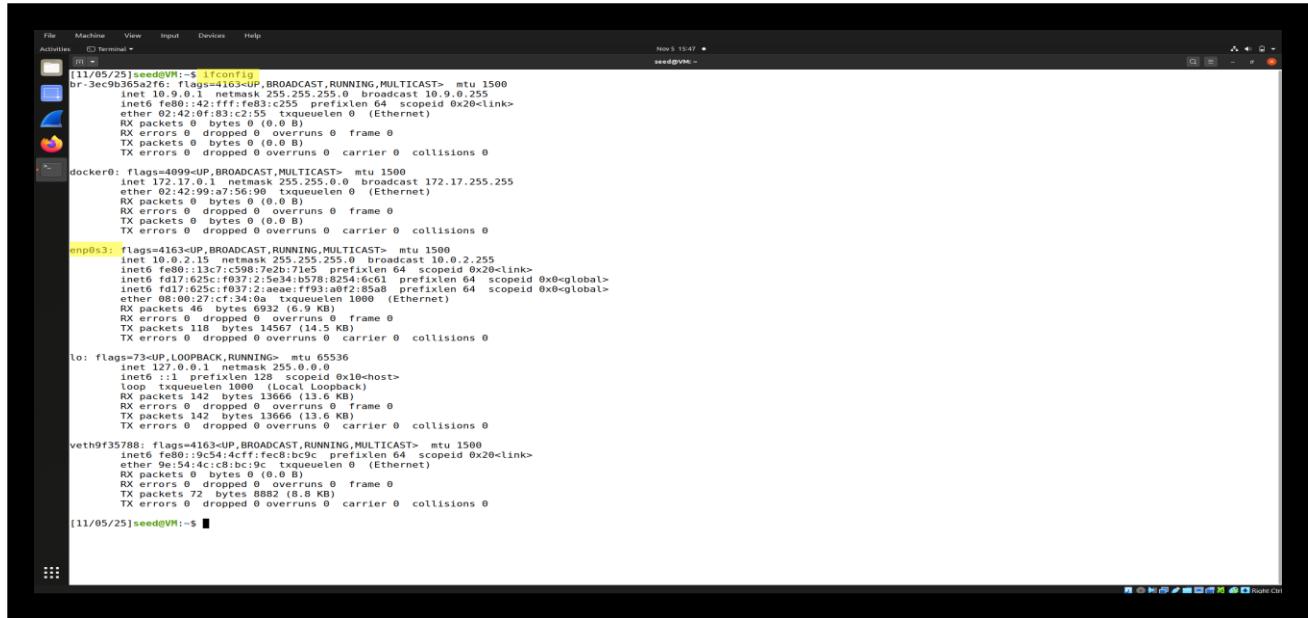


LAB- 03

By :- Faraz Ahmed

Task 1.1

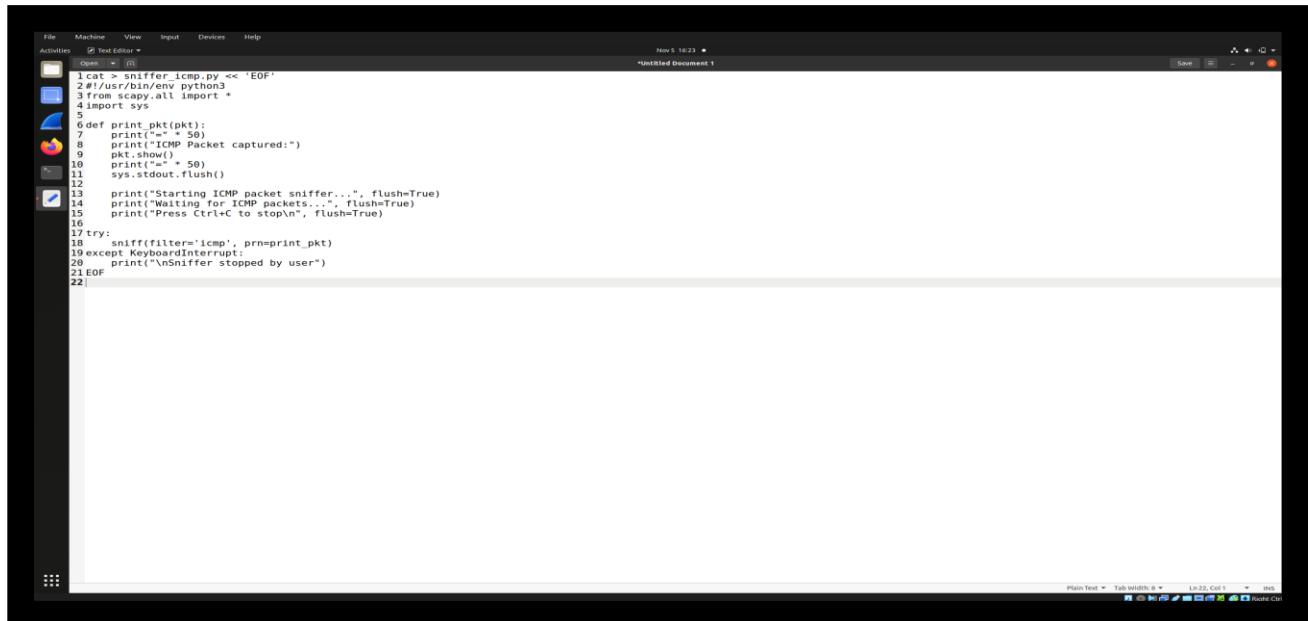
- Firstly, we use command “ifconfig” to find all interface in the network as highlighted in screenshot 1. It has enp0s3 (highlighted) which is our main network interface that has IP address 10.0.2.15.



```
[11/05/25]seed@VM:~$ ifconfig
br-3ec9b5a5x: flags=4163<UP,BROADCAST,MULTICAST> mtu 1500
    inet 10.0.0.1 brd 255.255.255.0 broadcast 10.0.0.255
        netmask 255.255.255.0
        ether 02:42:0f:83:c2:55 txqueuelen 0 (Ethernet)
        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
docker0: flags=4163<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 brd 255.255.255.255 broadcast 172.17.255.255
        netmask 255.255.255.255
        ether 02:42:99:a7:56:98 txqueuelen 0 (Ethernet)
        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
enp0s3: flags=163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 brd 255.255.255.0 broadcast 10.0.2.255
        netmask 255.255.255.0
        ether 08:00:27:c1:34:00 txqueuelen 1000 (Ethernet)
        RX packets 112 bytes 13666 (13.6 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 112 bytes 13666 (13.6 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 brd 127.0.0.1 netmask 255.0.0.0
        netmask 255.0.0.0
        ether 00:00:00:00:00:00 txqueuelen 0 (Host-Only Adapter)
        RX packets 142 bytes 13666 (13.6 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 142 bytes 13666 (13.6 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
veth9f3578b: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 6 fe80::9c54:4cff:fe8:bc9c brd ff:ff:ff:ff:ff:ff prefixlen 64 scopeid 0x20<link>
        ether 0e:54:4c:28:bc:9c txqueuelen 0 (Ethernet)
        RX packets 72 bytes 8882 (8.8 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[11/05/25]seed@VM:~$
```

Screenshot 1: Executing command “ifconfig” to find interface in the network.

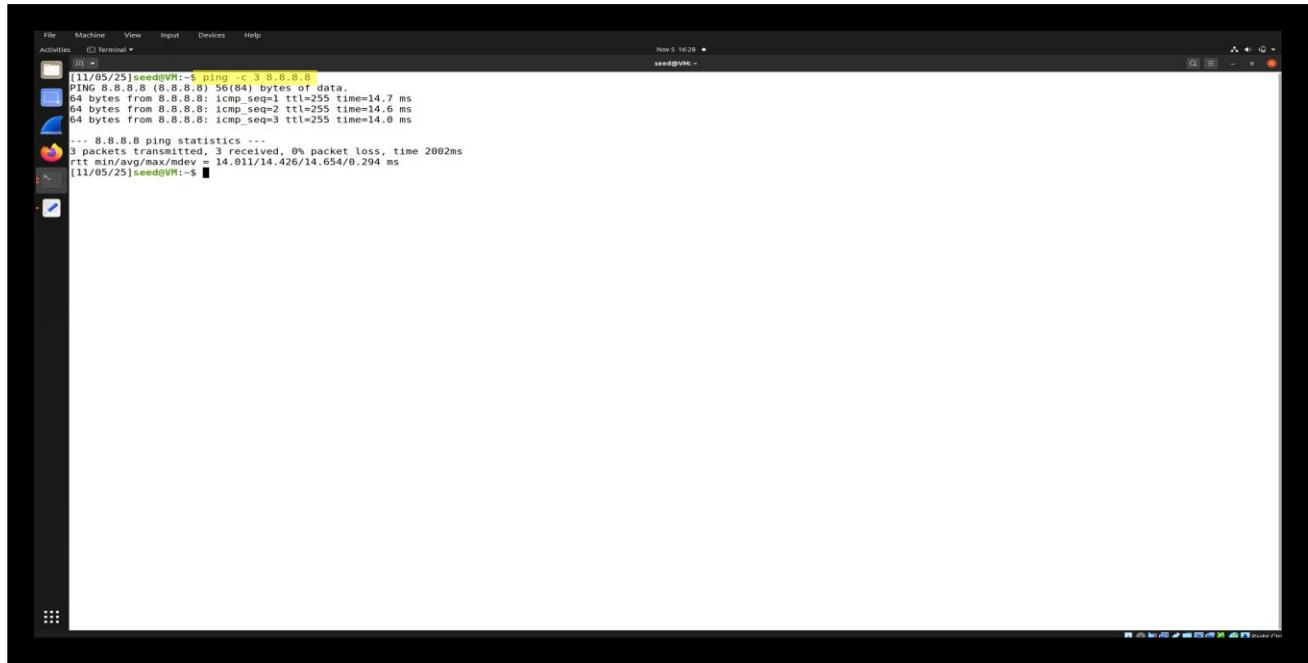
- Now we will create sniffer_icmp.py file which will contain code shown in screenshot 2.



```
1cat > sniffer_icmp.py << EOF
2#!/usr/bin/env python3
3from scapy.all import *
4import sys
5
6def print_pktpkt():
7    print("-----")
8    print("ICMP Packet captured:")
9    pkt.show()
10   print("-----")
11   sys.stdout.flush()
12
13   print("Starting ICMP packet sniffer...", flush=True)
14   print("Waiting for ICMP packets... ", flush=True)
15   print("Press Ctrl+C to stop\n", flush=True)
16
17 try:
18     sniff(filter='icmp', prn=print_pktpkt)
19 except KeyboardInterrupt:
20     print("\nSniffer stopped by user")
21EOF
22
```

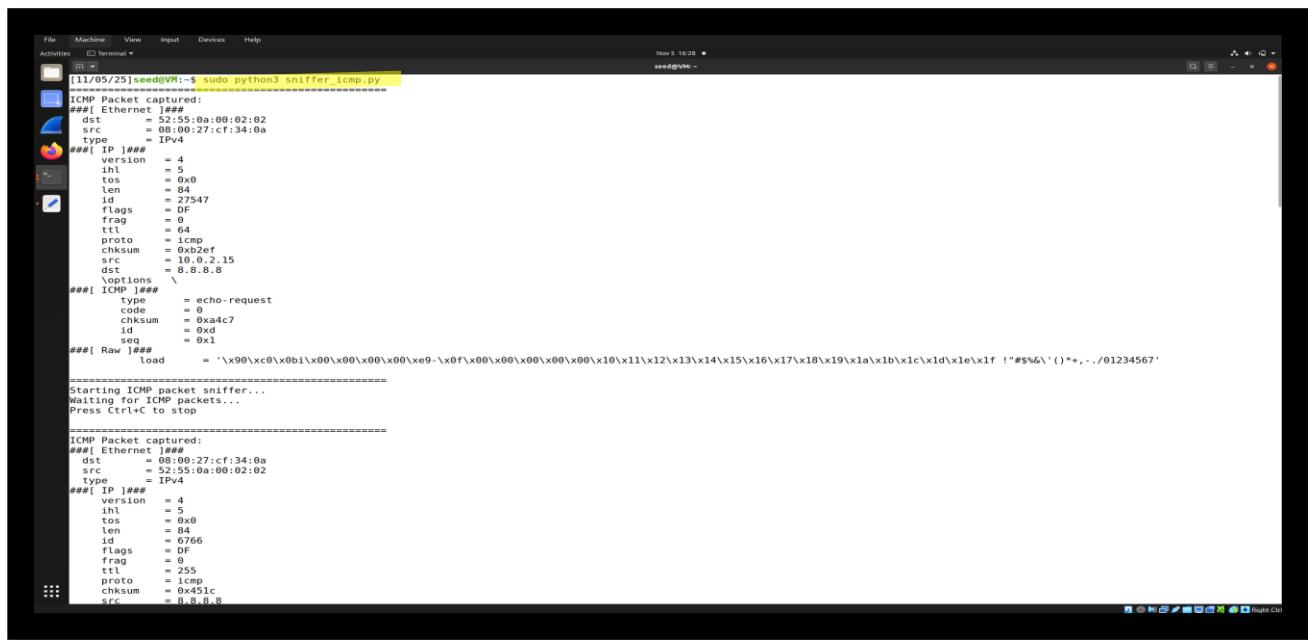
Screenshot 2: “Sniffer_icmp.py” python code which is newly created.

- Then we will execute command “sudo python3 sniffer_icmp.py” (highlighted in screenshot 4) which will start ICMP packet capture and then on another terminal and “ping -c 3 8.8.8.8” to start an ICMP ping as highlighted in screenshot 3.



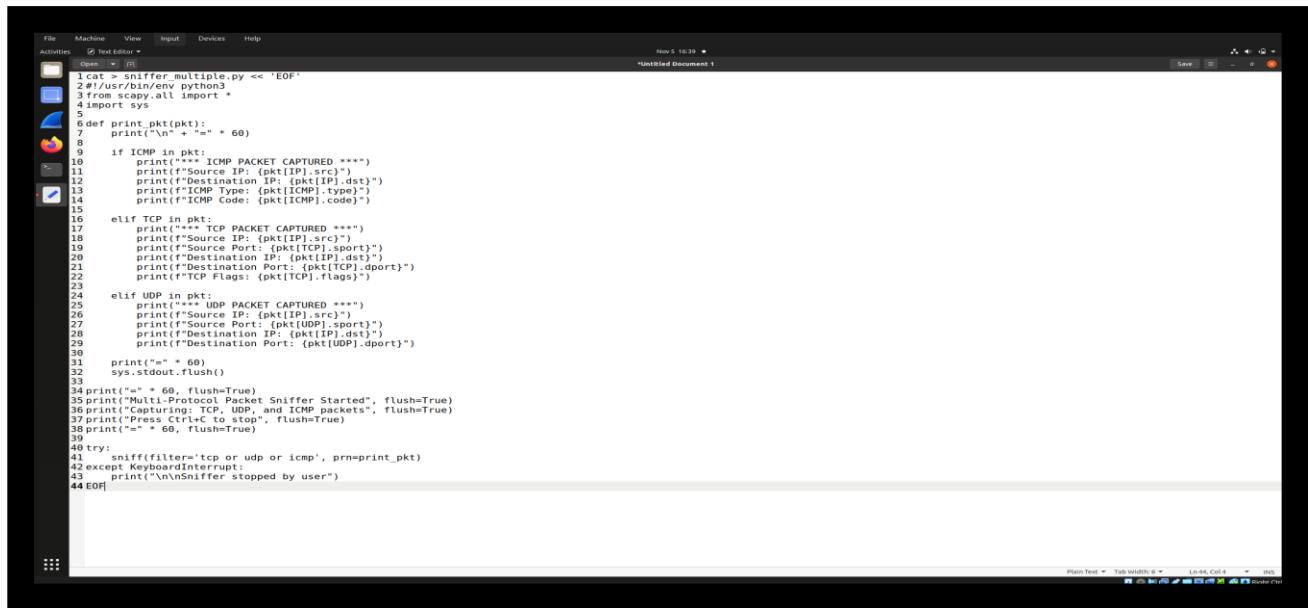
Screenshot 3: Command “ping -c 3 8.8.8.8” to start an ICMP ping.

- After running ICMP packet, we can see some activity in ICMP packet capture as observed in screenshot 4.



Screenshot 4: ICMP activity logged in ICMP packet capture.

- So, now we will capture three different packets from three different protocols like ICMP, TCP and UDP. So, for that we will use python code to create “sniffer_multiple.py” as observed in screenshot 5.



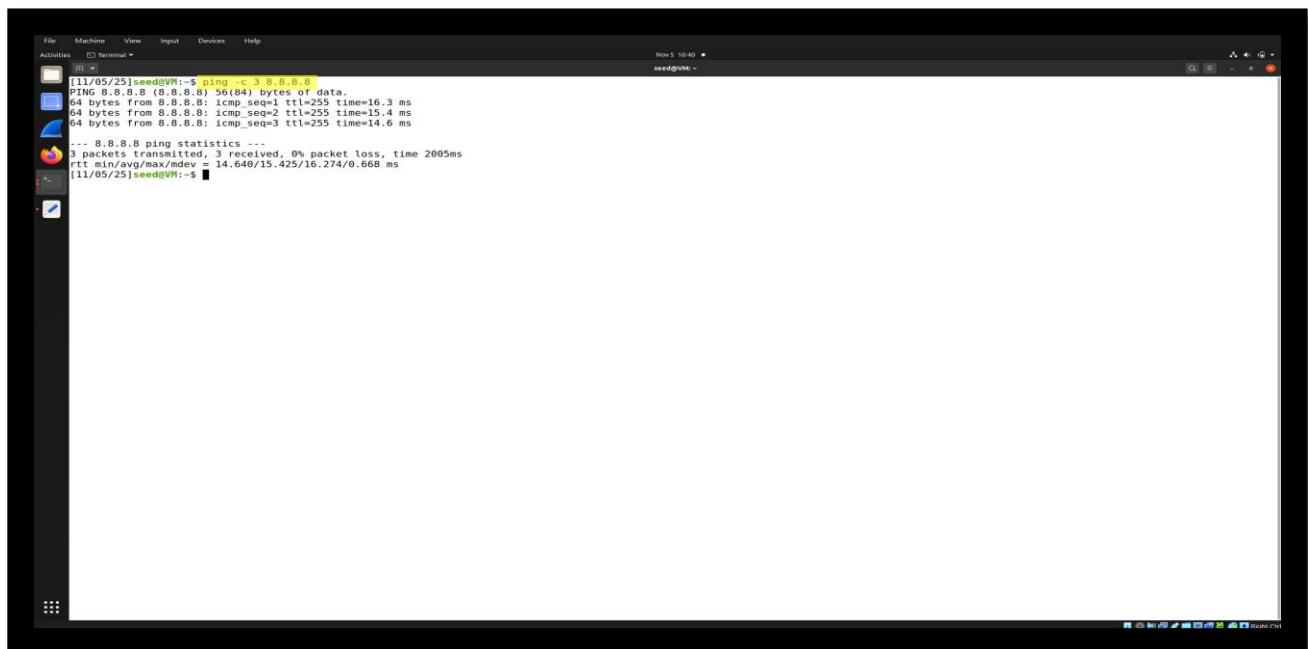
```

File Machine View Input Devices Help
Activities Text Editor
Open Untitled Document
Nov 5 10:39
1 #!/usr/bin/env python3
2 from scapy.all import *
3 import sys
4
5 def print_pkt(pkt):
6     print("\n" + "=" * 60)
7     print(pkt)
8
9     if ICMP in pkt:
10        print("ICMP PACKET CAPTURED ***")
11        print("Source IP: (pkt[IP].src)")
12        print("Destination IP: (pkt[IP].dst)")
13        print("ICMP Type: (pkt[ICMP].type)")
14        print("ICMP Code: (pkt[ICMP].code)")
15
16    elif TCP in pkt:
17        print("TCP PACKET CAPTURED ***")
18        print("Source IP: (pkt[IP].src)")
19        print("Source Port: (pkt[TCP].sport)")
20        print("Destination IP: (pkt[IP].dst)")
21        print("Destination Port: (pkt[TCP].dport)")
22        print("TCP Flags: (pkt[TCP].flags)")
23
24    elif UDP in pkt:
25        print("UDP PACKET CAPTURED ***")
26        print("Source IP: (pkt[IP].src)")
27        print("Source Port: (pkt[UDP].sport)")
28        print("Destination IP: (pkt[IP].dst)")
29        print("Destination Port: (pkt[UDP].dport)")
30
31    print("=" * 60)
32    sys.stdout.flush()
33
34 print("\n" + "=" * 60, flush=True)
35 print("Monitoring Protocol Sniffer Started", flush=True)
36 print("Capturing TCP, UDP, and ICMP packets", flush=True)
37 print("Press Ctrl+c to stop", flush=True)
38 print("\n" + "=" * 60, flush=True)
39
40 try:
41    sniff(filter='tcp or udp or icmp', prn=print_pkt)
42 except KeyboardInterrupt:
43    print("\n\nSniffer stopped by user")
44 EOF

```

Screenshot 5: Python code in “sniffer_multiple.py”.

- Now, we will ping one of the three protocols one by one and observe the packets which are captured by running “sudo python3 sniffer_multiple.py”. First, we will run ICMP ping as highlighted in screenshot 6 and note all ICMP packets captured in screenshot 7.



```

File Machine View Input Devices Help
Activities Terminal
Nov 5 10:40
seed@VM: ~
[11/05/23]seed@VM:~$ ping -c 3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=14.6 ms
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 14.640/15.425/16.274/0.668 ms
[11/05/23]seed@VM:~$ 

```

Screenshot 6: ICMP ping using “ping -c 3 8.8.8.8”.

```
[11/05/25]seed@VM:-$ sudo python3 sniffer_multiple.py
Multi-Protocol Packet Sniffer Started
Capturing: TCP, UDP, and ICMP packets
Press Ctrl+C to stop
*** ICMP PACKET CAPTURED ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP Type: 8
ICMP Code: 0

*** ICMP PACKET CAPTURED ***
Source IP: 8.8.8.8
Destination IP: 10.0.2.15
ICMP Type: 8
ICMP Code: 0

*** ICMP PACKET CAPTURED ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP Type: 8
ICMP Code: 0

*** ICMP PACKET CAPTURED ***
Source IP: 8.8.8.8
Destination IP: 10.0.2.15
ICMP Type: 8
ICMP Code: 0

*** ICMP PACKET CAPTURED ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP Type: 8
ICMP Code: 0

*** ICMP PACKET CAPTURED ***
Source IP: 8.8.8.8
Destination IP: 10.0.2.15
ICMP Type: 8
ICMP Code: 0
```

Screenshot 7: All captured ICMP packets from ICMP ping.

- Second, we will run TCP ping using “curl <http://www.example.com>” as highlighted in screenshot 8 and then observe all TCP packets captured in screenshot 9.

```
[11/05/25]seed@VM:-$ curl http://www.example.com
<!DOCTYPE html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Domain</h1><p>This domain is for use in documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></div></body></html>
[11/05/25]seed@VM:-$
```

Screenshot 8: TCP ping using “curl <http://www.example.com>”.

```
[11/05/25]seed@VM:~$ sudo python3 sniffer_multiple.py
Multi-Protocol Packet Sniffer Started
Capturing: TCP, UDP and ICMP packets
Press Ctrl+C to stop
=====
*** UDP PACKET CAPTURED ***
Source IP: 10.0.2.15
Source Port: 44563
Destination IP: 192.168.1.1
Destination Port: 53
=====
*** UDP PACKET CAPTURED ***
Source IP: 10.0.2.15
Source Port: 35636
Destination IP: 192.168.1.1
Destination Port: 53
=====
*** UDP PACKET CAPTURED ***
Source IP: 10.0.2.15
Source Port: 53
Destination IP: 10.0.2.15
Destination Port: 44563
=====
*** UDP PACKET CAPTURED ***
Source IP: 192.168.1.1
Source Port: 35636
Destination IP: 10.0.2.15
Destination Port: 35636
=====
*** TCP PACKET CAPTURED ***
Source IP: 10.0.2.15
Source Port: 53
Destination IP: 104.98.115.8
Destination Port: 80
TCP Flags: S
=====
*** TCP PACKET CAPTURED ***
Source IP: 104.98.115.8
Source Port: 80
Destination IP: 10.0.2.15
```

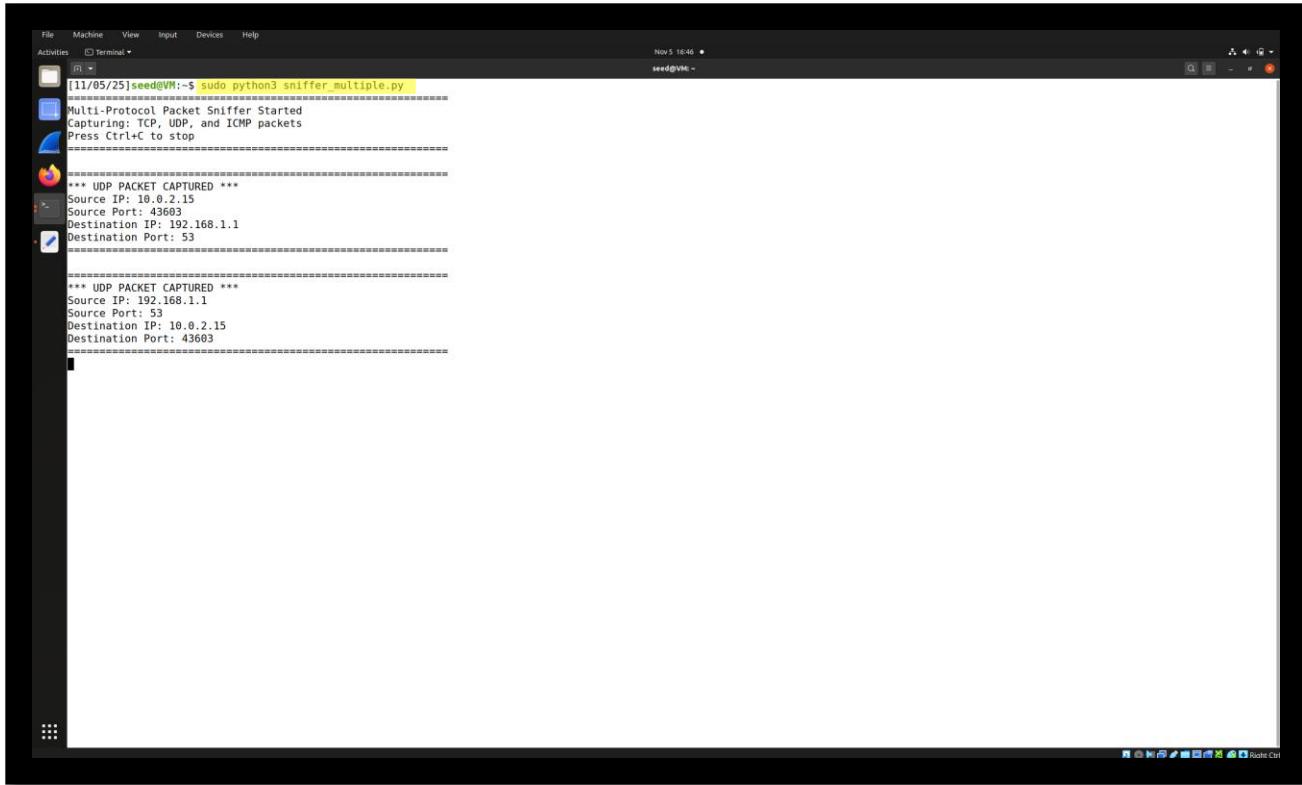
Screenshot 9: All captured TCP packets from TCP ping.

- Third, we will run UDP ping using “nslookup google.com” as highlighted in screenshot 10 and then observe all UDP packets captured in screenshot 11.

```
[11/05/25]seed@VM:~$ nslookup google.com
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
Name: google.com
Address: 142.250.80.78
Name: google.com
Address: 2607:f8b0:4006:823::200e
[11/05/25]seed@VM:~$
```

Screenshot 10: UDP ping using “nslookup google.com”.



The screenshot shows a terminal window titled "Terminal" with the command `sudo python3 sniffer_multiple.py` running. The output indicates a multi-protocol packet sniffer is active, capturing TCP, UDP, and ICMP packets. It specifically highlights two UDP captures:

```
[11/05/25]seed@VM:-$ sudo python3 sniffer_multiple.py
=====
Multi-Protocol Packet Sniffer Started
Capturing: TCP, UDP, and ICMP packets
Press Ctrl+C to stop
=====

*** UDP PACKET CAPTURED ***
Source IP: 10.0.2.15
Source Port: 43603
Destination IP: 192.168.1.1
Destination Port: 53
=====

*** UDP PACKET CAPTURED ***
Source IP: 192.168.1.1
Source Port: 53
Destination IP: 10.0.2.15
Destination Port: 43603
=====
```

Screenshot 11: All captured UDP packets from UDP ping.

Task 1.2

- Firstly, we have to install wireshark using “sudo apt-get install wireshark” and then set it up using “sudo dpkg-reconfigure wireshark-common” and select YES and add user to wireshark group using “sudo usermod -aG wireshark \$USER” as highlighted in screenshot 12.

```
veth9f3578b: Flags=4163-UP,BROADCAST,RUNNING,MULTICAST<-- mtu 1500
    link: fe80::54c:8bc9%veth9f3578b brd ff:ff:ff:ff:ff:ff
    ether 00:0c:8c:bc:9e:00 txqueuelen 0 (Ethernet)
      RX packets 0 bytes 0 (0.0 B)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 96 bytes 11080 (11.1 KB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[11/05/25]seed@VM:~$ sudo apt-get install wireshark
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wireshark is already the newest version (3.2.3-1).
The following packages were automatically installed and are no longer required:
  libpprint-2-tcl liblxml18
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[11/05/25]seed@VM:~$ sudo usermod -aG wireshark $USER
usermod: user $USER does not exist
[11/05/25]seed@VM:~$ sudo apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [74.6 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 DEP-11 Metadata [212 B]
Get:5 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [134 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [940 B]
Fetched 363 kB in 0s (1,028 kB/s)
0 packages can be upgraded or installed on this system.
[11/05/25]seed@VM:~$ sudo apt-get install wireshark -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wireshark is already the newest version (3.2.3-1).
The following packages were automatically installed and are no longer required:
  libpprint-2-tcl liblxml18
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[11/05/25]seed@VM:~$ sudo apt-get install wireshark -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wireshark is already the newest version (3.2.3-1).
The following packages were automatically installed and are no longer required:
  libpprint-2-tcl liblxml18
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[11/05/25]seed@VM:~$ sudo dpkg-reconfigure wireshark-common
[11/05/25]seed@VM:~$ sudo usermod -aG wireshark $USER
[11/05/25]seed@VM:~$ grep wireshark /etc/group
wireshark:x:137:seed
[11/05/25]seed@VM:~$
```

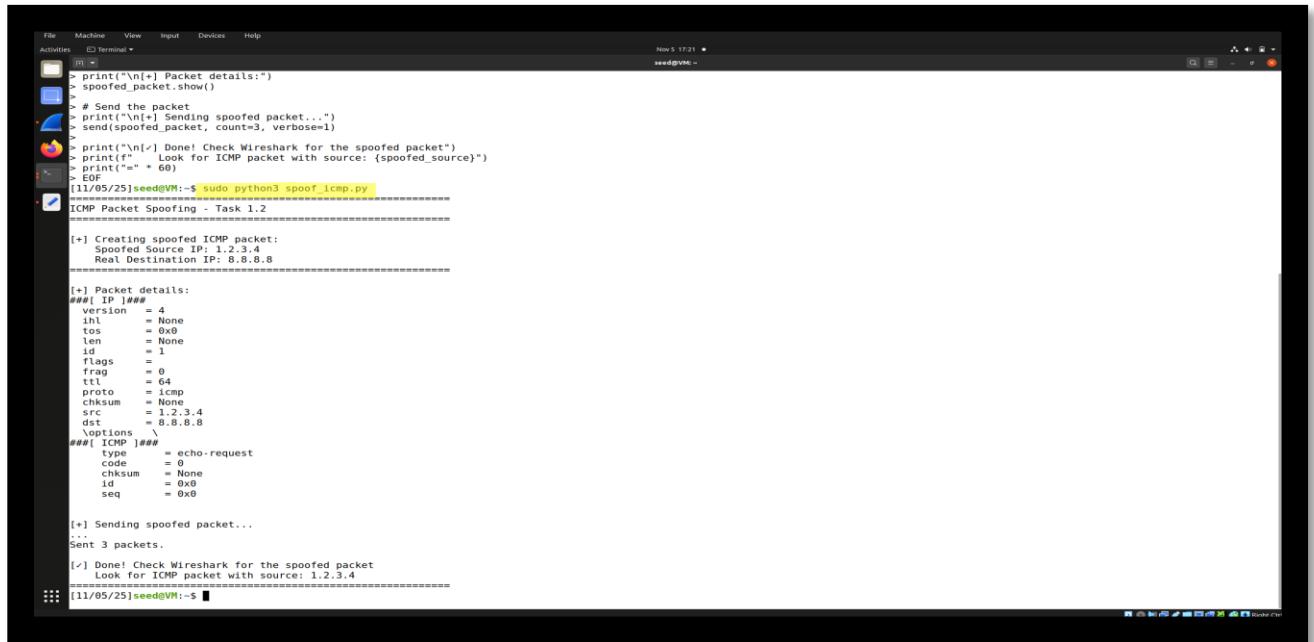
Screenshot 12: Commands used to install and setup Wireshark in the machine.

- Then we will create a new ICMP spoofing script of “spoof_icmp.py” which can be observed in screenshot 13.

```
File Machine View Input Devices Help
Activities Text Editor * Untitled Document *
File Machine View Input Devices Help
Activities Text Editor * Untitled Document *
1# Create ICMP layer
2#EOF
2#!/usr/bin/env python3
3from scapy.all import *
4
5print("=" * 60)
6print("ICMP Packet Spoofing - Task 1.2")
7print("=" * 60)
8
9# Define the spoofed source IP (fake IP)
10spoofed_source = "1.2.3.4"
11
12# Define the real destination IP
13destination = "8.8.8.8"
14
15print("[+] Creating spoofed ICMP packet:")
16spoofed_source_ip = IP(spoofed_source)
17real_destination_ip = IP(destination)
18print("Spoofed Source IP: " + spoofed_source_ip)
19print("Real Destination IP: " + real_destination_ip)
20print("=" * 60)
21
22# Create IP layer with spoofed source
23ip_layer = IP()
24ip_layer.src = spoofed_source_ip
25ip_layer.dst = real_destination_ip
26
27# Create ICMP layer (echo request)
28icmp_layer = ICMP()
29
30# Combine layers
31spoofed_packet = ip_layer / icmp_layer
32
33# Show packet details
34print("\n[+] Packet details:")
35spoofed_packet.show()
36
37# Send the packet
38print("[+] Sending spoofed packet...")
39send(spoofed_packet, count=5, verbose=1)
40
41print("\n[+] Done! Check Wireshark for the spoofed packet")
42print("Look for ICMP packet with source: " + spoofed_source_ip)
43#EOF
```

Screenshot 13: New ICMP spoofing script of “spoof_icmp.py”.

- Then we will run the spoofing script using “sudo python3 spoof_icmp.py” (highlighted in screenshot 14) which will send 3 packets as observed in screenshot 14.



```

File Machine View Input Devices Help
Activities Terminal Nov 5 17:21
[11/05/25] seeed@VM:~$ sudo python3 spoof_icmp.py
[sudo] password for seeed:
ICMP Packet Spoofing - Task 1.2
=====
[+] Creating spoofed ICMP packet:
  Spoofed Source IP: 1.2.3.4
  Real Destination IP: 8.8.8.8
=====
[+] Packet details:
  ### IP ###
    version = 4
    ihl     = None
    tos     = 0x0
    len     = None
    id      = 1
    flags   =
    frag    = 0
    ttl    = 64
    proto   = icmp
    cksum   = None
    src     = '1.2.3.4'
    dst     = '8.8.8.8'
    \options \
  ### ICMP ###
    type    = echo-request
    code    = 0
    cksum   = None
    id      = 0x0
    seq    = 0x0

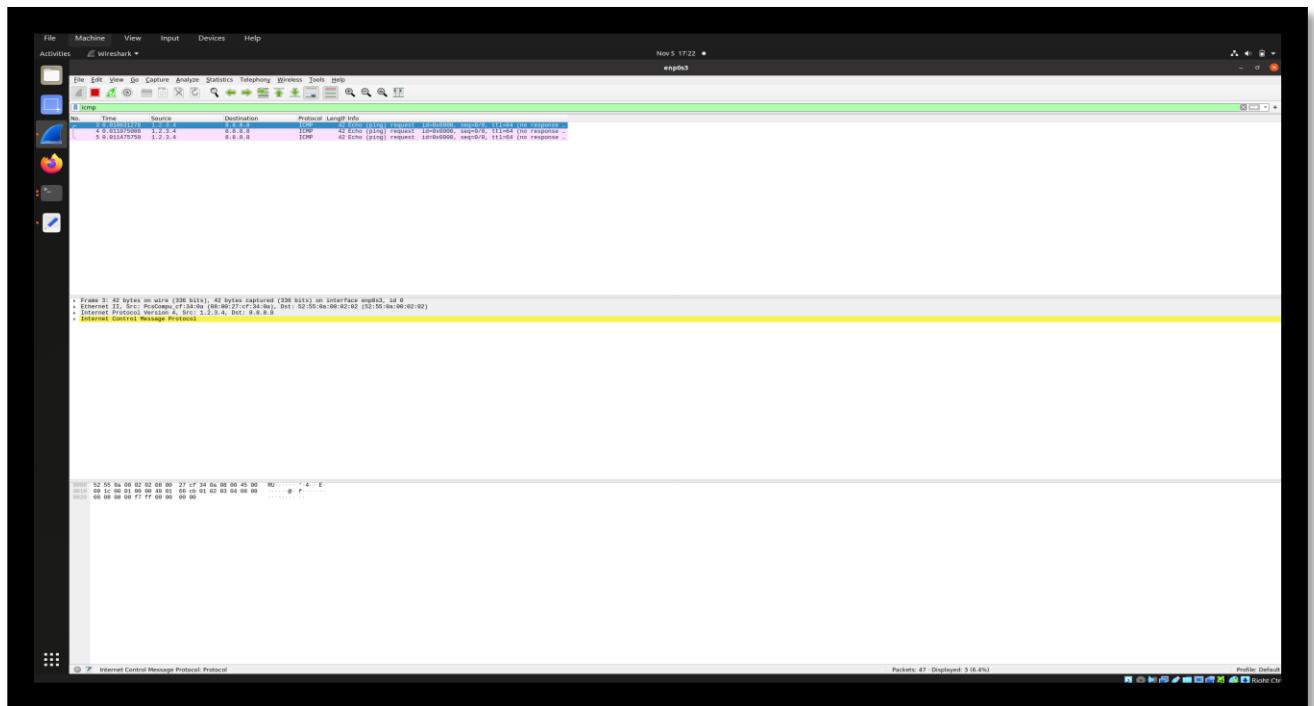
[+] Sending spoofed packet...
...
Sent 3 packets.

[!] Done! Check Wireshark for the spoofed packet
  Look for ICMP packet with source: 1.2.3.4
=====

[11/05/25] seeed@VM:~$ 
  
```

Screenshot 14: Running “sudo python3 spoof_icmp.py” to run spoofing script.

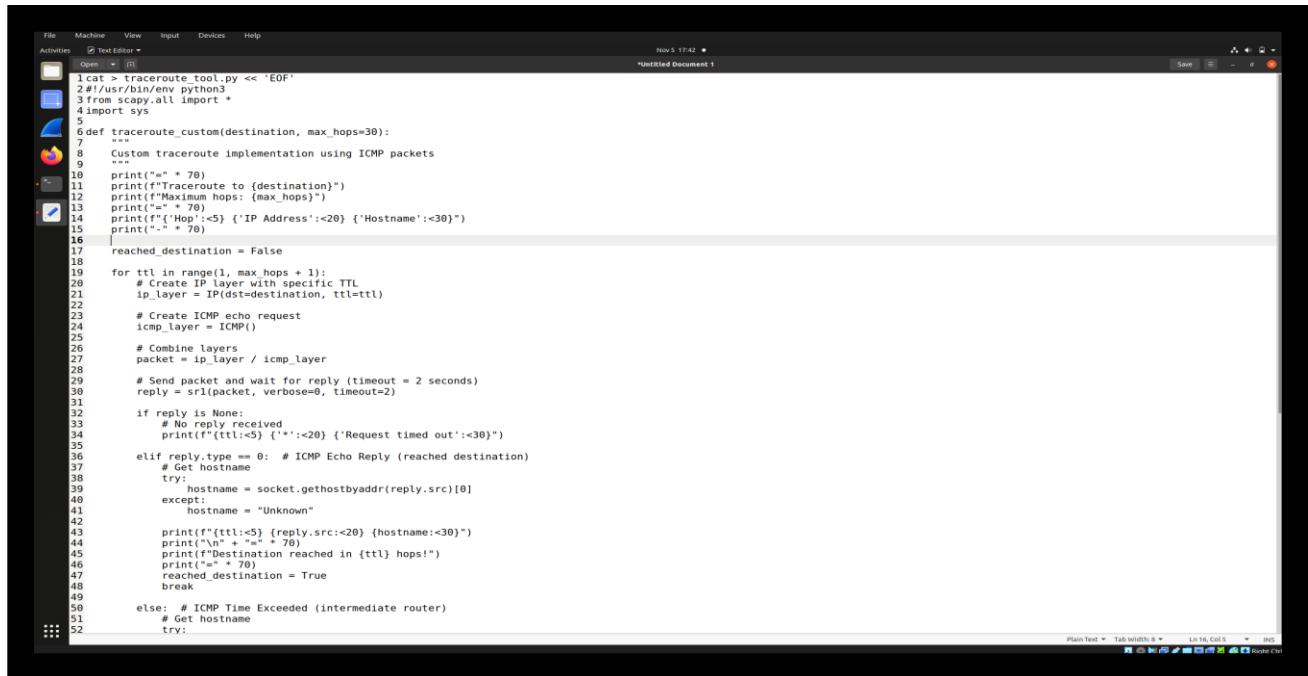
- Then we will open Wireshark using “sudo wireshark &” and observe all three generated packets as shown in screenshot 15.



Screenshot 15: All three generated packets in Wireshark.

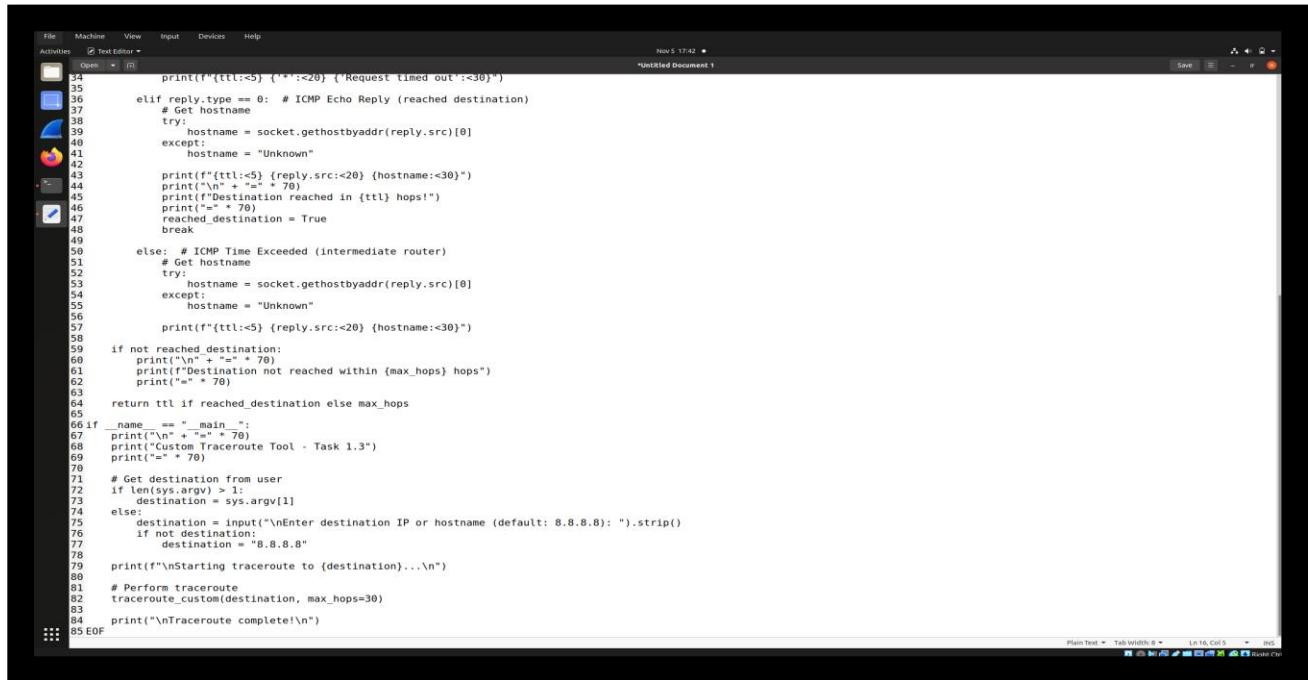
Task 1.3

- For this first, we will create an automated traceroute script “traceroute_tool.py” as observed in screenshot 16 and 17.



```
File Machine View Input Devices Help
Activities Text Editor
Open Untitled Document 1
Nov 5 17:42
1cat > traceroute_tool.py <<< EOF
2#!/usr/bin/env python3
3from scapy.all import *
4import sys
5
6def traceroute_custom(destination, max_hops=30):
7    """
8        Custom traceroute implementation using ICMP packets
9    """
10   print("*" * 70)
11   print(f"Traceroute to {destination}")
12   print(f"Maximum hops: {max_hops}")
13   print("*" * 70)
14   print(f"[Hop:<5] {'IP Address':<20} {'Hostname':<30}")
15   print("*" * 70)
16
17   reached_destination = False
18
19   for ttl in range(1, max_hops + 1):
20       # Create IP layer with specific TTL
21       ip_layer = IP(dst=destination, ttl=ttl)
22
23       # Create ICMP echo request
24       icmp_layer = ICMP()
25
26       # Combine layers
27       packet = ip_layer / icmp_layer
28
29       # Send packet and wait for reply (timeout = 2 seconds)
30       reply = sr1(packet, verbose=0, timeout=2)
31
32       if reply is None:
33           # No reply received
34           print(f"(ttl:<5) {'*'<20} {'Request timed out':<30}")
35
36       elif reply.type == 0: # ICMP Echo Reply (reached destination)
37           # Get hostname
38           try:
39               hostname = socket.gethostbyaddr(reply.src)[0]
40           except:
41               hostname = "Unknown"
42
43           print(f"(ttl:<5) {reply.src:<20} {hostname:<30}")
44           print("\n" + "=" * 70)
45           print(f"Destination reached in {ttl} hops!")
46           print("*" * 70)
47           reached_destination = True
48           break
49
50       else: # ICMP Time Exceeded (intermediate router)
51           # Get hostname
52           try:
53               hostname = socket.gethostbyaddr(reply.src)[0]
54           except:
55               hostname = "Unknown"
56
57           print(f"(ttl:<5) {reply.src:<20} {hostname:<30}")
58
59       if not reached_destination:
60           print("\n" + "=" * 70)
61           print(f"Destination not reached within {max_hops} hops")
62           print("*" * 70)
63
64   return ttl if reached_destination else max_hops
65
66if __name__ == "__main__":
67    print("\n" + "=" * 70)
68    print("Custom Traceroute Tool - Task 1.3")
69    print("\n" + "=" * 70)
70
71    # Get destination from user
72    if len(sys.argv) > 1:
73        destination = sys.argv[1]
74    else:
75        destination = input("\nEnter destination IP or hostname (default: 8.8.8.8): ").strip()
76    if not destination:
77        destination = "8.8.8.8"
78
79    print("\nStarting traceroute to {destination}...\n")
80
81    # Perform traceroute
82    traceroute_custom(destination, max_hops=30)
83
84    print("\nTraceroute complete!\n")
85EOF
```

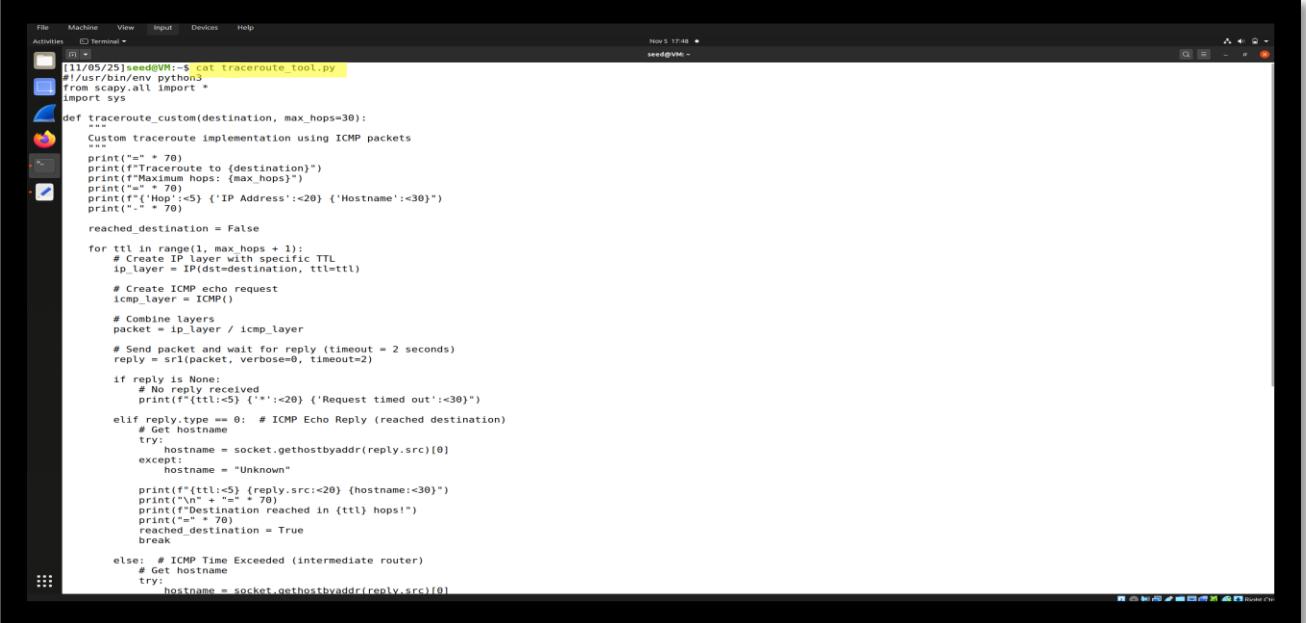
Screenshot 16: Create the Automated Traceroute Script- “traceroute_tool.py”.



```
File Machine View Input Devices Help
Activities Text Editor
Untitled Document 1
Nov 5 17:42
34   print(f"(ttl:<5) {'*'<20} {'Request timed out':<30}")
35
36   elif reply.type == 0: # ICMP Echo Reply (reached destination)
37       # Get hostname
38       try:
39           hostname = socket.gethostbyaddr(reply.src)[0]
40       except:
41           hostname = "Unknown"
42
43       print(f"(ttl:<5) {reply.src:<20} {hostname:<30}")
44       print("\n" + "=" * 70)
45       print(f"Destination reached in {ttl} hops!")
46       print("*" * 70)
47       reached_destination = True
48       break
49
50   else: # ICMP Time Exceeded (intermediate router)
51       # Get hostname
52       try:
53           hostname = socket.gethostbyaddr(reply.src)[0]
54       except:
55           hostname = "Unknown"
56
57       print(f"(ttl:<5) {reply.src:<20} {hostname:<30}")
58
59   if not reached_destination:
60       print("\n" + "=" * 70)
61       print(f"Destination not reached within {max_hops} hops")
62       print("*" * 70)
63
64   return ttl if reached_destination else max_hops
65
66if __name__ == "__main__":
67    print("\n" + "=" * 70)
68    print("Custom Traceroute Tool - Task 1.3")
69    print("\n" + "=" * 70)
70
71    # Get destination from user
72    if len(sys.argv) > 1:
73        destination = sys.argv[1]
74    else:
75        destination = input("\nEnter destination IP or hostname (default: 8.8.8.8): ").strip()
76    if not destination:
77        destination = "8.8.8.8"
78
79    print("\nStarting traceroute to {destination}...\n")
80
81    # Perform traceroute
82    traceroute_custom(destination, max_hops=30)
83
84    print("\nTraceroute complete!\n")
85EOF
```

Screenshot 17: Create the Automated Traceroute Script- “traceroute_tool.py”.

- So, we can verify the code once using “cat traceroute_tool.py” as highlighted in screenshot 18.



```
[11/05/25]seed@VM:-$ cat traceroute_tool.py
#!/usr/bin/env python3
from scapy.all import *
import sys

def traceroute_custom(destination, max_hops=30):
    """
    Custom traceroute implementation using ICMP packets
    """
    print("*" * 70)
    print(f"Traceroute to {destination}")
    print(f"Maximum hops: {max_hops}")
    print("*" * 70)
    print(f"({Hop}:<5) ('IP Address':<20) ('Hostname':<30)")
    print("-" * 70)

    reached_destination = False

    for ttl in range(1, max_hops + 1):
        # Create IP layer with specific TTL
        ip_layer = IP(dst=destination, ttl=ttl)

        # Create ICMP echo request
        icmp_layer = ICMP()

        # Combine layers
        packet = ip_layer / icmp_layer

        # Send packet and wait for reply (timeout = 2 seconds)
        reply = sr1(packet, verbose=0, timeout=2)

        if reply is None:
            # No reply received
            print(f"({ttl}:<5) (*:<20) ('Request timed out':<30)")

        elif reply.type == 0: # ICMP Echo Reply (reached destination)
            try:
                hostname = socket.gethostbyaddr(reply.src)[0]
            except:
                hostname = "Unknown"

            print(f"({ttl}:<5) ({reply.src:<20}) ({hostname:<30})")
            print("\n" * 70)
            print(f"Destination reached in {ttl} hops!")
            print("-" * 70)
            reached_destination = True
            break

        else: # ICMP Time Exceeded (intermediate router)
            try:
                hostname = socket.gethostbyaddr(reply.src)[0]
            except:
                hostname = "Unknown"

            print(f"({ttl}:<5) ({reply.src:<20}) ({hostname:<30})")
            print("\n" * 70)
            print(f"Destination reached in {ttl} hops!")
            print("-" * 70)
            reached_destination = True
            break

    else: # Get hostname
        try:
            hostname = socket.gethostbyaddr(reply.src)[0]
        except:
            hostname = "Unknown"

    print(f"({ttl}:<5) ({reply.src:<20}) ({hostname:<30})")
    print("\n" * 70)
    print(f"Destination reached in {ttl} hops!")
    print("-" * 70)
    reached_destination = True
    break

    else: # Get hostname
        try:
            hostname = socket.gethostbyaddr(reply.src)[0]
        except:
            hostname = "Unknown"

    print(f"({ttl}:<5) ({reply.src:<20}) ({hostname:<30})")
    print("\n" * 70)
    print(f"Destination reached in {ttl} hops!")
    print("-" * 70)
    reached_destination = True
    break
```

Screenshot 18: Verifying code using “cat traceroute_tool.py”.

- After that we make it executable using “chmod +x traceroute_tool.py” (highlighted below) and also use this to run the tool by using command “sudo python3 traceroute_tool.py” as highlighted in screenshot 19 and enter default google’s dns 8.8.8.8. This is how we will run the overall tool.

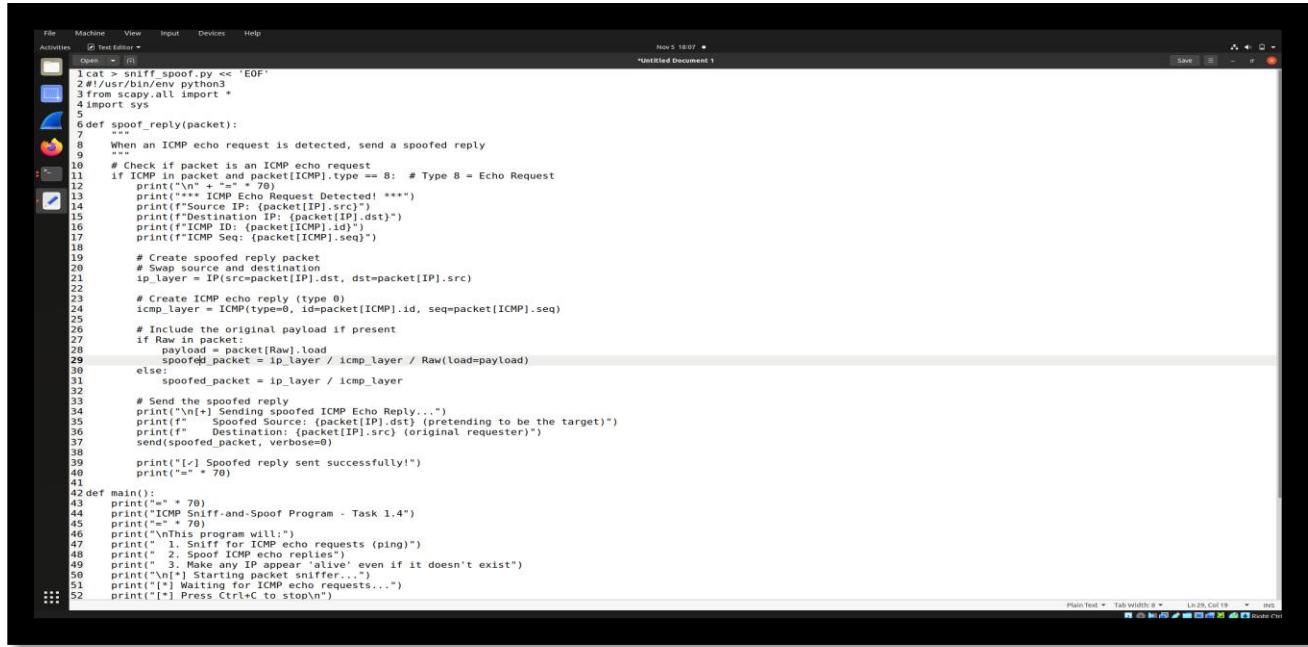


```
[11/05/25]seed@VM:-$ chmod +x traceroute_tool.py
[11/05/25]seed@VM:-$ sudo python3 traceroute_tool.py
Custom Traceroute Tool - Task 1.3
=====
Enter destination IP or hostname (default: 8.8.8.8):
Starting traceroute to 8.8.8.8...
=====
Traceroute to 8.8.8.8
Maximum hops: 30
=====
Hop IP Address Hostname
1 8.8.8.8 dns.google
=====
Destination reached in 1 hops!
=====
Traceroute complete!
[11/05/25]seed@VM:-$
```

Screenshot 19: Running the tool using “sudo python3 traceroute_tool.py” command.

Task 1.4

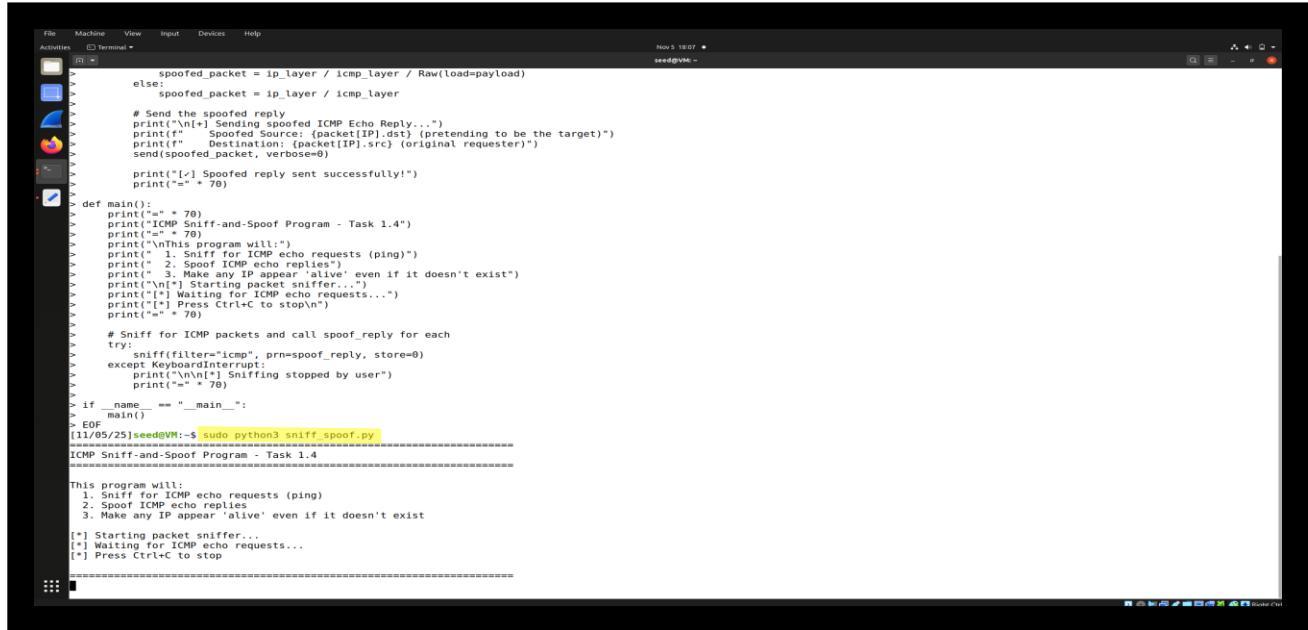
- Firstly, we will create the sniff and spoof program by creating file “sniff_spoof.py” and implementing code as observed in screenshot 20.



```
File Machine View Input Devices Help
Activities Text Editor * Untitled Document 1
Nov 5 10:07
Open: /home/seed/sniff_spoof.py << EOF
#!/usr/bin/python
from scapy.all import *
import sys
#def spoof_reply(packet):
#    """
#    When an ICMP echo request is detected, send a spoofed reply
#    """
#    if ICMP in packet and packet[ICMP].type == 8: # Type 8 = Echo Request
#        print("\n[*] -" * 70)
#        print("[" * 70 + " ICMP Echo Request Detected! ]")
#        print("Source IP: (packet[IP].src)")
#        print("Destination IP: (packet[IP].dst)")
#        print("ICMP ID: (packet[ICMP].id)")
#        print("ICMP Seq: (packet[ICMP].seq)")
#
#    # Create spoofed reply packet
#    # Swap source and destination
#    ip_layer = IP(src=packet[IP].dst, dst=packet[IP].src)
#
#    # Create ICMP echo reply (type 0)
#    icmp_layer = ICMP(type=0, id=packet[ICMP].id, seq=packet[ICMP].seq)
#
#    # Include the original payload if present
#    if Raw in packet:
#        Raw[ICMP].load = packet[Raw].load
#        spoofed_packet = ip_layer / icmp_layer / Raw(load=packet[Raw].load)
#    else:
#        spoofed_packet = ip_layer / icmp_layer
#
#    # Send the spoofed reply
#    print("\n[*] Sending spoofed ICMP Echo Reply...")
#    print("[" * 70 + " Spoofed Source: (packet[IP].dst) (pretending to be the target)"]
#    print("[" * 70 + " Destination: (packet[IP].src) (original requester)"]
#    send(spoofed_packet, verbose=0)
#
#    print("[*] Spoofed reply sent successfully!")
#    print("-" * 70)
#def main():
#    print("-" * 70)
#    print("ICMP Sniff-and-Spoof Program - Task 1.4")
#    print("-" * 70)
#    print("[" * 70 + " In this program will:")
#    print("[" * 70 + " 1. Sniff for ICMP echo requests (ping)")
#    print("[" * 70 + " 2. Spoof ICMP echo replies")
#    print("[" * 70 + " 3. Make any IP appear 'alive' even if it doesn't exist")
#    print("[" * 70 + " Starting packet sniffer...")
#    print("[" * 70 + " Waiting for ICMP echo requests...")
#    print("[" * 70 + " Press Ctrl+C to stop\n")
#    print("-" * 70)
#
#    # Sniff for ICMP packets and call spoof_reply for each
#    try:
#        sniff(filter="icmp", prn=spoof_reply, store=0)
#    except KeyboardInterrupt:
#        print("\n[*] Sniffing stopped by user")
#        print("-" * 70)
#
#    if __name__ == "__main__":
#        main()
#    EOF
[11/05/25]seed@VM:~$ sudo python3 sniff_spoof.py
=====
ICMP Sniff-and-Spoof Program - Task 1.4
=====
This program will:
1. Sniff for ICMP echo requests (ping)
2. Spoof ICMP echo replies
3. Make any IP appear 'alive' even if it doesn't exist
[*] Starting packet sniffer...
[*] Waiting for ICMP echo requests...
[*] Press Ctrl+C to stop
=====
```

Screenshot 20: Code used to create sniff and spoof program- “sniff_spoof.py”.

- Then we will run this program using command “sudo python3 sniff_spoof.py” as highlighted in screenshot 21.



```
File Machine View Input Devices Help
Activities Terminal * seed@VM:~$
File Machine View Input Devices Help
Activities Terminal * seed@VM:~$ sudo python3 sniff_spoof.py
=====
ICMP Sniff-and-Spoof Program - Task 1.4
=====
This program will:
1. Sniff for ICMP echo requests (ping)
2. Spoof ICMP echo replies
3. Make any IP appear 'alive' even if it doesn't exist
[*] Starting packet sniffer...
[*] Waiting for ICMP echo requests...
[*] Press Ctrl+C to stop
=====
```

Screenshot 21: To run the program using command “sudo python3 sniff_spoof.py”.

- Then, we will test three different scenarios, first one is of testing non-existing host on the internet by using “ping -c 4 1.2.3.4” as highlighted in screenshot 22 and we can observe the result of it in screenshot 23.

```
[11/05/25]seed@VM:~$ ping -c 4 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=17.9 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=16.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=18.2 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=20.9 ms
[11/05/25]seed@VM:~$
```

Screenshot 22: Executing ping command “ping -c 4 1.2.3.4”.

```
[11/05/25]seed@VM:~$ sudo python3 sniff_spoof.py
=====
ICMP Sniff-and-Spoof Program - Task 1.4
=====

This program will:
1. Sniff for ICMP echo requests (ping)
2. Spoof ICMP echo replies
3. Make any IP appear 'alive' even if it doesn't exist

[*] Starting packet sniffer...
[*] Waiting for ICMP echo requests...
[*] Press Ctrl+c to stop
=====

*** ICMP Echo Request Detected!
Source IP: 10.0.2.15
Destination IP: 1.2.3.4
ICMP ID: 1
ICMP Seq: 1

[+] Sending spoofed ICMP Echo Reply...
Spoofed Source: 1.2.3.4 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!

=====

*** ICMP Echo Request Detected!
Source IP: 10.0.2.15
Destination IP: 1.2.3.4
ICMP ID: 1
ICMP Seq: 2

[+] Sending spoofed ICMP Echo Reply...
Spoofed Source: 1.2.3.4 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!

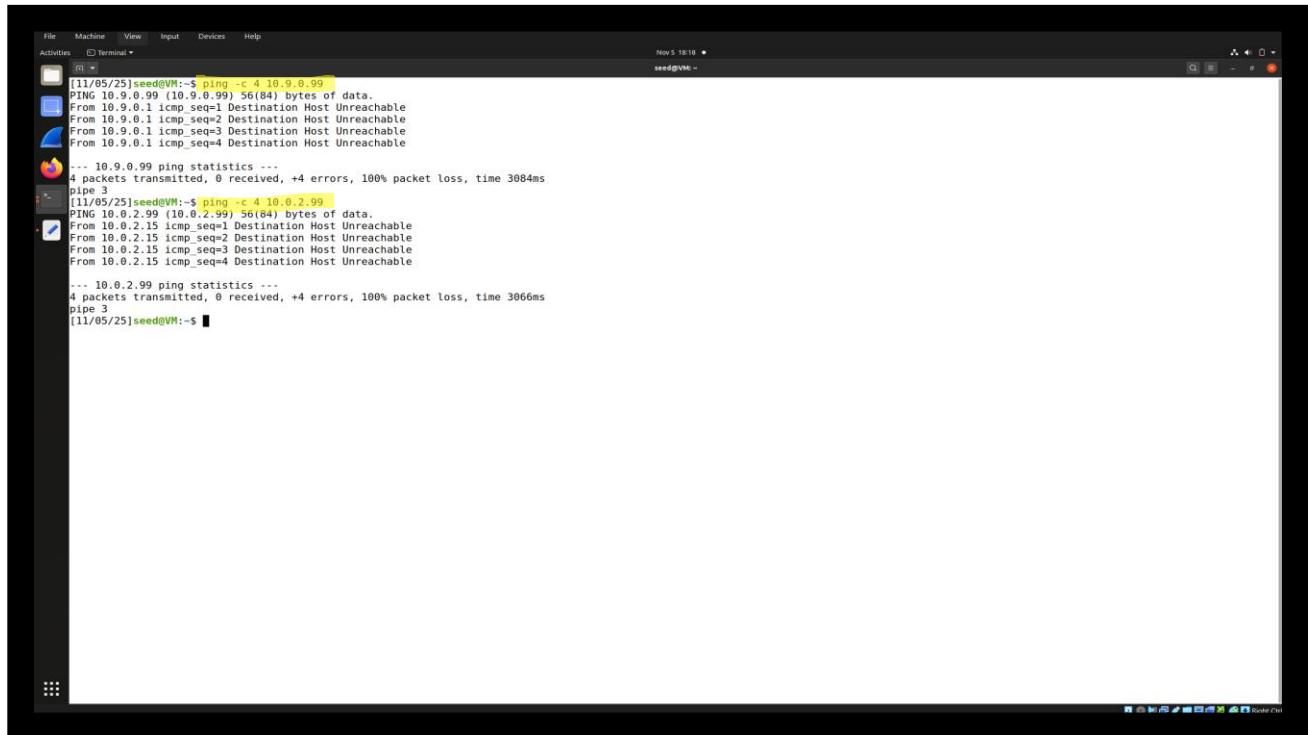
=====

*** ICMP Echo Request Detected!
Source IP: 10.0.2.15
Destination IP: 1.2.3.4
ICMP ID: 1
ICMP Seq: 3

[+] Sending spoofed ICMP Echo Reply...
Spoofed Source: 1.2.3.4 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!
```

Screenshot 23: Executing “sudo python3 sniff_spoof.py” to observe result from ping.

- Second one is of testing non-existing host on the LAN by using “ping -c 4 10.0.2.99” as highlighted in screenshot 24 and we can observe the result of it in screenshot 25.

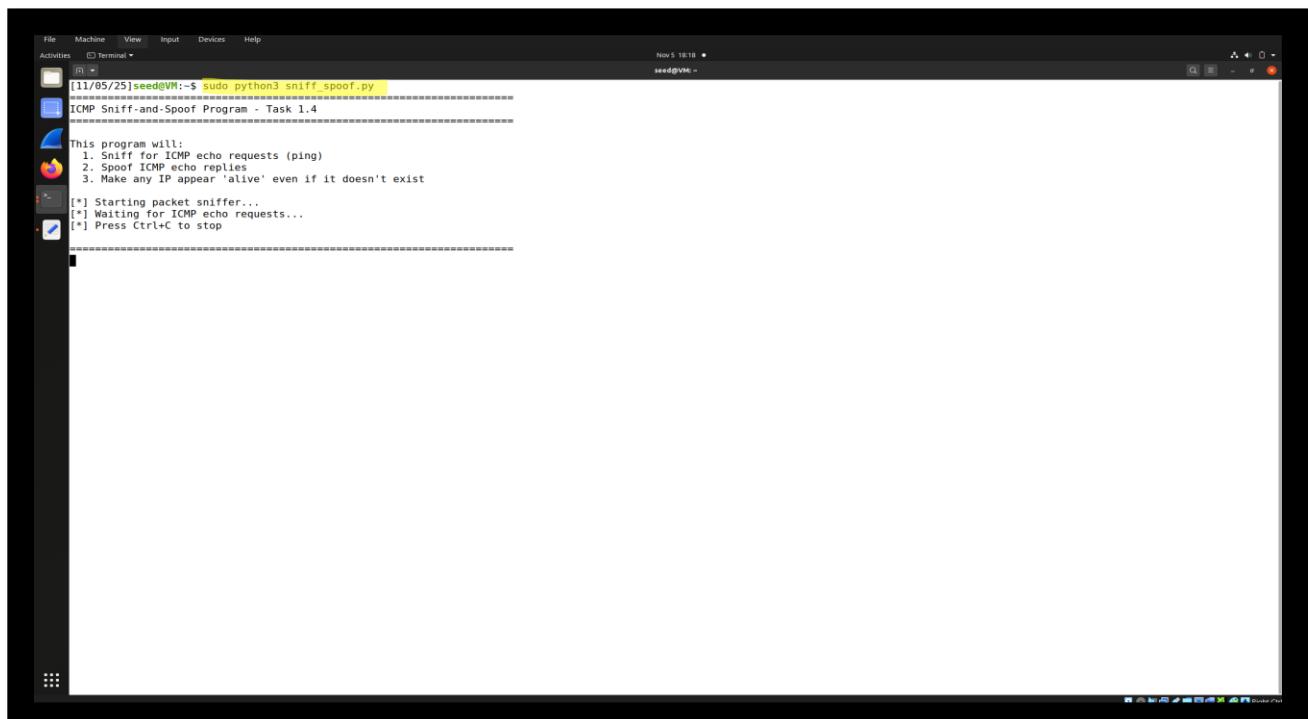


```
[11/05/25]seed@VM:~$ ping -c 4 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable

... 10.9.0.99 ping statistics ...
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3084ms
pipe 3
[11/05/25]seed@VM:~$ ping -c 4 10.0.2.99
PING 10.0.2.99 (10.0.2.99) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
From 10.0.2.15 icmp_seq=2 Destination Host Unreachable
From 10.0.2.15 icmp_seq=3 Destination Host Unreachable
From 10.0.2.15 icmp_seq=4 Destination Host Unreachable

... 10.0.2.99 ping statistics ...
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3066ms
pipe 3
[11/05/25]seed@VM:~$
```

Screenshot 24: Executing ping command “ping -c 4 10.0.2.99”.



```
[11/05/25]seed@VM:~$ sudo python3 sniff_spoof.py
=====
ICMP Sniff-and-Spoof Program - Task 1.4
=====
This program will:
 1. Sniff for ICMP echo requests (ping)
 2. Spoof ICMP echo replies
 3. Make any IP appear 'alive' even if it doesn't exist
[*] Starting packet sniffer...
[*] Waiting for ICMP echo requests...
[*] Press Ctrl+C to stop
=====
```

Screenshot 25: Executing “sudo python3 sniff_spoof.py” to observe result from ping.

- Third one is of testing existing host on the internet by using “ping -c 4 8.8.8.8” as highlighted in screenshot 26 and we can observe the result of it in screenshot 27.

```
[11/05/25]seed@VM:~$ ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=18.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=12.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=16.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=13.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=14.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=12.0 ms

... 8.8.8.8 ping statistics ...
4 packets transmitted, 4 received, +3 duplicates, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 11.952/14.735/18.090/2.028 ms
[11/05/25]seed@VM:~$
```

Screenshot 26: Executing ping command “ping -c 4 8.8.8.8”.

```
[11/05/25]seed@VM:~$ sudo python3 sniff_spoof.py
=====
ICMP Sniff-and-Spoof Program - Task 1.4
=====

This program will:
1. Sniff for ICMP echo requests (ping)
2. Spoof ICMP echo replies
3. Make any IP appear 'alive' even if it doesn't exist

[*] Starting packet sniffer...
[*] Waiting for ICMP echo requests...
[*] Press Ctrl+C to stop

=====
*** ICMP Echo Request Detected! ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP ID: 6
ICMP Seq: 1

[*] Sending spoofed ICMP Echo Reply...
Spoofed Source: 8.8.8.8 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!

=====
*** ICMP Echo Request Detected! ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP ID: 6
ICMP Seq: 2

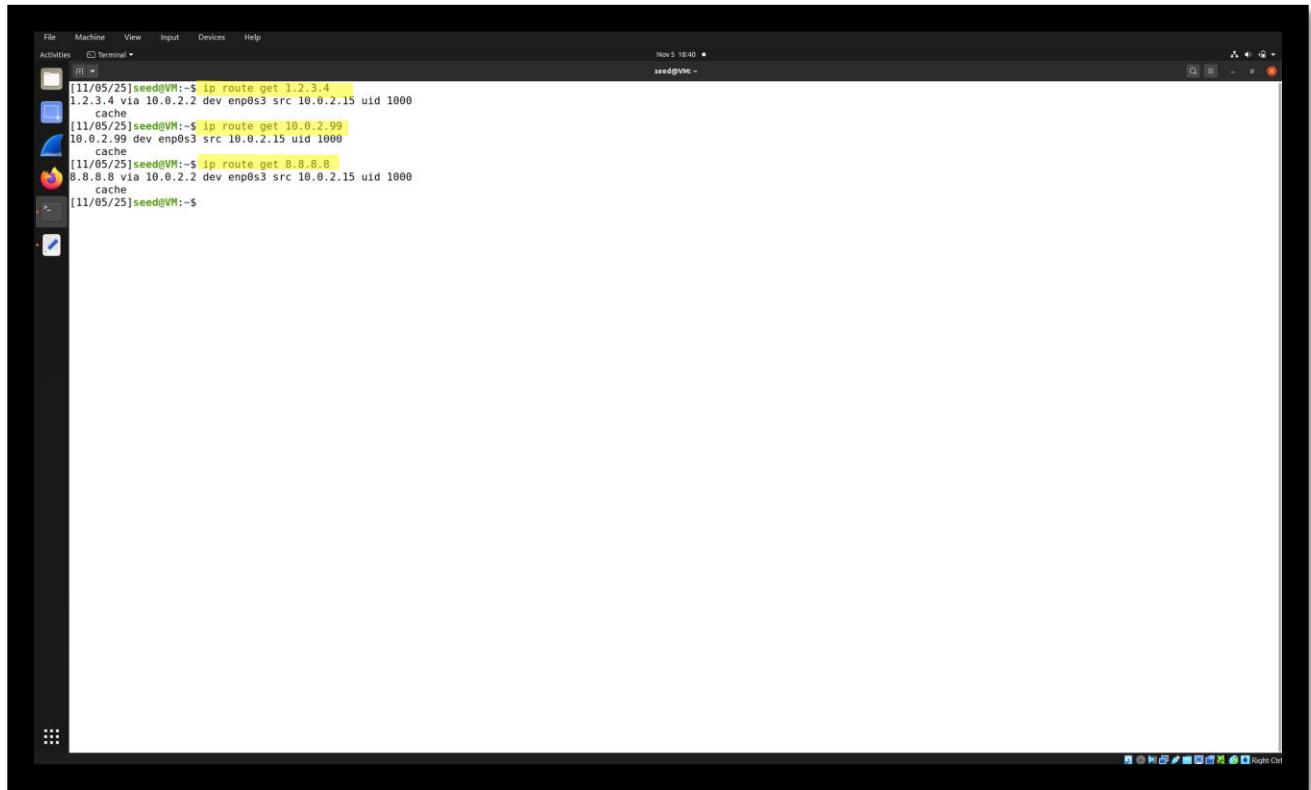
[*] Sending spoofed ICMP Echo Reply...
Spoofed Source: 8.8.8.8 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!

=====
*** ICMP Echo Request Detected! ***
Source IP: 10.0.2.15
Destination IP: 8.8.8.8
ICMP ID: 6
ICMP Seq: 3

[*] Sending spoofed ICMP Echo Reply...
Spoofed Source: 8.8.8.8 (pretending to be the target)
Destination: 10.0.2.15 (original requester)
[!] Spoofed reply sent successfully!
```

Screenshot 27: Executing “sudo python3 sniff_spoof.py” to observe result from ping.

- After that to verify the path where the packet to this IP address, we utilize “ip route get” which is highlighted in screenshot 28.



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window has a dark background and light-colored text. It displays three separate command-line entries, each starting with 'seed@VM:~\$ ip route get'. The first command is 'ip route get 1.2.3.4', the second is 'ip route get 10.0.2.99', and the third is 'ip route get 8.8.8.8'. Each command shows the route path from the source interface 'enp0s3' to the destination IP address via the gateway '10.0.2.15'. The terminal window is titled 'seed@VM:~\$' and is located in the 'Activities' workspace. The desktop interface includes a dock at the bottom with icons for various applications like a browser, file manager, and terminal.

```
[11/05/25]seed@VM:~$ ip route get 1.2.3.4
1.2.3.4 via 10.0.2.2 dev enp0s3 src 10.0.2.15 uid 1000
    cache
[11/05/25]seed@VM:~$ ip route get 10.0.2.99
10.0.2.99 dev enp0s3 src 10.0.2.15 uid 1000
    cache
[11/05/25]seed@VM:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.2 dev enp0s3 src 10.0.2.15 uid 1000
    cache
[11/05/25]seed@VM:~$
```

Screenshot 28: Utilizing “ip route get” to verify the full route of packets.