IP Spoofing and Session Hijacking Defense Report

Practical Demonstration of IP Spoofing and Session Hijacking Defense Mechanisms using open Source Tools

Nov 2024

Table of Contents

Executive Summary

This project aims to demonstrate practical methods for defending against IP spoofing and session hijacking attacks in a simulated environment. These two attack vectors are significant threats in network security, where attackers attempt to impersonate legitimate users or take over established sessions between users and servers. The goal is to explore how these attacks work, monitor them in real-time, and apply countermeasures using open-source tools like Wireshark, Snort, iptables, hping3 and Ettercap.

Objectives:

- Simulate **IP spoofing** and **SYN flood attacks** using hping3.
- Detect and prevent unauthorized session hijacking attempts.
- Monitor and log network activities with **Wireshark** and **Snort**.
- Block malicious traffic using **iptables**.

Expected Outcomes:

- Successful detection and mitigation of **IP spoofing** and **SYN flood attacks**.
- Real-time alerts and logging using Wireshark and Snort.
- Blocking of malicious traffic with **iptables**.
- A practical demonstration of **session hijacking defense**.

Scope of the project

- **Project Objectives:**
    - Simulate IP Spoofing and Session Hijacking attacks in a virtual environment.
    - Deploy open-source tools to detect, analyze, and mitigate these attacks.
    - Document attack patterns, responses, and defense efficacy.
    - Provide a practical understanding of how network security measures are applied to prevent unauthorized access.
- **Scope Limitations:**
    - Focus is limited to open-source tools, excluding proprietary software.
    - Testing will occur in a virtual environment, not a production-level deployment.
    - Demonstration and testing are limited to TCP/IP-based spoofing and session hijacking attacks.

Team Structure

- **Team Lead:** Aarav Saha
- **Team Members:**

1. **Prashant Kumar**
2. **Ayush Singh**
3. **Sachin Chauhan**
4. **Yash Kumar**
5. **Vishal Bansa**l
6. **Hritik Yadav**
7. **Aditya Sharma**
8. **Ritika Katiyar**

Project Timeline

- **Key Phases:**
  - **Phase 1**: Project Planning and Research
    - Define objectives, gather resources, assign roles.
  - **Phase 2**: Virtual Environment Setup
    - Set up and configure Ubuntu server, user, and attacker machine
  - **Phase 3**: Attack Simulation
    - Simulate IP spoofing and session hijacking attacks.
  - **Phase 4**: Defense Implementation
    - Configure Wireshark, Snort, iptables and arptables for packet inspection, alerts and blocking rules.
  - **Phase 5**: Testing and Documentation
    - Test defense mechanisms and document results.
  - **Phase 6**: Final Presentation and Report Submission
    - Prepare and submit the final project report and presentation.
- **Milestones:**
  - Completion of network setup and connectivity tests
  - Successful simulation of attack scenarios
  - Deployment of defense mechanisms
  - Final testing and adjustments report

Deliverables

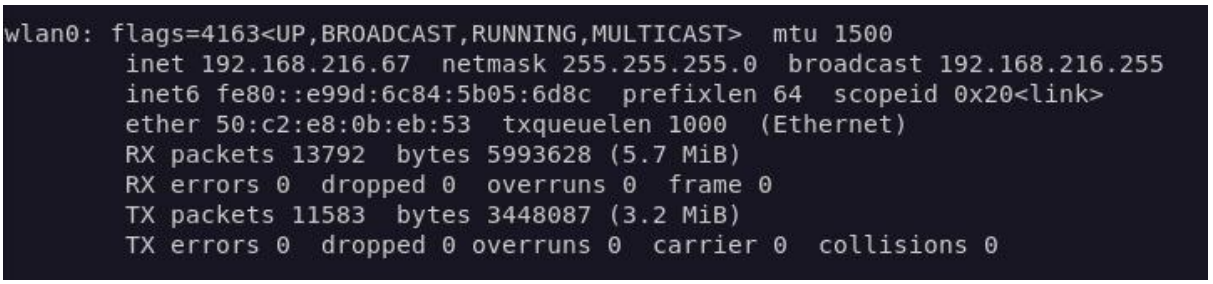
Project Deliverables

1. **Comprehensive Research Report**: An in-depth document detailing the concepts, techniques, and implications of IP spoofing and session hijacking, along with potential defense mechanisms.
2. **Simulation Scripts and Configuration Files**: A collection of scripts and configuration files for simulating IP spoofing and SYN flood attacks using hping3 and Ettercap.
3. **Traffic Monitoring and Analysis Documentation**: Detailed documentation on the setup and configuration of Wireshark and Snort for monitoring network traffic during simulations.
4. **Final Presentation**: A visual presentation summarising the project's objectives, methodologies, simulated outcomes, and recommendations for future work.
5. **Logs and Network Traffic Analysis**: Captured logs and documentation of network traffic during simulations, including identified anomalies and detected threats.

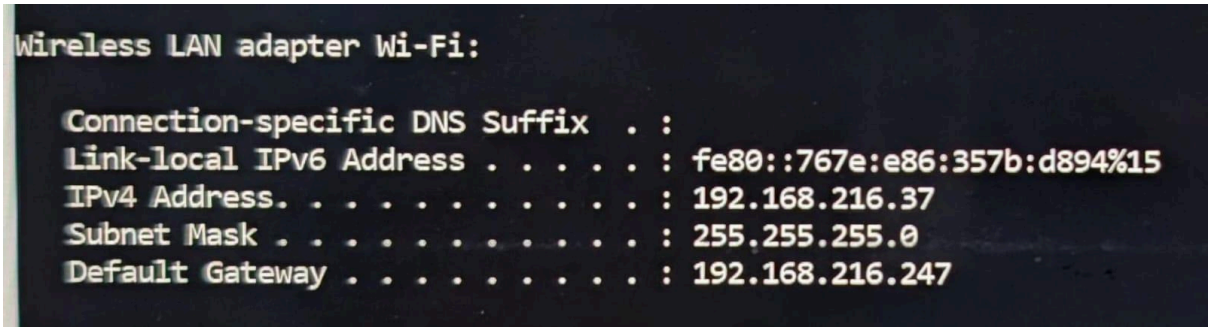
Methodology

Step 1: Set all three machines in the same network to allow communication between them.

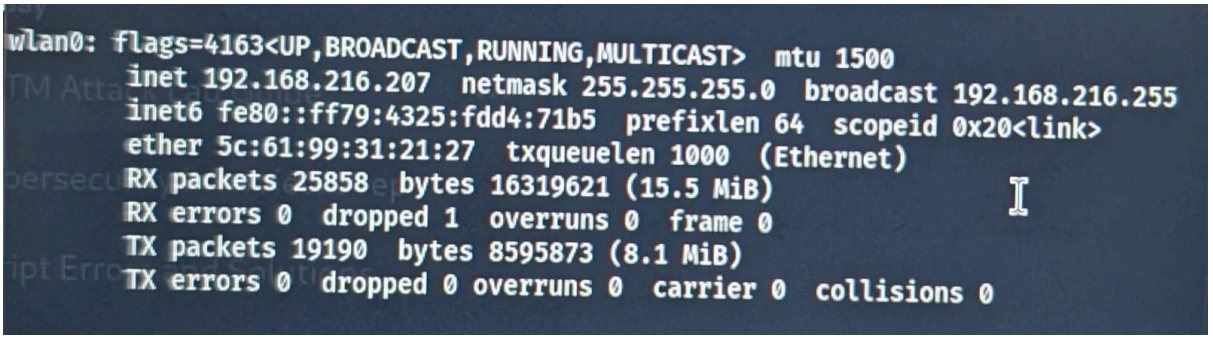
1. **Configure Network Settings**:
    - **Server Machine**: Set Linux as a Server machine, which will act as the target of attacks.
    - **User Machine**: Set Windows machine to simulate a legitimate user.
    - **Attacker Machine:** Using Kali linux machine for attacks.
2. **Configure Network Settings**:
    - Set all three machines in the same network to allow communication between them.
    - All three machine are connected through a mobile hotspot portraying as a router.
    - All machines are connected to each other through router.



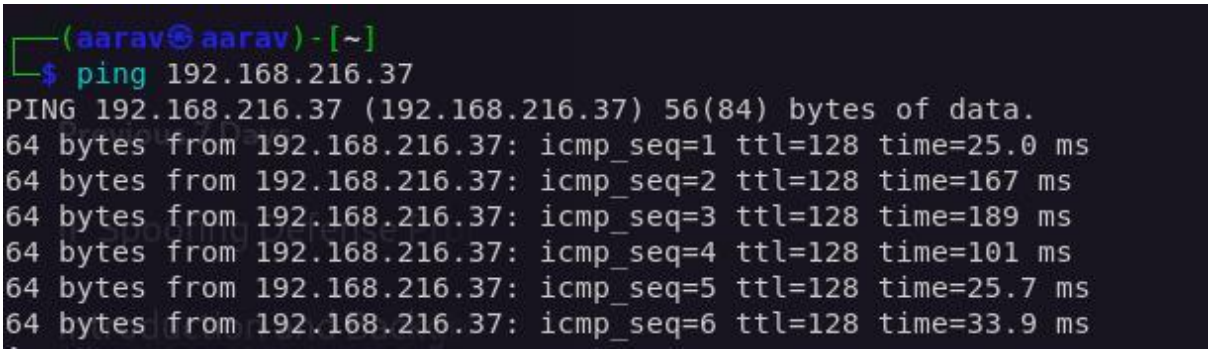
Attacker Machine IP



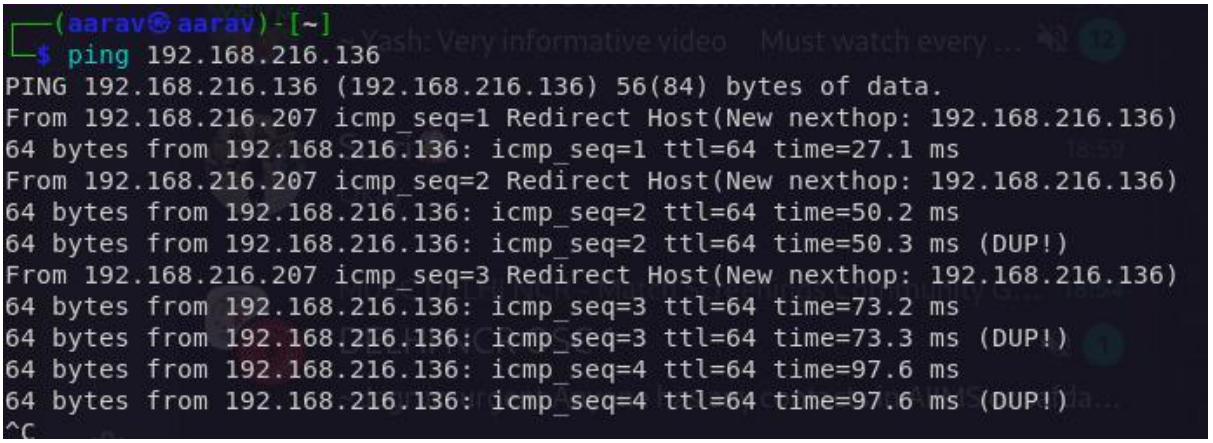
Victim Machine IP



Server Machine IP



This screenshot shows that attacker machine is pinging victim machine.



This screenshot shows attacking machine is pinging server machine.

This verifies all devices are successfully communicating with each other.

Step 2: Install Necessary Tools on Each Machine

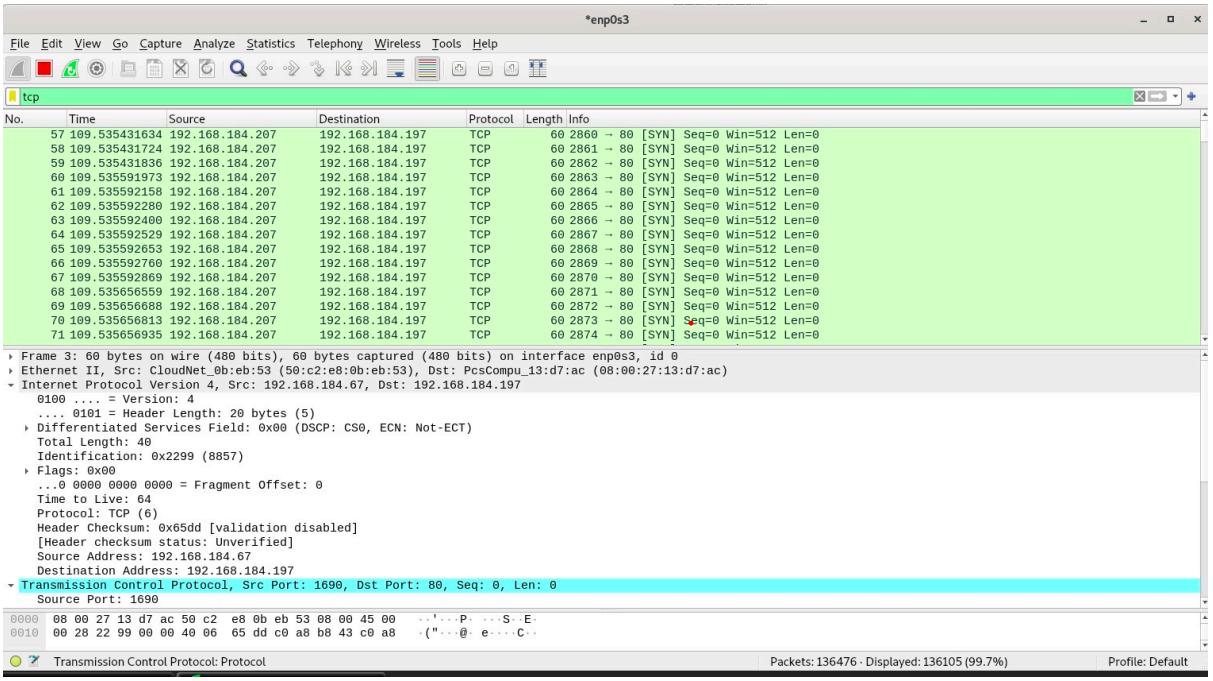
1. **User and Server**:
    - **Wireshark**: Install for packet capturing and network analysis.
    - **Snort**: Set up as an Intrusion Detection System (IDS) to monitor for suspicious activity.
    - **iptables**: Available by default on most Linux distributions for managing firewall rules.

2. **Attacker VM**:
    - **hping3**: Install for generating spoofed packets and SYN flood attacks.
    - **Ettercap**: Install for session hijacking attempts.

Step 3: Simulate IP Spoofing Attack

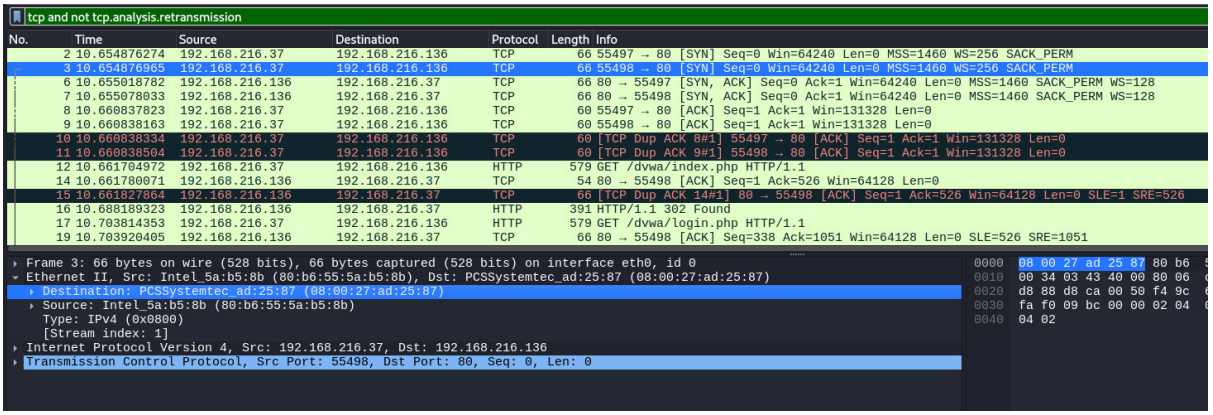
1. **Run hping3 on Attacker machine**:
    - Use hping3 to create packets with a spoofed IP address (the User Machine's IP) aimed at the Server Machine.

    - This sends SYN packets with a fake IP address to the server, simulating an IP Spoofing attack.
2. **Capture Traffic with Wireshark on Server machine**:
    - Start Wireshark and capture packets on the Server VM to observe the incoming spoofed packets.



This is the spoofed IP traffic.

Analyse the IP headers for inconsistencies in source IP.



This is the legitimate tcp traffic from the user.

Step 4: Detect and Block Suspicious Traffic Using Snort and iptables

1. **Configure Snort on the Server VM**:
    - Define custom Snort rules to detect IP Spoofing and high volumes of SYN packets (indicative of SYN floods).

```
remnux@remnux:/var/log/snort$ cat /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures. Put your local
# additions here.

# Detect SYN Flood Attack
alert tcp any any -> any 80 (msg:"Potential SYN Flood Attack"; flags:S; detection_filter:track by_src, count 20, seconds 1; sid:1000001; rev:1;)

# Detect IP Spoofing
alert ip any any -> any any (msg:"IP Spoofing Detected"; ipopts:rr; sid:1000002; rev:1;)

# Detect SYN Flood with Spoofed IPs
alert tcp any any -> any 80 (msg:"Possible SYN Flood with Spoofed IPs"; flags:S; detection_filter:track by_dst, count 50, seconds 1; sid:1000003; rev:1;)
```

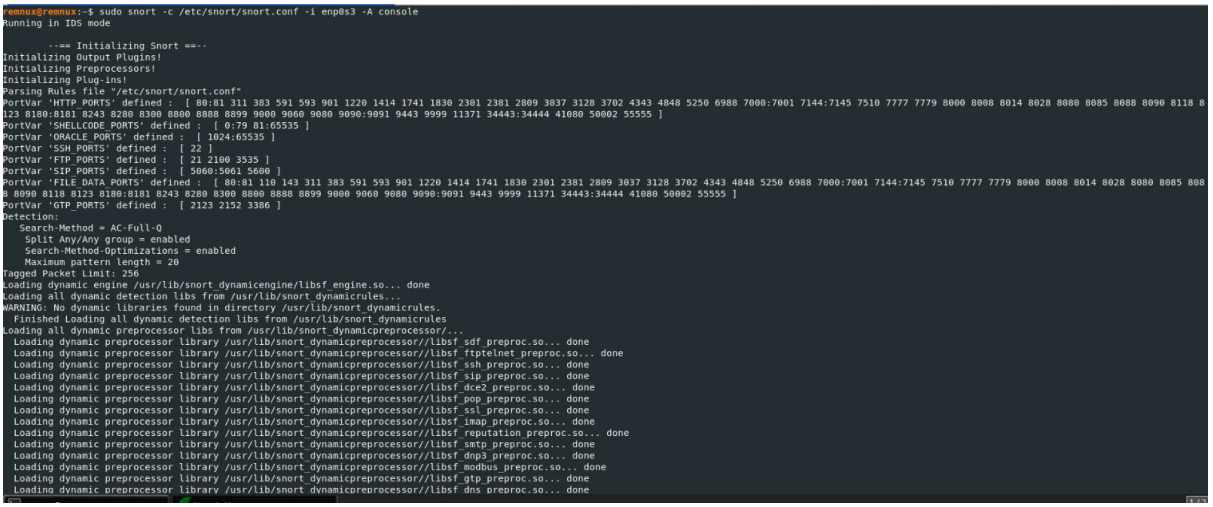
These rules say:

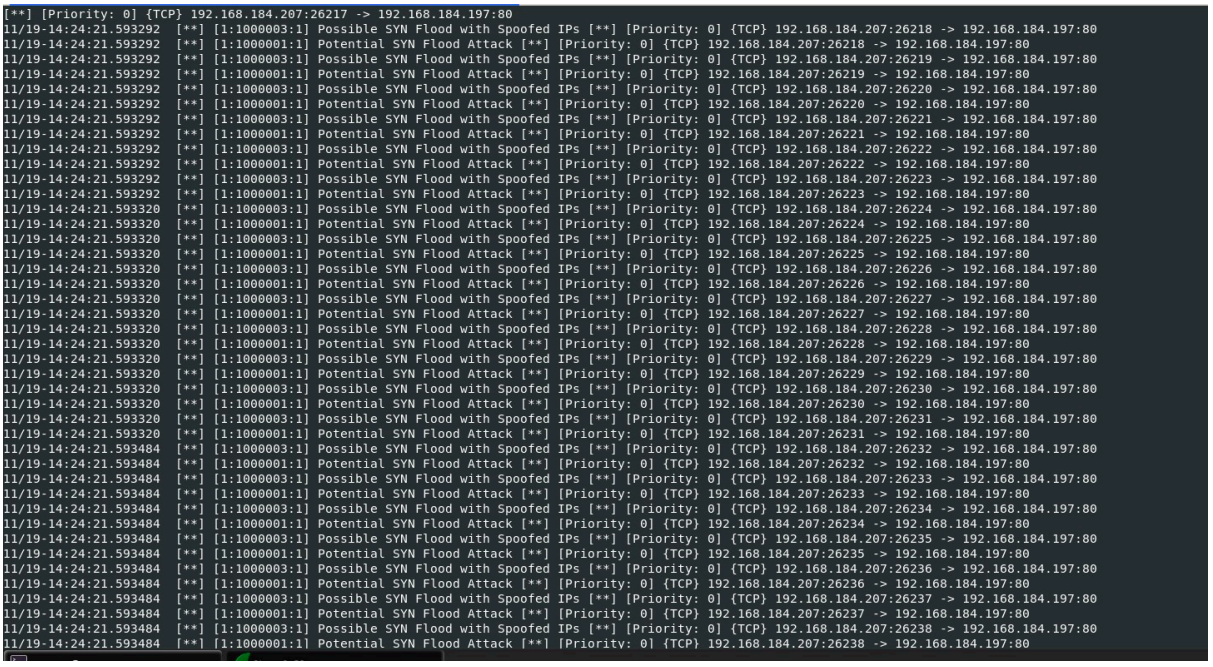
Detect SYN Flood Attacks: Monitors TCP traffic to detect potential SYN flood attacks by flagging TCP packets with the SYN flag and high request rates.

Detect IP Spoofing: Identifies packets with unusual IP options, such as record route (RR), which could indicate IP spoofing.

Detect SYN Flood with Spoofed IPs: Combines SYN flood detection and spoofing detection to identify high-rate SYN packets originating from spoofed IP addresses.

Running Snort in the terminal and then checking if it is responding to the IP spoofed SYN attacks.





As soon as we started flooding the server VM with SYN packets, Snort immediately alerted us. Now we will block this IP temporarily to stop this attack.

2. **Blocking with iptables automating with bash script**:

Bash Script

```
#!/bin/bash

CAPTURED_FILE="captured_ips.txt"
BLOCK_DURATION=300

> "$CAPTURED_FILE"

run_tcpdump() {
    sudo tcpdump -i enp0s3 -nn -tttt -U | \
    awk '/seq/ {print $0}' | \
    awk '{split($0, a, "seq "); if (a[2] > 132458000) print $4}' | \
    cut -d'.' -f1-4 | \
    while read ip; do
        if ! grep -qx "$ip" "$CAPTURED_FILE"; then
            echo "$ip" >> "$CAPTURED_FILE"
        fi
    done &
    TCPDUMP_PID=$!
}

block_ip_for_duration() {
    local ip=$1
    if ! sudo iptables -C INPUT -s "$ip" -j DROP 2>/dev/null; then
        sudo iptables -A INPUT -s "$ip" -j DROP
        sudo iptables -A INPUT -s "$ip" -j LOG --log-prefix "Dropped traffic: "
    fi
    sleep $BLOCK_DURATION
    sudo iptables -D INPUT -s "$ip" -j DROP
}

run_tcpdump

while true; do
    if [[ -s "$CAPTURED_FILE" ]]; then
        break
    fi
    sleep 2
done

cat "$CAPTURED_FILE" | sort | uniq | while read -r ip; do
    block_ip_for_duration "$ip" &
done

wait
```

This script is designed to **capture incoming IP addresses** using `tcpdump`, **analyze the captured data** to extract specific IPs (SYN attack IP), and **temporarily block these IPs** using `iptables`

Working of the script:

- Monitor network traffic using tcpdump.
- Identify suspicious IPs based on specific patterns (e.g., high sequence numbers or abnormal traffic volume).
- Automatically add blocking rules to iptables for these IPs.
- Unblock the IPs after a predefined time to avoid permanent denial of legitimate traffic.

Working of the script

```
remnux@remnux:~$ sudo bash ip_grab.sh
tcpdump is running in the background with PID 2236
Waiting for IPs to be captured...
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
Waiting for IPs to be captured...
Waiting for IPs to be captured...
Waiting for IPs to be captured...
Waiting for IPs to be captured...
Waiting for IPs to be captured...
Waiting for IPs to be captured...
Waiting for IPs to be captured...
IPs are being captured, starting processing...
Blocking incoming packets from IP: 192.168.175.207 for 300 seconds
Blocked IP: 192.168.175.207
```

Blocks the IP which was sending high number of SYN packets.

Stored that IP in the captured file and then add blocking rules on that ip in iptables.

```
remnux@remnux:~$ sudo tail -f captured_ips.txt
192.168.175.207
```

Can be verified by displaying the iptables rule

```
remnux@remnux: ~                                    remnux@remnux: ~
remnux@remnux:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 49 packets, 3332 bytes)
 pkts bytes target     prot opt in     out     source               destination
136K 5428K DROP       all  --  *      *       192.168.175.207      0.0.0.0/0
    0     0 LOG        all  --  *      *       192.168.175.207      0.0.0.0/0           LOG flags 0 level 4 prefix "Dropped traffic: "

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 49 packets, 3332 bytes)
 pkts bytes target     prot opt in     out     source               destination
remnux@remnux:~$
```

All packets coming from this IP are being dropped now temporarily.

Step 5: Simulate Session Hijacking

ARP spoofing is being performed as a pretext for session hijacking. The purpose of ARP spoofing is to intercept and monitor the network traffic flowing between the victim (user machine) and the router. By tricking both devices into associating the attacker's MAC address with the legitimate IP addresses

1. **Command Execution:**
   The following command was executed to initiate ARP spoofing between the victim machine (192.168.184.37) and the router (192.168.184.78):

```
┌──(aarav㉿aarav)-[~]
└─$ sudo ettercap -T -M arp:remote //192.168.184.37// //192.168.184.78//
```

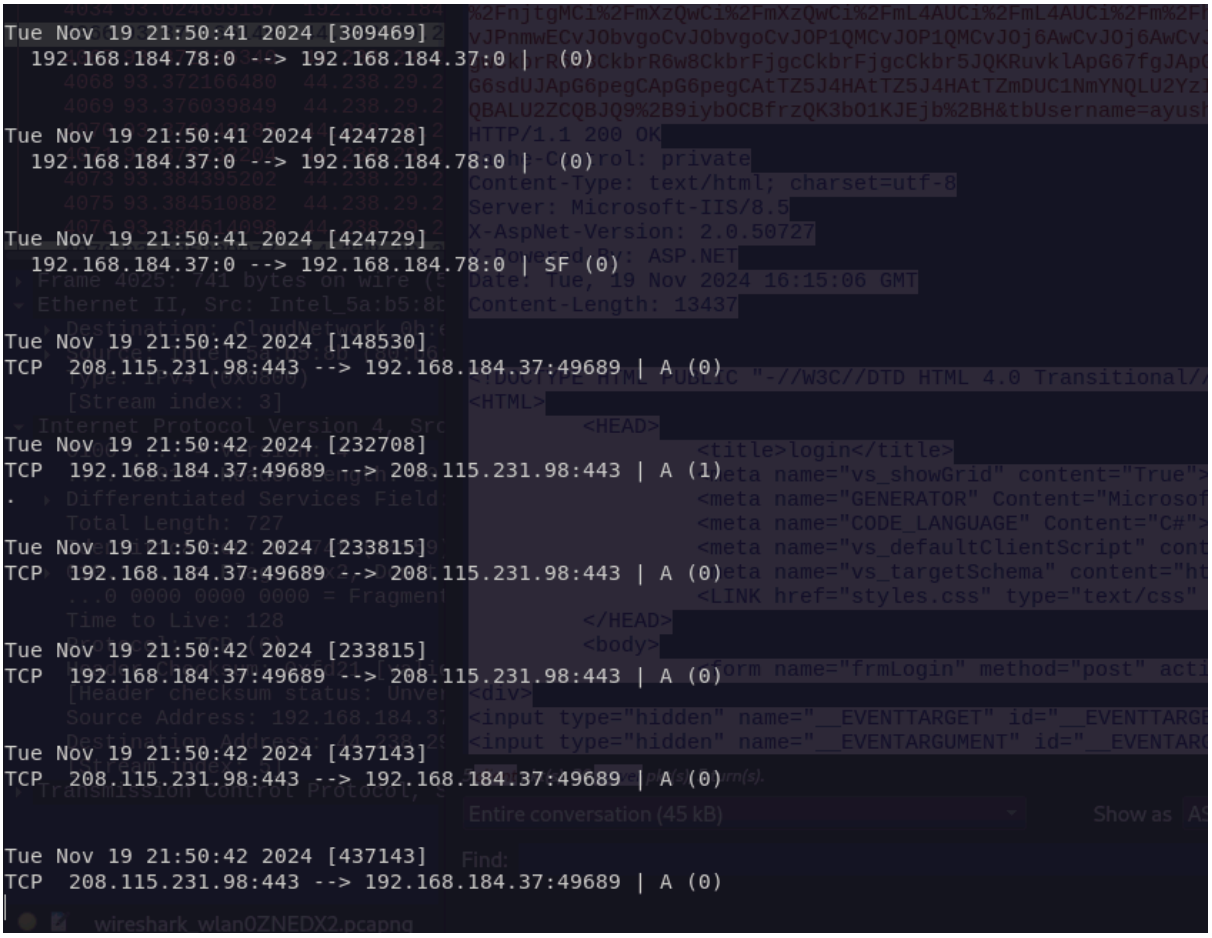
```
TCP  192.168.184.37:49734 --> 52.123.170.113:443 | AP (46)
....).......>..)....8".Uq8.Iy.......@....T.}

Tue Nov 19 21:48:31 2024 [372102]
TCP  192.168.184.37:49734 --> 52.123.170.113:443 | AP (108)
....g.......
.I.5.&d;k\....9..i4..D...K.B.T..#...P*D..M.e.6.E..A      .......-{.8y....a..L.1r7..&./0.....

Tue Nov 19 21:48:31 2024 [414072]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414073]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414073]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414190]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414190]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414191]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [414330]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | A (0)

Tue Nov 19 21:48:31 2024 [722309]
TCP  52.113.194.16:443 --> 192.168.184.37:49732 | AP (182)
.....d....j.s...%...l..."`.V..^....7.E..YT..u$.nKc.<...>..%....g......\{"..b.5..b.|....5..

Tue Nov 19 21:48:31 2024 [722310]
```

Perform ARP Spoofing

- Ettercap modified the ARP tables of both the victim and the router, tricking them into believing the attacker's machine is the other device.
- This enabled interception of traffic between the victim and the router.

2. **Exchanged IPs for Bi-Directional Spoofing**
   Reversed the IPs in a new command to ensure complete ARP spoofing in both directions:

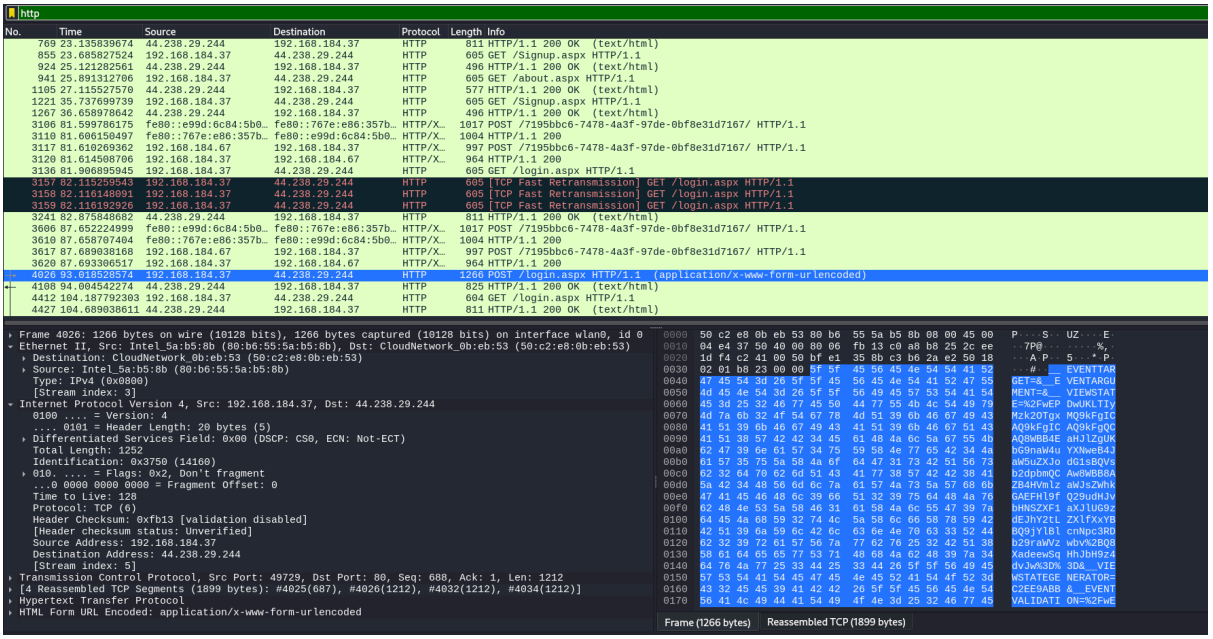
```
┌──(aarav㉿aarav)-[~]
└─$ sudo ettercap -T -M arp:remote //192.168.184.78// //192.168.184.37//
```

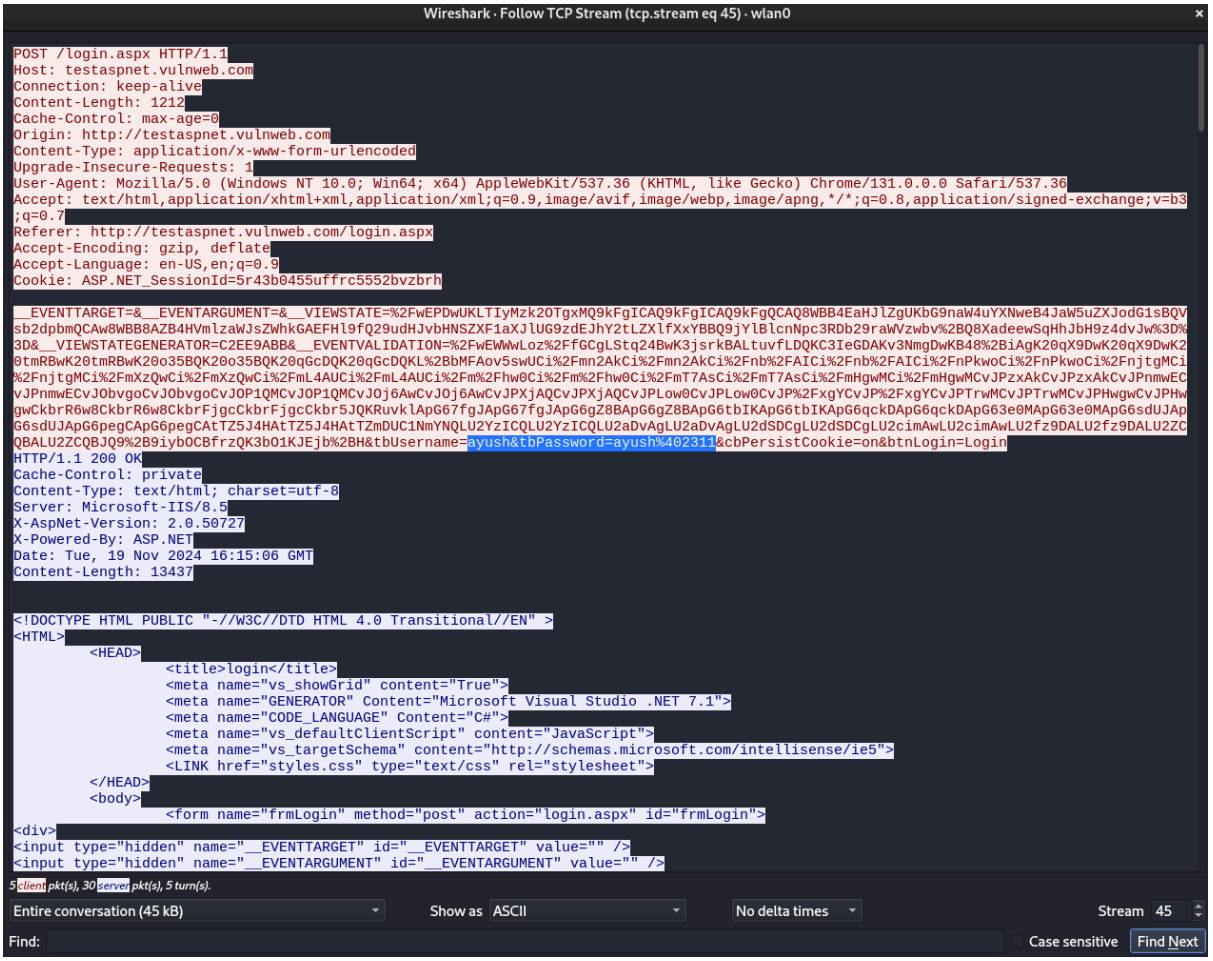


- This step ensured that traffic from both the victim to the router and the router to the victim was intercepted.

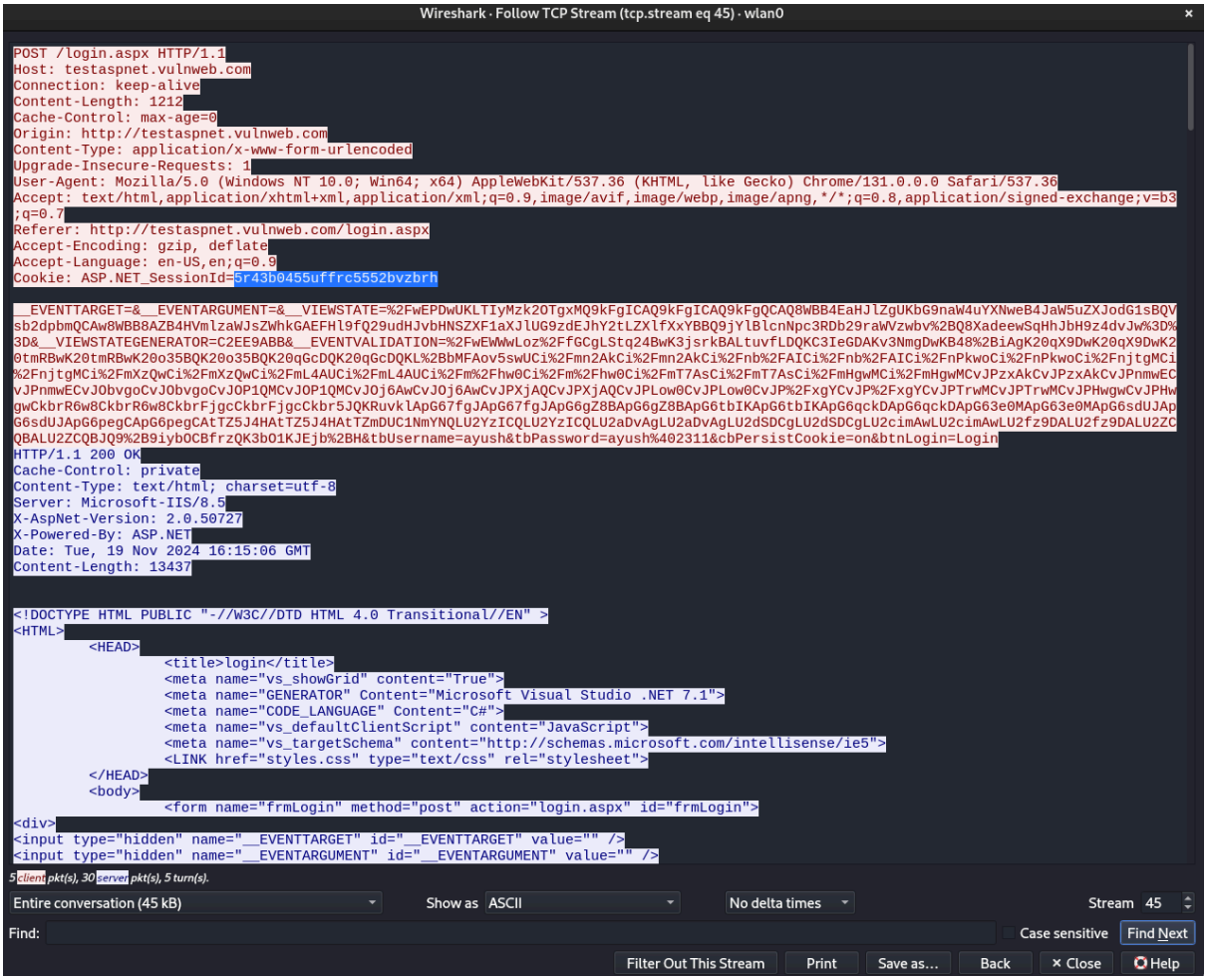
Traffic Interception:
By performing the above commands, the attacker successfully intercepted network traffic between the victim machine and the router. The attacker's machine became the "man-in-the-middle," enabling the capture of all data exchanged between these two devices.

The traffic is analysed in wireshark to find the session cookie





The user opened a website which was using insecure http protocol, we can see the username and password from the traffic we captured.



The user’s sessionID can also be stolen from here.

Step 6: Defense against arp-spoofing (Precursor to Session Hijacking)

Used a bash script to automate the process of detecting and blocking ARP spoofing attacks.

```bash
#!/bin/bash

LEGIT_MAC_TABLE="/tmp/legit_mac_table.txt"
TEMP_ARP_TABLE="/tmp/temp_arp_table.txt"

rule_exists() {
    local attacker_ip=$1
    local target_ip=$2
    sudo arptables -L INPUT -n | grep -q "$attacker_ip" | grep -q "$target_ip"
}

block_arp_spoofing() {
    local attacker_ip=$1
    local target_ip=$2
    if rule_exists "$attacker_ip" "$target_ip"; then
        echo "Rule already exists for Attacker IP: $attacker_ip and Target IP: $target_ip"
    else
        echo "Blocking ARP spoofing: Attacker IP $attacker_ip spoofing Target IP $target_ip"
        sudo arptables -A INPUT -d "$target_ip" -s "$attacker_ip" -j DROP
    fi
}

detect_arp_spoofing() {
    arp -n | awk '/^[1-9]/ {print $1,$3}' > "$TEMP_ARP_TABLE"
    while read -r ip mac; do
        if grep -q "$ip" "$LEGIT_MAC_TABLE"; then
            legit_mac=$(grep "$ip" "$LEGIT_MAC_TABLE" | awk '{print $2}')
            if [[ "$legit_mac" != "$mac" ]]; then
                echo "ARP spoof detected! IP: $ip is spoofed by MAC: $mac"
                block_arp_spoofing "$ip" "$ip"
            fi
        else
            echo "$ip $mac" >> "$LEGIT_MAC_TABLE"
            echo "Added legitimate IP-MAC pair: $ip -> $mac"
        fi
    done < "$TEMP_ARP_TABLE"
}

echo "Starting ARP spoof detection..."
> "$LEGIT_MAC_TABLE"
while true; do
    detect_arp_spoofing
    sleep 5
done
```

Working of the script

Starting ARP Detection:

- The script begins by creating an empty `LEGIT_MAC_TABLE` file.
- Continuously monitors the ARP table for changes.

Detecting Spoofing:

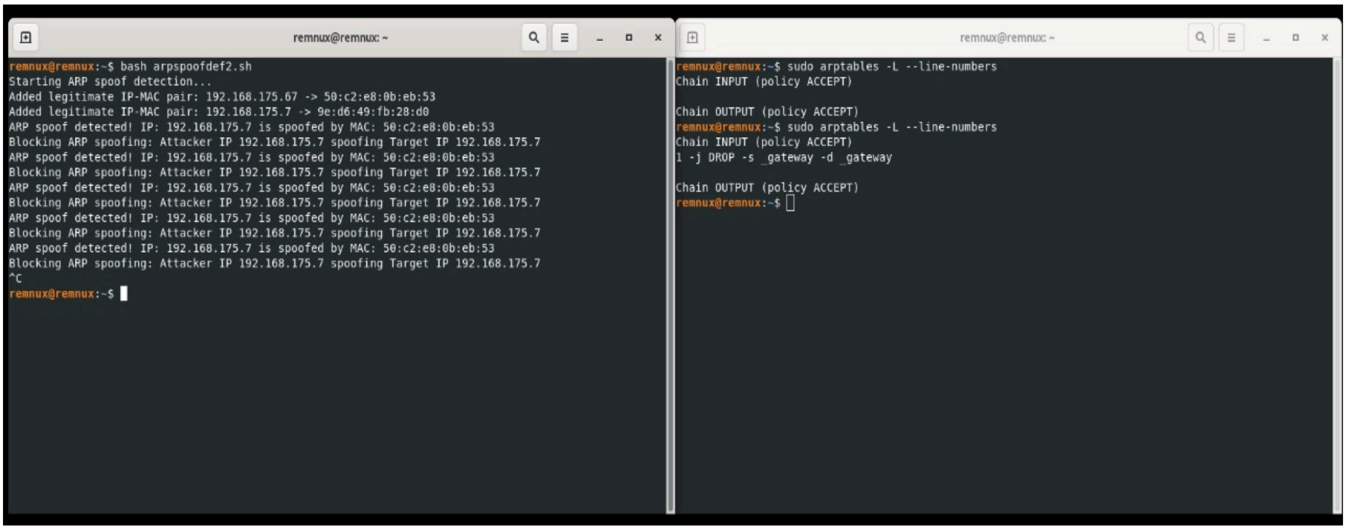
- Reads the current ARP table (`arp -n`) and extracts IP-MAC pairs.
- If an IP has a different MAC address than the one recorded in `LEGIT_MAC_TABLE`, it assumes ARP spoofing is happening.

Blocking Spoofing Attempts:

- Once ARP spoofing is detected, the script uses `arptables` to block packets from the spoofing source to the target.
- Logs the detected spoofing attempt for later analysis.

Updating Legitimate MAC Table:

- If a new IP-MAC pair is observed (i.e., not previously recorded), it gets added to `LEGIT_MAC_TABLE` as legitimate.



The left terminal shows us the log that it is adding legitimate IP-MAC pairs, detection of ARP-spoofing and then blocking that attack.

For each spoofing detection, the script dynamically added arptables rules to block packets from the attacker.

The right terminal lists the arptables rules. The blocking rule dynamically added by the script is visible:

```
-j DROP -s <attacker_ip> -d <target_ip>
```

Purpose of Each Step

- **Script Execution:** Starts ARP spoof detection and dynamically defends against attacks.
- **Detection:** Identifies spoofing by comparing observed IP-MAC pairs against the legitimate table.
- **Blocking:** Uses arptables to prevent spoofing attempts, ensuring the attacker cannot impersonate the legitimate IP.
- **Verification:** Confirms blocking rules have been correctly added to arptables.

Project Outcome

- **Summary of Results:**
    - Successful detection and mitigation of IP spoofing and SYN flood attacks through real-time monitoring and analysis.
    - Implementation of effective logging and alerting mechanisms using Wireshark and Snort.
    - Blocking of malicious traffic based on real-time analysis and pre-defined rules using iptables and arptables
- **Metrics of Success:**
    - **Detection Accuracy**: Number of spoofed or hijacked sessions detected.
    - **Response Time**: Time taken from attack detection to defense deployment.
    - **Documentation Quality**: Completeness and clarity of reports

Challenges and Solutions

Major Obstacles

- **Network Configuration**: Issues with VM connectivity in a simulated environment.
- **Inefficient way:** Attackers use multiple spoofed IPs, making blocking specific IPs ineffective.
- **Resource Constraints**: Limited processing power affected real-time packet analysis.

Solutions and Mitigations

- **Network Configuration**: Troubleshooting virtual network settings and ensuring correct IP assignments.
- **Rule Optimization**: Implement rate-limiting with `iptables` and Snort to dynamically detect and mitigate spoofed traffic based on patterns.
- **Testing in Phases**: Conducted testing in phases to optimize resource usage.