

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:

```
analyst@8ec2dba07be8:~$ 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 692  bytes 13720314 (13.0 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 374  bytes 34730 (33.9 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 89  bytes 11581 (11.3 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 89  bytes 11581 (11.3 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

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 eth0 -v -c5
tcpdump: eth0: 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.], cksum
    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:

```
analyst@8ec2dba07be8:~$ sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap &
[1] 12945
analyst@8ec2dba07be8:~$ tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
analyst@8ec2dba07be8:~$ █
```

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 -> 0xd4c), 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 -> 0xbfe), 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 0xdff5 (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 -> 0xaa4), 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 -> 0xaa2), 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:

```
analyst@C93f6ee8d3cbr:~$ sudo tcpdump -nn -r capture.pcap -X
reading from file capture.pcap, link-type EN10MB (Ethernet)
16:07:08.666963 IP 172.17.0.2.53360 > 173.194.213.138.80: Flags [S], seq 3701989374, win 65320, options [
  msg 1420, sackOK, TS val 90762411, ecr 0, nop, wscale 7], length 0
  0x0000: 4500 003c 825e 4000 4006 88ff ac11 0002  E...@.8.....
  0x0010: adc2 d58a d070 0050 dca7 dffe 0000 0000  ....P.P.....
  0x0020: a002 ff28 2f8f 0000 0204 058c 0402 080a  .../.....
  0x0030: 1d40 70ab 0000 0000 0103 0307  .@p.....
16:07:08.667720 IP 173.194.213.138.80 > 172.17.0.2.53360: Flags [S.], seq 540800173, ack 3701989375, win
  65535, options [msg 1420, sackOK, TS val 961793664, ecr 490762411, nop, wscale 8], length 0
  0x0000: 4500 003c 0000 4000 7e06 cd5b adc2 d58a  E...<..8-..[....
  0x0010: ac11 0002 0050 d070 203b f4ad dca7 dfff  ....P.p;.....
  0x0020: a012 ffff e0a5 0000 0204 058c 0402 080a  ....@p.....
  0x0030: 3953 ce80 1d40 70ab 0103 0308  9S...@p.....
16:07:08.667751 IP 172.17.0.2.53360 > 173.194.213.138.80: Flags [.], ack 1, win 511, options [nop,nop,TS
  val 490762411, ecr 961793664], length 0
  0x0000: 4500 0034 825d 4000 4006 8906 ac11 0002  E..4.]@.8.....
  0x0010: adc2 d58a d070 0050 dca7 dfff 203b f4ae  ....P.P.....;..
  0x0020: 8010 01ff 2f87 0000 0101 080a 1d40 70ab  ....//.....@p.
  0x0030: 3953 ce80 9S...
16:07:08.667822 IP 172.17.0.2.53360 > 173.194.213.138.80: Flags [P.], seq 1:86, ack 1, win 511, options [
  nop,nop,TS val 490762412, ecr 961793664], length 85: HTTP: GET / HTTP/1.1
  0x0000: 4500 0089 825e 4000 4006 88b0 ac11 0002  E...."@.8.....
  0x0010: adc2 d58a d070 0050 dca7 dfff 203b f4ae  ....P.P.....;..
  0x0020: 8018 01ff 2fde 0000 0101 080a 1d40 70ac  ....//.....@p.
  0x0030: 3953 ce80 4745 5420 2f20 4854 5450 2f31  9S..GET../HTTP/1
  0x0040: 2e31 0d0a 486f 7374 3a20 6f70 656e 736f  .1..Host:.openso
  0x0050: 7572 6365 2a67 6f6f 676c 652e 636f 6d0d  urce.google.com.
  0x0060: 0a55 7365 722d 4167 656e 743a 2063 7572  .User-Agent: cur
  0x0070: 6c2f 372e 3634 2e30 0d0a 4163 6365 7074  /7.64.0..Accept
  0x0080: 3a20 2a2f 2a0d 0a0d 0a  ./*/*....
16:07:08.667995 IP 173.194.213.138.80 > 172.17.0.2.53360: Flags [.], ack 86, win 256, options [nop,nop,TS
  val 961793664, ecr 490762412], length 0
  0x0000: 4500 0034 0000 4000 7e06 cd63 adc2 d58a  E..4..8-..c....
  0x0010: ac11 0002 0050 d070 203b f4ae dca7 e054  ....P.p;.....T
  0x0020: 8010 0100 0df5 0000 0101 080a 3953 ce80  ....//.....9S..
  0x0030: 1d40 70ac  .@p.
16:07:08.670260 IP 173.194.213.138.80 > 172.17.0.2.53360: Flags [P.], seq 1:594, ack 86, win 256, options
  [nop,nop,TS val 961793667, ecr 490762412], length 593: HTTP: HTTP/1.1 301 Moved Permanently
  0x0000: 4500 0285 0000 4000 7e06 cb12 adc2 d58a  E....8-.....
  0x0010: ac11 0002 0050 d070 203b f4ae dca7 e054  ....P.p;.....T
  0x0020: 8018 0100 1916 0000 0101 080a 3953 ce83  ....//.....9S..
  0x0030: 1d40 70ac 4854 5450 2e31 2e31 2033 3031  .@p.HTTP/1.1 301
  0x0040: 204d 6f76 6564 2050 6572 6d61 6e65 6e74  .Moved,Permanent
  0x0050: 6c79 0d0a 4c6f 6361 7469 6f6e 3a20 6874  y..Location:ht
  0x0060: 7470 733a 2f2f 6f70 656e 736f 7572 6365  tps://opensource
  0x0070: 2e67 6f6f 676c 652f 0d0a 4372 6f73 732d  .google/..Cross-
  0x0080: 4e72 696f 696e 2d52 6573 6f75 7263 652d  Origin-Resource-
  0x0090: 506f 6c69 6379 3a20 6372 6f73 732d 6f72  Policy..cross-or
  0x00a0: 6967 696e 0d0a 582d 436f 6e74 656e 742d  igin..X-Content-
  0x00b0: 5479 7065 2d4f 7074 696f 6e73 3a20 6e6f  Type-Options:.no
  0x00c0: 736e 6966 6e0d 0a53 6572 7665 723a 2073  sniff..Server:s
  0x00d0: 6666 650d 0a43 6f6e 7465 6e74 2d4c 656e  file..Content-Len
  0x00e0: 6274 681a 2032 3233 0d0a 582d 5853 532d  shh: 221 y-YSS..
```


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.