

TRAFFIC CAPTURE ANALYSIS WITH WIRESHARK AND TCPDUMP

OBJECTIVE: To learn how to utilize various packet capture and protocol analyzer tools, such as Wireshark and Tcpdump, and help identify risk, vulnerabilities, and threats associated with traffic.

Project Description: This project contains two traffic captures, one captured with Wireshark and the other with tcpdump. These captures were saved as packet capture (pcap) files and later used for protocol analysis using Wireshark filters and tcpdump capture filters, as further documented in this report.

Tools Used:

- Wireshark
- Tcpdump
- Snipping Tool
- Kali VM

Tools Used	Description
Wireshark	This is a cross-platform network protocol analyzer tool that provides a user-friendly Graphical User Interface(GUI) for users to analyze and filter traffic running on their networks, and it is perfect for anomaly detection, troubleshooting network issues, protocol analysis, and security analysis, such as spotting misconfigurations, identifying malware
Tcpdump	This is a Command Line Interface (CLI) Network packet capture and protocol analyzer that is capable of capturing live traffic and widely used for network troubleshooting, security monitoring, and is particularly suited to Unix-like operating Systems

Snipping Tool	This was used to capture various vital screenshots shown in this report.
Kali VM	This is the Virtual Machine OS used to practice tcpdump commands and capture its traffic.

REPORT

WIRESHARK

Wireshark is a cross-platform GUI protocol analyzer that provides various filters for analyzing protocols and traffic, and also supports colorization to identify packet types. And by default, they are:

Color in Wireshark	Packet Types
Light Purple	TCP
Light Blue	UDP
Black	Packet with errors
Light Green	HTTP traffic
Light Yellow	Window-Specific traffic, including server message block (SMB) and NetBios
Dark Yellow	Routing
Dark Gray	TCP, SYN, FIN, and ACK traffic

The Wireshark dashboard, displayed in the snippet, presents the various interfaces available for traffic capture. Interfaces currently experiencing traffic flow are indicated by zigzag lines. To begin the capture process, click the icon resembling the main Wireshark icon. For optimal results, it is crucial to set the network interface to **promiscuous mode**. To do this, click "**Capture**" on the taskbar, select "**Options**," and then ensure that "**Enable Promiscuous mode on all interfaces**" is checked, as if not selected the network card will only capture traffic sent to its own network addresses.

The screenshot shows two windows of the Wireshark Network Analyzer. The top window is the main interface with a toolbar at the top, followed by a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help), a search bar, and a status bar indicating 'Apply a display filter... <Ctrl-/>' and a connection icon. Below the toolbar is a 'Welcome to Wireshark' message and a 'Capture' section. It includes a 'Capture' button, a 'using this filter:' dropdown with a placeholder 'Enter a capture filter ...', and a 'All interfaces shown' dropdown. A list of available interfaces is displayed, with 'Local Area Connection* 12' selected. Other listed interfaces include Wi-Fi 2, VMware Network Adapter VMnet8, VMware Network Adapter VMnet1, VirtualBox Host-Only Network, Adapter for loopback traffic capture, Local Area Connection* 11, Local Area Connection* 10, Bluetooth Network Connection 2, Local Area Connection* 4, Local Area Connection* 3, Ethernet 2, and Event Tracing for Windows (ETW) reader: etwdump. To the right of the interface list is a packet list pane showing several captured frames. The bottom window is the 'Capture Options' dialog box, which has tabs for Input, Output, and Options. The Input tab is active, showing a table of capture settings for various interfaces. The table columns are: Interface, Traffic, Link-layer Header, Promisc, Snaplen (B), Buffer (MB), Monitor, and Capture Filter. Most interfaces have 'Ethernet' as the link-layer header and 'Promisc' checked. The 'Event Tracing for Windows (ETW) reader: etwdump' interface has 'BSD loopback' as the link-layer header and 'DLT_ETW' as the capture filter. At the bottom of the dialog are checkboxes for 'Enable promiscuous mode on all interfaces' (checked) and 'Enable monitor mode on all 802.11 interfaces' (unchecked), along with 'Manage Interfaces...', 'Capture filter for selected interfaces' (with a placeholder 'Enter a capture filter ...'), 'Compile BPFs', and buttons for 'Start', 'Close', and 'Help'.

The project utilizes a pre-existing PCAP file for analysis. This file, which contains **1,195,997** captured packets of various traffic types, was collected over a three-hour duration during a live YouTube program.

November_first_capture.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
2	0.000182	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
3	0.000251	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
4	0.000321	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
5	0.000392	192.168.137.237	150.171.22.12	UDP	440	60095 → 443 Len=398
6	0.000500	192.168.137.237	150.171.22.12	UDP	414	60095 → 443 Len=372
7	0.006974	150.171.22.12	192.168.137.237	UDP	75	443 → 60095 Len=33
8	0.008298	150.171.22.12	192.168.137.237	UDP	80	443 → 60095 Len=38
9	0.008298	150.171.22.12	192.168.137.237	UDP	80	443 → 60095 Len=38
10	0.008298	150.171.22.12	192.168.137.237	UDP	75	443 → 60095 Len=33
11	0.008597	192.168.137.237	150.171.22.12	UDP	88	60095 → 443 Len=38
12	0.208143	150.171.22.12	192.168.137.237	UDP	1514	443 → 60095 Len=1472
13	0.208143	150.171.22.12	192.168.137.237	UDP	1123	443 → 60095 Len=1081
14	0.215313	192.168.137.237	150.171.22.12	UDP	85	60095 → 443 Len=43
15	0.230570	150.171.22.12	192.168.137.237	UDP	72	443 → 60095 Len=30
16	0.239931	192.168.137.237	150.171.22.12	UDP	81	60095 → 443 Len=39

```
> Frame 1: Packet, 1292 bytes on wire (1033 bits)
> Ethernet II, Src: Intel_14:4e:43 (7:...), Dst: (08:00:27:00:00:00)
> Internet Protocol Version 4, Src: 192.168.137.237 (192.168.137.237), Dst: 150.171.22.12 (150.171.22.12)
> User Datagram Protocol, Src Port: 6 (6), Dst Port: 443 (443)
> Data (1250 bytes)

0000  fe 5c 68 03 e9 bd 7c 2a 31 14 4e 43 08 45 00  \h...[*] 1 NC-E-
0010  04 fe b2 ff 40 00 80 11 00 00 c0 a8 89 ed 96 ab  ..@.....
0020  16 0c ea b7 01 bb 04 ea fc 44 d9 c8 84 37 2e  .....HD-7.
0030  18 75 d9 1d 8f 36 4c 24 da 67 2d 39 bd ba 27 37  ..u-6LS_g-9-'7
0040  41 dc ee a0 21 0d 8a 34 ac 9f 88 04 a2 z6 d6  A-!..4...'.
0050  3f 1c 97 b5 d2 50 c8 99 f1 fd 89 98 81 01 ?...P.....
0060  af 64 87 0b c4 5d 5c a8 2c 55 p3 43 27 c3 58 68  d-!` ,SC-Xh
0070  45 e0 07 28 02 29 da e2 da 6d 1d 05 05 98 66 E-(...)..m....f
0080  5e 05 6a 7e 90 e7 1c c8 19 97 96 0f 44 b3 95 cf ^j.....oD...
0090  0d df f7 1f 6a 66 21 74 90 e3 f5 c0 49 32 69 7e ...jflt...I2i~
00a0  6b f7 82 b3 8b 8d 1c f2 f3 af 79 dc ad 03 k.....y-M...
00b0  e9 86 46 de 68 be 84 27 c9 63 77 2f 7f 7e 65 d6 ..F-h-' cw-e...
00c0  1c 65 c9 19 eb 0e 99 a0 f4 d7 5c 2f 63 e2 b9 91 e-.....\c...
00d0  40 44 89 d4 03 da 67 ac 44 58 cf 28 71 38 55 eb @o...g DX-(qBU
00e0  99 0d 0e d9 02 e2 33 6b bc 76 f7 a3 43 9b f0 .....3k vx C...
00f0  56 5b 34 5e c9 ec ac 08 bb fa 86 af a0 19 8c 68 V|4^.....h...
0100  7a 68 a8 84 61 43 9b 82 ea d2 37 86 fb 6b 1c zh-al=...7-k...
0110  df ff dd 71 0f 61 f2 4b 52 bb 4b 3d 46 2e e2 3d ..q-a-K R-K=F,-
```

Packets: 1195997 | Profile: Default

Various Color codes are displayed through the following snippet, ranging from **Light Yellow**, showing the packet type of Window-Specific traffic.

21	2.000000	192.168.137.1	192.168.137.237	TCP	1292	MSEARCH [HTTP/1.1]
22	4.2681240	192.168.137.1	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
23	4.259452	3.33.252.61	192.168.137.237	TCP	138	80 → 62444 [PSH, ACK] Seq=1 Ack=1 Win=476 Len=84
24	4.259465	192.168.137.237	3.33.252.61	TCP	54	62444 → 80 [ACK] Seq=1 Ack=85 Win=257 Len=0

To **Light Blue**, which shows the packet type of UDP:

November_first_capture.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
2	0.000182	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
3	0.000251	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
4	0.000321	192.168.137.237	150.171.22.12	UDP	1292	60095 → 443 Len=1250
5	0.000392	192.168.137.237	150.171.22.12	UDP	440	60095 → 443 Len=398
6	0.000500	192.168.137.237	150.171.22.12	UDP	414	60095 → 443 Len=372
7	0.006974	150.171.22.12	192.168.137.237	UDP	75	443 → 60095 Len=33
8	0.008298	150.171.22.12	192.168.137.237	UDP	80	443 → 60095 Len=38
9	0.008298	150.171.22.12	192.168.137.237	UDP	80	443 → 60095 Len=38
10	0.008298	150.171.22.12	192.168.137.237	UDP	75	443 → 60095 Len=33
11	0.008597	192.168.137.237	150.171.22.12	UDP	88	60095 → 443 Len=38
12	0.208143	150.171.22.12	192.168.137.237	UDP	1514	443 → 60095 Len=1472
13	0.208143	150.171.22.12	192.168.137.237	UDP	1123	443 → 60095 Len=1081
14	0.215313	192.168.137.237	150.171.22.12	UDP	85	60095 → 443 Len=43
15	0.230570	150.171.22.12	192.168.137.237	UDP	72	443 → 60095 Len=30
16	0.239931	192.168.137.237	150.171.22.12	UDP	81	60095 → 443 Len=39

```
> Frame 1: Packet, 1292 bytes on wire (1033 bits)
> Ethernet II, Src: Intel_14:4e:43 (7:...), Dst: (08:00:27:00:00:00)
> Internet Protocol Version 4, Src: 192.168.137.237 (192.168.137.237), Dst: 150.171.22.12 (150.171.22.12)
> User Datagram Protocol, Src Port: 6 (6), Dst Port: 443 (443)
> Data (1250 bytes)

0000  fe 5c 68 03 e9 bd 7c 2a 31 14 4e 43 08 45 00  \h...[*] 1 NC-E-
0010  04 fe b2 ff 40 00 80 11 00 00 c0 a8 89 ed 96 ab  ..@.....
0020  16 0c ea b7 01 bb 04 ea fc 44 d9 c8 84 37 2e  .....HD-7.
0030  18 75 d9 1d 8f 36 4c 24 da 67 2d 39 bd ba 27 37  ..u-6LS_g-9-'7
0040  41 dc ee a0 21 0d 8a 34 ac 9f 88 04 a2 z6 d6  A-!..4...'.
0050  3f 1c 97 b5 d2 50 c8 99 f1 fd 89 98 81 01 ?...P.....
0060  af 64 87 0b c4 5d 5c a8 2c 55 p3 43 27 c3 58 68  d-!` ,SC-Xh
0070  45 e0 07 28 02 29 da e2 da 6d 1d 05 05 98 66 E-(...)..m....f
0080  5e 05 6a 7e 90 e7 1c c8 19 97 96 0f 44 b3 95 cf ^j.....oD...
0090  0d df f7 1f 6a 66 21 74 90 e3 f5 c0 49 32 69 7e ...jflt...I2i~
00a0  6b f7 82 b3 8b 8d 1c f2 f3 af 79 dc ad 03 k.....y-M...
00b0  e9 86 46 de 68 be 84 27 c9 63 77 2f 7f 7e 65 d6 ..F-h-' cw-e...
00c0  1c 65 c9 19 eb 0e 99 a0 f4 d7 5c 2f 63 e2 b9 91 e-.....\c...
00d0  40 44 89 d4 03 da 67 ac 44 58 cf 28 71 38 55 eb @o...g DX-(qBU
00e0  99 0d 0e d9 02 e2 33 6b bc 76 f7 a3 43 9b f0 .....3k vx C...
00f0  56 5b 34 5e c9 ec ac 08 bb fa 86 af a0 19 8c 68 V|4^.....h...
0100  7a 68 a8 84 61 43 9b 82 ea d2 37 86 fb 6b 1c zh-al=...7-k...
0110  df ff dd 71 0f 61 f2 4b 52 bb 4b 3d 46 2e e2 3d ..q-a-K R-K=F,-
```

To **Light Green**, which shows the packet type of HTTP traffic

21	2.000000	192.168.137.237	192.168.137.237	TCP	1292	Initial, DCID=f27098045950910, PKN: 1, PING, PADDING, CRYPTO, PADDING, CRYPTO, PING, PADDIN
22	4.2681240	192.168.137.237	192.168.137.237	TCP	138	80 → 62444 [PSH, ACK] Seq=1 Ack=1 Win=476 Len=84
23	4.259452	3.33.252.61	192.168.137.237	TCP	54	62444 → 80 [ACK] Seq=1 Ack=85 Win=257 Len=0
24	4.259465	192.168.137.237	3.33.252.61	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
25	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
26	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
27	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
28	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
29	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
30	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
31	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
32	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
33	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
34	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
35	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
36	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
37	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
38	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
39	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
40	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
41	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
42	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
43	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
44	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
45	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
46	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
47	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
48	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
49	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
50	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
51	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
52	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
53	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
54	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
55	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
56	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=257 Len=79
57	4.259465	192.168.137.237	192.168.137.237	TCP	133	62444 → 80 [PSH, ACK] Seq=1 Ack=85 Win=

No.	Time	Source	Destination	Protocol	Length	Info
1983	22.431937	95.101.35.11	192.168.137.237	TLSv1.3	971	[TCP Window Full] , Continuation Data
1984	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1671597 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1985	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [PSH, ACK] Seq=1672997 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1986	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1674397 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1987	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1675797 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1988	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1677197 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1989	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1678597 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1990	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [PSH, ACK] Seq=1679997 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1991	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1681397 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1992	22.431937	95.101.35.11	192.168.137.237	TLSv1.3	1454	[TCP Out-Of-Order] , Application Data Application Data
1993	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1684197 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1994	22.431937	95.101.35.11	192.168.137.237	TCP	1454	[TCP Out-Of-Order] 443 -> 63358 [ACK] Seq=1685597 Ack=4071 Win=62720 Len=1400 [TCP PDU reassembly]
1995	22.432316	192.168.137.237	95.101.35.11	TCP	90	[TCP Dup ACK 1540#89] 63358 -> 443 [ACK] Seq=4071 Ack=1652351 Win=531456 Len=0 SLE=2131627 SRE=
1996	22.432316	192.168.137.237	95.101.35.11	TCP	90	[TCP Dup ACK 1540#90] 63358 -> 443 [ACK] Seq=4071 Ack=1652351 Win=531456 Len=0 SLE=2131627 SRE=
1997	22.432346	192.168.137.237	95.101.35.11	TCP	90	[TCP Dup ACK 1540#91] 63358 -> 443 [ACK] Seq=4071 Ack=1652351 Win=531456 Len=0 SLE=2166627 SRE=
1998	22.432375	192.168.137.237	95.101.35.11	TCP	90	[TCP Dup ACK 1540#92] 63358 -> 443 [ACK] Seq=4071 Ack=1652351 Win=531456 Len=0 SLE=2131627 SRE=

The Wireshark Packet Capture Dashboard

The Wireshark packet capture dashboard consists of three main sessions, which include: the packet list pane, the packet details pane, packet bytes pane

The Packet List Pane

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.137.237	150.171.22.12	UDP	1292	60095 -> 443 Len=1250
2	0.000182	192.168.137.237	150.171.22.12	UDP	1292	60095 -> 443 Len=1250
3	0.000251	192.168.137.237	150.171.22.12	UDP	1292	60095 -> 443 Len=1250
4	0.000321	192.168.137.237	150.171.22.12	UDP	1292	60095 -> 443 Len=1250
5	0.000392	192.168.137.237	150.171.22.12	UDP	440	60095 -> 443 Len=398
6	0.000500	192.168.137.237	150.171.22.12	UDP	414	60095 -> 443 Len=372
7	0.000674	150.171.22.12	192.168.137.237	UDP	75	443 -> 60095 Len=33
8	0.0008298	150.171.22.12	192.168.137.237	UDP	80	443 -> 60095 Len=38
9	0.0008298	150.171.22.12	192.168.137.237	UDP	80	443 -> 60095 Len=38
10	0.0008298	150.171.22.12	192.168.137.237	UDP	75	443 -> 60095 Len=33
11	0.0008597	192.168.137.237	150.171.22.12	UDP	80	60095 -> 443 Len=38
12	0.208143	150.171.22.12	192.168.137.237	UDP	1514	443 -> 60095 Len=1472
13	0.208143	150.171.22.12	192.168.137.237	UDP	1123	443 -> 60095 Len=1081
14	0.215313	192.168.137.237	150.171.22.12	UDP	85	60095 -> 443 Len=43
15	0.230570	150.171.22.12	192.168.137.237	UDP	72	443 -> 60095 Len=30
16	0.239931	192.168.137.237	150.171.22.12	UDP	81	60095 -> 443 Len=39

Frame 162289: Packet, 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface \Device\NPF_{8D048A7-DF70-4D96-A

```

0000: fc 5c 68 03 e9 bd 7c 2a 31 14 4e 43 88 00 45 00 .\h...
0010: 00 4f 89 34 40 88 00 00 00 c0 48 89 4d d8 3a 0@...
0020: d7 2e e2 b1 bb e5 ef 35 63 9c 7d 3d 33 50 18 ..@...
0030: 01 01 fa 40 00 00 17 03 03 00 22 4a 9c f8 3c 44 ..@...
0040: 4a 1f c4 12 04 7d ab 02 97 63 d4 f8 6a 41 73 fc J...|n-c-p...
0050: 6e b7 63 93 70 13 c1 63 1c 9d 04 72 24

```

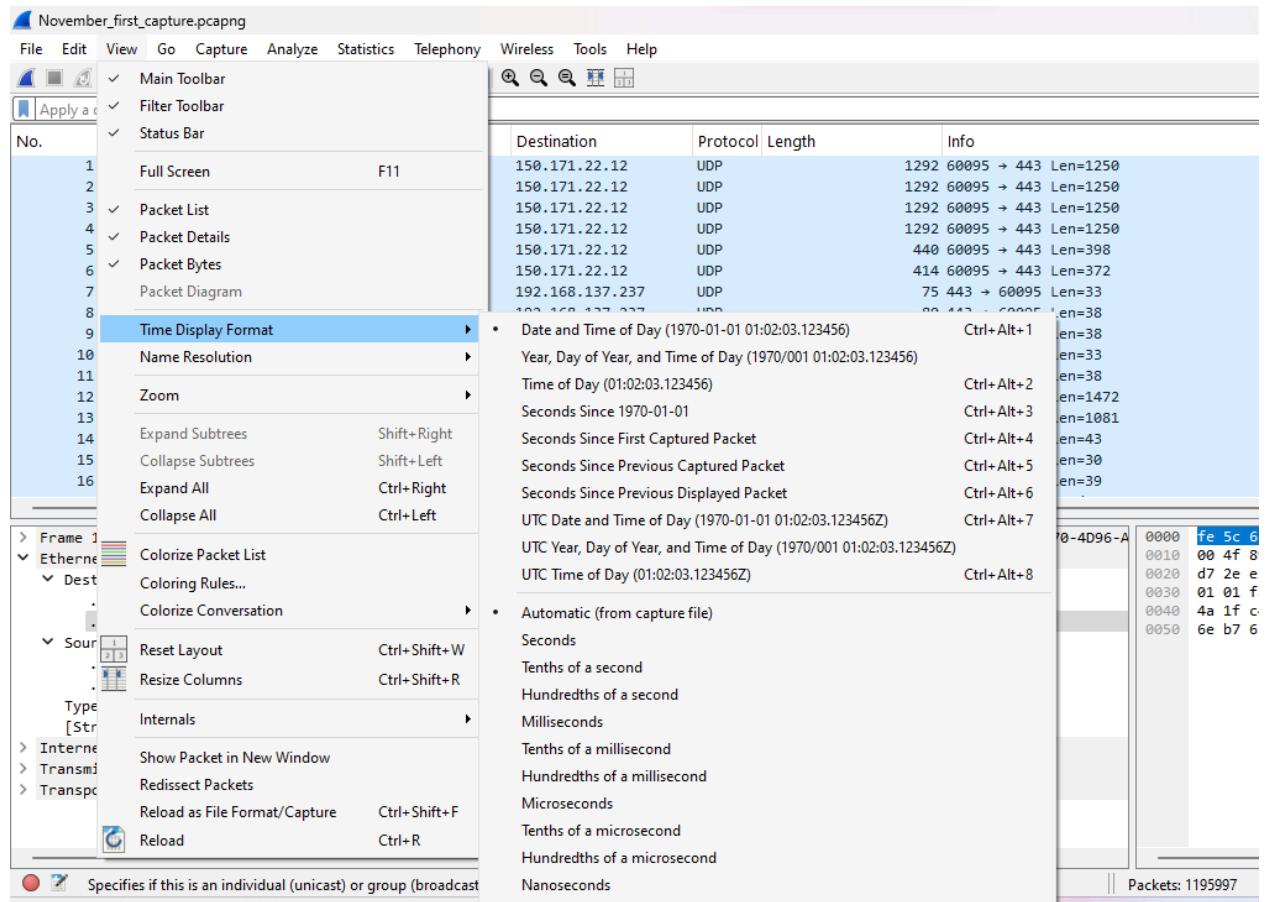
Specifies if this is an individual (unicast) or group (broadcast/multicast) address (eth.dst.ig), 1 bit

Packets: 1195997 | Profile: Default

The packet list pane is the portion of the snippet mapped out by the green marker, and it gives the summary view of the packet capture, as it would display a list of all packets captured in the file and contain the following details

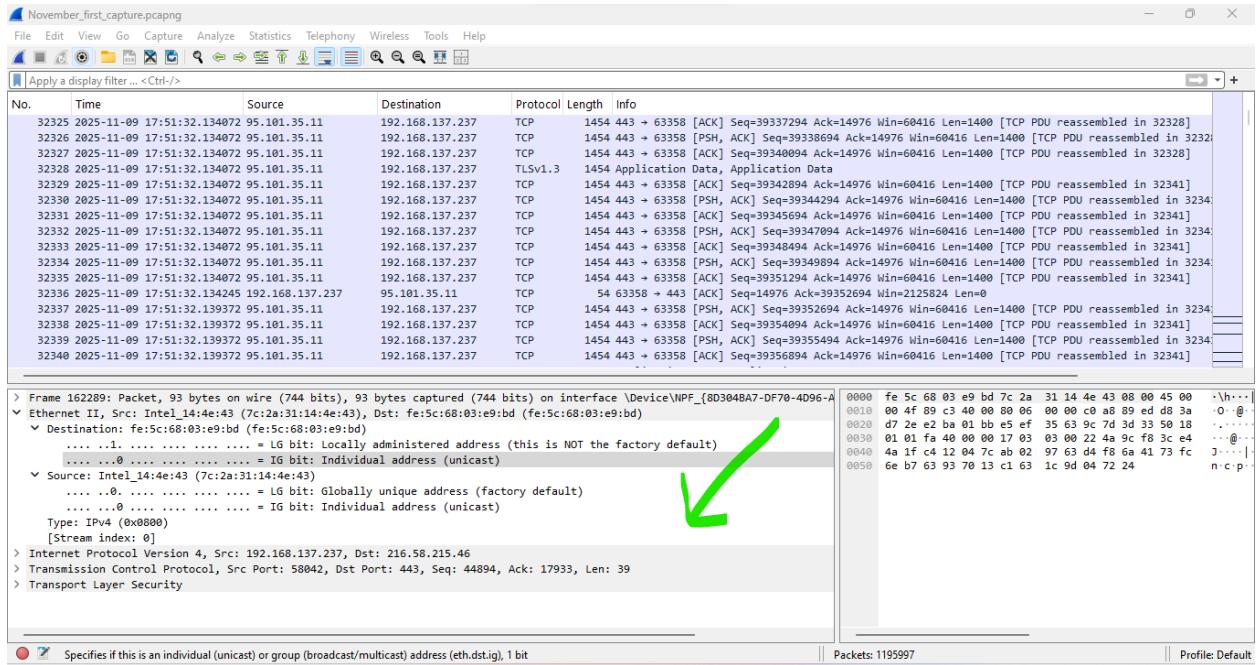
- No. (Number):** This is the index number assigned to a packet as it is received and serves the purpose of a serial number.
- Time:** This is the timestamp of when the packet was captured, and it is measured in **microseconds** by default. To adjust the time

display format, go to **View**, then select **Time Display format** and choose the option that best suits your needs.



- Source:** This is the IP address of the endpoint sending the packet
- Destination:** This is the IP address of the endpoint receiving the packet
- Protocol:** This is a protocol (established language for communication) used to communicate between the sending and receiving endpoints, such as ARP, DNS, TCP, UDP, and many more
- Length:** This represents the size of the packet in bytes
- Info:** This would provide a brief and short summary of the content of the packet that can help us understand the purpose or what is happening with the packet

The Packets Details pane



The packet's details pane gives a more granular detail of the packet by **dissecting it layer by layer**, following the OSI model.

- With the **Frame** representing the **physical layer**, giving details about the capture interface, total bits/bytes captured on the wire, and the Frame length

In the snippet below, we can see that the capture interface is **Wi-Fi 2**, and also the arrival time in a different format, and the Frame number is **1**, indicating that it is the first frame captured by Wireshark, and the frame length and the capture length as **1292 bytes** representing the total size of the packet as transmitted on the **physical medium which is Wi-Fi** in this case and the actual size of the packet as saved to file on wireshark respectively and then it further provided information about whether the **frame is marked or ignored** which in this case is **false for both**, and the **Protocols in frame section** gives a view of the protocol stack used in the packet indicating the layers of encapsulation showing its movement from **etherent to IP to UDP and finally to data**

```

▼ Frame 1: Packet, 1292 bytes on wire (10336 bits), 1292 bytes captured (10336 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
  Section number: 1
  ▼ Interface id: 0 (\Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87})
    Interface name: \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}
    Interface description: Wi-Fi 2
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 9, 2025 17:49:41.625767000 W. Central Africa Standard Time
    UTC Arrival Time: Nov 9, 2025 16:49:41.625767000 UTC
    Epoch Arrival Time: 1762706981.625767000
    [Time shift for this packet: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 1292 bytes (10336 bits)
    Capture Length: 1292 bytes (10336 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:data]
    Character encoding: ASCII (0)
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]

```

The **Ethernet II** section presents the **Data Link layer** and deals with MAC addresses, and the major components in this section are the **Source**, which represents the MAC address of the sending endpoint, and the **Destination**, which means the MAC address of the receiving endpoint, the **IG bit**, means the **individual/Group bit** and determines if the address is for a single device (Unicast) when *IG bit* = 0 or to a group (multicast) where *IG bit* = 1. The **LG bit** means **Local/Global bit**, which determines how the address was assigned, whether it was manually or dynamically changed by the operating system or by virtualization software when *LG bit* = 1, or it is the factory-burned-in MAC address when *LG bit* = 0. The **Type field with parameter IPV4(0x0800)** tells the receiving device the protocol encapsulated inside the Ethernet frame and the next protocol layer to handle the packets. The **Stream index** is used for tracking related packets beginning with a single connection. The snippet below is a capture of Ethernet II

```

▼ Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
  ▼ Destination: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
    .... ..1. .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: Intel_14:4e:43 (7c:2a:31:14:4e:43)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
  [Stream index: 0]

```

The **Internet Protocol Version 4** section represents the **Network layer** that handles logical addressing. In the snippet below, the **version: 4** confirms that it is an *IPv4* protocol, then the **Header length** tells us about the length of the IPv4 header, which is **20 bytes**.

Differentiated Services Field is used in classifying and managing the network traffic, allowing certain types of data to be prioritized over others, **The Differentiated services Codepoint: Default (0)** indicates

that this current packet has default priority which is best effort, **The Explicit Congestion Notification: Not ECN-Capable Transport(0)** indicates a mechanism for devices used to signal congestion and reduce retransmission, in this case the packet is not participating in ECN, **Total Length: 1278** identify the size of the entire IP Datagram as **1278 bytes**, **Identification: 0xb2ff (45823)** help to inform destination endpoint about fragment belonging to the same original packet and the **number (45823)** is unique to fragments of the same transmission, The **Flags: 0x2**, is used to control how packet being fragmented is handled depending on how the *Reserved bit*, *Don't Fragment*, *More fragment* is set, in this case the “1.. = Don't fragment: Set.” meaning that the packet is not to be fragmented, **The Fragment Offset** normally represent the fragment number, in this case it is 0 meaning that it is not fragmented and no other fragment is expected as this is the entire packet.

The Time to Live: 128 represents a counter that prevents packets from circulating endlessly. As the value decreases by 1 and it reaches 0, the packet is dropped. **Protocol: UDP (17)**, the section tells the receiving machine which protocol is to be handled in layer 4, which corresponds to UDP. In this case, the **Header Checksum** is used to verify if the IP header has not been corrupted during transit, and in this case, the validation was disabled, and the checksum is not Unverified.

Ensuring that validation is enabled is important for identifying malformed packets. **Source Address and Destination Addresses** represent the IP address for the sending and receiving endpoints, respectively, and the **Stream index** is used for tracking related packets beginning with a single connection, as previously stated

```

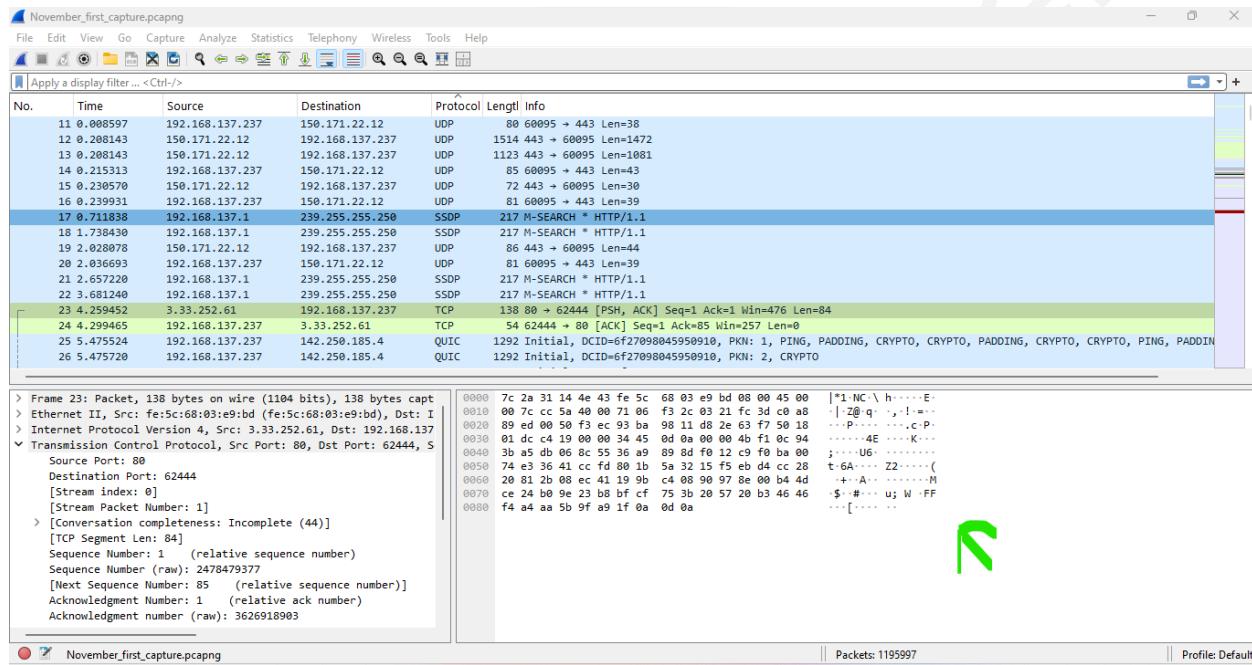
-----+
`- Internet Protocol Version 4, Src: 192.168.137.237, Dst: 150.171.22.12
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
`- Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  0000 00.. = Differentiated Services Codepoint: Default (0)
  .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 1278
  Identification: 0xb2ff (45823)
`- 010. .... = Flags: 0x2, Don't fragment
  0... .... = Reserved bit: Not set
  .1.. .... = Don't fragment: Set
  ..0. .... = More fragments: Not set
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: UDP (17)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.137.237
  Destination Address: 150.171.22.12
  [Stream index: 0]

```

The **Transmission Control Protocol**, which in other cases can be the User Datagram Protocol section, represents the **Transport layer**, and it determines the connection reliability. In a further section of this documentation, there will be a detailed explanation of TCP, UDP, and other protocols analyzed in this project.

The next section, which mostly represents the **Application layer**, is treated differently and will be discussed further in another part of the report

The Packets Bytes pane



The **Packet Byte pane** shows the exact data as the computer sees it, so the packet byte pane gives the raw data view, On the left side of the byte pane is data formatted in **base 16 (Hexadecimal)** which is what the machine reads and On the right side of the byte pane is data formatted in **ASCII** and it translates the base-16 data on the left to human readable text, the ASCII text may produce human readable text or not depending on the protocol being evaluated, the snippet above is not so readable unlike the one below

```

219528 1331.655987 216.58.212.99 192.168.137.237 HTTP 277 HTTP/1.1 304 Not Modified
219530 1331.678862 192.168.137.237 216.58.212.99 HTTP 254 GET /r/r4.crl HTTP/1.1
219556 1331.864164 192.168.137.237 216.58.212.99 HTTP 254 GET /r/crl HTTP/1.1

> Frame 219530: Packet, 254 bytes on wire (2032 bits), 254 bytes
> Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5
> Internet Protocol Version 4, Src: 192.168.137.237, Dst: 216.58.
Transmission Control Protocol, Src Port: 58441, Dst Port: 80, S
Source Port: 58441
Destination Port: 80
[Stream index: 231]
[Stream Packet Number: 7]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 200]
Sequence Number: 203 (relative sequence number)
Sequence Number (raw): 3841318330
[Next Sequence Number: 403 (relative sequence number)]
Acknowledgment Number: 224 (relative ack number)
Acknowledgment number (raw): 3125507630

```

Filtering with Wireshark

Most times, capturing network traffic would provide an enormous amount of data, and as a network administrator or security professional scanning through this data is largely a waste of time, so we can utilize various filters in Wireshark to make our search more seamless and effective when carrying out protocol analysis and packet capture, such as:

Filtering with the protocol: One of the easiest ways to filter through the packet capture is to directly filter through the protocol name, such as DNS, ARP, TCP, HTTP, and so on

No.	Time	Source	Destination	Protocol	Length	Info
160869	842.500486	Intel_14:4e:43	Broadcast	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
160871	843.493570	Intel_14:4e:43	Broadcast	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
160873	843.873315	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
210555	1238.501095	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
210569	1238.572919	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
210707	1246.004766	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
210708	1246.093624	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
210781	1252.998156	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
210782	1253.143558	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
408665	2768.872066	a2:7a:0d:e6:20:8e	Broadcast	ARP	42	Who has 192.168.137.1? Tell 192.168.137.197
408922	2770.106470	a2:7a:0d:e6:20:8e	Broadcast	ARP	42	ARP Announcement for 192.168.137.197
448289	3015.997716	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
448291	3016.001508	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
453584	3061.006229	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237
453585	3061.010597	fe:5c:68:03:e9:bd	Intel_14:4e:43	ARP	46	192.168.137.1 is at fe:5c:68:03:e9:bd
463650	3151.495547	Intel_14:4e:43	fe:5c:68:03:e9:bd	ARP	42	Who has 192.168.137.1? Tell 192.168.137.237

Filtering with Addresses: We can also filter traffic via addresses, such as **source and destination MAC addresses**, **source and destination IP addresses**. The common syntax to filter with addresses would be

```

eth.addr == 00:00:5e:00:53:00, eth.src == fe:5c:68:03:e9:bd, eth.dst
== fe:5c:68:03:e9:bd, ip.addr == 192.168.137.1, ip.src ==
192.168.137.1, ip.dst == 192.168.137.1

```

Common comparison operators to use in Wireshark

Abbreviation	Operator	Meaning
eq	==	Equal
ne	!=	Not Equal
gt	>	Greater Than
lt	<	Less Than
ge	>=	Greater than or Equal to
le	<=	Less than or Equal to

Search and Match operators

Operator	Description
contains	Tests whether a protocol, field, or slice contains a specific value
matches, ~	Checks if the string matches the specified Perl-compatible regular expression, ignoring case.

Filtering based on various protocol fields: we can filter based on various properties in the protocol fields, such as ports, responses, and so much more. Common syntax is: `tcp.port == 80, udp.port == 53, http.request.method == "GET", dns.flags.response == 1, arp.opcode == 1 (for ARP request), arp.opcode == 2 (for ARP reply), ip.proto == 6 (for TCP), ip.proto == 17 (for UDP), icmp.type == 8 (for Echo request), icmp.type == 0 (for Echo reply).`

Filtering based on frame properties: We can filter packets based on their inherent characteristics, irrespective of the encapsulated protocols:

- **Filter by size/length:** `frame.len > 1500` (shows packets larger than the standard Ethernet MTU), `frame.len <= 100` (shows very small packets).
- **Filter by frame number:** `frame.number == 1000` (shows the 1000th captured packet), `frame.number > 5000` and `frame.number < 5100` (shows a range of packets).
- **Filter by errors:** `expert.severity == error` (identifies packets flagged by Wireshark's expert system as having errors, often useful for troubleshooting malformed or corrupted packets).

Logical Operators

Combining filters using logical operators allows for complex and precise traffic analysis:

- **AND (&&):** Requires both conditions to be true. Example: `ip.addr == 192.168.1.100 && tcp.port == 443` (Traffic to/from 192.168.1.100 on HTTPS port).
- **OR (||):** Requires at least one condition to be true. Example: `dns || arp` (Shows all DNS or ARP packets).

Protocol Analysis with Wireshark

This section of the report takes certain protocols and dives into a deeper explanation of the components of these protocols as defined in Wireshark.

Protocol analysis of UDP

User Datagram Protocol(UDP) is a connectionless-oriented protocol that operates at layer 4 of the OSI model (Transport Layer), and it is responsible for carrying data or packets that require less reliability and are ideal for applications where occasional lost packets are tolerated, such as in DNS queries and video streaming. It has the protocol number of 17

```

▼ User Datagram Protocol, Src Port: 60095, Dst Port: 443
  Source Port: 60095
  Destination Port: 443
  Length: 1258
  Checksum: 0xfc48 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Stream Packet Number: 1]
  ▼ [Timestamps]
    [Time since first frame: 0.00000000 seconds]
    [Time since previous frame: 0.00000000 seconds]
  UDP payload (1250 bytes)
  ▼ Data (1250 bytes)
    Data [...]: 44d9c884372e1875d91d8f364c24da672d39bdb273741dceea00a210d8a34acc98f04a32ed63f1c97a7b5d250c899f1f2db89988101af64870bc45d5ca82ca5534327c3
    [Length: 1250]

```

From the snippet above, the UDP contains the Source port of **60095**, which represents the source port used by the sending endpoint to communicate with the service of the receiving endpoint, which has the destination port of **443**, which is widely recognized as the HTTPS protocol, and the **Length** represents the length of the entire UDP payload. The **Checksum** represents a mathematical value used in detecting errors in the packet's header and data. The checksum is unverified, which means the capture device didn't calculate it; the stream index is used to keep track of traffic within the same stream or thread, and the **Timestamps** show the relative timing measurements of the packets sent.

The UDP payload contains the actual application data being carried inside the UDP packet,

UDP Packet with SSDP as Data payload

```

forward, hold to see history | 
> Frame 17: Packet, 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id
> Ethernet II, Src: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
> Internet Protocol Version 4, Src: 192.168.137.1, Dst: 239.255.255.250
> User Datagram Protocol, Src Port: 59967, Dst Port: 1900
  ▼ Simple Service Discovery Protocol
    > M-SEARCH * HTTP/1.1\r\n
      HOST: 239.255.255.250:1900\r\n
      MAN: "ssdp:discover"\r\n
      MX: 1\r\n
      ST: urn:dial-multiscreen-org:service:dial:1\r\n
      USER-AGENT: Google Chrome/118.0.5993.120 Windows\r\n
      \r\n
      [Full request URI: http://239.255.255.250:1900*]

```

This capture shows a UDP packet with the payload as **SSDP**, which is short for **Simple Service Discovery Protocol**, which is commonly used by devices on a local network to discover services offered by other devices, it is important to notice that the destination ip is for a

multicast(239.255.255.250) and not designated to a single device, The SSDP data is structured using HTTP-like headers, encapsulated within the UDP packet. **M-SEARCH * HTTP/1.1:** This is the method used. M-SEARCH stands for Method Search and is the request sent by a client to discover available devices or services. **HOST: 239.255.255.250:1900:** Confirms the request is targeted at the standard SSDP multicast address and port.

MAN: "ssdp: discover": The MAN (Mandatory) header specifies the search method, confirming that the client wants to discover services. **MX: 1:** The **Maximum Wait Time** in seconds. Devices should wait a random amount of time up to 1 second before responding to prevent all devices from responding simultaneously and flooding the network.

ST: urn:dial-multiscreen-org:service:dial:1: The Service Type (ST) header is the core of the request, asking specifically for a device or service that supports the DIAL (Discovery and Launch) protocol, version 1.

USER-AGENT: Google Chrome/118.0.5993.120 Windows: This identifies the client initiating the search.

UDP packets with QUIC as payload

```
> User Datagram Protocol, Src Port: 63400, Dst Port: 443
  < QUIC IETF
    < QUIC Connection information
      [Connection Number: 0]
      [Packet Length: 1250]
      [.... .... = Header Form: Long Header (1)
       .1... .... = Fixed Bit: True
       ..00 .... = Packet Type: Initial (0)
       [.... 00.. = Reserved: 0]
      [.... ..00 = Packet Number Length: 1 bytes (0)]
      Version: 1 (0x00000001)
      Destination Connection ID Length: 8
      Destination Connection ID: 6f27098045950910
      Source Connection ID Length: 0
      Token Length: 70
      Token: 001fc73a58740bf1d644372ddaa6cc1a22302a160d7b46fb80cdb76e27fe04ba63c0fcbb020ca3bda1701308bd7cba244fec66a0f0bdaa2344da1ef82c9ef14b8bc4358fdf
      Length: 1161
      [Packet Number: 1]
      Payload [-]: 4467fdf6a0c49053a85ff211f8744c0af0ff788112cc709747dd9d78ac981f475100f29af83f6d8a72cacde4e7bac02b6012b16e90cfa100cd852e720218f1c3598
    > PING
    > PADDING Length: 1
    > CRYPTO
    > CRYPTO
    ... --
```

This snippet shows a packet using **QUIC**, which means **Quick UDP Internet Connections Protocol**, which is a modern transport layer network protocol developed by Google and it is to replace TCP/TLS for HTTP traffic to make internet connections faster and more secure. The following components forms the QUIC protocol: **Destination Connection**

ID which is the unique identifier for the connection chosen by the server and this is used to route packets to the connection handler, the **Token with length 70** is a cryptographic value used by the server to validate the client's address which forms a defense against DOS attacks, the payload in the QUIC is encrypted and contains components such as:

- **Packet Number:** the sequence number for the QUIC packet
- **PING:** A frame used to confirm the connection is still active and can receive data
- **PADDING LENGTH:** This is the length of the padding frames, which is sometimes used to make a packet meet the minimum size
- **CRYPTO:** This frame carries encrypted handshake data used to negotiate the security keys for the connection

UDP packet with DNS as payload

DNS Query

```
> Frame 84: Packet, 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
> Internet Protocol Version 4, Src: 192.168.137.237, Dst: 192.168.137.1
> User Datagram Protocol, Src Port: 57985, Dst Port: 53
└ Domain Name System (query)
  Transaction ID: 0xe0f0
  > Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  < Queries
    > vod-adaptive-ak.vimeocdn.com: type HTTPS, class IN
  [Response In: 89]
```

The snippet above describes a **DNS Query** to resolve a hostname by asking a local DNS server to find the IP address for the hostname vod-adaptive-ak.vimeocdn.com, The structure of DNS Query request is defined with the **Flags: 0x0100** Standard query indicates that the packet is a standard query, **Transaction ID: 0xe0f0** represents a unique identifier assigned to this request, **Questions: 1** shows that the client is asking for the resolution of one single hostname, **Answer RRs: 0 / Authority RRs: 0 / Additional RRs: 0**

These fields are all zero because this is a query, not a response. In a response packet, these fields would indicate the number of resource records (RRs) being sent back. **Queries > vod-adaptive-ak.vimeocdn.com:** This is the hostname (domain name) the client wants resolved to an IP address

DNS response

```
> Frame 88: Packet, 220 bytes on wire (1760 bits), 220 bytes captured (1760 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd), Dst: Intel_14:4e:43 (7c:2a:31:14:4e:43)
> Internet Protocol Version 4, Src: 192.168.137.1, Dst: 192.168.137.237
> User Datagram Protocol, Src Port: 53, Dst Port: 58066
└ Domain Name System (response)
    Transaction ID: 0x086f
    > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 2
    Authority RRs: 1
    Additional RRs: 0
    < Queries
        > acrobat.adobe.com: type HTTPS, class IN
    < Answers
        > acrobat.adobe.com: type CNAME, class IN, cname acrobat.adobe.com.i.edgekey.net
        > acrobat.adobe.com.i.edgekey.net: type CNAME, class IN, cname e29329.dsca.akamaiedge.net
    > Authoritative nameservers
        [Request In: 86]
    [Time: 74.910000 milliseconds]
```

The snippet above shows a **DNS response** that has a simple structure with the DNS Query.

Protocol Analysis of ICMP

Internet Control Message Protocol (ICMP) is a network layer (Layer 3) protocol used by network devices, such as routers and hosts, to send error messages and operational information, such as whether a requested service is available or if a host or router can be reached. **ICMP** is crucial for network troubleshooting and diagnostics, most famously utilized by the **ping** and **traceroute** utilities. ICMP messages are encapsulated within an IP packet. Key components of an ICMP packet include the **Type** and **Code** fields, which together specify the nature of the message (e.g., Echo Request, Echo Reply, Destination Unreachable, Time Exceeded).

```
> Frame 849456: Packet, 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
> Internet Protocol Version 4, Src: 192.168.137.237, Dst: 8.8.8.8
└ Internet Control Message Protocol
    Type: Echo (ping) request (8)
    Code: 0
    Checksum: 0x4d42 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 25 (0x0019)
    Sequence Number (LE): 6400 (0x1900)
    [Response frame: 849476]
└ Data (32 bytes)
    Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869
    [Length: 32]
```

ICMP Echo Request

This snippet displays an **ICMP Echo Request** message, commonly known as a **ping request**, sent from a source host to a destination host to check connectivity.

- **Type: 8 (Echo (ping) request):** This field explicitly identifies the packet as an ICMP Echo Request.
- **Code: 0:** This indicates that the request is a standard Echo Request.
- **Checksum: 0x98f8 [correct]:** This is the value calculated to ensure the integrity of the ICMP header and data during transit. The **[correct]** annotation confirms that the calculated checksum matches the value in the packet.
- **Identifier (BE): / Identifier (LE):** : The Identifier is used by the sending host to match Echo Requests with their corresponding Echo Replies. The (BE) and (LE) values represent the identifier in Big-Endian and Little-Endian byte order, respectively, which is useful for different system architectures.
- **Sequence Number (BE): / Sequence Number (LE)::** The Sequence Number is a counter that increases with each subsequent ICMP message sent. It helps the sending host determine if packets were lost or received out of order.

- **Data (32 bytes):** This field contains the arbitrary data payload sent within the ICMP request

```
> Frame 849476: Packet, 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd), Dst: Intel_14:4e:43 (7c:2a:31:14:4e:43)
> Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.137.237
  <-- Internet Control Message Protocol
    Type: Echo (ping) reply (0)
    Code: 0
    Checksum: 0x5542 [correct]
      [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 25 (0x0019)
    Sequence Number (LE): 6400 (0x1900)
      [Request frame: 849456]
    [Response time: 46.577 ms]
  <-- Data (32 bytes)
    Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869
      [Length: 32]
```

ICMP Echo Reply

This snippet displays an **ICMP Echo Reply** message, and it has a similar structure to ICMP Echo Request, which is the response to an Echo Request, confirming that the destination host is reachable.

Protocol Analysis of ARP

Address Resolution Protocol (ARP) is a critical Layer 2 (Data Link) protocol used to map an Internet Protocol (IP) address to its corresponding physical hardware address, known as the **Media Access Control (MAC) address**. This translation is essential for devices to communicate with each other within a local network segment (LAN). Since IP addresses are used for routing at Layer 3, and MAC addresses are required for actual frame delivery at Layer 2, ARP acts as the bridge between these two addressing schemes.

ARP Request

```
> Frame 826585: Packet, 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
  <-- Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Intel_14:4e:43 (7c:2a:31:14:4e:43)
    Sender IP address: 192.168.137.237
    Target MAC address: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
    Target IP address: 192.168.137.1
```

An ARP transaction begins with an **ARP Request**. This request is broadcast across the entire local network segment to all connected devices.

- **Opcode: 1 (Request)**: This field identifies the packet as an ARP Request.
- **Sender MAC Address** and **Sender IP Address**: The addresses of the host initiating the request.
- **Target IP Address**: The IP address for which the MAC address is being sought.
- **Target MAC Address**: This field is set to **00:00:00:00:00:00** (or similar null value) because the MAC address is unknown.

When a device receives an ARP Request, it compares the Target IP Address in the packet with its own IP address. Only the device whose IP address matches the target IP will respond.

ARP Reply

```
> Frame 871786: Packet, 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id 0
> Ethernet II, Src: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd), Dst: Intel_14:4e:43 (7c:2a:31:14:4e:43)
> 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 0
▼ Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
  Sender IP address: 192.168.137.1
  Target MAC address: Intel_14:4e:43 (7c:2a:31:14:4e:43)
  Target IP address: 192.168.137.237
```

The device that recognizes the Target IP Address as its own will send an **ARP Reply** directly back to the requesting host (Unicast).

- **Opcode: 2 (Reply)**: This field identifies the packet as an ARP Reply.
- **Sender MAC Address** and **Sender IP Address**: The MAC and IP addresses of the responding host (which were the Target addresses in the Request).
- **Target MAC Address** and **Target IP Address**: The MAC and IP addresses of the original requesting host.

Protocol Analysis of TCP

Transmission Control Protocol (TCP) is a core, connection-oriented protocol operating at Layer 4 (Transport Layer) of the OSI model. Unlike UDP, TCP guarantees reliable, ordered, and error-checked delivery of data between applications. This reliability is achieved through mechanisms like the three-way handshake for connection establishment, sequencing, acknowledgments (ACKs), and flow control, making it ideal for applications that require high integrity, such as web browsing (HTTP/HTTPS), email transfer (SMTP), and file transfer (FTP).

```
> Ethernet II, Src: Intel_14:4e:43 (7c:2a:31:14:4e:43), Dst: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd)
> Internet Protocol Version 4, Src: 192.168.137.237, Dst: 216.58.214.78
> Transmission Control Protocol, Src Port: 51984, Dst Port: 80, Seq: 1, Ack: 1, Len: 322
    Source Port: 51984
    Destination Port: 80
    [Stream index: 33]
    [Stream Packet Number: 4]
    > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 322]
    Sequence Number: 1 (relative sequence number)
    Sequence Number (raw): 426781745
    [Next Sequence Number: 323 (relative sequence number)]
    Acknowledgment Number: 1 (relative ack number)
    Acknowledgment number (raw): 4088837560
    Acknowledgment number (raw): 4088837560
    0101 .... = Header Length: 20 bytes (5)
    > Flags: 0x010 (PSH, ACK)
    Window: 257
    [Calculated window size: 65792]
    [Window size scaling factor: 256]
    Checksum: 0xfa7b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    > [Timestamps]
    > [SEQ/ACK analysis]
    [Client Contiguous Streams: 1]
    [Server Contiguous Streams: 1]
    TCP payload (322 bytes)
    Hypertext Transfer Protocol
    > [...] GET /collect?v=1&id=UA-65945523-14&cid=HP_Audio_3ca93b7f-3836-4ffc-88e6-22f5b694a5a2&cs=SystemTray&an=HPAudioSwitch&av=1.0.150.0&ip=1&cm=Preinstalled&sr=1366*768&t=event&cd=outp
    Host: www.google-analytics.com\r\n
    Connection: Keep-Alive\r\n
```

TCP Components

The TCP header contains several vital components necessary for maintaining the connection and ensuring reliable delivery:

- **Source Port** and **Destination Port**: These fields identify the application or service sending and receiving the data, respectively.
- **Sequence Number**: A field representing the sequence number of the first data byte in the segment. It is used to reassemble the data in the correct order.
- **Acknowledgment Number (ACK)**: A field containing the next sequence number the sender of the segment is expecting to receive from the other side, confirming receipt of all prior data.
- **Header Length**: Specifies the size of the TCP header.
- **Flags/Control Bits**: These 9 individual bits are crucial for connection management and flow control:
 - **URG (Urgent)**: Indicates the Urgent Pointer field is significant.

- **ACK (Acknowledgment):** Indicates the Acknowledgment Number field is significant (always set after the initial handshake SYN).
 - **PSH (Push):** Forces the sending application to deliver data to the receiving application immediately.
 - **RST (Reset):** Terminates a connection abruptly due to an error.
 - **SYN (Synchronize):** Used to initiate a connection (first step of the three-way handshake).
 - **FIN (Finish):** Used to gracefully terminate a connection.
- **Window Size:** A field used for flow control, indicating the amount of receive buffer space (in bytes) available, informing the sender how much data it can transmit before receiving an acknowledgment.
- **Checksum:** A field used for error-checking the TCP header and data payload.
- **Urgent Pointer:** Points to the byte offset in the sequence number where urgent data ends.
- **Options:** Allows for optional features, such as Maximum Segment Size (MSS) negotiation.

PACKET CAPTURE WITH TCPDUMP

This is a Command Line Interface (CLI) Network packet capture and protocol analyzer that is capable of capturing live traffic and widely used for network troubleshooting, security monitoring, and is particularly suited to Unix-like operating Systems, it is similar to wireshark in its functionality in capturing packets

Common Commands used in Tcpdump and their uses

Sudo tcpdump -D : this command would list out all interface on which tcpdump can capture traffic on

```
zsh: corrupt history file /home/kali/.zsh_history
[~] (kali㉿kali)-[~]
└─$ sudo tcpdump -D
[sudo] password for kali:
1.eth0 [Up, Running, Connected]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
7 dbus-system (D-Bus system bus) [none]
8 dbus-session (D-Bus session bus) [none]
```

```
[~] (kali㉿kali)-[~]
└─$ █
```

Sudo tcpdump -i eth0: This command tells tcpdump to capture traffic on the eth0 interface

```
[~] (kali㉿kali)-[~]
└─$ sudo tcpdump -i eth0
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v] ... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Sudo tcpdump -n -i eth0: this command tells tcpdump to capture traffic on the eth0 interface and ensures that name resolution is disabled

```
[~] (kali㉿kali)-[~]
└─$ sudo tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v[v] ... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
04:51:37.509622 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 53, length 64
04:51:37.515851 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 53, length 64
04:51:38.511471 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 54, length 64
04:51:38.521995 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 54, length 64
04:51:39.514864 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 55, length 64
04:51:39.521535 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 55, length 64
04:51:40.517851 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 56, length 64
04:51:40.525573 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 56, length 64
04:51:41.674043 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
04:51:41.677240 ARP, Reply 10.0.2.2 is-at 52:55:0a:00:02:02, length 50
04:51:41.678482 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 57, length 64
04:51:41.845596 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 57, length 64
04:51:43.499540 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 58, length 64
04:51:43.506009 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 58, length 64
^X04:51:44.501927 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 59, length 64
04:51:44.518827 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 59, length 64
04:51:46.313232 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 60, length 64
^C bytes from 8.8.8.8: icmp_seq=73 ttl=255 time=15.5 ms
17 packets captured, 17 packets received by filter
18 packets received by kernel
04 packets dropped by kernel seq=76 ttl=255 time=6.96 ms
```

Sudo tcpdump -c10 -i eth0: this command tells tcpdump to capture

traffic on the eth0 interface and ensures to only capture 10 packets.

```
(kali㉿kali)-[~] 8: ~$ sudo tcpdump -c10 -i eth0
$ sudo tcpdump -c10 -i eth0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
64 bytes from 8.8.8.8: icmp_seq=230 ttl=255 time=6.38 ms
64 bytes from 8.8.8.8: icmp_seq=231 ttl=255 time=6.44 ms
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
64 bytes from 8.8.8.8: icmp_seq=234 ttl=255 time=92.8 ms
64 bytes from 8.8.8.8: icmp_seq=235 ttl=255 time=6.49 ms
64 bytes from 8.8.8.8: icmp_seq=236 ttl=255 time=9.86 ms
04:55:00.772812 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 223, length 64
04:55:00.780063 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 223, length 64
04:55:01.774892 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 224, length 64
04:55:01.781622 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 224, length 64
04:55:02.775966 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 225, length 64
04:55:02.782597 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 225, length 64
04:55:03.777733 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 226, length 64
04:55:03.783680 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 226, length 64
04:55:04.784069 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 227, length 64
04:55:04.791720 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 227, length 64
10 packets captured 81 bytes 64 bytes received by filter
42 packets received by kernel
26 packets dropped by kernel
```

Sudo tcpdump -w file.pcap: This command tells tcpdump to store the captured packets in file.pcap

```
(kali㉿kali)-[~] 9: ~$ sudo tcpdump -w file.pcap
$ sudo tcpdump -w file.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
64 bytes from 8.8.8.8: icmp_seq=199 ttl=255 time=7.06 ms
64 bytes from 8.8.8.8: icmp_seq=200 ttl=255 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=201 ttl=255 time=7.82 ms
64 bytes from 8.8.8.8: icmp_seq=202 ttl=255 time=6.89 ms
64 bytes from 8.8.8.8: icmp_seq=203 ttl=255 time=12.0 ms
64 bytes from 8.8.8.8: icmp_seq=204 ttl=255 time=10.7 ms
64 bytes from 8.8.8.8: icmp_seq=205 ttl=255 time=6.97 ms
^C69 packets captured
51 packets received by filter
0 packets dropped by kernel
(kali㉿kali)-[~] 10: ~$ ls
Desktop  Downloads  Karatu  Karatu-PKI  packages.microsoft.gpg  Project_Nov_capture2.pcap  Public  testFolder  Videos    wget-log  wget-log.2
Documents  file.pcap  Karatu-Assymmetric  Music      Pictures        Project_Nov_capture.pcap  Templates  testFolder2  vscode.deb  wget-log.1  wget-log.3
```

Sudo tcpdump -r file.pcap: This command tells tcpdump to read into the file.pcap file and displays it on the screen

```
(kali㉿kali)-[~] 81: icmp_seq=630 ttl=255 time=7.49 ms
$ sudo tcpdump -r file.pcap
reading from file file.pcap, link-type EN10MB (Ethernet), snapshot length 262144
04:57:35.428613 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 377, length 64
04:57:35.435417 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 377, length 64
04:57:36.430272 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 378, length 64
04:57:36.438776 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 378, length 64
04:57:37.431787 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 379, length 64
04:57:37.446639 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 379, length 64
04:57:38.433612 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 380, length 64
04:57:38.440321 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 380, length 64
04:57:39.434684 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 381, length 64
04:57:39.446933 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 381, length 64
04:57:40.436501 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 382, length 64
04:57:40.445422 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 382, length 64
04:57:41.437912 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 383, length 64
04:57:41.444296 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 383, length 64
04:57:42.438577 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 384, length 64
04:57:42.449598 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 384, length 64
04:57:43.440001 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 385, length 64
04:57:43.447076 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 385, length 64
04:57:44.441972 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 386, length 64
04:57:44.448907 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 386, length 64
04:57:45.444009 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 387, length 64
04:57:45.450635 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 387, length 64
04:57:46.445662 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 388, length 64
04:57:46.480326 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 388, length 64
04:57:47.447802 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 389, length 64
04:57:47.454477 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 389, length 64
04:57:48.449936 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 390, length 64
04:57:48.457854 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 390, length 64
04:57:49.452012 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 391, length 64
```

There are so many **tcpdump** command options to explore when dealing with **tcpdump**, and you get a cheatsheet at <https://www.comparitech.com/net-admin/tcpdump-cheat-sheet/>

Summary: This project report provided a detailed analysis of network traffic capture and protocol analysis using two essential tools: **Wireshark** and **Tcpdump**. The primary objective was to demonstrate the practical application of these tools in identifying risks, vulnerabilities, and threats within network traffic.

The report first established the context, detailing the cross-platform GUI-based capabilities of **Wireshark** for in-depth protocol analysis and filtering, alongside the CLI-based efficiency of **Tcpdump** for live traffic capture, particularly suited for Unix-like operating systems.

The core of the analysis focused on dissecting a captured PCAP file using **Wireshark**. The report meticulously detailed the **Wireshark** interface, including the Packet List Pane, Packet Details Pane (explaining the OSI layer dissection), and Packet Bytes Pane (showing

Hexadecimal and ASCII data). Crucial filtering techniques were explored, utilizing protocols, addresses, comparison operators, and logical operators (AND/OR) to streamline the analysis of large datasets.

In the **Protocol Analysis** section, key transport and network layer protocols were examined:

- **UDP:** Its connectionless nature was highlighted, alongside analysis of payloads such as **SSDP** (multicast service discovery) and **QUIC** (modern, fast, secure internet connections).
- **DNS:** The structure and distinction between DNS Queries and Responses were outlined.
- **ICMP:** Its role in network diagnostics (ping/traceroute) was covered, with a breakdown of Echo Request and Echo Reply messages.
- **ARP:** The Layer 2 process of mapping IP to MAC addresses via ARP Request (broadcast) and ARP Reply (unicast) was explained.
- **TCP:** As a reliable, connection-oriented protocol, its essential components, like Sequence Numbers, Acknowledgment Numbers, Flags (SYN, ACK, FIN, RST), and Window Size, were documented.

Finally, the report provided a cheatsheet of common **Tcpdump** commands for interface selection, packet count limits, name resolution disabling, and file input/output operations, confirming its utility for command-line-based network security monitoring and troubleshooting.

Conclusion: The successful execution of this project has demonstrated a solid foundational understanding of network traffic analysis using industry-standard tools. By performing deep dives into the packet structures of various protocols (UDP, ICMP, ARP, TCP, DNS, QUIC, SSDP), the report successfully illustrated how specific protocol fields and headers inform network behavior, connectivity, and potential security issues.

The ability to effectively filter traffic in Wireshark and manage captures using **Tcpdump** commands is paramount for network administrators and security professionals. These skills are directly applicable to:

- 1. Troubleshooting:** Quickly isolating connectivity issues (e.g., failed TCP handshakes, ICMP errors) or application latency (e.g., analyzing DNS response times).
- 2. Security Analysis:** Identifying anomalous traffic patterns, detecting unauthorized service discovery (SSDP), spotting malformed packets (indicated by Wireshark errors), or recognizing suspicious connection attempts (analyzing TCP flags).
- 3. Vulnerability Identification:** Pinpointing unencrypted communication (e.g., clear-text HTTP, older protocols) or observing behaviors that could indicate a Man-in-the-Middle or reconnaissance attack (excessive ARP or DNS queries).

In conclusion, Wireshark and Tcpdump are indispensable assets in the cybersecurity and network engineering toolkit. Mastery of these tools is crucial for ensuring network integrity, optimizing performance, and proactively defending against a constantly evolving threat landscape.