Tcpdump Essentials 101

Objective

Tcpdump is an uber network analysis and packet dump tool, first released in 1988. Like tshark, tcpdump is a command-line equivalent of Wireshark while having advantages over Wireshark in some areas.

During pen-testing, you often get restricted shells or attack system, where installing a new tool or having a GUI access is prohibited. or if you want to write an automated packet sniffer, tcpdump comes handy in such situations.

This lab requires you to perform below listed operations on the pcap files located in /root/Desktop/pcap-analysis

1. Identify first appearance of a MAC address
2. Filter DNS traffic from a local pcap file
3. Extract packet's sequence number
4. Identify User-Agent of the target host
5. Discover corresponding port for the target host
6. Perform Logical 'AND/ OR' operations to filter target host traffic

Useful link:
https://danielmiessler.com/study/tcpdump/

TABLE OF CONTENTS

Flag 1

Q. When was the MAC (`ec:1a:59:43:3f:fd`) first appeared in the *`iphone6s-iOS12_1.pcapng`* file?

```
tcpdump -e -n -r iphone6s-iOS12_1.pcapng| grep -i "ec:1a:59:43:3f:fd"
```

`-e`	Prints Ethernet (MAC) address, or the link-level header on each dump
`-n`	Do not resolve IP addresses, port numbers to domain
`-r` `<src>`	Reads packet capture file

TIP: You can also combine the cli flags for better representation. For example, the same command will give same results if written as:

```
tcpdump -enr iphone6s-iOS12_1.pcapng | grep -i "ec:1a:59:43:3f:fd"
```

Simply running ***`tcpdump -enr iphone6s-iOS12_1.pcapng`*** will give you a plethora of packet capture information like this:

```
18:31:57.447901 9c:f4:8e:9d:44:de > 00:c0:ca:5a:34:b6, ethertype ARP (0x0806)
18:31:57.447932 00:c0:ca:5a:34:b6 > 9c:f4:8e:9d:44:de, ethertype ARP (0x0806)
18:31:57.449529 9c:f4:8e:9d:44:de > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800)
18:31:57.449556 9c:f4:8e:9d:44:de > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800)
18:31:57.450864 00:c0:ca:5a:34:b6 > 9c:f4:8e:9d:44:de, ethertype IPv4 (0x0800)
18:31:57.459414 9c:f4:8e:9d:44:de > 33:33:ff:83:16:29, ethertype IPv6 (0x86dd)
18:31:57.459496 9c:f4:8e:9d:44:de > 33:33:ff:83:16:29, ethertype IPv6 (0x86dd)
18:31:57.461948 9c:f4:8e:9d:44:de > 33:33:00:00:00:02, ethertype IPv6 (0x86dd)
18:31:57.461954 9c:f4:8e:9d:44:de > 33:33:00:00:00:02, ethertype IPv6 (0x86dd)
18:31:57.464283 9c:f4:8e:9d:44:de > 00:c0:ca:5a:34:b6, ethertype ARP (0x0806)
18:31:57.464373 00:c0:ca:5a:34:b6 > 9c:f4:8e:9d:44:de, ethertype ARP (0x0806)
```

We then pass this output via Bash pipe to grep to filter results for `ec:1a:59:43:3f:fd` MAC address. Which gives us the following result.

Here you can see the timestamp of first appearance of **`ec:1a:59:43:3f:fd`**.

```
18:31:57.138584 9c:f4:8e:9d:44:de > ec:1a:59:43:3f:fd, ethertype IPv4 (0x0800), length
105: 192.168.2.80.52064 > 17.142.163.21.993: Flags [FP.], seq 3508880369:3508880408,
ack 3901183005, win 2048, options [nop,nop,TS val 376973149 ecr 4026141990], length 39
```

Flag 2

Q. Save the DNS traffic from file android-oneplus5.pcapng as DNS.pcap and calculate the md5hash of the resulting file.

Tell tcpdump to show packet transferred via **`port 53`** (dns), reading (**`-r`**) the input from packet capture file: **`android-oneplus5.pcapng`** and write (**`-w`**) it to **`dns.pcap`** in current working directory.

```
tcpdump port 53 -r android-oneplus5.pcapng -w dns.pcap
```

Upon successful write, we calculate the hash of the resulting dns.pcap file

```
md5sum dns.pcap
42c93d67933f9a1e6fc353ad3dc1b57c  dns.pcap
```

Flag 3

Q. What is the sequence number for the fifth packet found in the kerberos connection *kpasswd_tcp.pcap*

```
tcpdump -nc 5 -r kpasswd_tcp.pcap
```

-n	Do not resolve IP addresses, port numbers to domain
-c <count>	Show first '*n*' number of packets. Is a whole number
-r <src>	Reads packet capture file

```
02:04:27.018000 IP 192.168.43.232.1047 > 10.2.20.154.464: Flags [S], seq 1308371789,
win 64240, options [mss 1460], length 0

02:04:27.018000 IP 10.2.20.154.464 > 192.168.43.232.1047: Flags [S.], seq 1782935237,
ack 1308371790, win 16384, options [mss 1460], length 0

02:04:27.018000 IP 192.168.43.232.1047 > 10.2.20.154.464: Flags [.], ack 1, win 64240,
length 0

02:04:27.019000 IP 192.168.43.232.1047 > 10.2.20.154.464: Flags [P.], seq 1:1461, ack
1, win 64240, length 1460

02:04:27.020000 IP 192.168.43.232.1047 > 10.2.20.154.464: Flags [P.], seq 1461:2921,
ack 1, win 64240, length 1460
```

If you look at the last (fifth) line of the output, you'll notice the sequence number in the response. Enter the number **1461:2921** in the corresponding flag value.

Flag 4

Q. Find the **User-Agent** of the iOS device trying to connect to the Captive Network portal.

```
tcpdump -An -r iphone7-iOS12_1_2.pcapng | grep 'User-Agent'
```

-A	Print each packet (minus its link level header) in ASCII. Handy for capturing web pages.
-n	Do not resolve IP addresses, port numbers to domain
-r <src>	Read packet capture file from <src>

Grep searches an exact match of string '**User-Agent**'. You can also ignore the case by using **-i** flag with grep (***grep -i "user-agent"***).

```
User-Agent: CaptiveNetworkSupport-355.200.27 wispr
User-Agent: CaptiveNetworkSupport-355.200.27 wispr
User-Agent: CaptiveNetworkSupport-355.200.27 wispr
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_1_2 like Mac OS X) AppleWebKit/
605.1.15 (KHTML, like Gecko) Mobile/16D5024a
User-Agent: CaptiveNetworkSupport-355.200.27 wispr
User-Agent: CaptiveNetworkSupport-355.200.27 wispr
```

Flag 5

Q. In windows-10.pcapng, Which port number does IP **`10.0.0.203`** made **25** requests from?

First, we print http traffic from windows-10.pcapng with host name resolution disabled, for faster results.

```
tcpdump port http -nr windows-10.pcapng
```

Which results in a response like this:

```
11:23:23.078991 IP 10.0.0.203.53931 > 13.107.4.52.80: Flags [S], seq 2432615504, win 17520,
options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0

11:23:23.079155 IP 13.107.4.52.80 > 10.0.0.203.53931: Flags [S.], seq 3139394652, ack 2432615505,
win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0

11:23:23.087866 IP 10.0.0.203.53931 > 13.107.4.52.80: Flags [.], ack 1, win 68, length 0

11:23:23.092539 IP 10.0.0.203.53931 > 13.107.4.52.80: Flags [P.], seq 1:112, ack 1, win 68,
length 111: HTTP: GET /connecttest.txt HTTP/1.1

11:23:23.092613 IP 13.107.4.52.80 > 10.0.0.203.53931: Flags [.], ack 112, win 229, length 0

11:23:23.107323 IP 13.107.4.52.80 > 10.0.0.203.53931: Flags [P.], seq 1:494, ack 112, win 229,
length 493: HTTP: HTTP/1.1 302 Found

11:23:23.107430 IP 13.107.4.52.80 > 10.0.0.203.53931: Flags [F.], seq 494, ack 112, win 229,
length 0
```

On a closer look, we discover that all the IP addresses have the third position in the space-delimited output.

So, we filter that using ***`awk`***. Then ***`sort`*** the output by unique values with a count (***`uniq -c`***) on each of them.

```
tcpdump port http -nr windows-10.pcapng | awk '{print $3}' | sort | uniq -c
```

```
     27 10.0.0.1.80
      5 10.0.0.203.53931
      5 10.0.0.203.53932
      4 10.0.0.203.53934
      4 10.0.0.203.53935
      6 10.0.0.203.53937
      6 10.0.0.203.53939
     25 10.0.0.203.53940
      6 10.0.0.203.53945
     18 13.107.4.52.80
      4 172.217.160.227.80
```

Flag 6

Q. Identify the IPv4 address from **`windows-10.pcapng`** that made highest number of http or mdns type requests and is neither `10.0.0.1` nor `10.0.0.203`.

If you've completed the Text Searching And Manipulation lab, you'd know how to parse IP address pattern using Regular Expression with grep and sort the output by unique values to filter garbage repetitive values.

You can read the detailed working of the ***`grep`*** | ***`sort`*** | ***`uniq -c`*** command in the solution of Text Searching And Manipulation lab. Just go to the Solutions tab and open the solution to read it.

We use logical AND and logical OR in this flag.

First we tell tcpdump to read from the source *pcapng* file, followed by the port http. Then we put a choice, it can be either port http or port DNS. This way we get packets for both `HTTP` and `MDNS` in our output.

Since the question tells us that it is neither `10.0.0.203`, nor `10.0.0.1` (router), we add this condition to our logic and tell *tcpdump* to show packets which are either `HTTP` or `MDNS`, **AND** the source (`src`) is not `10.0.0.203` **AND** src is also not `10.0.0.1`

We then pass the output to awk to filter IP addresses for us, sorts the output and filters only unique values with a count on total no. of times they were discovered during processing.

```
tcpdump -nr windows-10.pcapng port http or port mdns and src not 10.0.0.203 and src not 10.0.0.1 | awk '{print $3}' | grep  -E -o "([0-9]{1,3}[\.]){3}[0-9]{1,3}" | sort | uniq -c | sort -r
```

```
reading from file windows-10.pcapng, link-type EN10MB (Ethernet)
     18 13.107.4.52
      4 172.217.160.227
```