *-X* option which displays the hexadecimal and ASCII output format packet data. Such output could be analyzed to detect patterns or anomalies during malware analysis or forensic analysis.