

ARP Spoofing + Man in the Middle Attack

Project Demonstration

Course: CSE 406

Anik Saha (2005001)

Supervisor:

Jaber Ahmed Deeder (2005023)

Abdur Rashid Tushar

Lecturer, CSE, BUET

Introduction

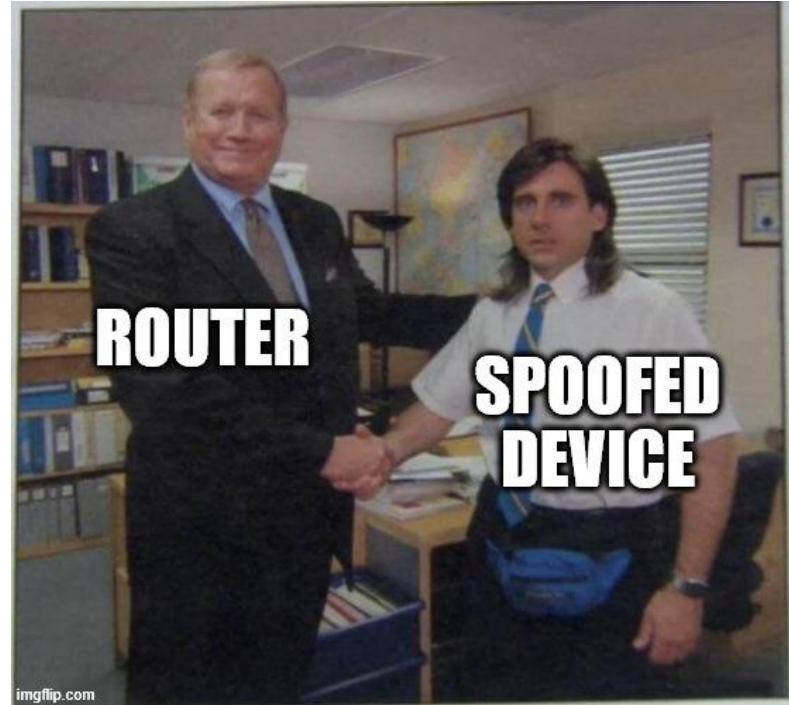
- ARP = Address Resolution Protocol.
- It's a network protocol used to find the **physical (MAC) address** associated with an **IP address** on a local network.
- Devices maintain their ARP tables as cache for quick translation

But **ARP blindly trusts replies!** Any device can send fake ARP messages.



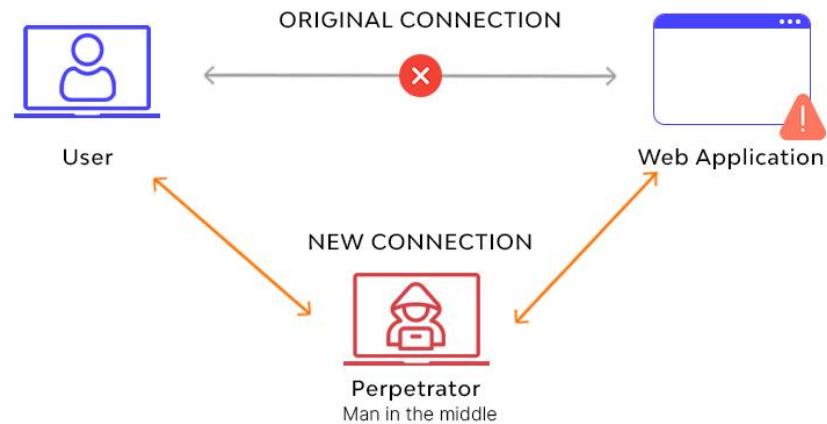
Introduction

- Spoofing means pretending to be a different device on a network (in this context)
- This can be used to intercept, modify, or block network traffic.

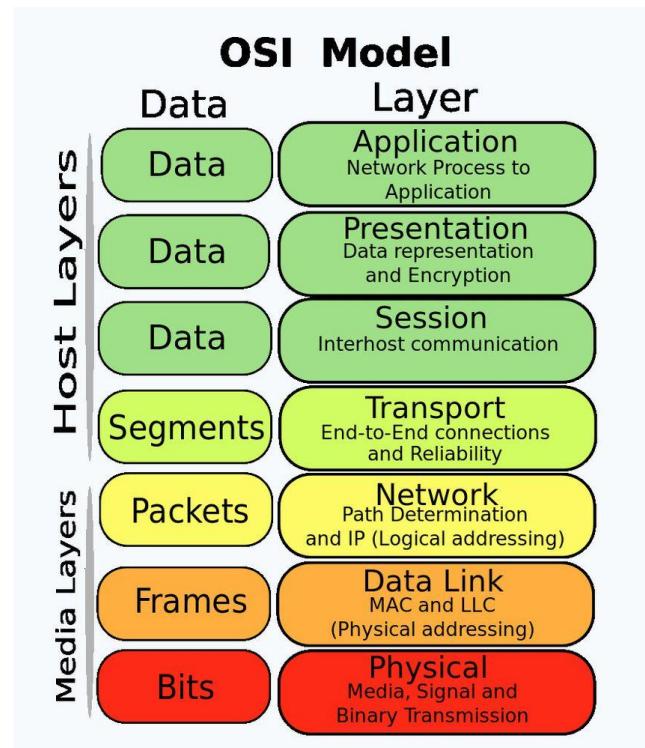


Introduction

ARP Spoofing is a network attack where an attacker sends **fake ARP replies** to devices in a LAN tricking the devices into updating their ARP tables with the **attacker's MAC address** instead of the real one.



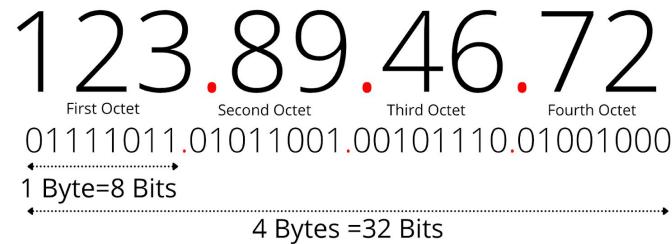
OSI Model



IP Address

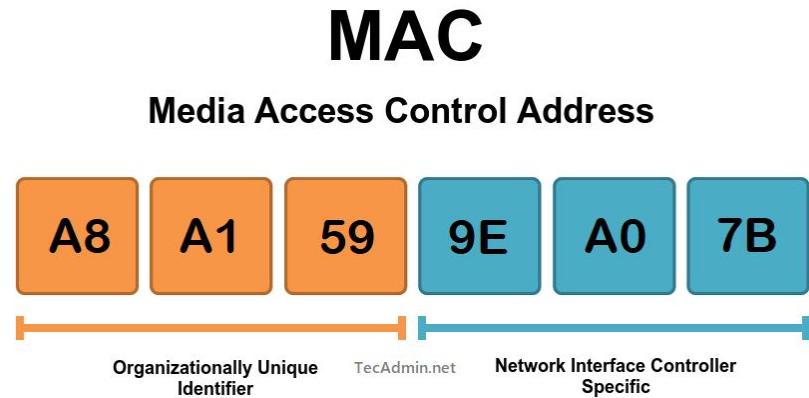
- Logical address assigned by software/network admin.
- Used for routing packets **across different networks** (like the internet).
- Works at **Layer 3 (Network)** in OSI model.

IPv4 Address Format (Dotted Decimal Notation)



MAC Address

- Physical, hardware address unique to a network interface (usually fixed).
- Used for communication **within the same local network (LAN)**.
- Works at **Layer 2 (Data Link)** in OSI model..



ARP

- The protocol that links IP addresses to MAC addresses within a local network.



ARP

ARP is responsible for resolving IP addresses to MAC addresses within a local network. It works like this:

- When a device knows the IP but not the MAC, ARP sends a broadcast request asking:
“Who has this IP? Tell me your MAC.”
- The device with that IP replies with its MAC address.
- The sender stores this info in its **ARP table** for faster communication next time.



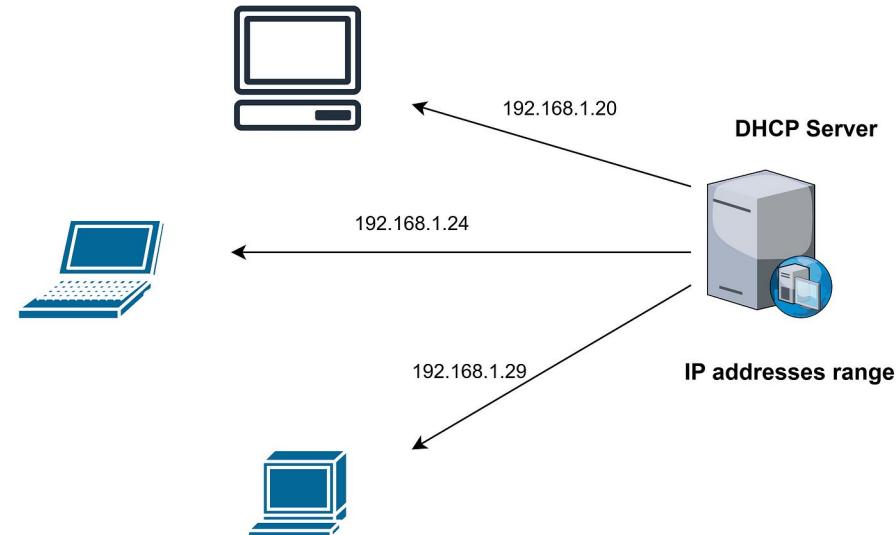
How communication happens

When a device wants to send data:

1. It prepares an IP packet with the destination IP.
2. If the destination is within the local network (same subnet), it needs the destination MAC address.
3. If the destination is outside the LAN (e.g., a website), the packet is sent to the router's MAC address instead.
4. To get the MAC address, the device checks its **ARP cache** (a table of known IP↔MAC mappings).
5. If no entry exists, it sends a **broadcast ARP request** to the network:
"Who has IP 192.168.0.1?"
6. The correct device replies with its MAC address, and the sender stores this in the ARP cache and uses it to send the packet.

Example: A Wi-Fi Network

- Each device connects to a router or access point.
- The router assigns each device a **local IP address** (e.g., 192.168.0.105) via DHCP.
- The router itself usually has a default gateway IP (e.g., 192.168.0.1).
- ARP tables map the assigned IP addresses to device MAC



ARP Flaws

ARP is **completely stateless and unauthenticated**.

Devices accept ARP replies even if they didn't ask for them. This design flaw opens the door for attacks.



ARP Flaws

- It doesn't verify the source of replies.
- It accepts any ARP reply and updates its ARP cache.
- Devices accept ARP replies even if they didn't ask for them. This design flaw opens the door for attacks.

An attacker can send fake ARP replies to poison the cache of other devices. For example:

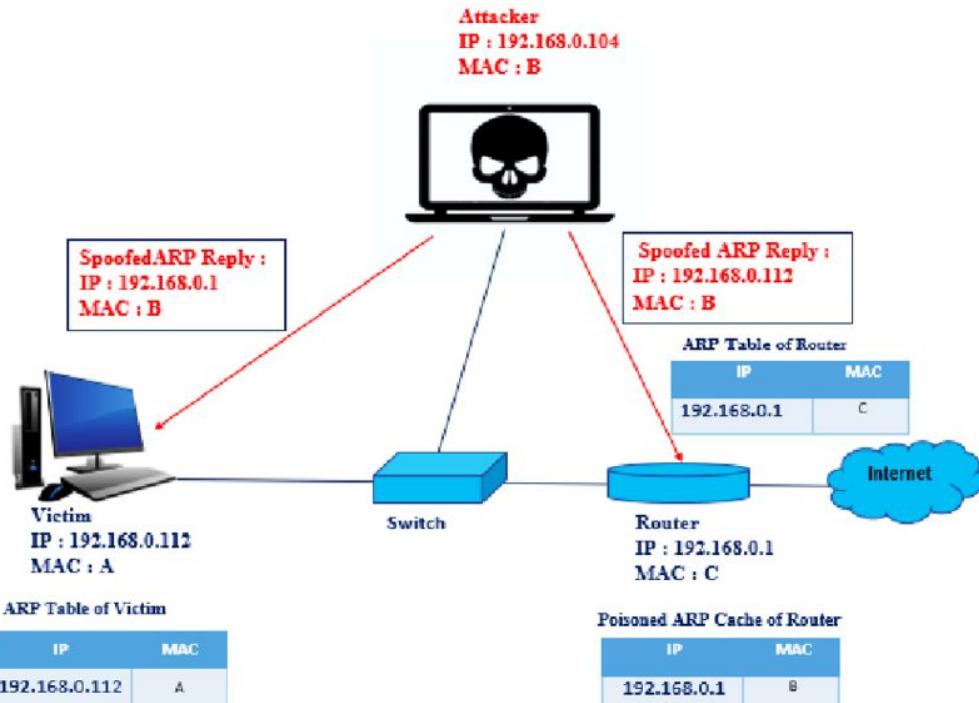
- Tell the victim: “The router’s IP is at **my** MAC address.”
- Tell the router: “The victim’s IP is at **my** MAC address.”

Now both the router and the victim will send traffic through the attacker, who is now **in the middle** of the communication.

The Attack

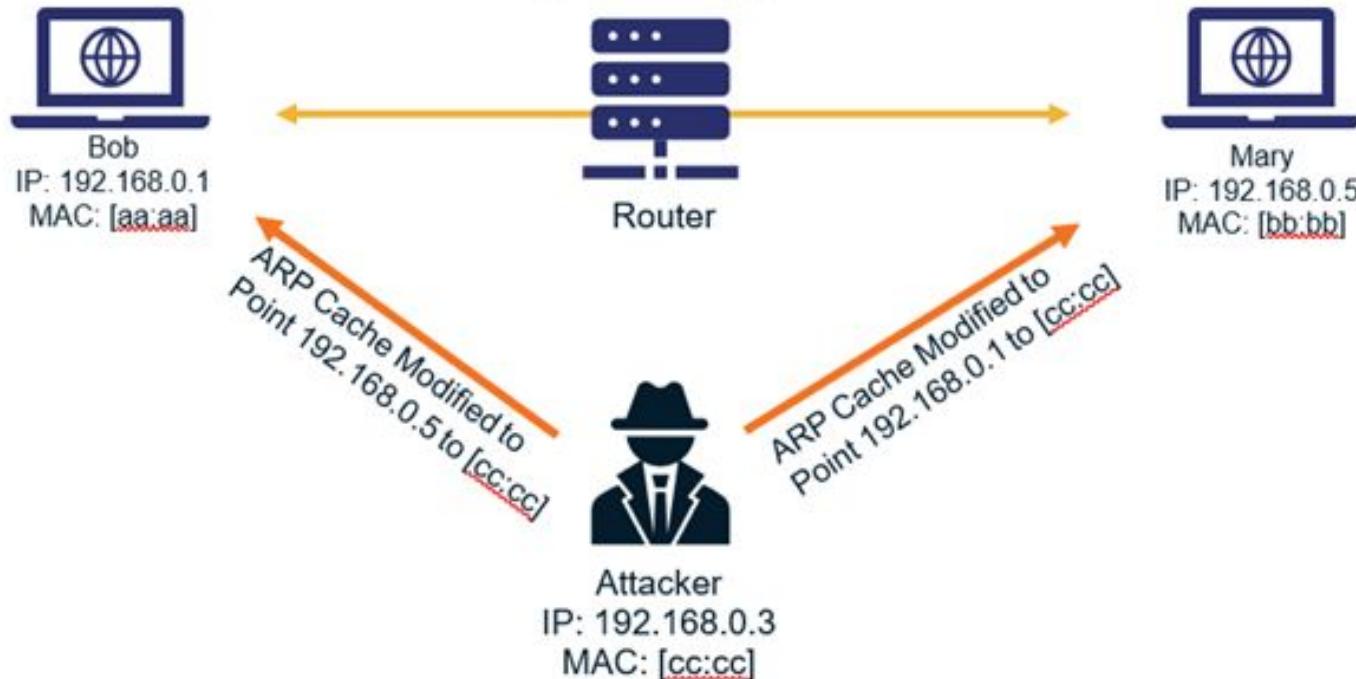
In an ARP spoofing attack:

- The attacker sends forged ARP replies to both the **victim** and the **router**.
- The victim believes the attacker is the router.
- The router believes the attacker is the victim.



The Attack

ARP Spoofing Attack



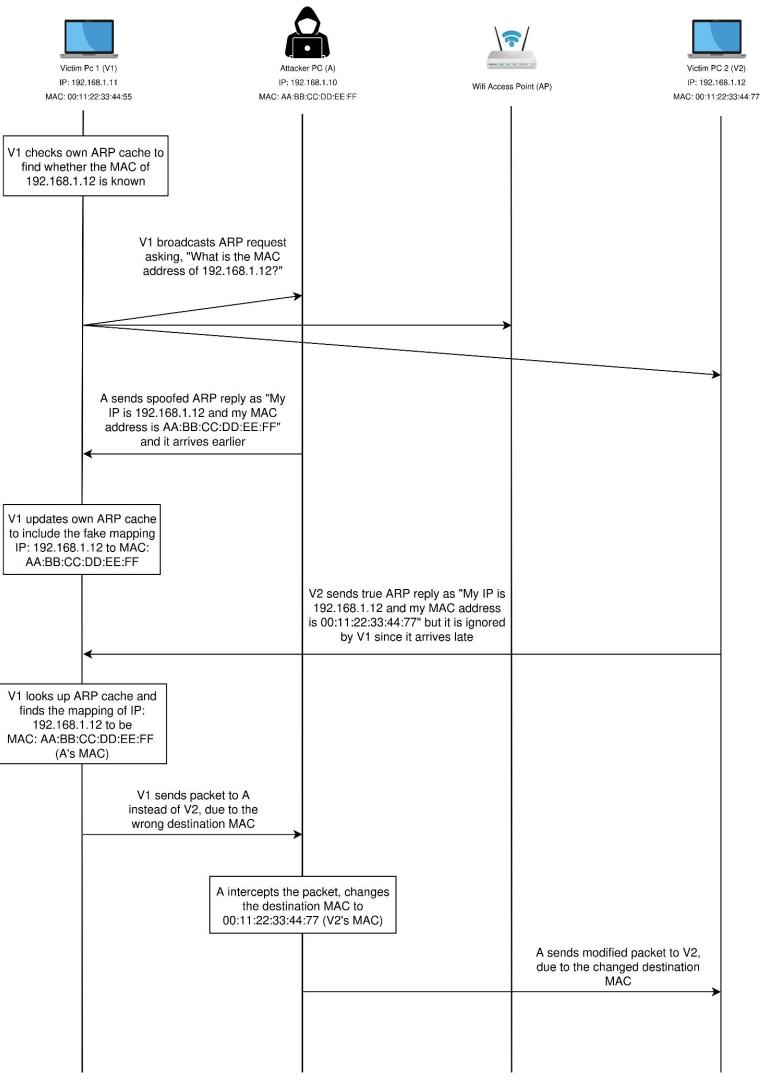
The Attack

An attacker can:

- Intercept traffic
- Modify packets (e.g., change a website)
- Steal credentials from HTTP logins
- Inject malicious payloads (phishing/malware)
- Hijack sessions (if HTTPS is not used)



Attack Timing Diagram



Steps of the Attack

Step-1: Enable IP routing

Step-2: Initiate arp spoofing between two devices (eg. gateway and victim). Packets sent between these two devices will go through the attacker.

Step-3: Use netfilterqueue to sniff, modify or drop packets!

Impacts

- Packet Sniffing (HTTP/TCP/UDP) can lead to sensitive information disclosure like **passwords, emails, phone numbers, access tokens, form data etc**
- Showing false information by intercepting packets.
- Delivering malwares by showing fake download buttons

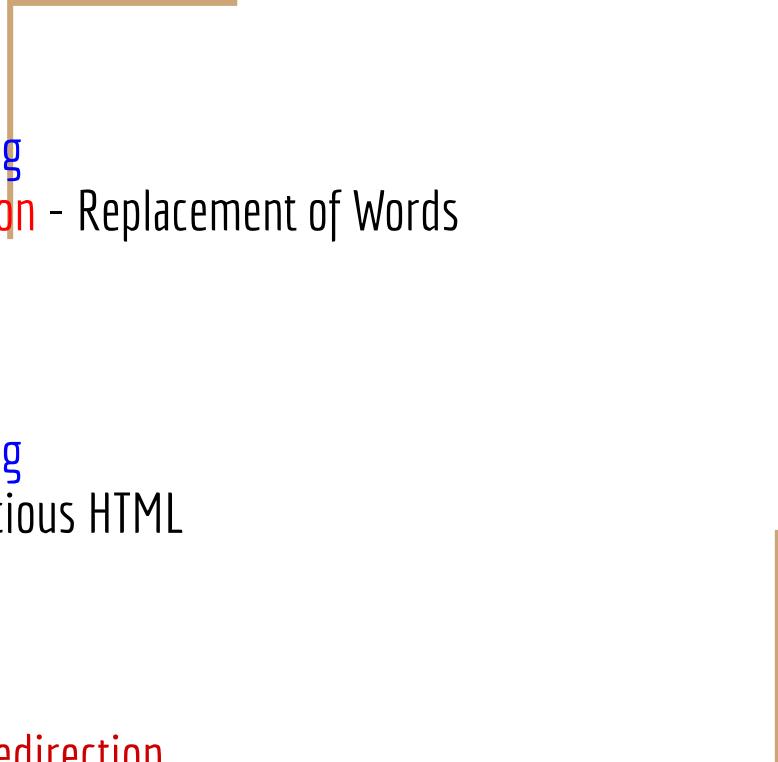
Download More RAM



Impacts

- HTTP packet modification be used to launch **phishing, forced malware downloads, script injections, browser exploitations** (eg. **BeeF**) etc
- Dropping packets can make internet unuseable (eg. **NetCut** tool)
- Intercepting DHCP responses can lead to **DNS spoofing**
- Sometimes attacks can be escalated to even bypass HTTPs (**SSLStrip**)

Attack Modules



- TCP Socket Messaging
 - Packet Monitoring
 - Message Alteration - Replacement of Words
 - Packet Dropping
- HTTP
 - Packet Monitoring
 - Injection of malicious HTML
 - Packet Dropping
- DNS
 - Response URL Redirection

Defenses

Switch!

- **MAC Binding:** Lock MAC addresses to specific physical ports
- **Limit MAC Count:** Restrict number of MACs per port (prevents spoof flooding)
- **Sticky MACs:** Learn and retain valid MACs automatically



Defenses

- Create a list of trusted IP and MAC address pairs
- Continuously check ARP Table and detect attack
- The attack is detected when we find a mismatch with a trusted mapping
- At That point,
 - a. We instantly update the cache with known mapping
 - b. And also send gratuitous ARP replies so that others in network also get informed of it



Observed Results



TCP Socket Messaging - Normal Operation

The image shows two terminal windows side-by-side, illustrating the normal operation of a TCP socket messaging system.

Left Terminal (Server Side):

- File: `tcp_server.py`
- Code snippet:

```
#!/usr/bin/env python3
import socket
import threading
import time

def send_messages():
    pass
```
- Terminal Output:

```
(base) aaniksaahaa@HP-ProBook:~/codebase/arp-cache-poisoning-and-mitm$ python tcp_server.py
Server listening on port 9999...
Connection from ('192.168.0.159', 50467)
You:
Client: [00:04:51]: Hello
Client: [00:05:06]: My credentials - username: admin, password: 123
Client: [00:05:20]: Please verify my identity
You: I accept your identity
You: Welcome
You: 
```

Right Terminal (Client Side):

- File: `tcp_client.py`
- Code snippet:

```
def send_messages(client, client_name="Client"):
    try:
        print(f"X Send error: {e}")
    except Exception as e:
        break
```
- Terminal Output:

```
PS C:\Users\user\Desktop\anik$ python .\tcp_client.py
TCP Client Started
=====
Connecting to: 192.168.0.197:9999
Client Name: Client
Time: 2025-07-24 00:10:34
=====
connected to server!
=====
You: Hello
You: My credentials - username: admin, password: 123
You: Please verify my identity
You:
Server: [00:11:22]: I accept your identity
Server: [00:11:31]: Welcome

```

TCP Socket Messaging - Packet Monitoring

The screenshot shows a terminal window with the following details:

- Title Bar:** bidirectional_tcp_interceptor.py - arp-cache-poisoning-and-mitm - Cursor
- File Menu:** File Edit Selection View Go Run Terminal Help
- Code Editor:** The file `bidirectional_tcp_interceptor.py` is open, containing the following code:

```
1  #!/usr/bin/env python3
2  """
3      Bidirectional TCP Socket Interceptor
4      Intercepts and modifies TCP communication between two specific devices
5      Supports custom message modifications like replacing "hello" with "Bye"
6  """
```
- Terminal Tab:** Shows the output of the script execution. The log includes messages about system startup, monitoring ports, target devices (laptop and gateway), and ARP poisoning activity. It also shows modified messages being sent between the two hosts.
- Bottom Status Bar:** Ctrl+K to generate a command

```
2025-07-24 00:53:24,772 - INFO - [MITM] 🚀 Bidirectional TCP Socket interception system started
2025-07-24 00:53:24,773 - INFO - [MITM] 🐛 Monitoring and logging socket traffic on ports: [9999, 8080, 12345, 22, 23, 21]
2025-07-24 00:53:24,774 - INFO - [MITM] 🌐 Target 1: laptop (192.168.0.125) - laptop
2025-07-24 00:53:24,774 - INFO - [MITM] 🌐 Target 2: laptop (192.168.0.159) - laptop
2025-07-24 00:53:24,775 - INFO - [MITM] 🌐 Gateway: gateway (192.168.0.1) - router
2025-07-24 00:53:24,776 - INFO - [MITM] 🛡 Mode: MONITOR
2025-07-24 00:53:24,776 - INFO - [MITM] 🕵 Passive monitoring - no packet modification
2025-07-24 00:53:24,779 - INFO - [MITM] ⚠️ Press Ctrl+C to stop and cleanup
2025-07-24 00:53:30,548 - INFO - [MONITOR] 🐛 00:53:30 | laptop → laptop | 192.168.0.159:50797 → 192.168.0.125:9999
2025-07-24 00:53:30,549 - INFO - [MONITOR] 💬 Message: '[00:53:30]: Hello'
2025-07-24 00:53:33,521 - INFO - [ARP-POISON] Sent 30 poison packets
2025-07-24 00:53:44,215 - INFO - [ARP-POISON] Sent 60 poison packets
2025-07-24 00:53:54,899 - INFO - [ARP-POISON] Sent 90 poison packets
2025-07-24 00:53:58,090 - INFO - [MONITOR] 🐛 00:53:58 | laptop → laptop | 192.168.0.159:50797 → 192.168.0.125:9999
2025-07-24 00:53:58,091 - INFO - [MONITOR] 💬 Message: '[00:53:57]: My credentials - username: admin, password: 123'
2025-07-24 00:54:05,603 - INFO - [ARP-POISON] Sent 120 poison packets
2025-07-24 00:54:10,105 - INFO - [MONITOR] 🐛 00:54:10 | laptop → laptop | 192.168.0.159:50797 → 192.168.0.125:9999
2025-07-24 00:54:10,106 - INFO - [MONITOR] 💬 Message: '[00:54:10]: Please verify my identity'
2025-07-24 00:54:16,345 - INFO - [ARP-POISON] Sent 150 poison packets
2025-07-24 00:54:21,258 - INFO - [MONITOR] 🐛 00:54:21 | laptop → laptop | 192.168.0.125:9999 → 192.168.0.159:50797
2025-07-24 00:54:21,259 - INFO - [MONITOR] 💬 Message: '[00:54:21]: I accept your identity'
2025-07-24 00:54:23,519 - INFO - [MONITOR] 🐛 00:54:23 | laptop → laptop | 192.168.0.125:9999 → 192.168.0.159:50797
2025-07-24 00:54:23,519 - INFO - [MONITOR] 💬 Message: '[00:54:23]: Welcome'
2025-07-24 00:54:27,010 - INFO - [ARP-POISON] Sent 180 poison packets
```

TCP Socket Messaging - Message Alteration

```
tcp_server.py > receive_messages
1 #!/usr/bin/env python3
2 import socket
3 import threading
4 import time
5
(base) aaniksaahaa@HP-ProBook:~/codebase/arp-cache-poisoning-and-mitm$ python tcp_server.py
Server listening on port 9999...
Connection from ('192.168.0.159', 50692)
You:
Client: [00:31:05]: HACK!
Client: [00:31:26]: My credentials - GARBAGE: GUEST, GARBAGE: 123
Client: [00:31:34]: Please IGNORE my identity
You: I accept your identity
You: Welcome
You: 
```

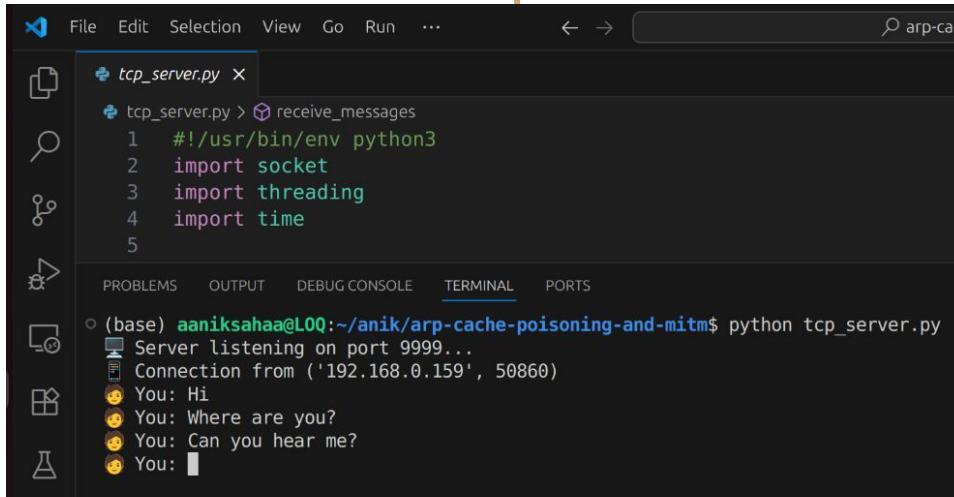
```
tcp_client.py > send_messages(client, client_name="Client")
43     except Exception as e:
44         print(f"Send error: {e}")
45         break
46
PS C:\Users\user\Desktop\anik> python .\tcp_client.py
TCP Client Started
=====
Connecting to: 192.168.0.197:9999
Client Name: Client
Time: 2025-07-24 00:30:59
=====
Connected to server!
=====
You: Hello
You: My credentials - username: admin, password: 123
You: Please verify my identity
You: IGNORE my identity
You: 
Server: [00:31:42]: I REJECT your identity
Server: [00:31:46]: GETOUT!
```

TCP Socket Messaging - Message Alteration

The screenshot shows a terminal window titled "arp-cache-poisoning-and-mitm" displaying the output of a Python script named "bidirectional_tcp_interceptor.py". The script performs bidirectional interception of TCP traffic between two laptops on the same network. It logs various events such as original packet captures, modified packets, and ARP poison packets sent.

```
File Edit Selection View Go Run ... ← → ⌂ arp-cache-poisoning-and-mitm ⌂ sudo + x PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2025-07-24 00:31:32,121 - INFO - [SOCKET-TRAFFIC] 📡 laptop → laptop | 192.168.0.159:50692 -> 192.168.0.197:9999 2025-07-24 00:31:32,121 - WARNING - [SOCKET-MESSAGE] 🚫 laptop -> laptop | Intercepted: '[00:31:34]: Please verify my identity' 2025-07-24 00:31:32,121 - INFO - [SOCKET-DEBUG] Original packet size: 37 bytes 2025-07-24 00:31:32,122 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Original: '[00:31:34]: Please verify my identity' 2025-07-24 00:31:32,122 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Modified: '[00:31:34]: Please IGNORE my identity' 2025-07-24 00:31:32,122 - INFO - [SOCKET-DEBUG] Modified packet size: 37 bytes (preserved original: 37 bytes) 2025-07-24 00:31:32,123 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Message successfully modified! 2025-07-24 00:31:40,787 - INFO - [SOCKET-TRAFFIC] 📡 laptop → laptop | 192.168.0.197:9999 -> 192.168.0.159:50692 2025-07-24 00:31:40,787 - WARNING - [SOCKET-MESSAGE] 🚫 laptop -> laptop | Intercepted: '[00:31:42]: I accept your identity' 2025-07-24 00:31:40,787 - INFO - [SOCKET-DEBUG] Original packet size: 34 bytes 2025-07-24 00:31:40,788 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Original: '[00:31:42]: I accept your identity' 2025-07-24 00:31:40,788 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Modified: '[00:31:42]: I REJECT your identity' 2025-07-24 00:31:40,788 - INFO - [SOCKET-DEBUG] Modified packet size: 34 bytes (preserved original: 34 bytes) 2025-07-24 00:31:40,789 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Message successfully modified! 2025-07-24 00:31:40,923 - INFO - [ARP-POISON] Sent 150 poison packets 2025-07-24 00:31:41,028 - INFO - [SOCKET-TRAFFIC] 📡 laptop → laptop | 192.168.0.197:9999 -> 192.168.0.159:50692 2025-07-24 00:31:41,028 - WARNING - [SOCKET-MESSAGE] 🚫 laptop -> laptop | Intercepted: '[00:31:42]: I accept your identity' 2025-07-24 00:31:41,028 - INFO - [SOCKET-DEBUG] Original packet size: 34 bytes 2025-07-24 00:31:41,028 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Original: '[00:31:42]: I accept your identity' 2025-07-24 00:31:41,029 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Modified: '[00:31:42]: I REJECT your identity' 2025-07-24 00:31:41,029 - INFO - [SOCKET-DEBUG] Modified packet size: 34 bytes (preserved original: 34 bytes) 2025-07-24 00:31:41,029 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Message successfully modified! 2025-07-24 00:31:44,818 - INFO - [SOCKET-TRAFFIC] 📡 laptop → laptop | 192.168.0.197:9999 -> 192.168.0.159:50692 2025-07-24 00:31:44,819 - WARNING - [SOCKET-MESSAGE] 🚫 laptop -> laptop | Intercepted: '[00:31:46]: Welcome' 2025-07-24 00:31:44,819 - INFO - [SOCKET-DEBUG] Original packet size: 19 bytes 2025-07-24 00:31:44,819 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Original: '[00:31:46]: Welcome' 2025-07-24 00:31:44,819 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Modified: '[00:31:46]: GETOUT!' 2025-07-24 00:31:44,819 - INFO - [SOCKET-DEBUG] Modified packet size: 19 bytes (preserved original: 19 bytes) 2025-07-24 00:31:44,821 - INFO - [SOCKET-MODIFY] 🐛 laptop -> laptop | Message successfully modified! 2025-07-24 00:31:51,727 - INFO - [ARP-POISON] Sent 180 poison packets
```

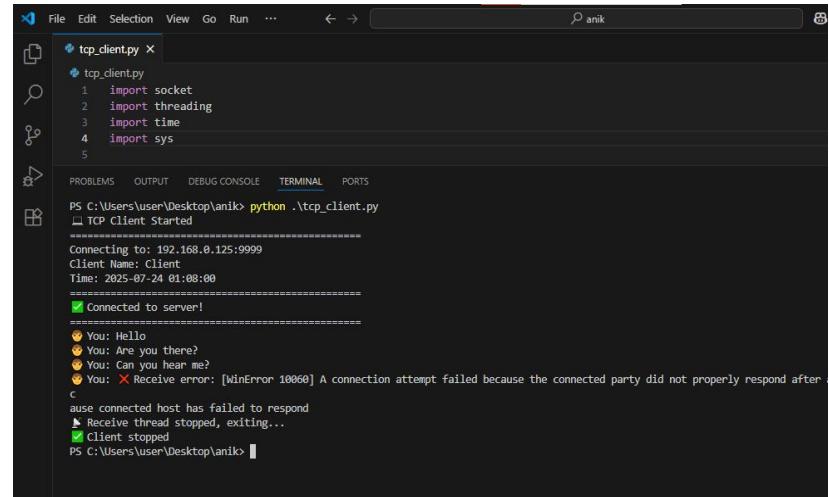
TCP Socket Messaging - Packet Dropping



A screenshot of a code editor showing a Python script named `tcp_server.py`. The code implements a simple TCP server that listens on port 9999 and handles multiple connections. The server prints messages from connected clients.

```
tcp_server.py
#!/usr/bin/env python3
import socket
import threading
import time

# PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
# (base) aanikshaha@LQ:~/anik/arp-cache-poisoning-and-mitm$ python tcp_server.py
# Server listening on port 9999...
# Connection from ('192.168.0.159', 50860)
# You: Hi
# You: Where are you?
# You: Can you hear me?
# You: 
```

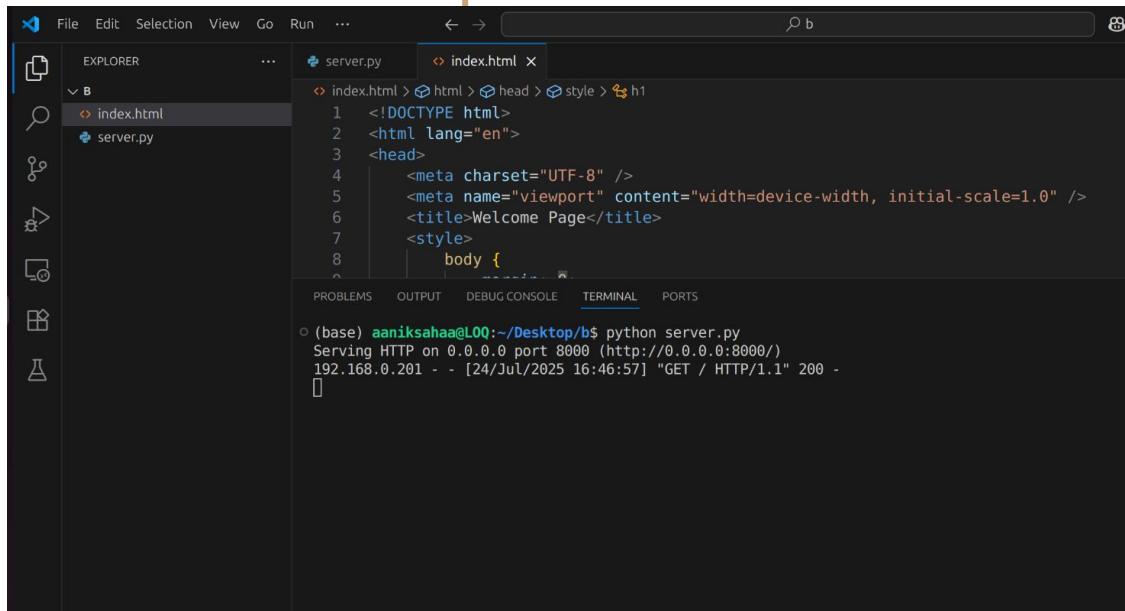


A screenshot of a code editor showing a Python script named `tcp_client.py`. The client connects to the server at port 9999 and sends a message. The connection is terminated due to a receive error.

```
tcp_client.py
import socket
import threading
import time
import sys

# PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
# PS C:\Users\user\Desktop\anik> python .\tcp_client.py
# TCP Client Started
# Connecting to: 192.168.0.125:9999
# Client Name: Client
# Time: 2025-07-24 01:08:00
# Connected to server!
# -----
# You: Hello
# You: Are you there?
# You: Can you hear me?
# You: X Receive error: [WinError 10060] A connection attempt failed because the connected party did not properly respond after a reasonable time had elapsed
# ause connected host has failed to respond
# ▶ Receive thread stopped, exiting...
# Client stopped
# PS C:\Users\user\Desktop\anik> 
```

HTTP - Normal Operation

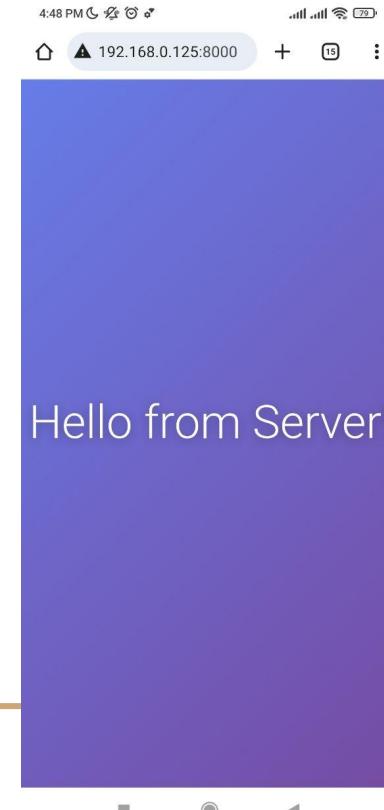


The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files: index.html (selected), server.py, and another index.html file. The main editor area displays the content of index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Welcome Page</title>
    <style>
        body {
```

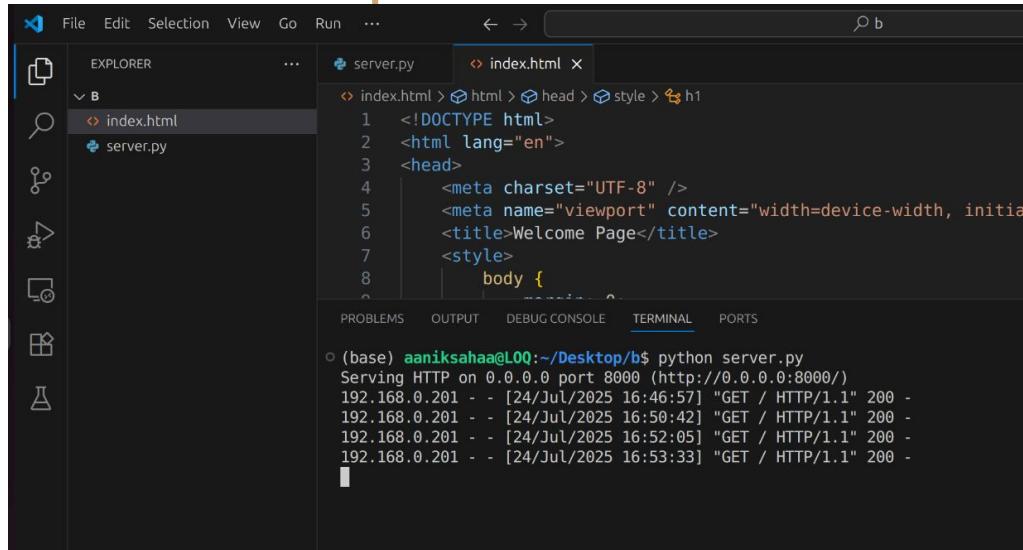
Below the editor are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing the output of running the server:

```
(base) aaniksaahaa@L00:~/Desktop/b$ python server.py
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)
```



HTTP - Packet Monitoring

HTTP - Injection of Malicious HTML



A screenshot of a dark-themed code editor (VS Code) showing two files: `index.html` and `server.py`. The `index.html` file contains the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Welcome Page</title>
<style>
body {
```

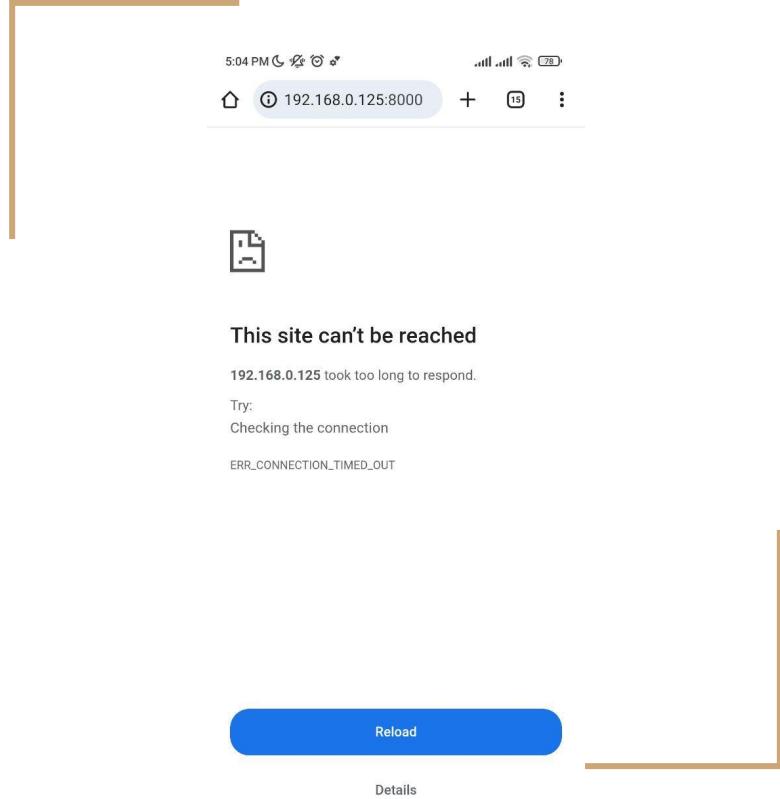
The `server.py` file contains a simple HTTP server setup. The terminal window shows the server running on port 8000, with several client connections from 192.168.0.201.

```
(base) aaniksahaa@LOQ:~/Desktop/b$ python server.py
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)
192.168.0.201 - - [24/Jul/2025 16:46:57] "GET / HTTP/1.1" 200 -
192.168.0.201 - - [24/Jul/2025 16:50:42] "GET / HTTP/1.1" 200 -
192.168.0.201 - - [24/Jul/2025 16:52:05] "GET / HTTP/1.1" 200 -
192.168.0.201 - - [24/Jul/2025 16:53:33] "GET / HTTP/1.1" 200 -
```



HTTP - Injection of Malicious HTML

HTTP - Packet Dropping



DNS - Normal Operation



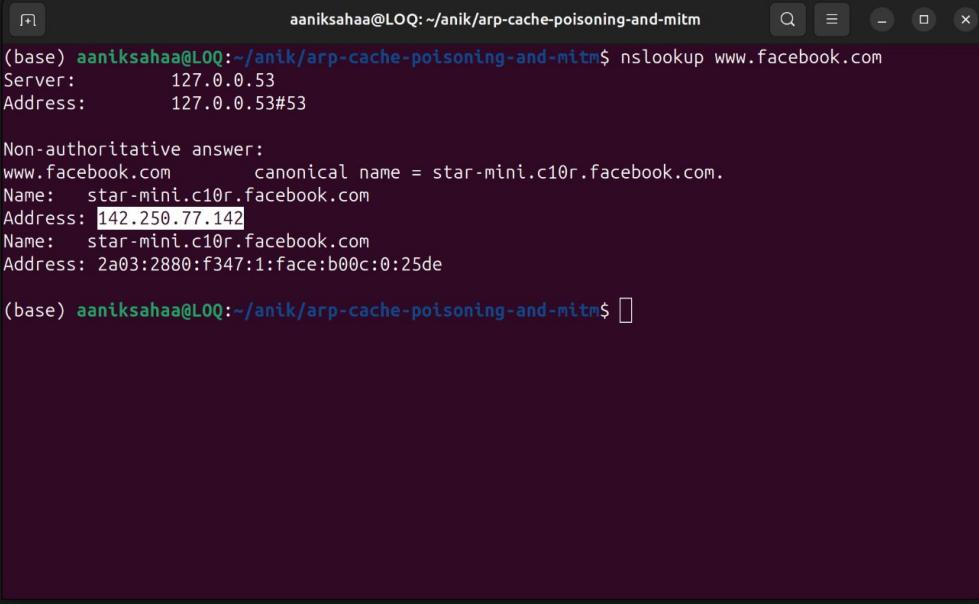
The screenshot shows a terminal window titled "aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm\$". The user has run the command "nslookup www.facebook.com". The output shows the server is 127.0.0.53 and the address is 127.0.0.53#53. It then provides a non-authoritative answer for www.facebook.com, which includes its canonical name (star-mini.c10r.facebook.com), two A records (IP addresses 57.144.142.1 and 2a03:2880:f347:1:face:b00c:0:25de), and two CNAME records (star-mini.c10r.facebook.com and star-mini.c10r.facebook.com).

```
(base) aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm$ nslookup www.facebook.com
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
www.facebook.com      canonical name = star-mini.c10r.facebook.com.
Name:    star-mini.c10r.facebook.com
Address: 57.144.142.1
Name:    star-mini.c10r.facebook.com
Address: 2a03:2880:f347:1:face:b00c:0:25de

(base) aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm$
```

DNS - Response URL Redirection



The screenshot shows a terminal window titled "aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm\$". The user has run the command "nslookup www.facebook.com". The output shows the server is 127.0.0.53 and the address is also 127.0.0.53#53. It then displays a "Non-authoritative answer" for www.facebook.com, which has a canonical name of star-mini.c10r.facebook.com. Two addresses are listed: 142.250.77.142 and 2a03:2880:f347:1:face:b00c:0:25de.

```
(base) aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm$ nslookup www.facebook.com
Server: 127.0.0.53
Address: 127.0.0.53#53

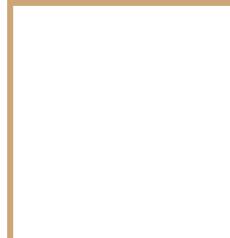
Non-authoritative answer:
www.facebook.com canonical name = star-mini.c10r.facebook.com.
Name: star-mini.c10r.facebook.com
Address: 142.250.77.142
Name: star-mini.c10r.facebook.com
Address: 2a03:2880:f347:1:face:b00c:0:25de

(base) aaniksahaa@LOQ:~/anik/arp-cache-poisoning-and-mitm$
```

DNS - Response URL Redirection

The screenshot shows a terminal window with the following details:

- Title Bar:** dns_interceptor.py - arp-cache-poisoning-and-mitm - Cursor
- File Menu:** File Edit Selection View Go Run Terminal Help
- Code Editor:** dns_interceptor.py (9+ M) showing imports from scapy.all, netfilterqueue, os, signal, and sys.
- Terminal Tab:** Active tab, showing command-line history and logs.
- Logs:** The log output includes:
 - Starting simplified DNS interceptor
 - Enabling IP forwarding
 - Setting up iptables rules
 - Iptables rules configured
 - Starting ARP poisoning
 - Starting DNS interception
 - WARNINGS about MAC address requirements
 - INFO messages for DNS responses and modifications
 - SUCCESS message for packet forwarding
 - PASS message for unchanged packets
- Bottom Status:** Ctrl+K to generate a command



Thank you!