**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.

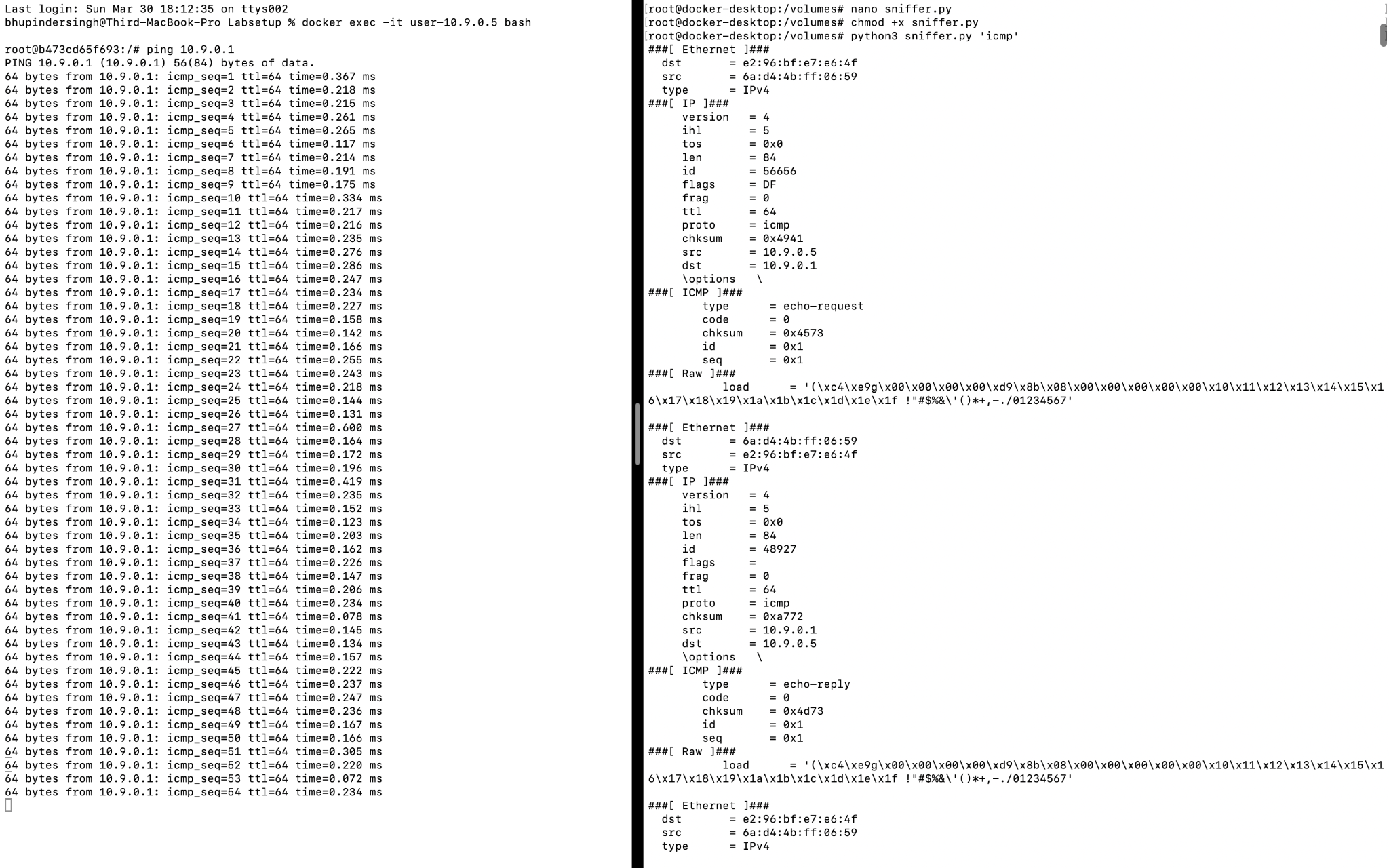
**Task N2.a : Capture only ICMP packets**

**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.

**Figure 1: Screenshot of N2.A**

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

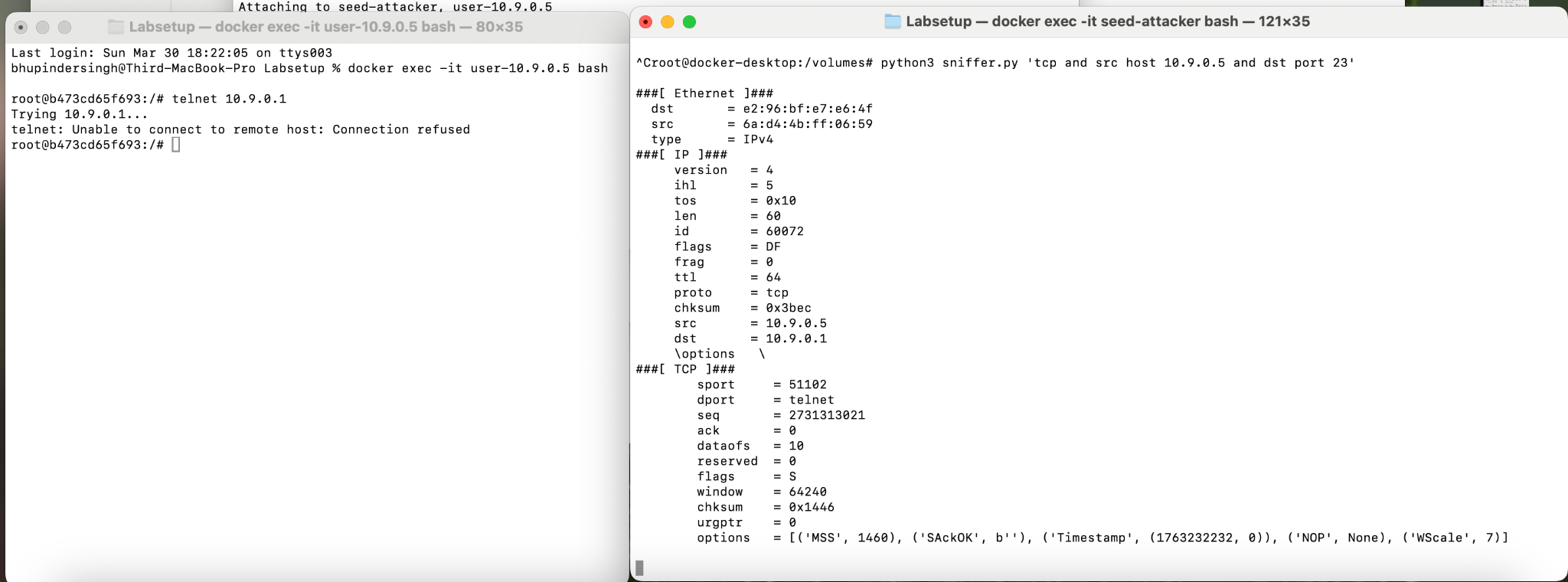
**Filter used:**

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.



**Figure 2: Screenshot of N2.B**

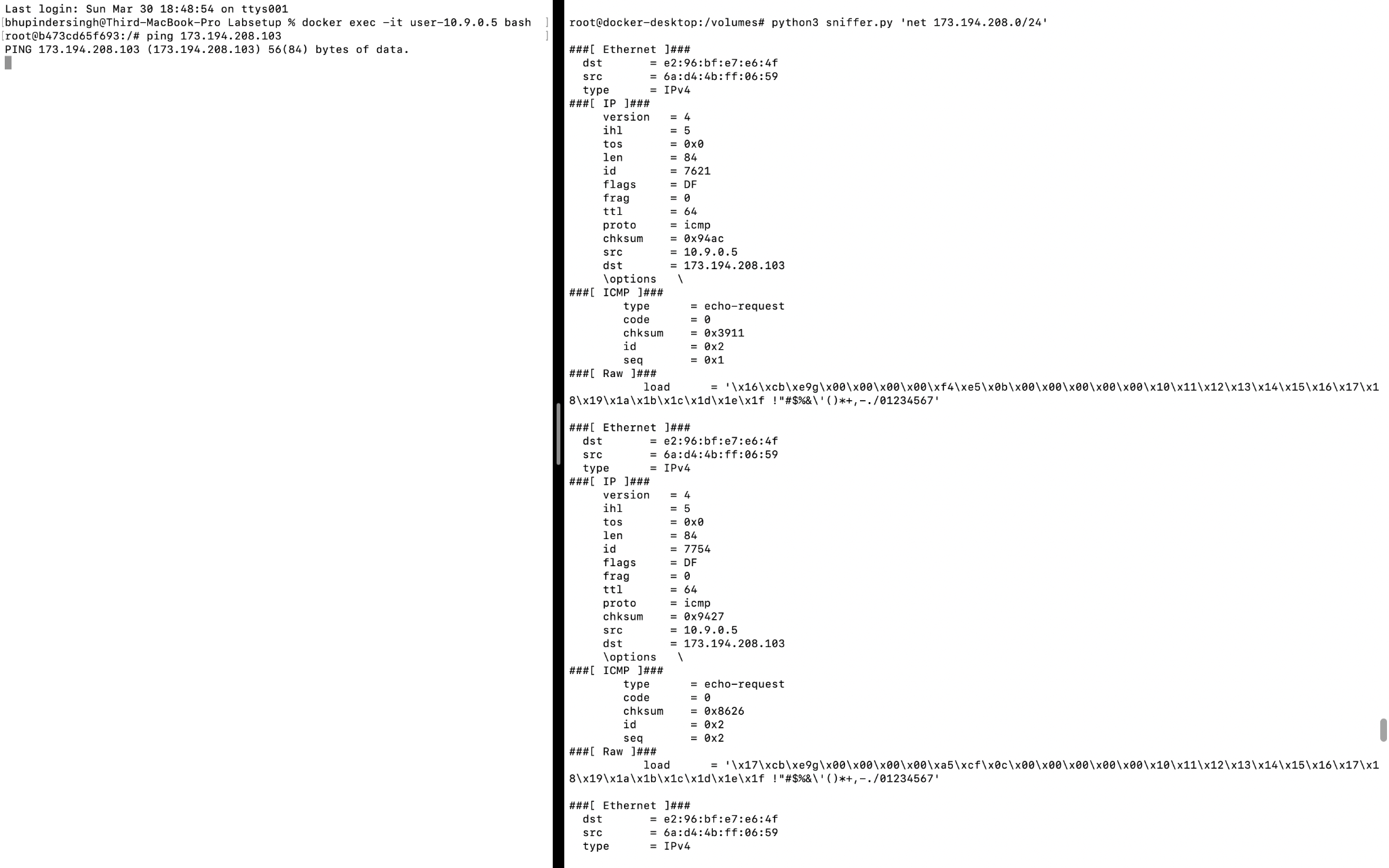
**Task N2.c (4 pts): Capture packets involving subnet 173.194.208.0/24**

**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.

**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:

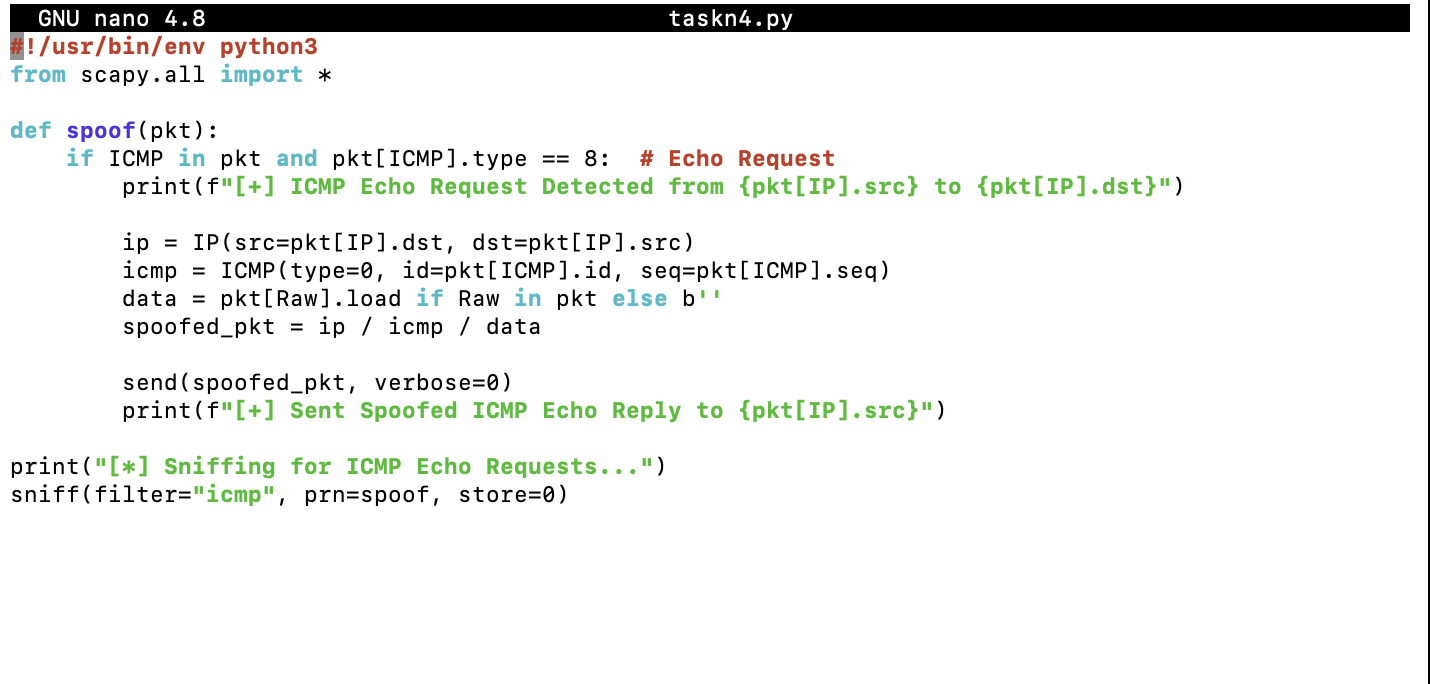
* On the left: Output from the SpoofingDriverProgram.py script, confirming ICMP packets were sent from both attacker containers.
* On the right: tcpdump successfully captured incoming spoofed ICMP echo request packets in real time.

### **Screenshot (Evidence):**

**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.

**Figure 4: Screenshot of taskn4.py**

# taskn4.py (simplified explanation)

- sniff() captures ICMP Echo Requests

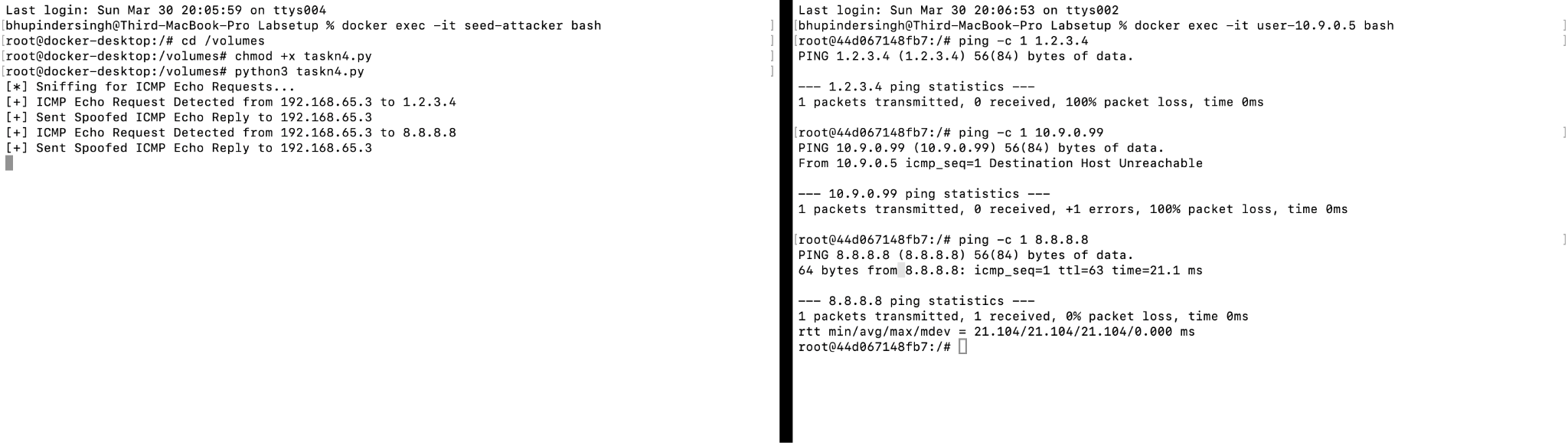
- For each packet:

- Extracts source and destination IPs

- Constructs a spoofed ICMP Echo Reply

- Sends the spoofed packet using send()

**b) Screenshot of Terminal Demonstration**



**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.