TASK 2:

Sniffing and Snooping Using pcap library in C.

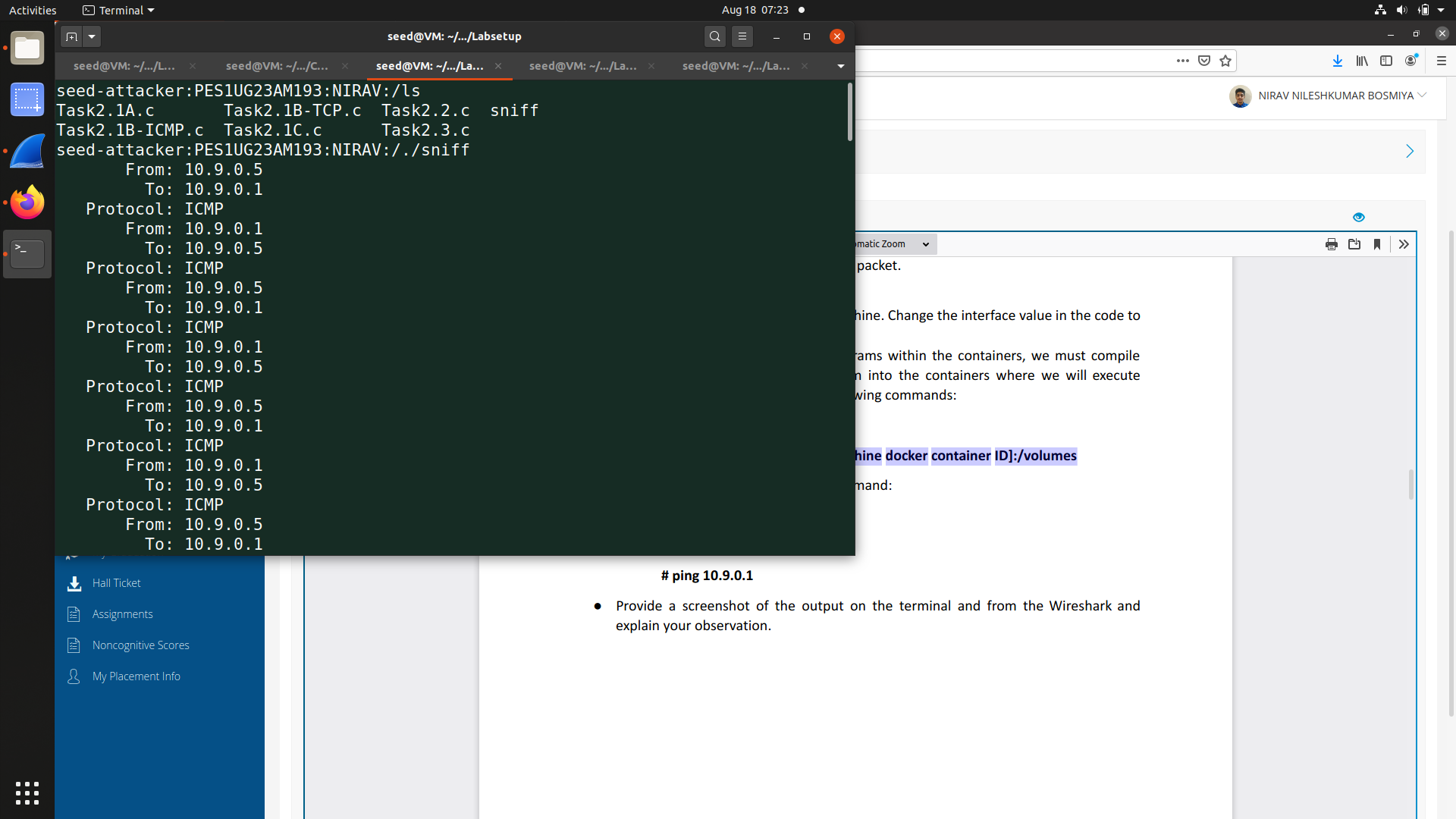
Name: Nirav N Bosmiya

SRN: PES1UG23AM193

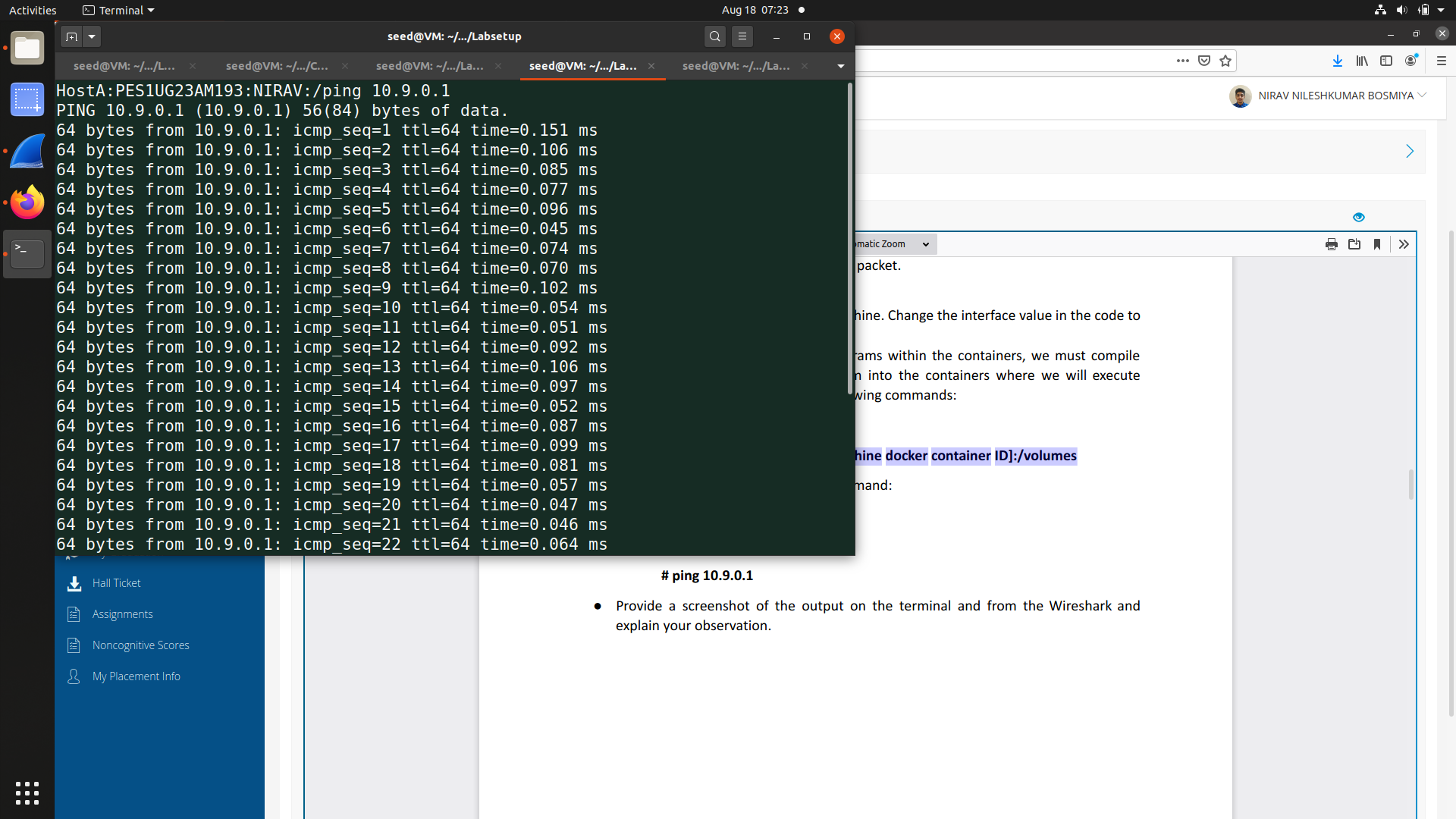
Section: D

Task1.1A:

Attacker Machine



Host Machine



Wireshark



A network sniffer on the attacker's container (10.9.0.1) and Wireshark on Host A (10.9.0.5) both successfully captured the ICMP Echo Requests and Replies exchanged between the two machines. This confirms that the sniffer is functioning correctly and that both tools are consistently capturing the same network traffic.

Question 1: Describe the sequence of the library calls that are essential for sniffer programs using pcap

Ans :

pcap\_open\_live(): Initializes a network device for live packet capture, configuring snapshot length, promiscuous mode, and a timeout.

pcap\_compile(): Translates a human-readable filter expression into a kernel-level BPF program for efficient filtering.

pcap\_setfilter(): Applies the pre-compiled BPF filter to the capture handle to restrict which packets are captured.

pcap\_loop(): Captures packets continuously in a loop, processing them one by one until a specified count or timeout is reached.

pcap\_close(): Shuts down the packet capture session and frees all associated resources.

Question 2: On the Attacker container run the command :

# su seed (this is used to switch from root to normal user)

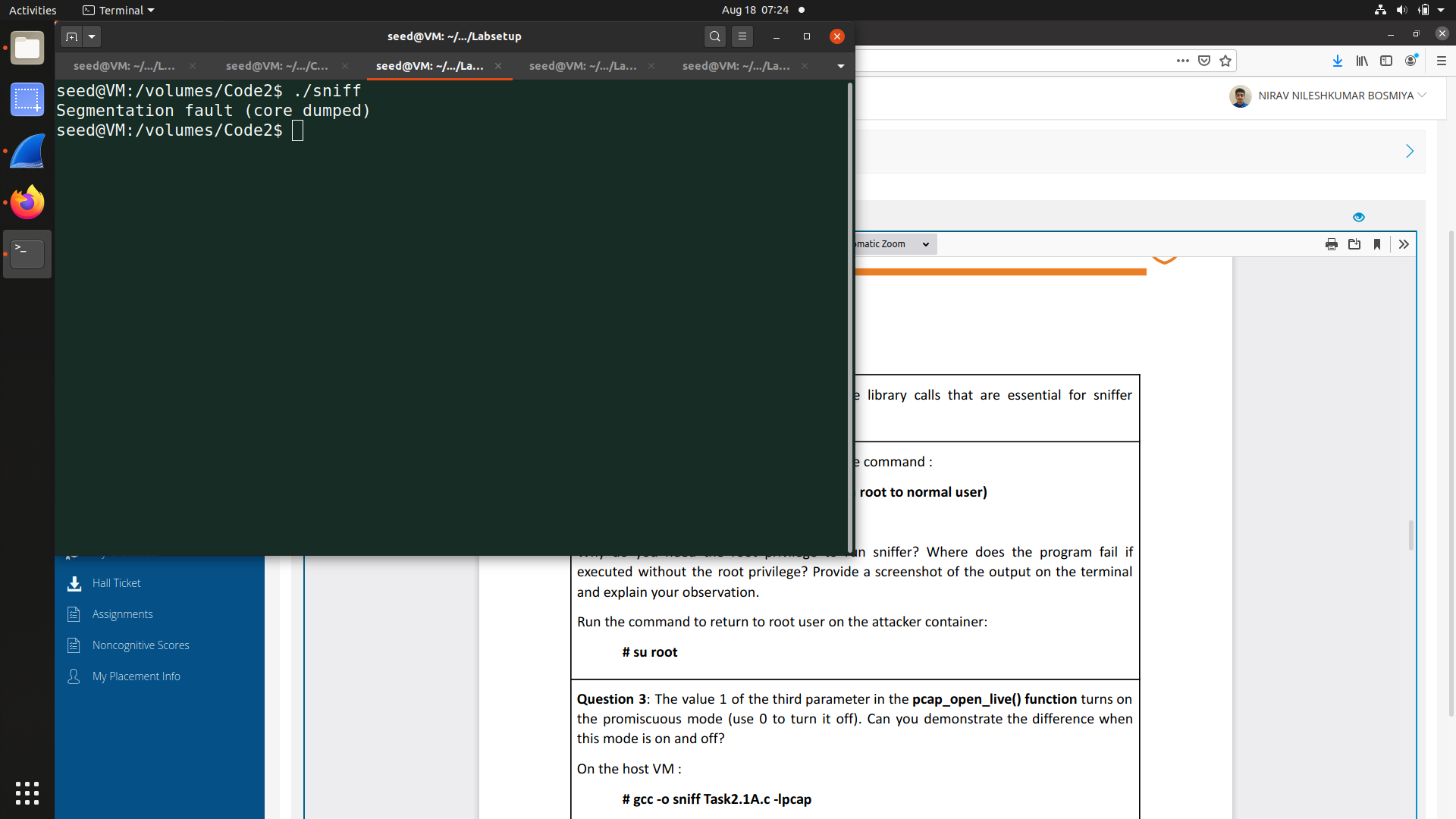
# ./sniff

Why do you need the root privilege to run sniffer? Where does the program fail if

executed without the root privilege? Provide a screenshot of the output on the terminal

and explain your observation

Ans:



Sniffer programs require root privileges to operate in raw capture mode. Without these permissions, the call to pcap\_open\_live() fails and returns a NULL value. The program, if it doesn't handle this error, will attempt to use the invalid handle, causing a segmentation fault when it tries to access restricted memory. With root access, the function succeeds and the program runs correctly.

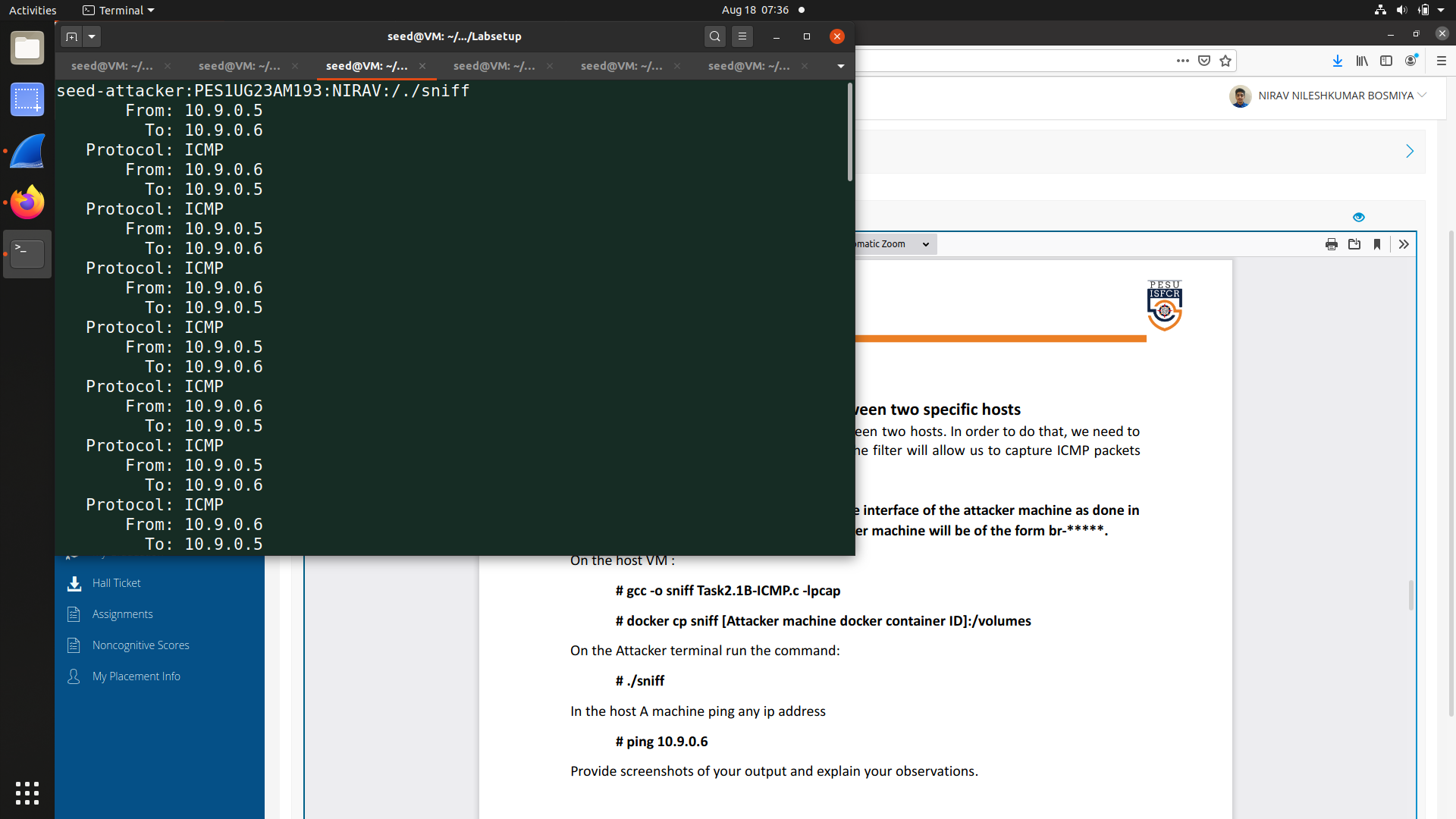
Question 3: The value 1 of the third parameter in the pcap\_open\_live() function turns on

the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when

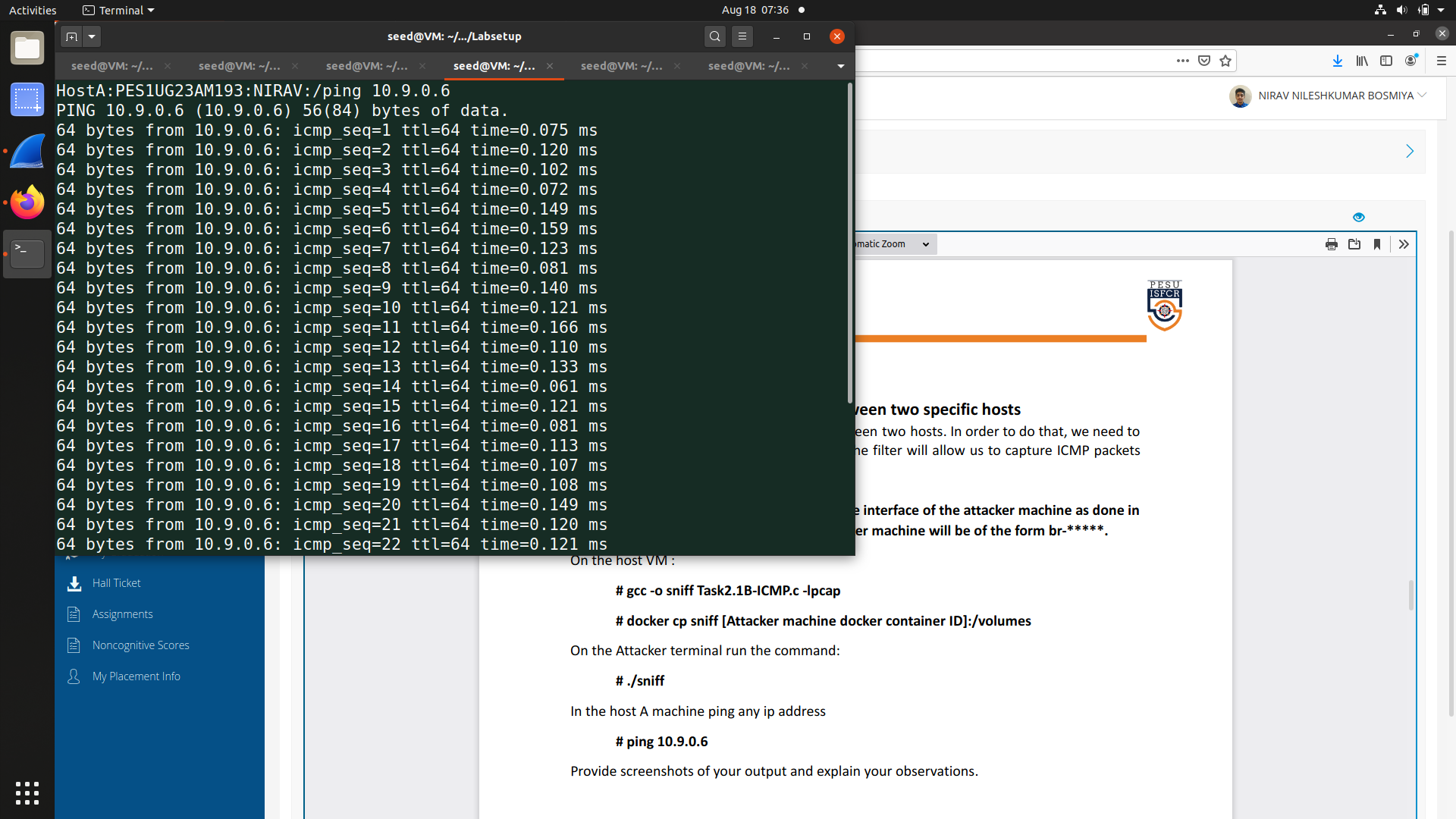
this mode is on and off?

Task2.1B: Step 1:

Attacker machine



Host machine



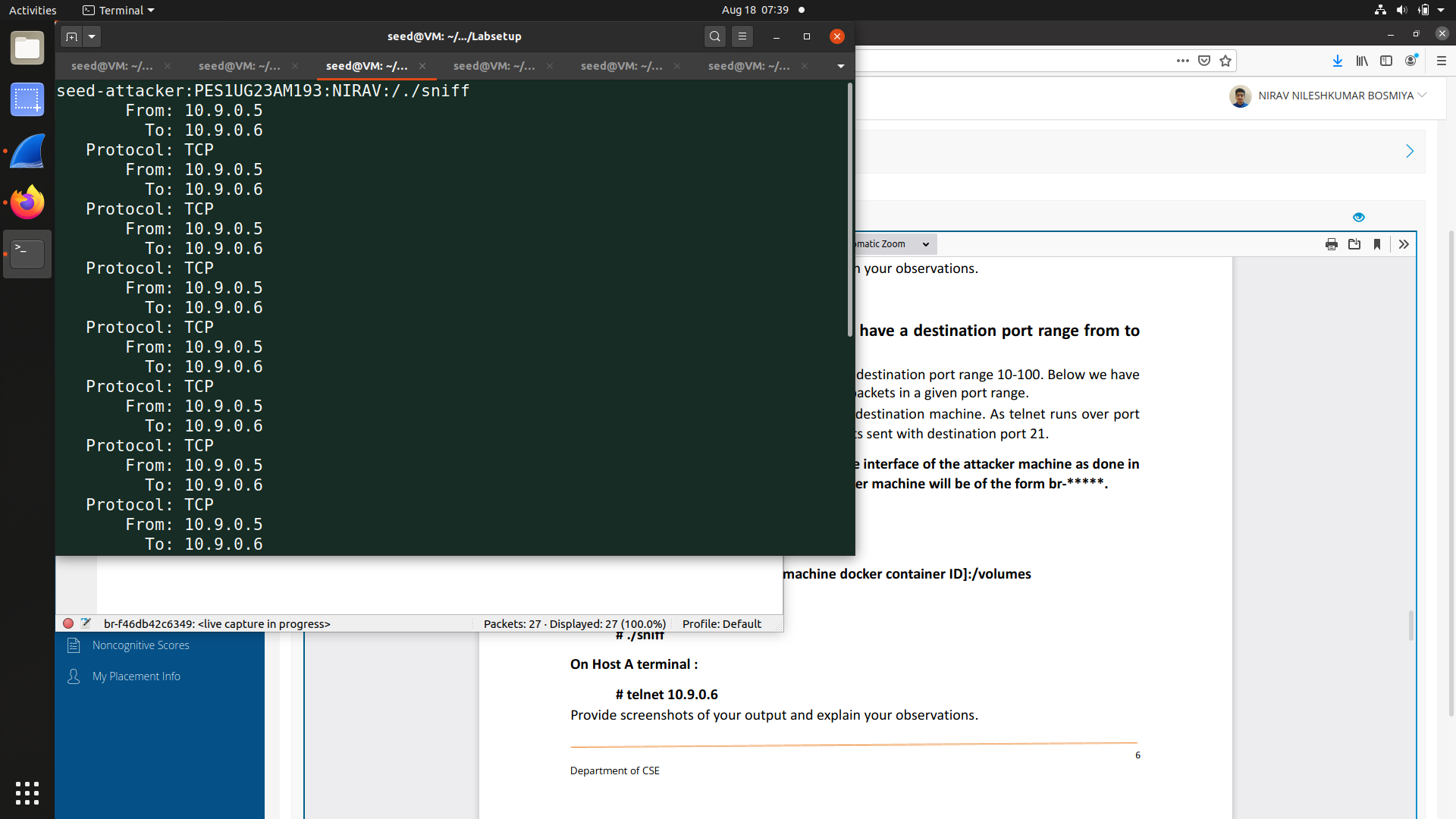
Wireshark  


The sniffer successfully demonstrated its intended functionality by exclusively capturing ICMP traffic. When Host A initiated a ping to the machine at 10.9.0.6, the sniffer's output provided a clean, concise log of only the ICMP Echo Requests and their corresponding Echo Replies. This targeted capture proves that the ICMP filter was correctly implemented and is working precisely as designed, effectively isolating the desired packet type from all other network noise.

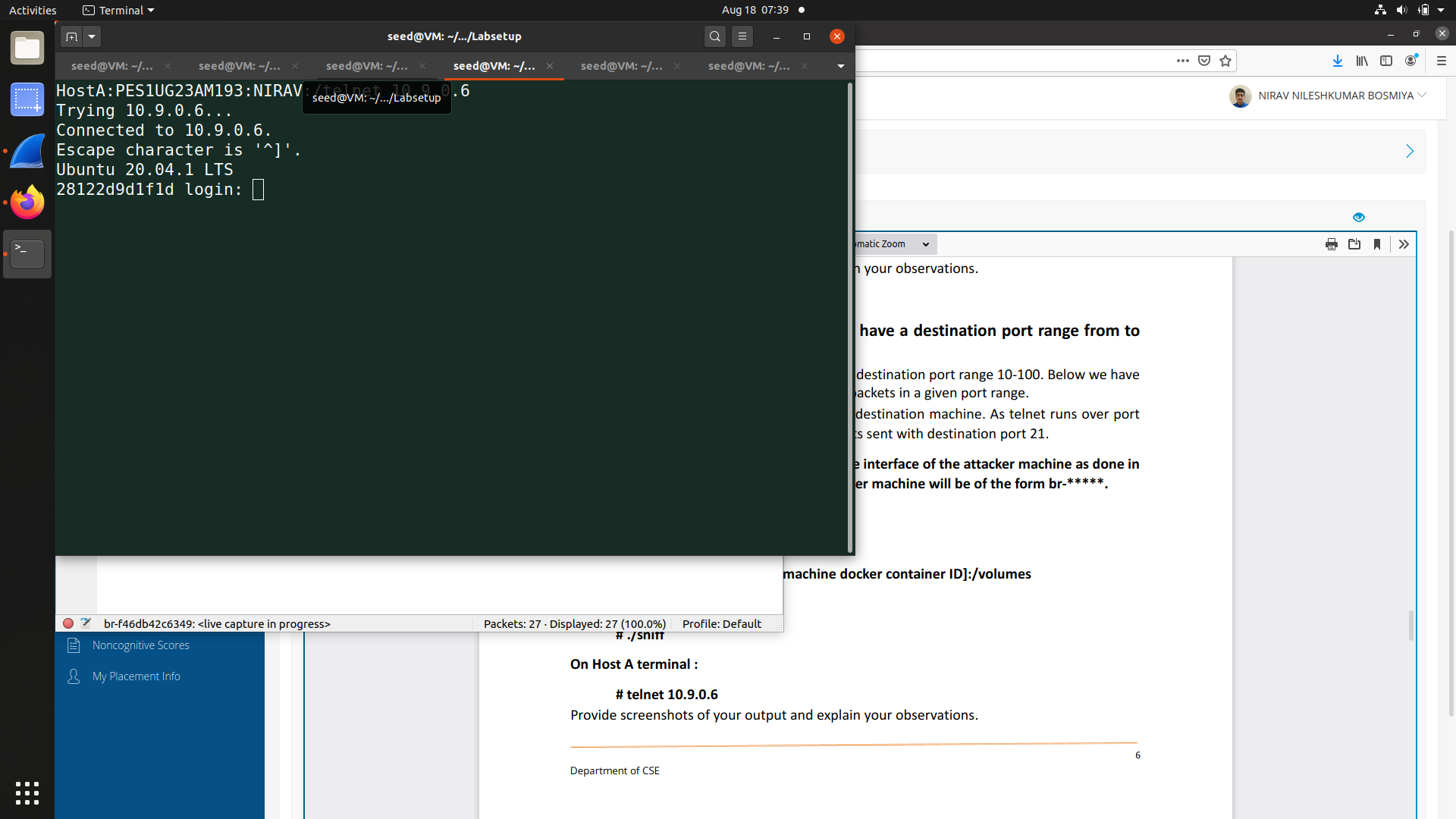
Filter: char filter\_exp[] = "proto ICMP and (host 10.9.0.5 and 10.9.0.6)";

Step 2:

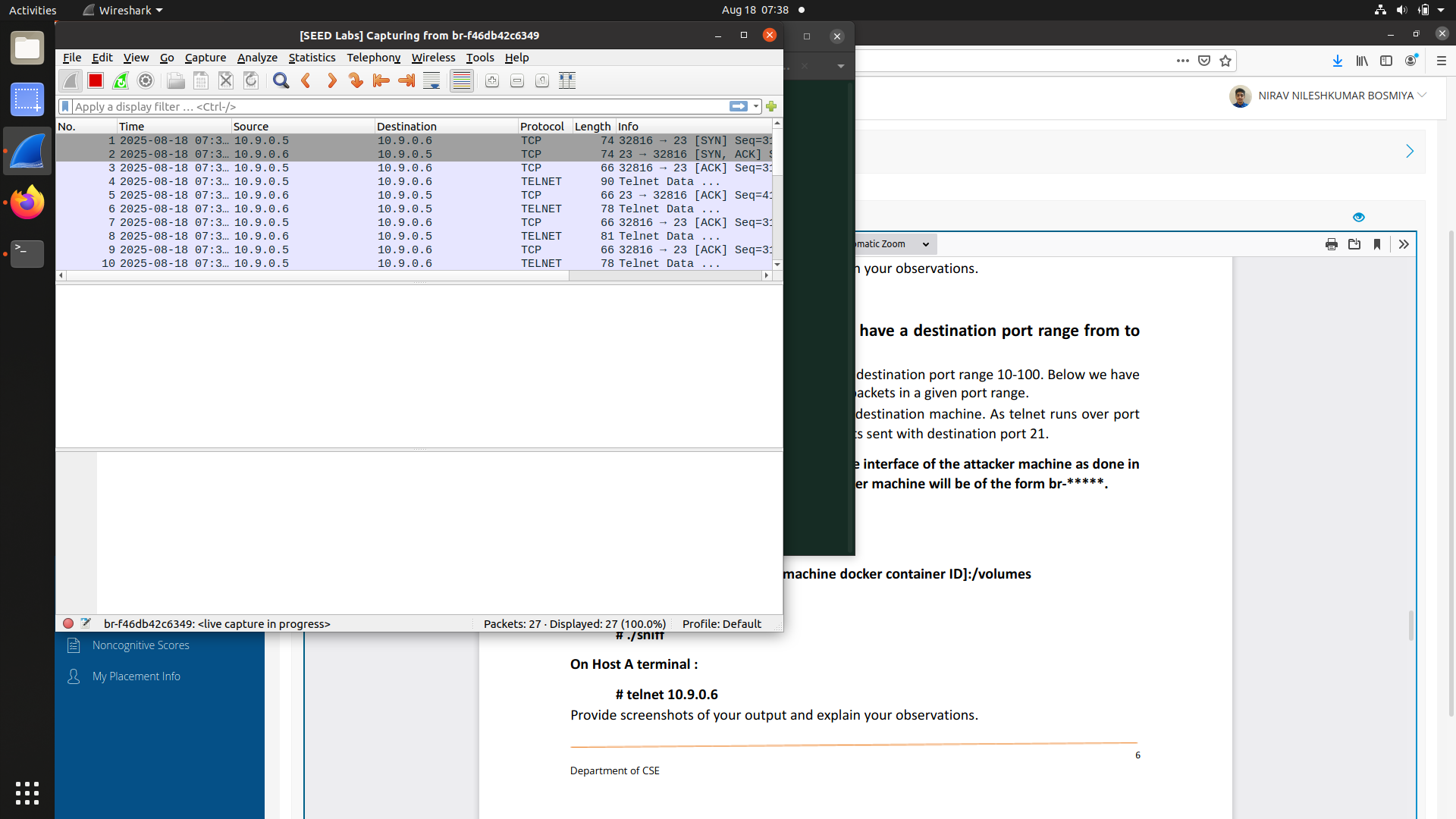
Attacker Machine



Host machine



WireShark

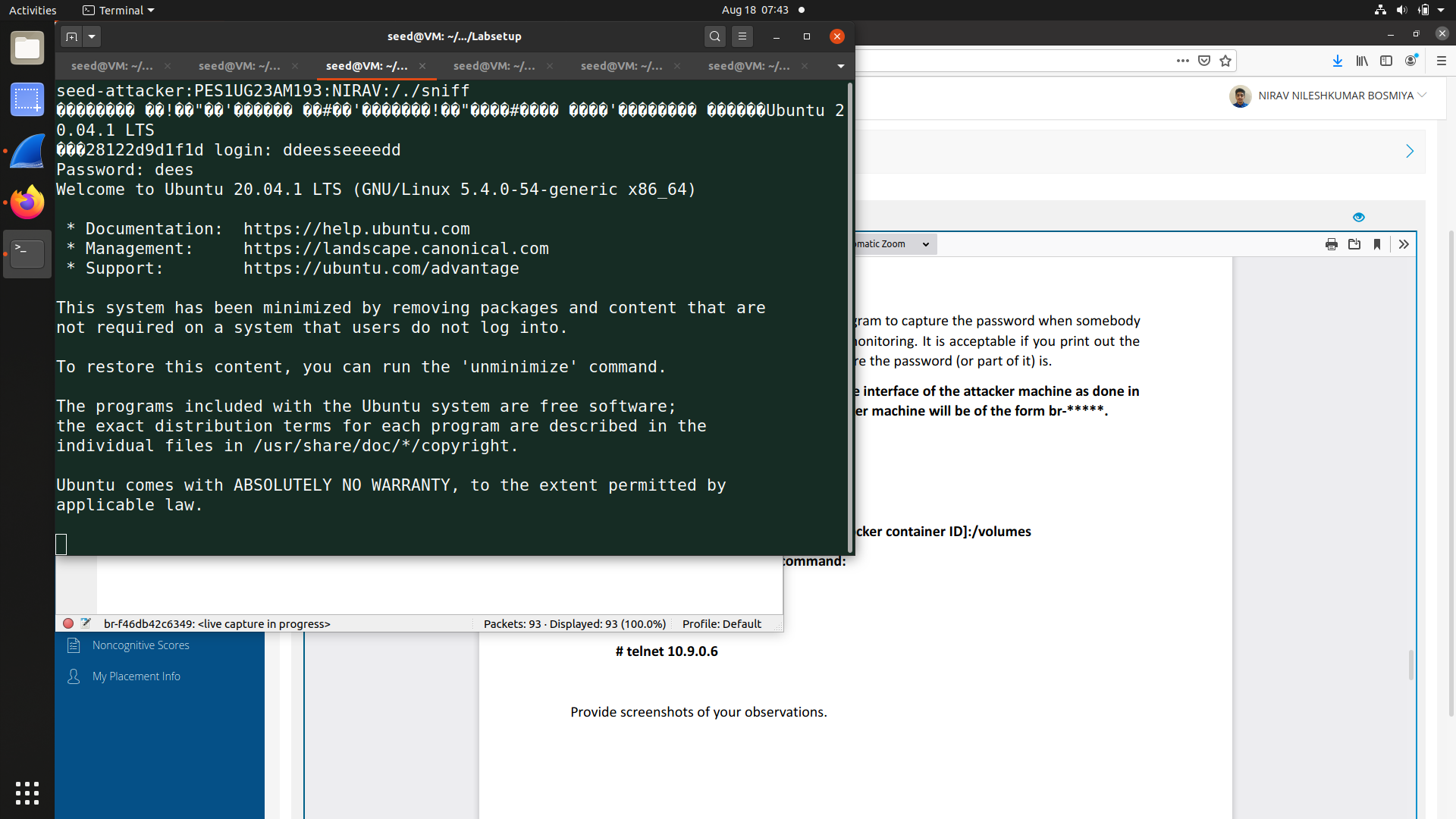


When executed, the sniffer displays only TCP packets with destination ports in the range 10–100, such as port 23, confirming that the filter is working correctly. No ICMP or UDP packets appear, showing that the sniffer selectively captures traffic based on the specified port range. The output includes source and destination IP addresses along with port numbers, illustrating the details of the filtered TCP connections

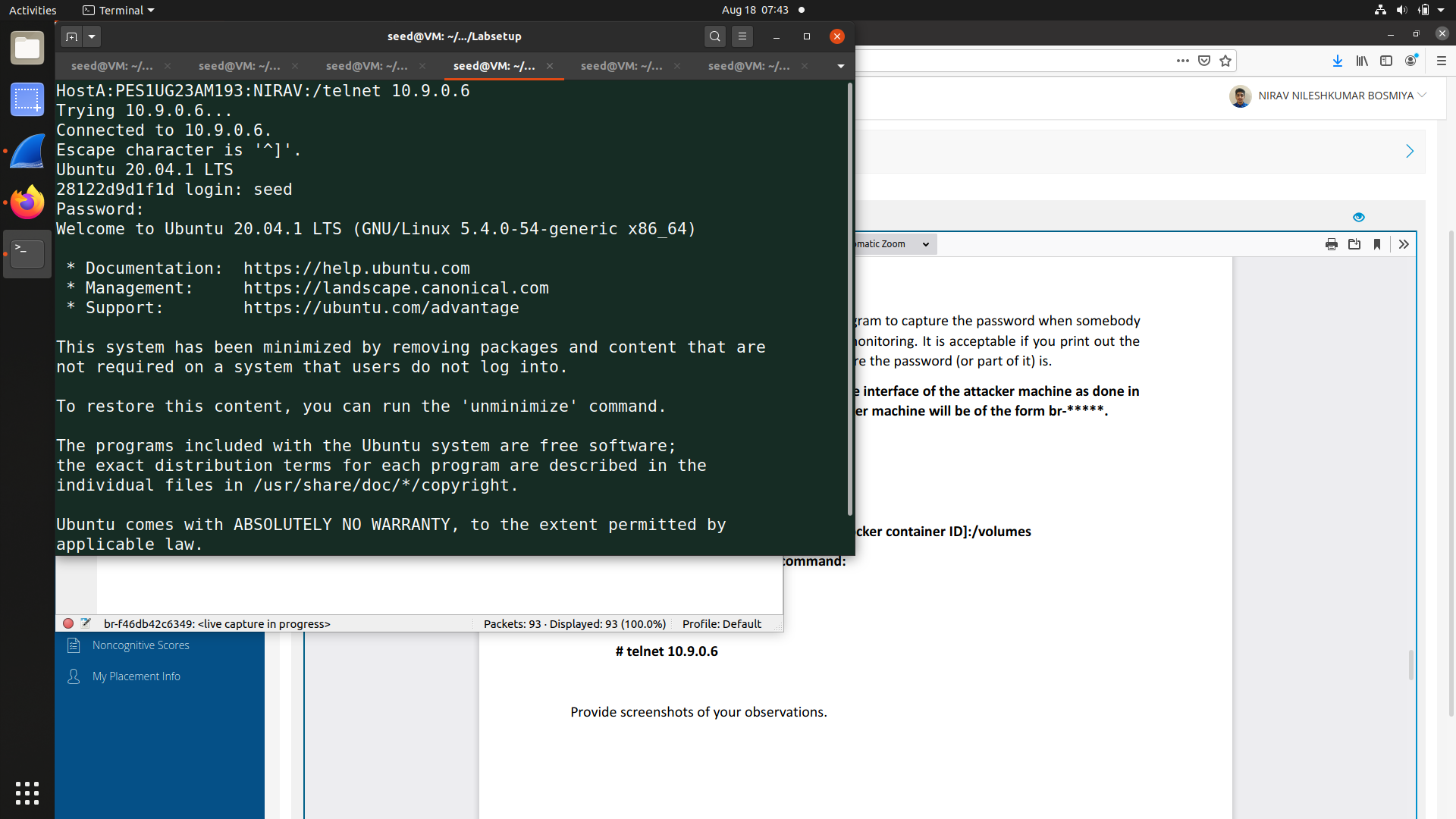
Filter:char filter\_exp[] = "proto TCP and dst portrange 10-100";

Task 2.1C:

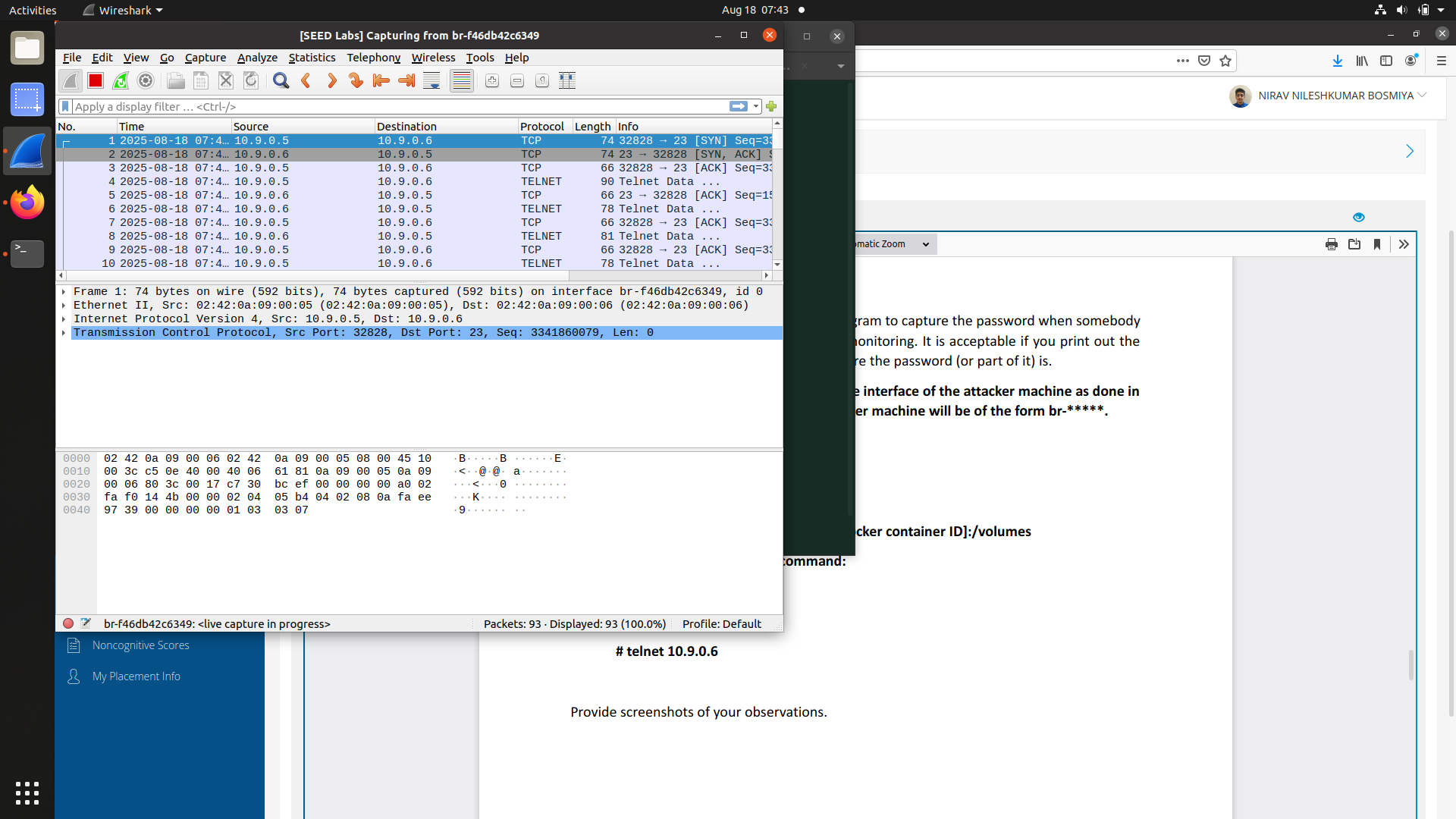
Attacker machine



Host machine

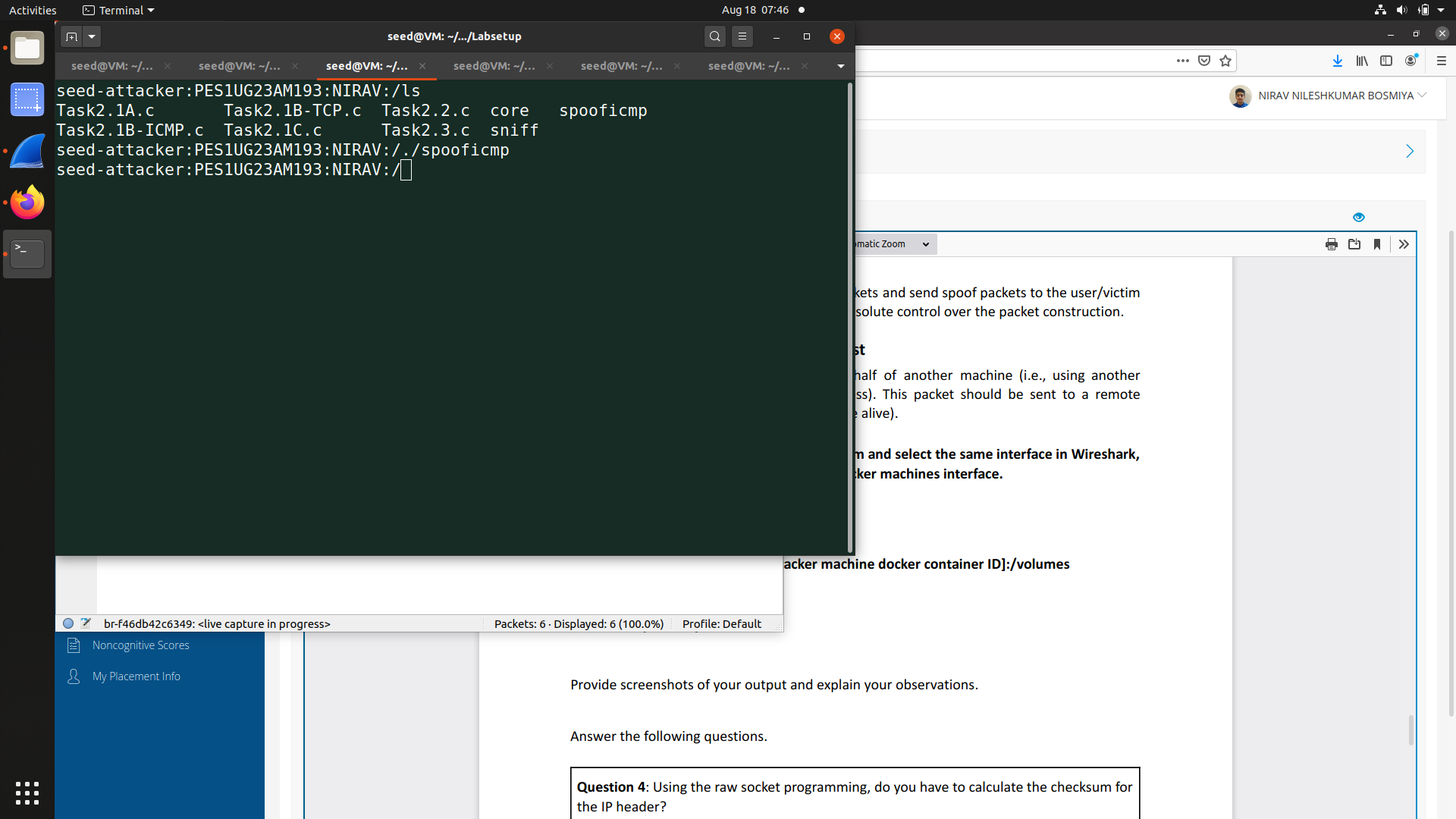


WireShark

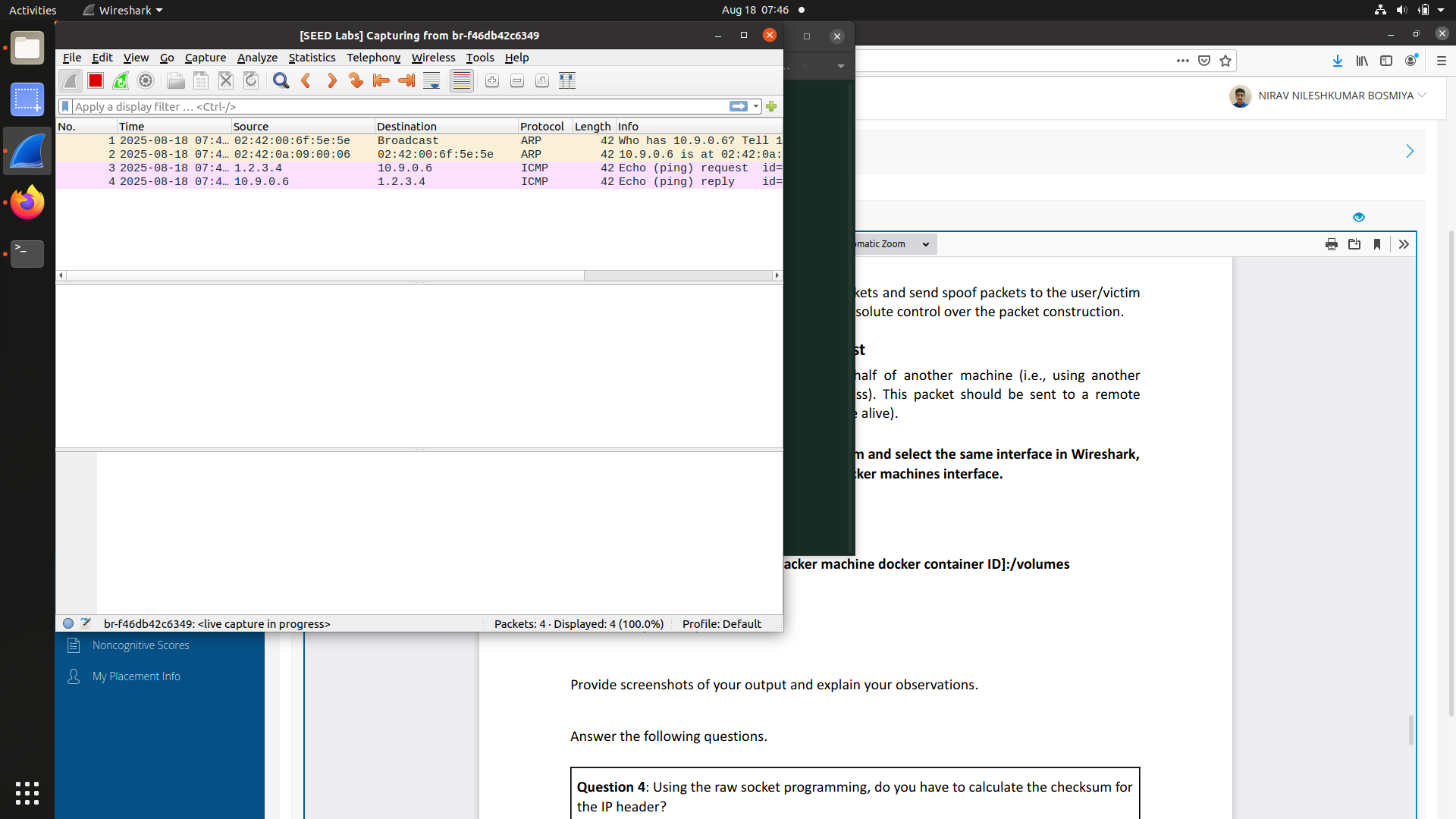


The sniffer captures TCP packets on port 23, and the payload clearly shows the username and password transmitted during the Telnet login. This demonstrates that Telnet sends credentials in plain text, making it insecure since an attacker monitoring the network can easily extract sensitive information

Task2.2:

Attacker machine  


WireShark



Wireshark shows that the attacker successfully generated and sent an ICMP Echo Request with a spoofed source IP address. This confirms that the raw socket program provided full control over packet construction, including manual creation of IP and ICMP headers. The result demonstrates how raw sockets can be used to craft custom packets and manipulate header fields that are normally handled by the operating system.

Question 4: Using the raw socket programming, do you have to calculate the checksum for the IP header?

Ans:

In raw socket programming, you normally do not have to calculate the checksum for the IP header, because the kernel takes care of it automatically before transmitting the packet.

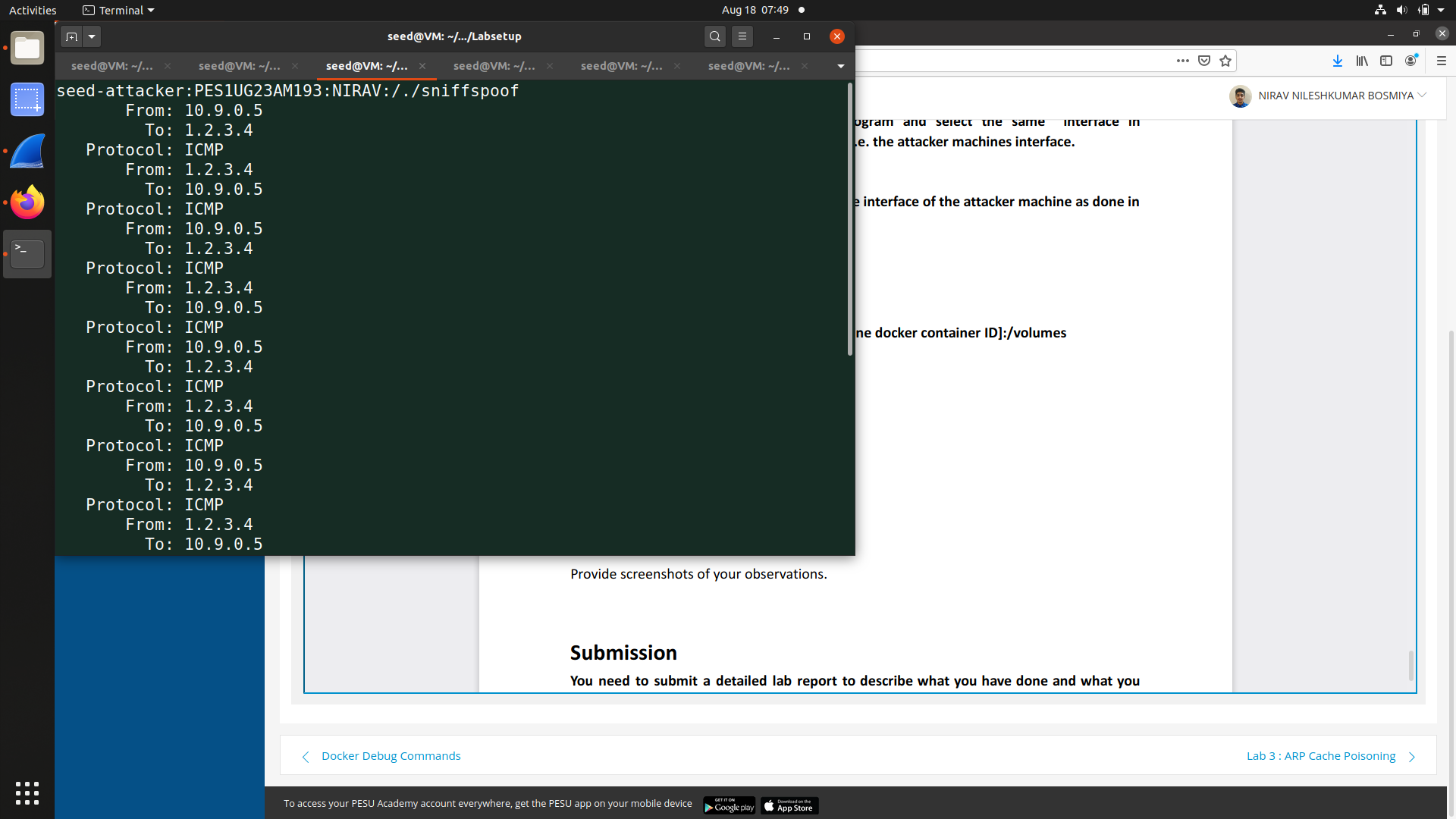
Question 5: Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

Ans:

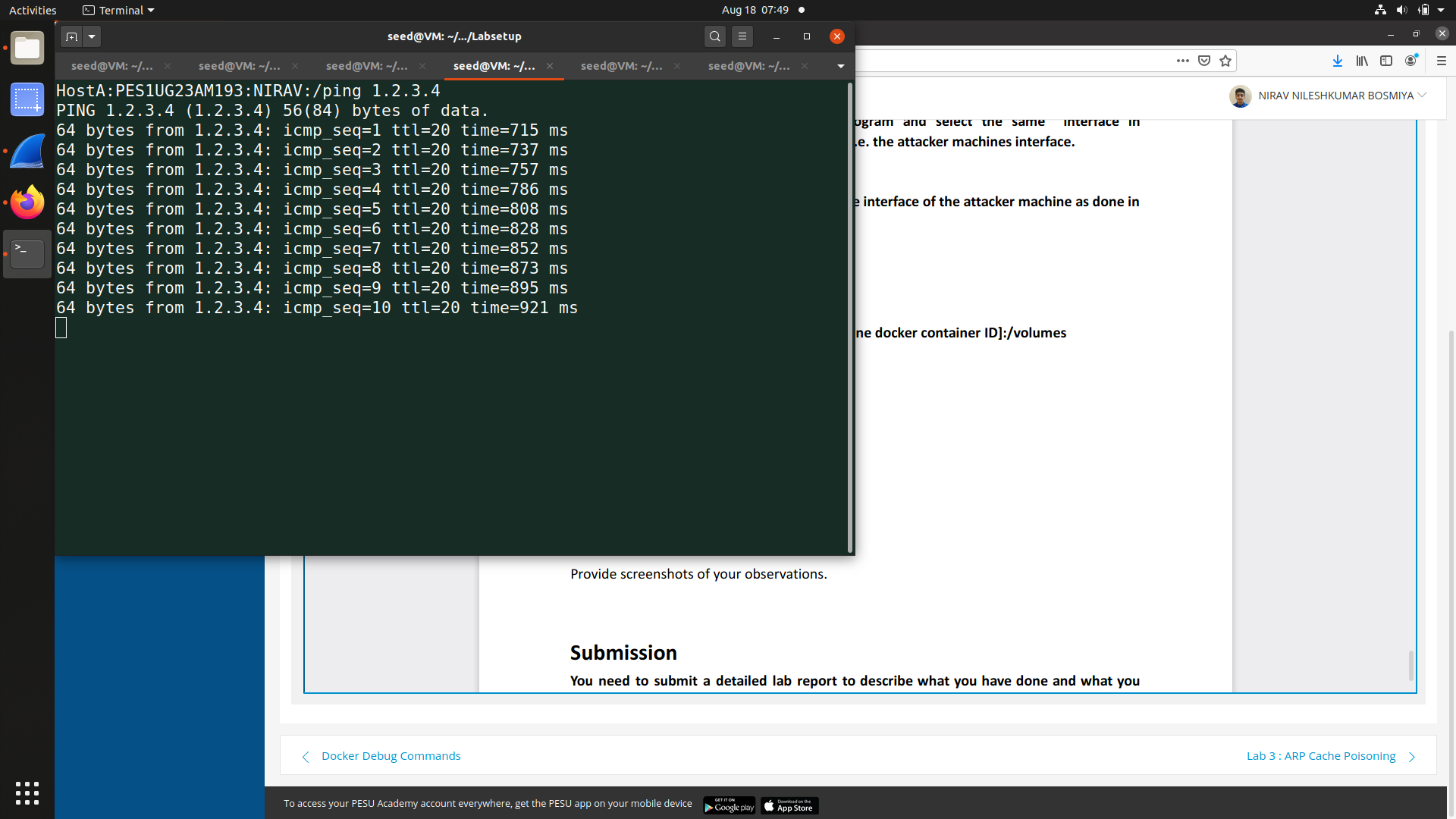
Programs that use raw sockets require root privileges because creating and manipulating such sockets gives direct control over packet construction and transmission. Allowing normal users to do this would be a major security risk, as it could let them spoof packets, sniff traffic, or interfere with the network stack. For this reason, the operating system restricts raw socket access to privileged users.  
If the program is executed without root privileges, it will fail at the call to create the raw socket.

Task 2.3:

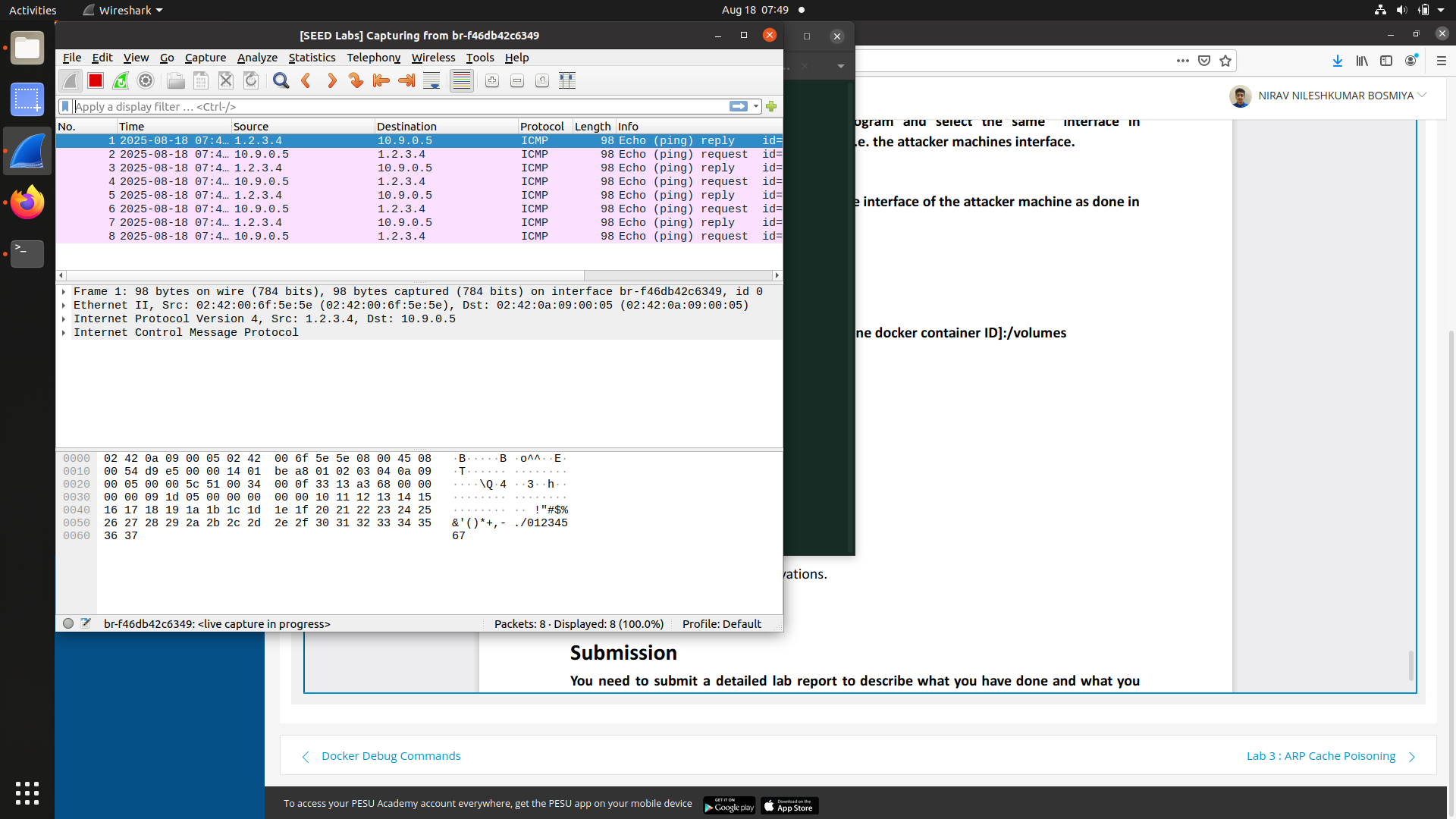
Attacker machine



Host machine



WireShark



When the victim pings the non-existent IP 1.2.3.4, the attacker’s program captures the echo request and forges an echo reply using raw sockets by swapping the source and destination addresses. The victim then receives replies that appear to come from 1.2.3.4, making the host seem alive even though it does not exist. Wireshark shows the request and the spoofed reply packets, confirming the success of the attack