LOGIX ACADEMY

Lab: tcpdump

Purpose

In this lab, we are going to demonstrate how **tcpdump**, a very popular packet sniffer can be used to monitor different types of traffic on network interfaces.

Pre-Requisite

Before you can start the lab, you need to run the lab script which will setup everything. Open the **Labs** folder on Desktop then right-click and "Open Terminal Here". Or open a terminal and cd to Desktop/Labs folder, then issue the command:

```
sudo ./main_script.sh
```

Select **tcpdump Lab** option from the lab menu.

Listening To All Traffic

```
sudo tcpdump -i docker0 -v
```

(`-i` – interface on which tcpdump should listen)
(`-v` – verbose mode, shows more details of packet headers)

This command will display all traffic on docker0 interface. Now open a new tab for the terminal and do the following to trigger some traffic then check the tcpdump logs:

```
ping 172.17.0.2
```

```
nmap -p 22 172.17.0.2
```

```
┌──(kali㉿vbox)-[~/Desktop/Labs]
└─$ sudo tcpdump -i docker0 -v
tcpdump: listening on docker0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
17:28:44.611981 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.17.0.2 tell 172.17.0.1, length 28
17:28:44.612050 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.17.0.2 is-at 02:42:ac:11:00:02 (oui Unknown), length 2
8
17:28:44.612050 IP (tos 0x0, ttl 64, id 38457, offset 0, flags [DF], proto ICMP (1), length 84)
    172.17.0.1 > 172.17.0.2: ICMP echo request, id 2, seq 1, length 64
17:28:44.612254 IP (tos 0x0, ttl 64, id 46009, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.2 > 172.17.0.1: ICMP echo reply, id 2, seq 1, length 64
17:28:45.612514 IP (tos 0x0, ttl 64, id 38529, offset 0, flags [DF], proto ICMP (1), length 84)
    172.17.0.1 > 172.17.0.2: ICMP echo request, id 2, seq 2, length 64
17:28:45.612537 IP (tos 0x0, ttl 64, id 46067, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.2 > 172.17.0.1: ICMP echo reply, id 2, seq 2, length 64
17:28:46.627283 IP (tos 0x0, ttl 64, id 38608, offset 0, flags [DF], proto ICMP (1), length 84)
    172.17.0.1 > 172.17.0.2: ICMP echo request, id 2, seq 3, length 64
17:28:46.627303 IP (tos 0x0, ttl 64, id 46253, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.2 > 172.17.0.1: ICMP echo reply, id 2, seq 3, length 64
17:28:49.858917 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.17.0.1 tell 172.17.0.2, length 28
17:28:49.858927 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.17.0.1 is-at 02:42:b7:99:20:37 (oui Unknown), length 2
8
17:28:54.466693 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.17.0.2 tell 172.17.0.1, length 28
17:28:54.466708 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.17.0.2 is-at 02:42:ac:11:00:02 (oui Unknown), length 2
8
17:28:54.534685 IP (tos 0x0, ttl 50, id 44545, offset 0, flags [none], proto TCP (6), length 44)
    172.17.0.1.41036 > 172.17.0.2.ssh: Flags [S], cksum 0x9f05 (correct), seq 1843760819, win 1024, options [mss 1460
], length 0
```

LOGIX ACADEMY

Filter on Destination Port

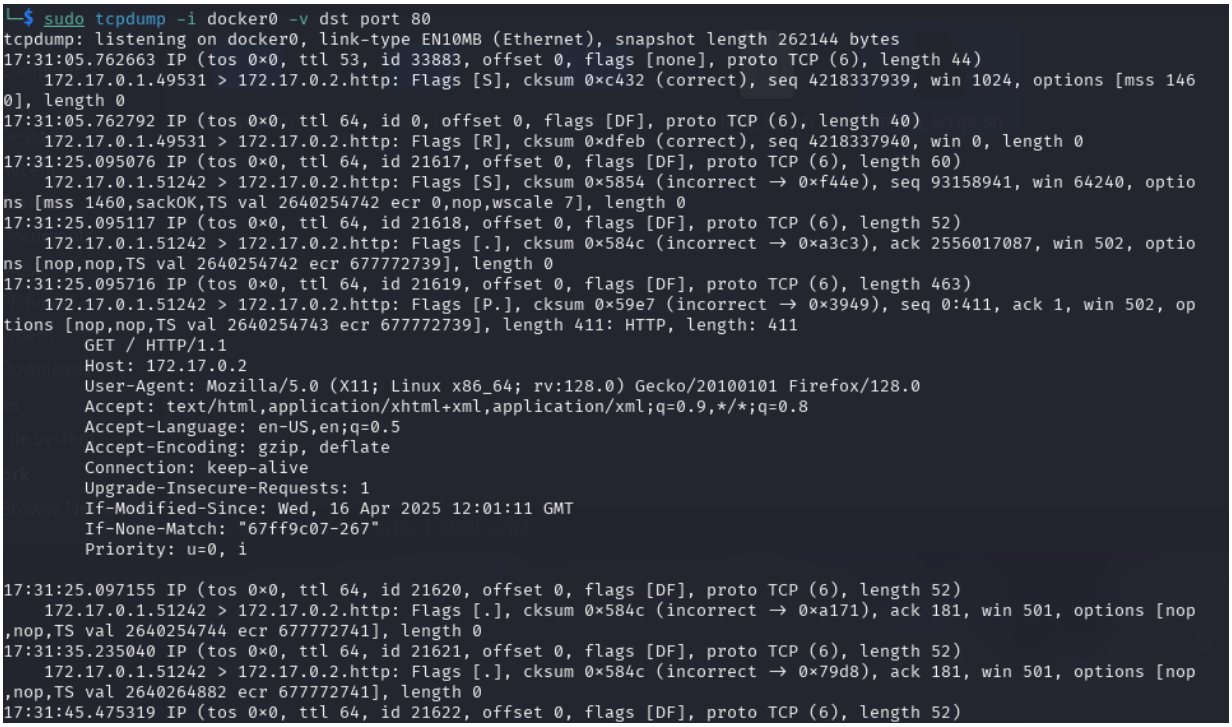
```
sudo tcpdump -i docker0 -v dst port 80
```

This command will display all traffic on docker0 interface that is going to destination port 80. Now do the following to trigger some traffic then check the tcpdump logs:

`nmap -p 22 172.17.0.2 (should not see any capture)`

`nmap -p 80 172.17.0.2 (should see capture)`

`Open the browser and type http://172.17.0.2 (should see capture)`



```
└─$ sudo tcpdump -i docker0 -v dst port 80
tcpdump: listening on docker0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
17:31:05.762663 IP (tos 0×0, ttl 53, id 33883, offset 0, flags [none], proto TCP (6), length 44)
    172.17.0.1.49531 > 172.17.0.2.http: Flags [S], cksum 0×c432 (correct), seq 4218337939, win 1024, options [mss 146
0], length 0
17:31:05.762792 IP (tos 0×0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    172.17.0.1.49531 > 172.17.0.2.http: Flags [R], cksum 0×dfeb (correct), seq 4218337940, win 0, length 0
17:31:25.095076 IP (tos 0×0, ttl 64, id 21617, offset 0, flags [DF], proto TCP (6), length 60)
    172.17.0.1.51242 > 172.17.0.2.http: Flags [S], cksum 0×5854 (incorrect → 0×f44e), seq 93158941, win 64240, optio
ns [mss 1460,sackOK,TS val 2640254742 ecr 0,nop,wscale 7], length 0
17:31:25.095117 IP (tos 0×0, ttl 64, id 21618, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.1.51242 > 172.17.0.2.http: Flags [.], cksum 0×584c (incorrect → 0×a3c3), ack 2556017087, win 502, optio
ns [nop,nop,TS val 2640254742 ecr 677772739], length 0
17:31:25.095716 IP (tos 0×0, ttl 64, id 21619, offset 0, flags [DF], proto TCP (6), length 463)
    172.17.0.1.51242 > 172.17.0.2.http: Flags [P.], cksum 0×59e7 (incorrect → 0×3949), seq 0:411, ack 1, win 502, op
tions [nop,nop,TS val 2640254743 ecr 677772739], length 411: HTTP, length: 411
        GET / HTTP/1.1
        Host: 172.17.0.2
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        If-Modified-Since: Wed, 16 Apr 2025 12:01:11 GMT
        If-None-Match: "67ff9c07-267"
        Priority: u=0, i

17:31:25.097155 IP (tos 0×0, ttl 64, id 21620, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.1.51242 > 172.17.0.2.http: Flags [.], cksum 0×584c (incorrect → 0×a171), ack 181, win 501, options [nop
,nop,TS val 2640254744 ecr 677772741], length 0
17:31:35.235040 IP (tos 0×0, ttl 64, id 21621, offset 0, flags [DF], proto TCP (6), length 52)
    172.17.0.1.51242 > 172.17.0.2.http: Flags [.], cksum 0×584c (incorrect → 0×79d8), ack 181, win 501, options [nop
,nop,TS val 2640264882 ecr 677772741], length 0
17:31:45.475319 IP (tos 0×0, ttl 64, id 21622, offset 0, flags [DF], proto TCP (6), length 52)
```

Filter on Host (dst or source)

```
sudo tcpdump -i docker0 -v host 172.17.0.2
```

This command will display traffic on docker0 interface that is coming from or going to ip address 172.17.0.2. Now do the following to trigger some traffic then check the tcpdump logs:

`Open the browser and type http://172.17.0.2 (should see capture both directions)`

Filter on Host (dst)

```
sudo tcpdump -i docker0 -v dst host 172.17.0.2
```

This command will display traffic on docker0 interface that is going to ip address 172.17.0.2. Now do the following to trigger some traffic then check the tcpdump logs:

`nmap -p 22 172.17.0.2 (should see traffic going to nginx container)`

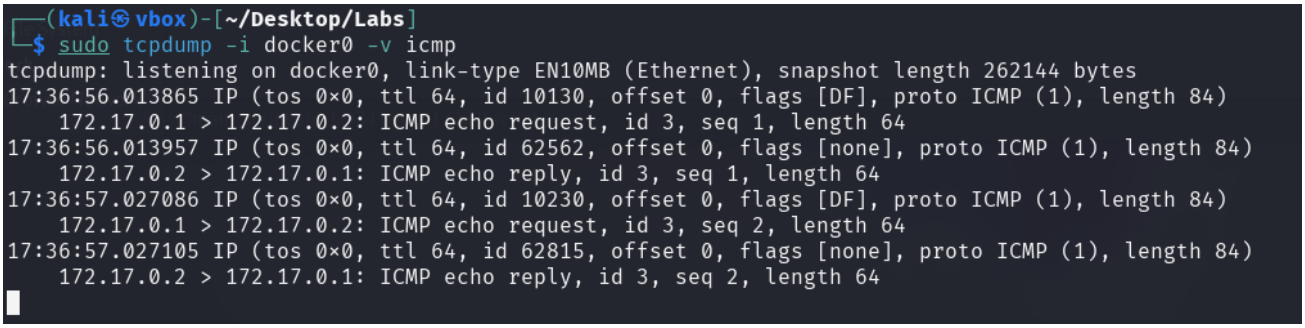
LOGIX ACADEMY

Filter on Protocol (icmp)

```
sudo tcpdump -i docker0 -v icmp
```

This command will display traffic on docker0 interface that uses the icmp protocol regardless of the direction. Now do the following to trigger some traffic then check the tcpdump logs:

```
ping 172.17.0.2 (should see traffic in both directions)
```

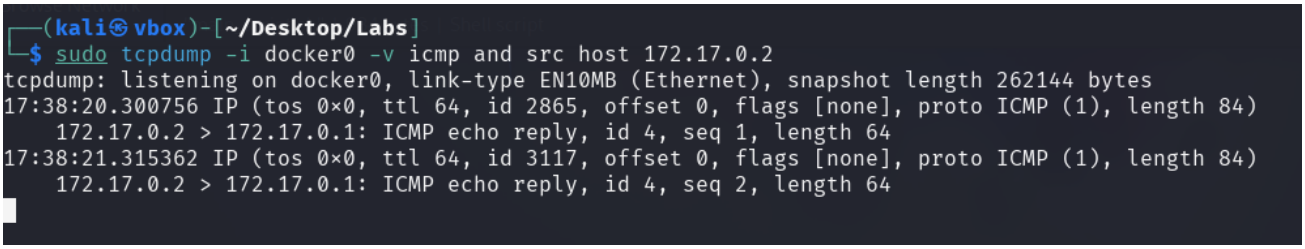


Filter on Protocol and source (icmp Reply messages)

```
sudo tcpdump -i docker0 -v icmp and src host 172.17.0.2
```

This command will display traffic on docker0 interface that uses the icmp protocol. However, since we have specified the nginx container ip (172.17.0.2) as the source, therefore we should only see ping reply traffic going from nginx container (172.17.0.2) to kali (172.17.0.1) Now do the following to trigger some traffic then check the tcpdump logs:

```
ping 172.17.0.2 (should see only ping reply traffic)
```



LOGIX ACADEMY

Reading pcap (packet capture files)

An interesting capability of tcpdump is that it can read and process packet capture files. We have included a sample packet capture file with the lab. The file is located at tcpdump/capture.pcap.

This capture file includes packet captures that we have recorded between kali Linux and nginx container. Try the same command for filtering on protocol and source but use the pcap file:

```
sudo tcpdump -r tcpdump/capture.pcap -v icmp and src host 172.17.0.2
```

Note that you do not need to trigger any traffic as the file already contains the traffic traces. You should see ping replies from nginx container.

```
┌──(kali㉿vbox)-[~/Desktop/Labs]
└─$ sudo tcpdump -r tcpdump/capture.pcap icmp and src host 172.17.0.2
reading from file tcpdump/capture.pcap, link-type IPV4 (Raw IPv4), snapshot length 65535
09:42:56.452251 IP 172.17.0.2 > 172.17.0.1: ICMP echo reply, id 1234, seq 0, length 24
09:42:57.452251 IP 172.17.0.2 > 172.17.0.1: ICMP echo reply, id 1235, seq 0, length 19
```

Filter on Protocol and Destination Port (DNS messages from Kali to nginx)

```
sudo tcpdump -r tcpdump/capture.pcap -v udp and dst port 53 and src host
172.17.0.1 and dst host 172.17.0.2
```

```
└─$ sudo tcpdump -r tcpdump/capture.pcap -v udp and dst port 53
reading from file tcpdump/capture.pcap, link-type IPV4 (Raw IPv4), snapshot length 65535
09:42:58.352251 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 57)
    172.17.0.1.54321 > 172.17.0.2.domain: 4919+ A? example.com. (29)
09:42:59.352251 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 56)
    172.17.0.1.54322 > 172.17.0.2.domain: 4920+ A? google.com. (28)
```

Challenge

Note: you need to run these against capture.pcap file (already included in the lab) not on docker0 interface.

1. Run the command to show only **DNS responses** from Nginx to Kali Linux (DNS responses also originate from port 53 on nginx)
2. Your team has realized that there has been a data exfiltration attempt from the nginx container to a remote malicious command and control station. Your manager has shared the packet captures and they want you to help them identify the name of the remote malicious server and the ip. You know that a DNS query must have been sent from nginx container to Google's dns servers to resolve it. Please run the command that parses the capture.pcap file and identifies:
    - DNS query (udp) originating from nginx, going to destination port 53 then find out the name of the remote malicious server
    - DNS response (udp) from Google (we don't know the ip of Google DNS server, so ignore that ip), coming from source port 53 and going to nginx container. Then find out the ip address of the remote malicious host

(Solution in next lecture)