CSCE4930 - Network Security

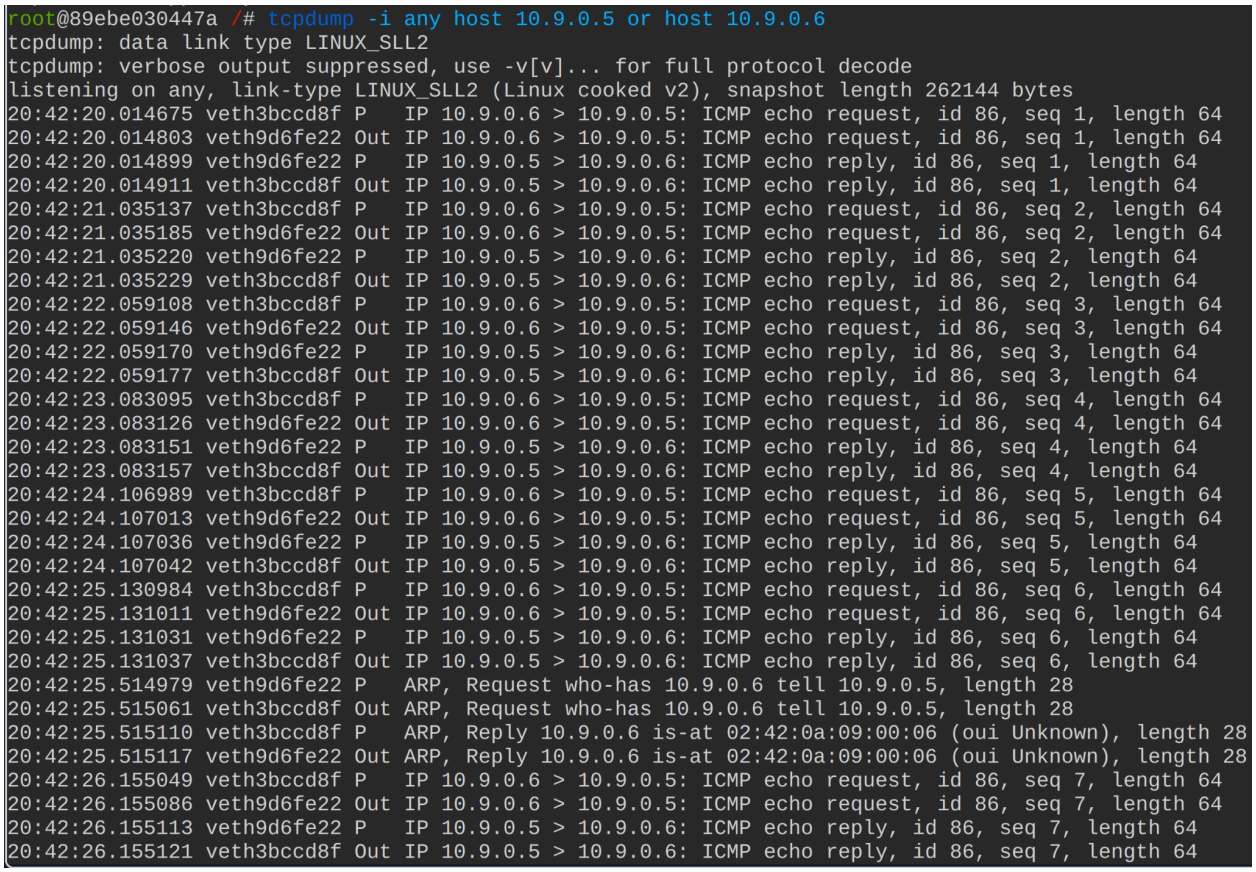
Assignment 1

Name: Mario Ghaly

ID: 900202178

1. Packet Sniffing

1. The user is pinging the victim, and the attacker can sniff all types of packets.



```
root@89ebe030447a /# tcpdump -i any host 10.9.0.5 or host 10.9.0.6
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
20:42:20.014675 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 1, length 64
20:42:20.014803 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 1, length 64
20:42:20.014899 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 1, length 64
20:42:20.014911 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 1, length 64
20:42:21.035137 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 2, length 64
20:42:21.035185 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 2, length 64
20:42:21.035220 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 2, length 64
20:42:21.035229 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 2, length 64
20:42:22.059108 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 3, length 64
20:42:22.059146 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 3, length 64
20:42:22.059170 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 3, length 64
20:42:22.059177 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 3, length 64
20:42:23.083095 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 4, length 64
20:42:23.083126 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 4, length 64
20:42:23.083151 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 4, length 64
20:42:23.083157 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 4, length 64
20:42:24.106989 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 5, length 64
20:42:24.107013 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 5, length 64
20:42:24.107036 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 5, length 64
20:42:24.107042 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 5, length 64
20:42:25.130984 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 6, length 64
20:42:25.131011 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 6, length 64
20:42:25.131031 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 6, length 64
20:42:25.131037 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 6, length 64
20:42:25.514979 veth9d6fe22 P   ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
20:42:25.515061 veth3bccd8f Out ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
20:42:25.515110 veth3bccd8f P   ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06 (oui Unknown), length 28
20:42:25.515117 veth9d6fe22 Out ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06 (oui Unknown), length 28
20:42:26.155049 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 7, length 64
20:42:26.155086 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 86, seq 7, length 64
20:42:26.155113 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 7, length 64
20:42:26.155121 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 86, seq 7, length 64
```

2. This command filters the packets to be of type ICMP only, and this is why the ARP packets no longer appear.

```
root@89ebe030447a / [1]# tcpdump -i any icmp and \( host 10.9.0.5 or host 10.9.0.6 \)

tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
21:02:13.676194 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 1, length 64
21:02:13.676541 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 1, length 64
21:02:13.676690 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 1, length 64
21:02:13.676727 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 1, length 64
21:02:14.699037 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 2, length 64
21:02:14.699086 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 2, length 64
21:02:14.699149 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 2, length 64
21:02:14.699165 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 2, length 64
21:02:15.723076 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 3, length 64
21:02:15.723114 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 3, length 64
21:02:15.723146 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 3, length 64
21:02:15.723154 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 3, length 64
21:02:16.747011 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 4, length 64
21:02:16.747051 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 4, length 64
21:02:16.747083 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 4, length 64
21:02:16.747094 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 4, length 64
21:02:17.771040 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 5, length 64
21:02:17.771081 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 5, length 64
21:02:17.771113 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 5, length 64
21:02:17.771121 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 5, length 64
21:02:18.795064 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 6, length 64
21:02:18.795129 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 6, length 64
21:02:18.795161 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 6, length 64
21:02:18.795172 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 6, length 64
21:02:19.819250 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 7, length 64
21:02:19.819328 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 88, seq 7, length 64
21:02:19.819361 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 7, length 64
21:02:19.819371 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 88, seq 7, length 64
```

3. This use case shows a TCP connection between 10.9.0.6 using Telnet with 10.9.0.5 as a destination on port 23. The first command can sniff this and it outputs the ARP packets in addition, but the new command used here is to filter the sniffing only on TCP packets destined for port 23, which is Telnet.

   Victim Side>>

```
root@330c866e11c4 /# nc -lvnp 23

Listening on 0.0.0.0 23
Connection received on 10.9.0.6 35440
 !"'hi
```

   User Side>>

```
root@d77e9d03bb98 /# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
hi
```

Attacker Side>>

```
root@89ebe030447a /# tcpdump -i any tcp port 23
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
22:27:13.476290 veth3bccd8f P   IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [S], seq 2526265313, win 64240, options [mss 1460,sackOK,TS val 119627599 ecr
 0,nop,wscale 7], length 0
22:27:13.476478 veth9d6fe22 Out IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [S], seq 2526265313, win 64240, options [mss 1460,sackOK,TS val 119627599 ecr
 0,nop,wscale 7], length 0
22:27:13.476731 veth9d6fe22 P   IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [S.], seq 451343991, ack 2526265314, win 65160, options [mss 1460,sackOK,TS v
al 933078595 ecr 119627599,nop,wscale 7], length 0
22:27:13.476750 veth3bccd8f Out IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [S.], seq 451343991, ack 2526265314, win 65160, options [mss 1460,sackOK,TS v
al 933078595 ecr 119627599,nop,wscale 7], length 0
22:27:13.476806 veth3bccd8f P   IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [.], ack 1, win 502, options [nop,nop,TS val 119627599 ecr 933078595], length
 0
22:27:13.476815 veth9d6fe22 Out IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [.], ack 1, win 502, options [nop,nop,TS val 119627599 ecr 933078595], length
 0
22:27:13.477073 veth3bccd8f P   IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [P.], seq 1:25, ack 1, win 502, options [nop,nop,TS val 119627600 ecr 9330785
95], length 24 [telnet DO SUPPRESS GO AHEAD, WILL TERMINAL TYPE, WILL NAWS, WILL TSPEED, WILL LFLOW, WILL LINEMODE, WILL NEW-ENVIRON, DO STATUS]
22:27:13.477085 veth9d6fe22 Out IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [P.], seq 1:25, ack 1, win 502, options [nop,nop,TS val 119627600 ecr 9330785
95], length 24 [telnet DO SUPPRESS GO AHEAD, WILL TERMINAL TYPE, WILL NAWS, WILL TSPEED, WILL LFLOW, WILL LINEMODE, WILL NEW-ENVIRON, DO STATUS]
22:27:13.477113 veth9d6fe22 P   IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [.], ack 25, win 509, options [nop,nop,TS val 933078596 ecr 119627600], lengt
h 0
22:27:13.477119 veth3bccd8f Out IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [.], ack 25, win 509, options [nop,nop,TS val 933078596 ecr 119627600], lengt
h 0
22:27:38.353814 veth3bccd8f P   IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [P.], seq 25:29, ack 1, win 502, options [nop,nop,TS val 119652476 ecr 933078
596], length 4
22:27:38.353875 veth9d6fe22 Out IP 10.9.0.6.60914 > 10.9.0.5.telnet: Flags [P.], seq 25:29, ack 1, win 502, options [nop,nop,TS val 119652476 ecr 933078
596], length 4
22:27:38.353992 veth9d6fe22 P   IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [.], ack 29, win 509, options [nop,nop,TS val 933103473 ecr 119652476], lengt
h 0
22:27:38.354013 veth3bccd8f Out IP 10.9.0.5.telnet > 10.9.0.6.60914: Flags [.], ack 29, win 509, options [nop,nop,TS val 933103473 ecr 119652476], lengt
h 0
```

The first 6 lines are for establishing the Telnet session using the TCP 3 way handshake, and the 4 lines after are for defining how the data would be sent, and the final 4 lines are for pushing data and acknowledging it.

4. Running this command gives the subnet to be 24.

```
root@89ebe030447a /# ifconfig
br-0e2b115c61bd: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
```

The command below filters the sniffed packets to be from or to the attacker's subnet. When the user pings the victim (or any IP address), the ARP and ICMP packets are still shown since both the user and victim have the same subnet.

```
root@89ebe030447a /# tcpdump -i any net 10.9.0.0/24

tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
22:53:36.498280 veth3bccd8f B   ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
22:53:36.498303 veth9d6fe22 Out ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
22:53:36.498280 br-0e2b115c61bd B   ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
22:53:36.498351 veth9d6fe22 P   ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05 (oui Unknown), length 28
22:53:36.498355 veth3bccd8f Out ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05 (oui Unknown), length 28
22:53:36.498364 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 1, length 64
22:53:36.498425 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 1, length 64
22:53:36.498496 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 1, length 64
22:53:36.498505 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 1, length 64
22:53:37.515076 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 2, length 64
22:53:37.515105 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 2, length 64
22:53:37.515139 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 2, length 64
22:53:37.515150 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 2, length 64
22:53:38.539047 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 3, length 64
22:53:38.539077 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 3, length 64
22:53:38.539102 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 3, length 64
22:53:38.539109 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 3, length 64
22:53:39.563111 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 4, length 64
22:53:39.563162 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 4, length 64
22:53:39.563200 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 4, length 64
22:53:39.563214 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 4, length 64
22:53:40.587136 veth3bccd8f P   IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 5, length 64
22:53:40.587188 veth9d6fe22 Out IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 104, seq 5, length 64
22:53:40.587228 veth9d6fe22 P   IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 5, length 64
22:53:40.587238 veth3bccd8f Out IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 104, seq 5, length 64
```

2. Packet Spoofing

1. On the attacker side, the first command generates and sends packets with a spoofed source IP address, that has been noticed from sniffing in the previous task (in this case, it is the user 10.9.0.6).

```
root@89ebe030447a / [127]# python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> packet = IP(src = "10.9.0.6", dst = "10.9.0.5") / ICMP()
>>> send(packet)
.
Sent 1 packets.
```

2. The following command is run on the victim side to capture ICMP packets leaving or coming to their machine, and as can be seen, it seems that 10.9.0.6 is pinging 10.9.0.5 although the attacker is the one who sent it.

```
root@8457a5571474 /# tcpdump -i eth0 icmp -nn
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
21:45:44.370552 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
21:45:44.370592 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 0, seq 0, length 8
```

Comparing the spoofed packets and normal packets

```
root@8457a5571474 /# tcpdump -i eth0 icmp -nn
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
21:45:44.370552 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
21:45:44.370592 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 0, seq 0, length 8
21:54:31.795707 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 283, seq 1, length 64
21:54:31.795825 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 283, seq 1, length 64
21:54:32.811485 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id 283, seq 2, length 64
21:54:32.811540 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 283, seq 2, length 64
```

It is worth mentioning that the packet sent with a spoofed source IP address has an ID and *sequence number* both equal to 0 since they were not specified. Thus, for the networks to detect spoofing, they may validate if the sequence number was not sequential after the last sequence number of the packet sent form the actual source address because

if the *seq* was not incremental, this might mean that the source IP address has been spoofed. Also, the ID is unique per ICMP session, so it could help in detecting spoofing; however, the attacker can spoof the ID as well.

3. This time, the attacker spoofs a made up source IP address

```
root@89ebe030447a /# python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> packet = IP(src = "193.154.145.12", dst = "10.9.0.5") / ICMP()
>>> send(packet)
.
Sent 1 packets.
```

Victim Side>>

```
root@8457a5571474 /# tcpdump -i eth0 icmp -nn
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
22:12:05.424716 IP 193.154.145.12 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
22:12:05.424865 IP 10.9.0.5 > 193.154.145.12: ICMP echo reply, id 0, seq 0, length 8
```

If the attacker tries to spoof google.com source address, for example:

Attacker User Victim

```
root@89ebe030447a /# python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> import random
>>> packet = IP(src = "8.8.8.8", dst = "10.9.0.5") / ICMP(id = random.randint(0, 65000), seq = random.randint(0, 65000))
>>> send(packet)
.
Sent 1 packets.
>>>
```

Victim Side>>

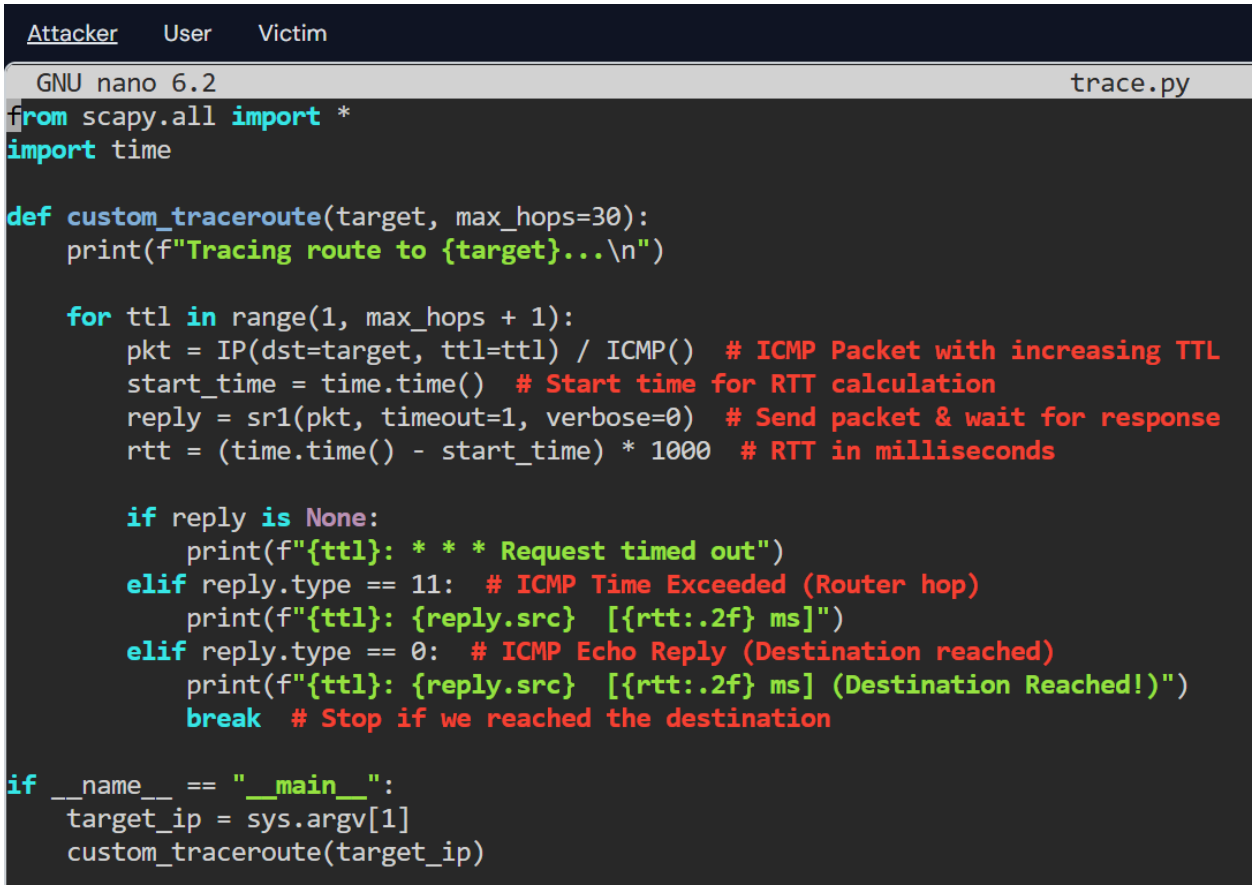
Attacker User Victim

```
root@8457a5571474 /# tcpdump -i eth0 icmp -nn
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
22:17:52.527935 IP 8.8.8.8 > 10.9.0.5: ICMP echo request, id 5365, seq 34140, length 8
22:17:52.528054 IP 10.9.0.5 > 8.8.8.8: ICMP echo reply, id 5365, seq 34140, length 8
```

Which means that the attacker can generate and send packets with any IP address, with specifying the *ID* and *seq* if needed

3. Traceroute Analysis

1. The implemented traceroute tool using scrapy is shown in the code below. TTL is the field preventing infinite loops in routers. Each router forwarding the packet decreases the TTL by 1, and when its value reaches 0, the router drops the packet and sends an ICMP reply indicating “Time Exceeded.” Thus, when sending a packet to a destination, the TTL field can be manipulated and specified as 1 in the beginning to trace the first hop it reached. Then, increment by 1 so that the packet can now travel to 2 routers instead of only 1, and so on until it reaches the destination. And each time TTL reaches 0, the sender will get the ICMP time exceeded reply and output the IP address of that hop and the trip round time.



This is an example of traceroute to Google.com

```
Attacker    User    Victim
root@89ebe030447a /lab1# python3 trace.py 8.8.8.8
Tracing route to 8.8.8.8...

1: 10.0.111.1  [63.55 ms]
2: 10.253.2.101  [25.10 ms]
3: 10.0.66.2  [32.32 ms]
4: 213.248.70.176  [32.53 ms]
5: 62.115.136.194  [28.01 ms]
6: 62.115.136.146  [25.82 ms]
7: 62.115.124.117  [40.22 ms]
8: 62.115.151.27  [29.13 ms]
9: 108.170.236.173  [34.33 ms]
10: 142.250.214.193  [27.35 ms]
11: 8.8.8.8  [28.48 ms] (Destination Reached!)
```

2. This is the standard traceroute command, and as can be seen, there are some similar hops in the way, but the standard command is much faster and shorter in number of hops

```
root@89ebe030447a /lab1# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  vmi2489842 (10.0.111.1)  0.222 ms  0.049 ms  0.049 ms
 2  10.253.1.101 (10.253.1.101)  0.935 ms  0.733 ms 10.253.2.101 (10.253.2.101)  0.544 ms
 3  10.0.66.2 (10.0.66.2)  0.422 ms 10.0.66.1 (10.0.66.1)  0.443 ms 10.0.66.2 (10.0.66.2)  0.350 ms
 4  laut-b1-link.ip.twelve99.net (213.248.67.10)  3.814 ms 212.133.82.97 (212.133.82.97)  8.432 ms laut-b1-link.ip.twelve99.net (213.248.67.10)  3
.485 ms
 5  laut-b1-link.ip.twelve99.net (62.115.136.194)  3.374 ms ae1.3121.edge8.frf1.neo.colt.net (171.75.8.25)  4.857 ms  4.736 ms
 6  142.250.165.106 (142.250.165.106)  5.920 ms 15169-3356-par.sp.lumen.tech (4.68.71.138)  7.485 ms *
 7  * google-ic-324085.ip.twelve99-cust.net (62.115.153.213)  3.107 ms google-ic-319726.ip.twelve99-cust.net (62.115.151.25)  2.934 ms
 8  dns.google (8.8.8.8)  7.342 ms  3.678 ms 142.251.64.181 (142.251.64.181)  14.479 ms
```

Using this method, an attacker can map routers, gateways, and firewalls along the path to the destination, identifying the network architecture. For example, if a hop suddenly stops responding, it could indicate a firewall or security filter blocking specific traffic.