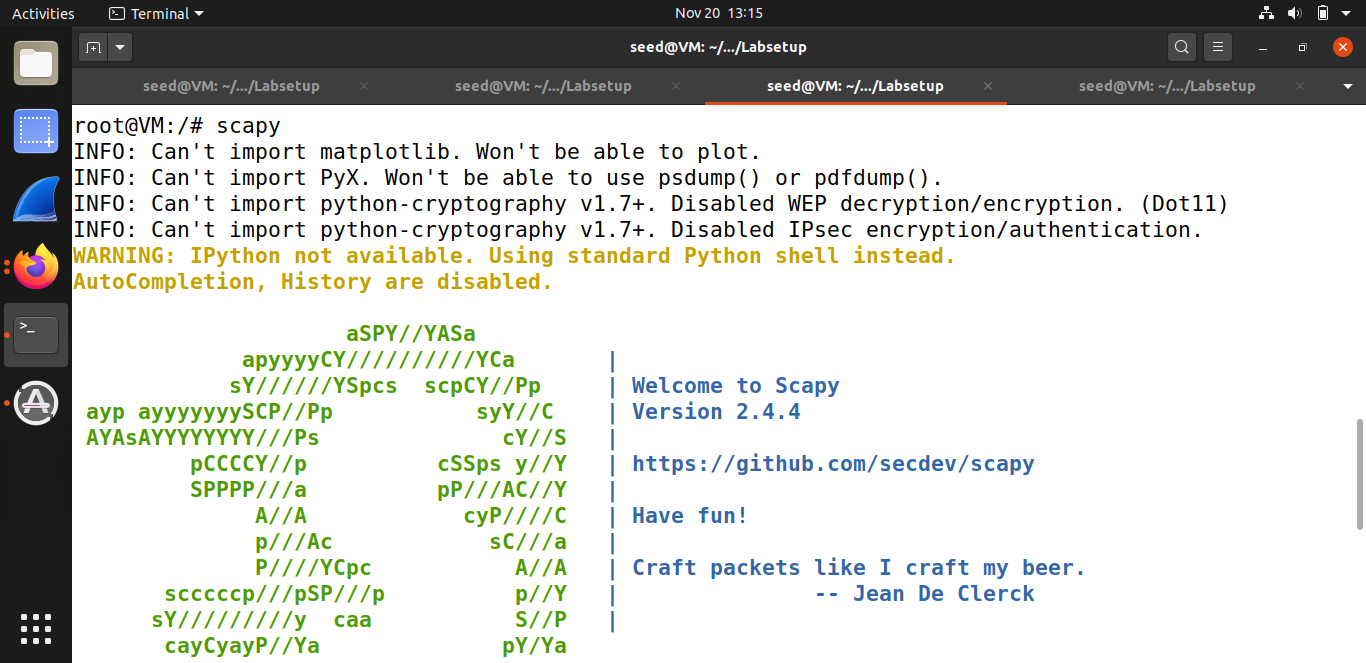
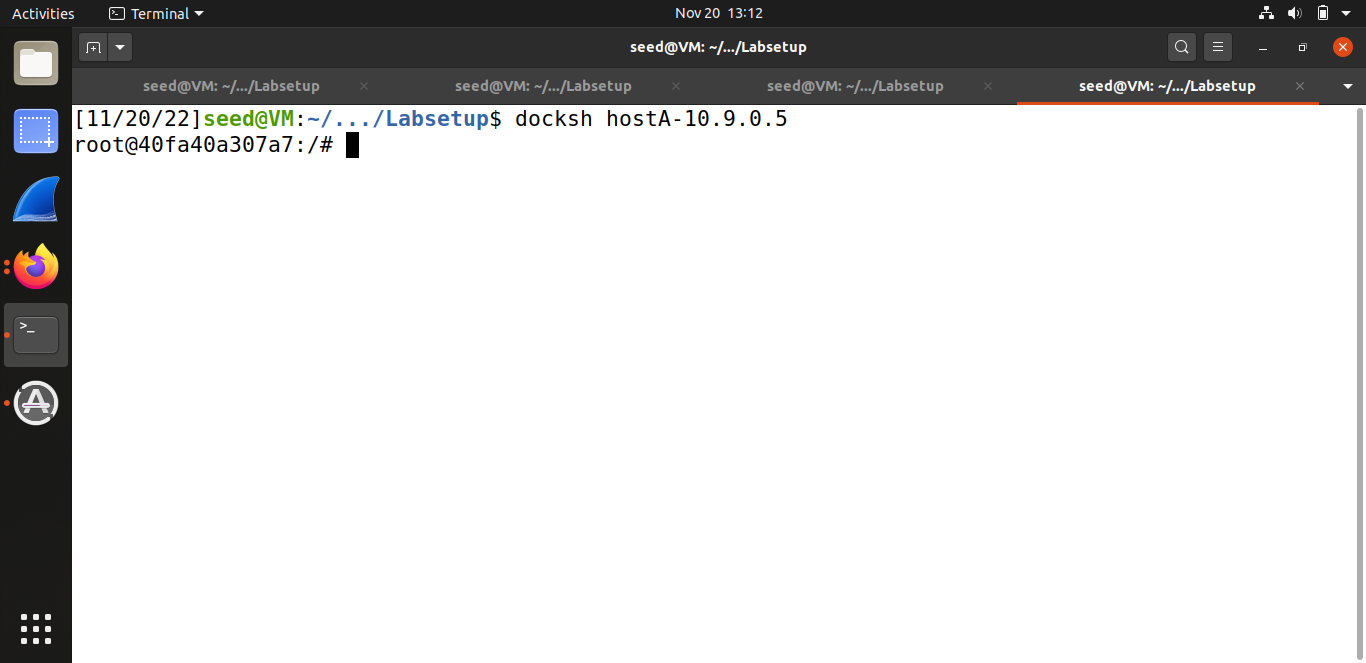
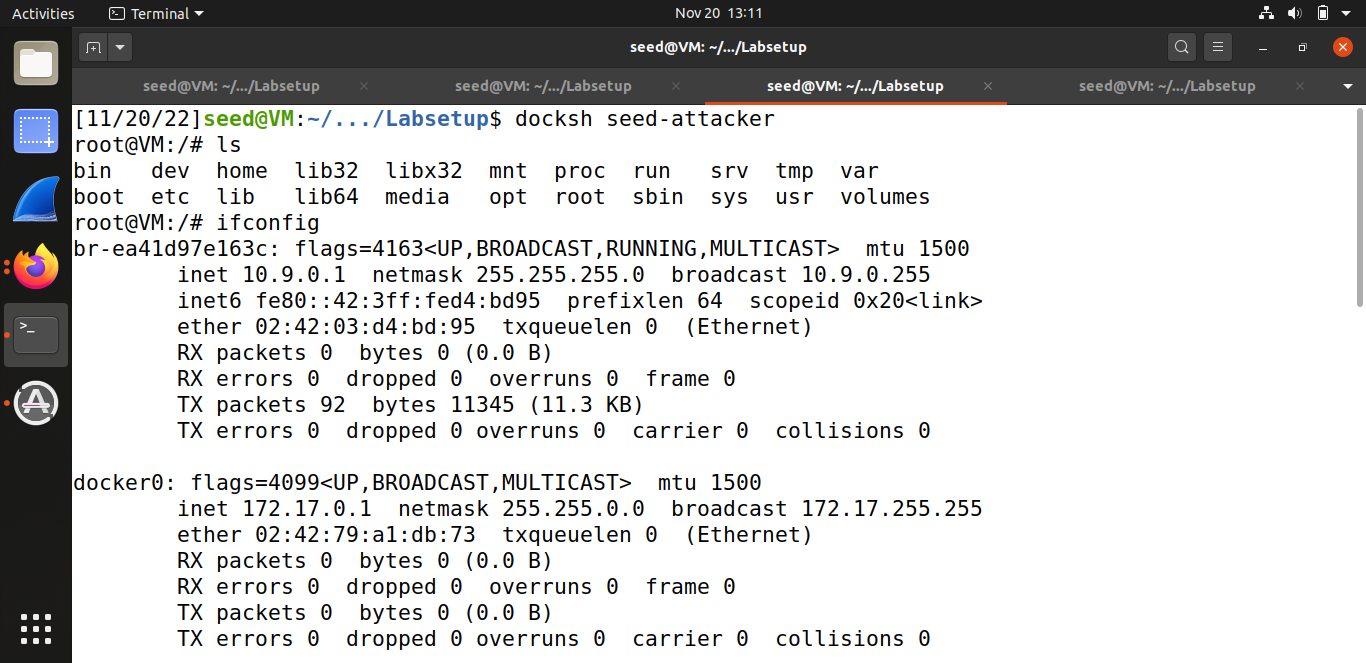
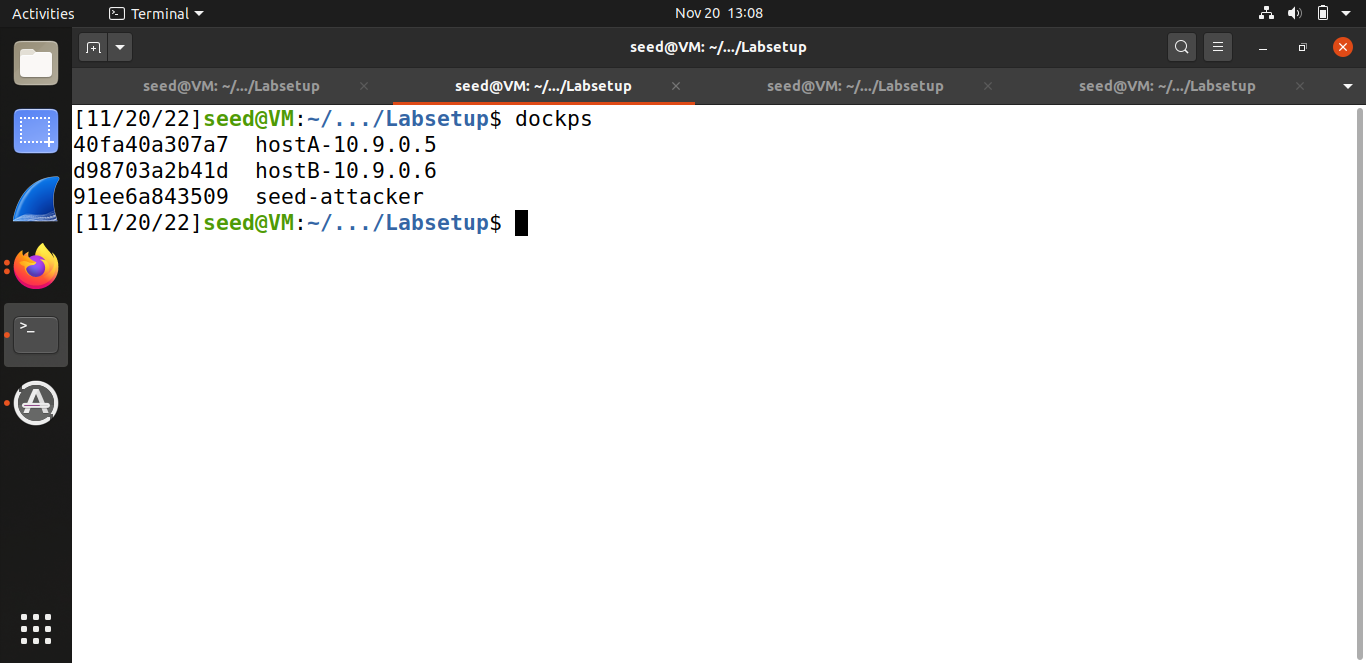
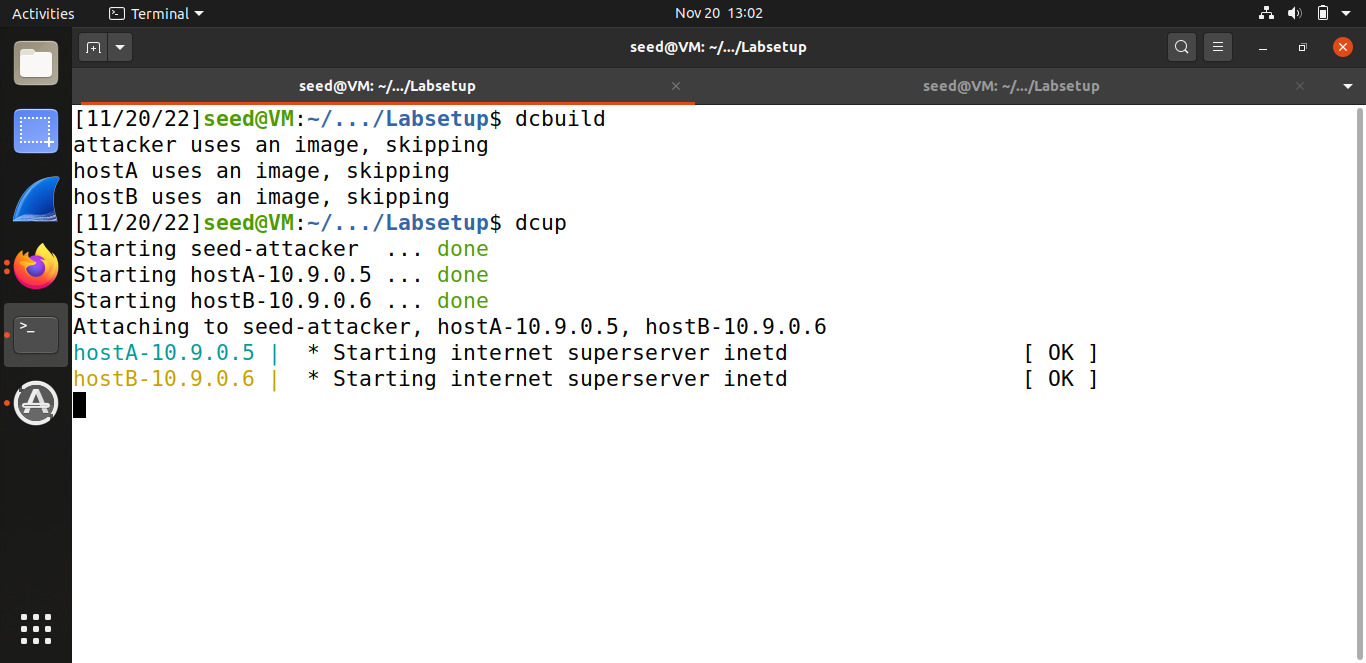
**IS LAB ASSIGNMENT#04**

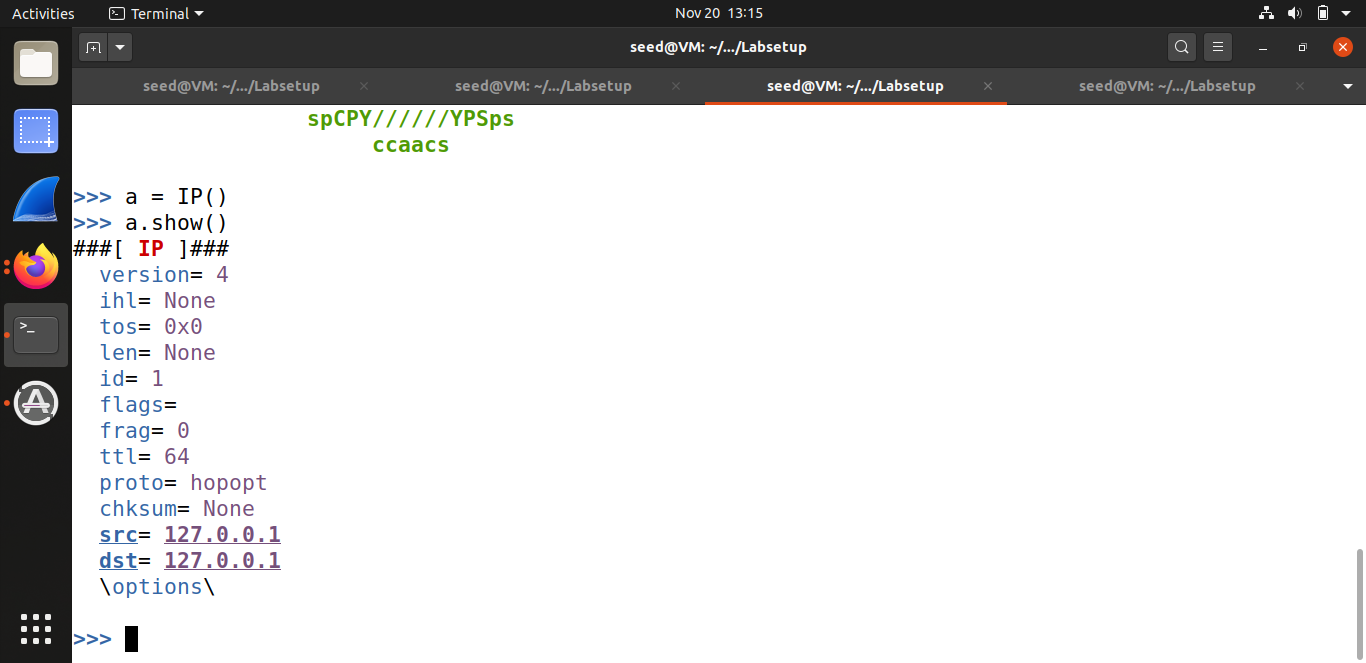
**Name: Saman Khan**

**ID: 19K-0354**

**Section: 7H**

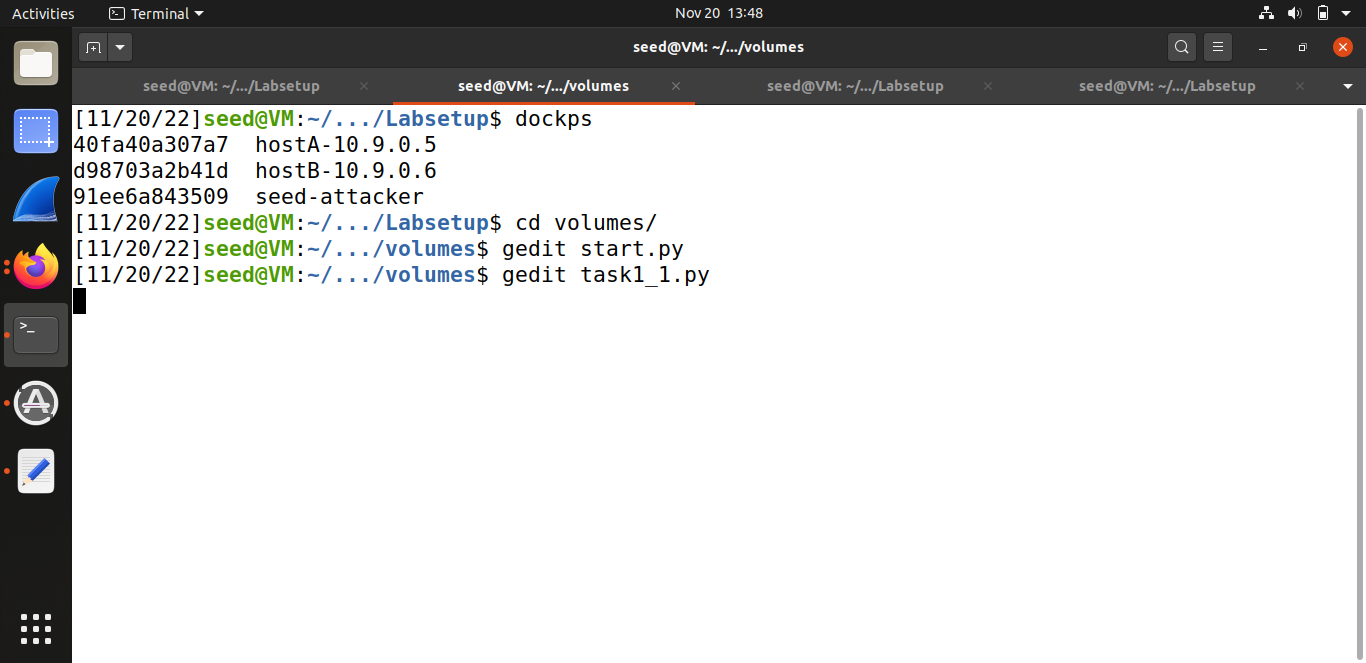
**Lab Environment Setup:**

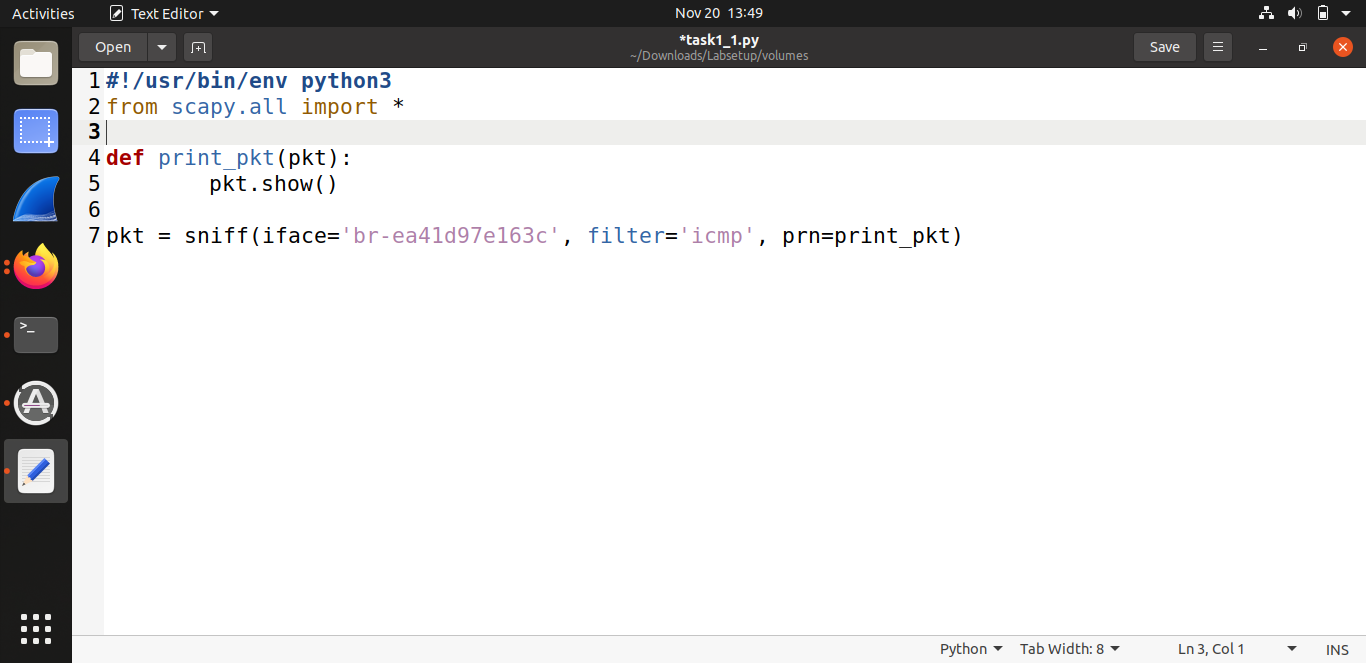


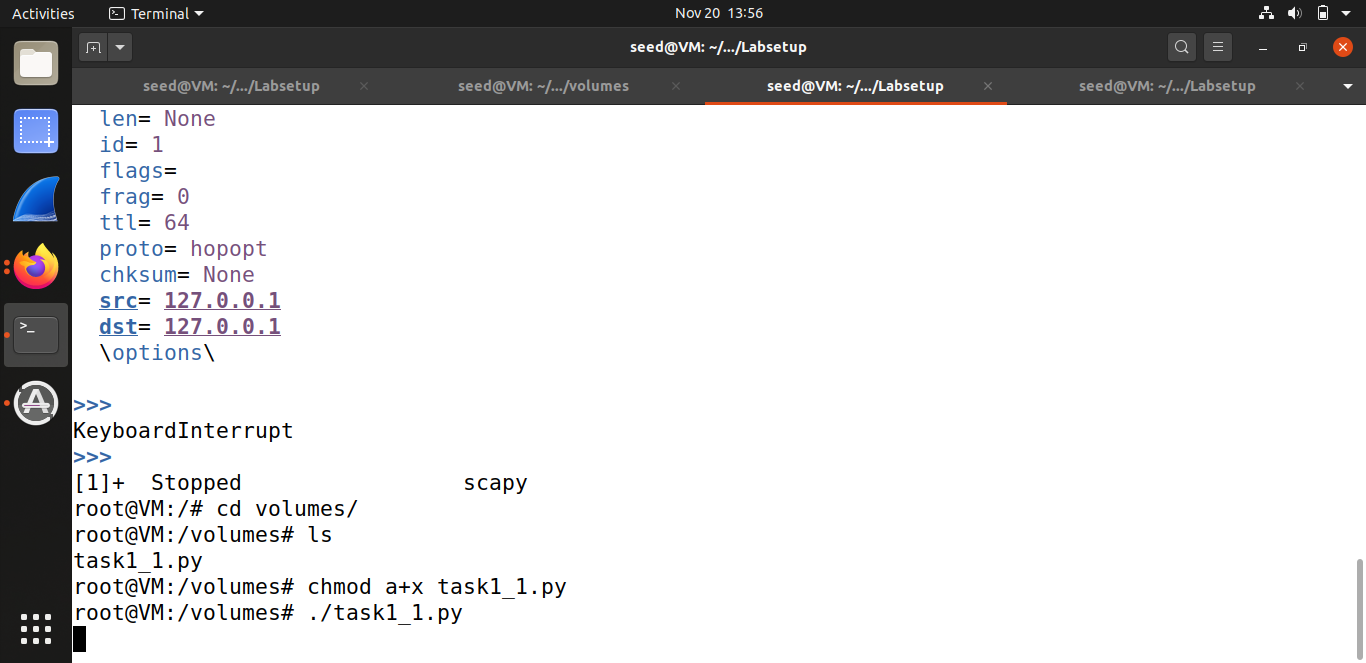
Scapy is a packet manipulation tool craft and send packets. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies. In this code (init.py), the IP() method creates and returns a new IP default and empty packet. If we execute the show() command it will display the contents of the packet. 

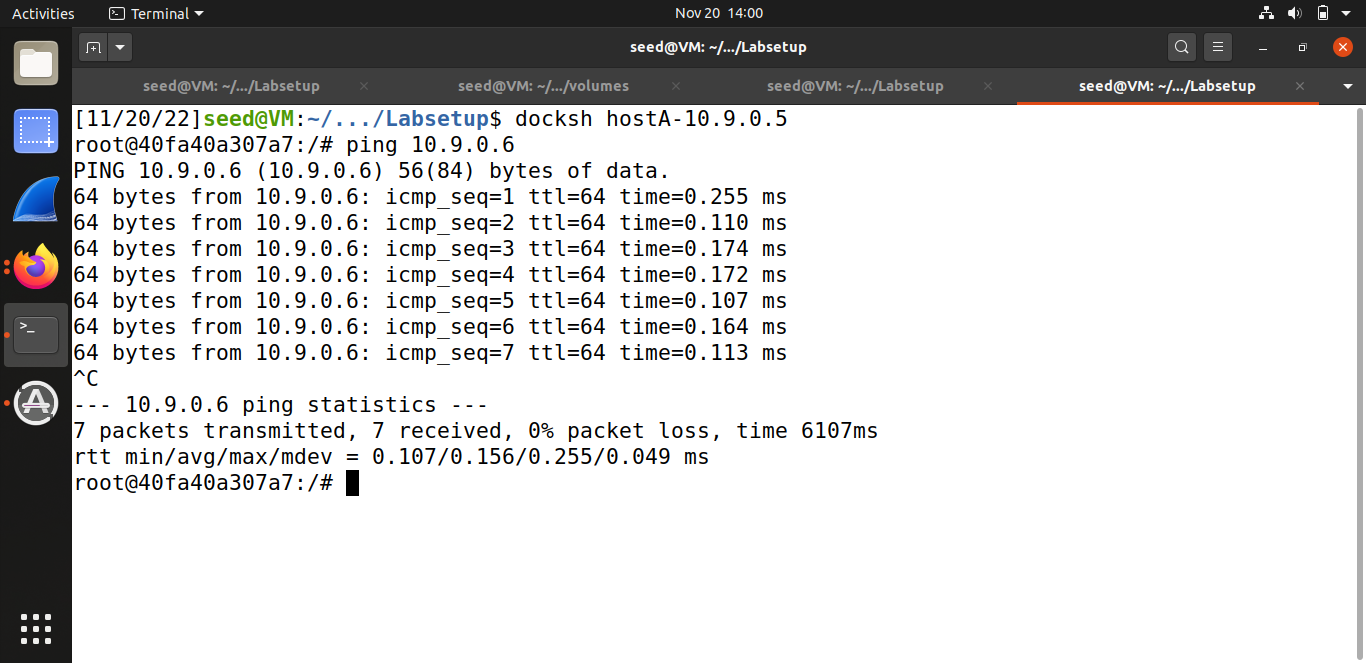
**Task#1.1A:**

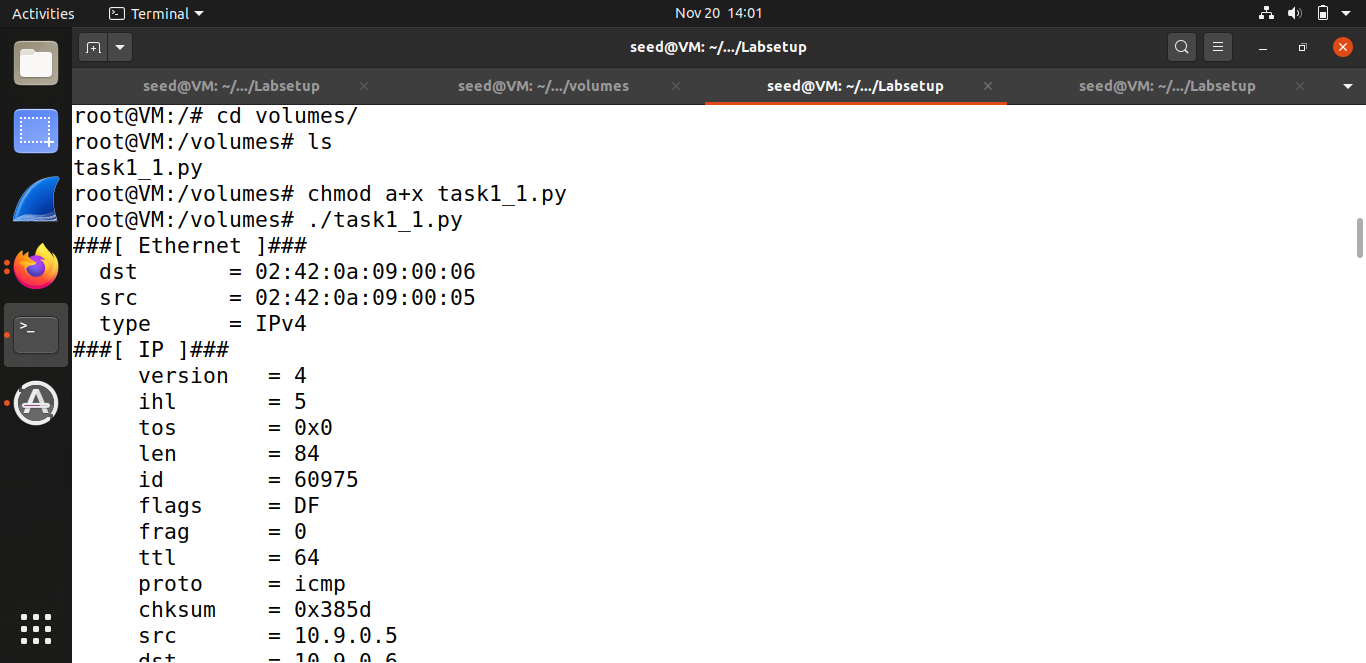
Using the the command ‘ifconfig’ in terminal and mentioned them in a list of the interfaces I wish to snip the packets from. Using the filter, Scapy will show us only ICMP packets. In order to run the program with the root privilege, I made this possible with the help of the following command: ‘sudo chmod a+x sniffer.py’, the ‘a+x’ means: execute + all. I used the ‘ping’ command with ‘google.com’ for ICMP echo request and reply packets. I ran the program again, but this time, without using the root privilege. As we know, the ‘sudo’ method (superuser do) requires that the user be set up with the power to run "as root". So this time, we will run the program without ‘sudo’. As we can see we got a PermissionError - “operation not permitted”. So, if we want to sniff packets, we need root privileges to see the traffic and to capture the relevant packets.

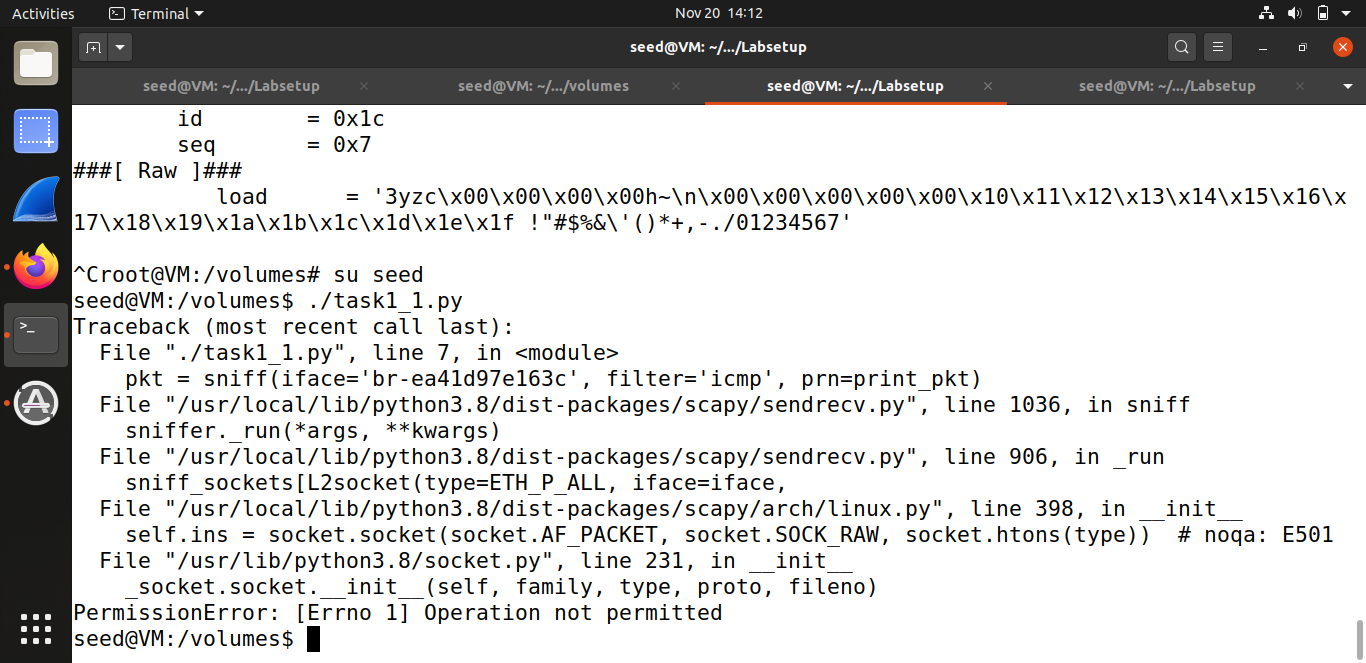








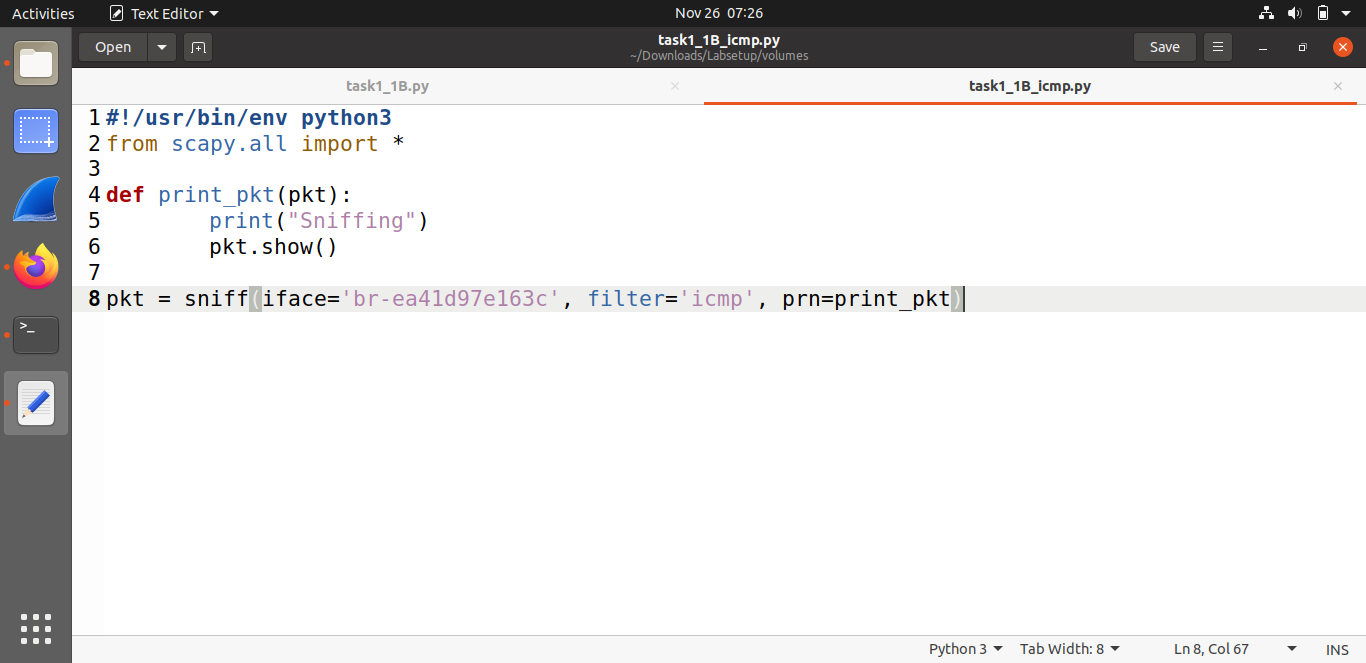


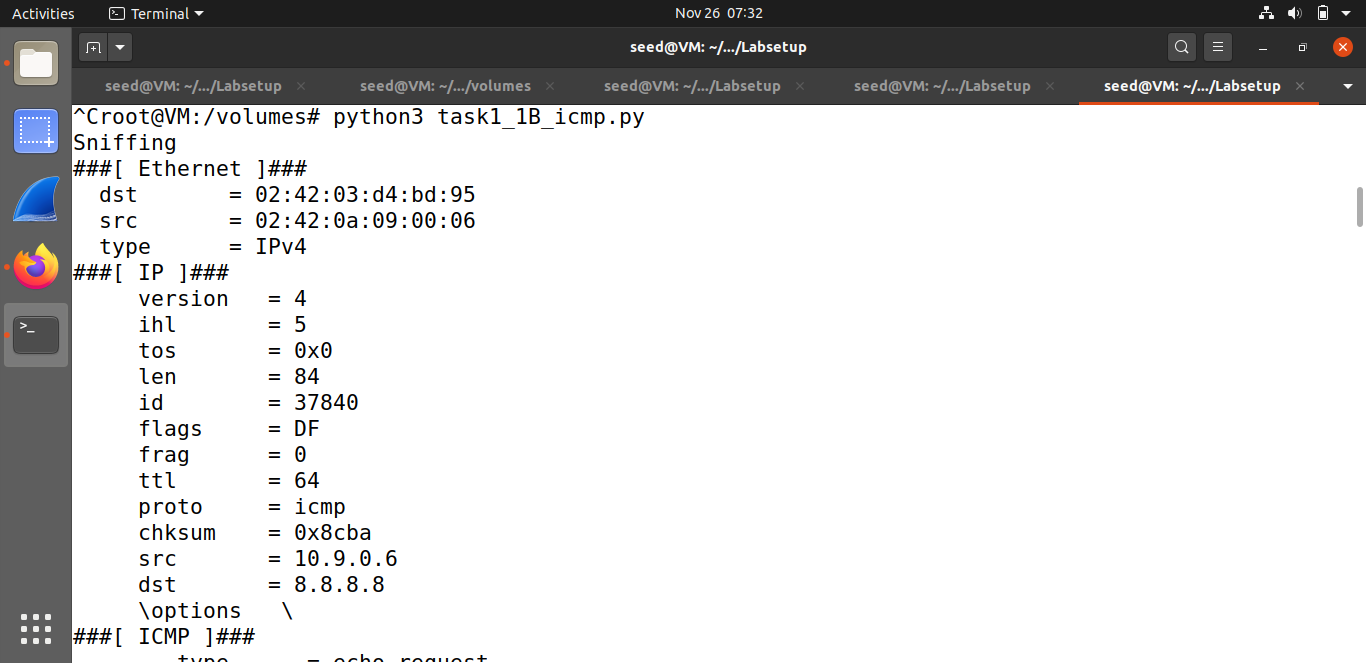


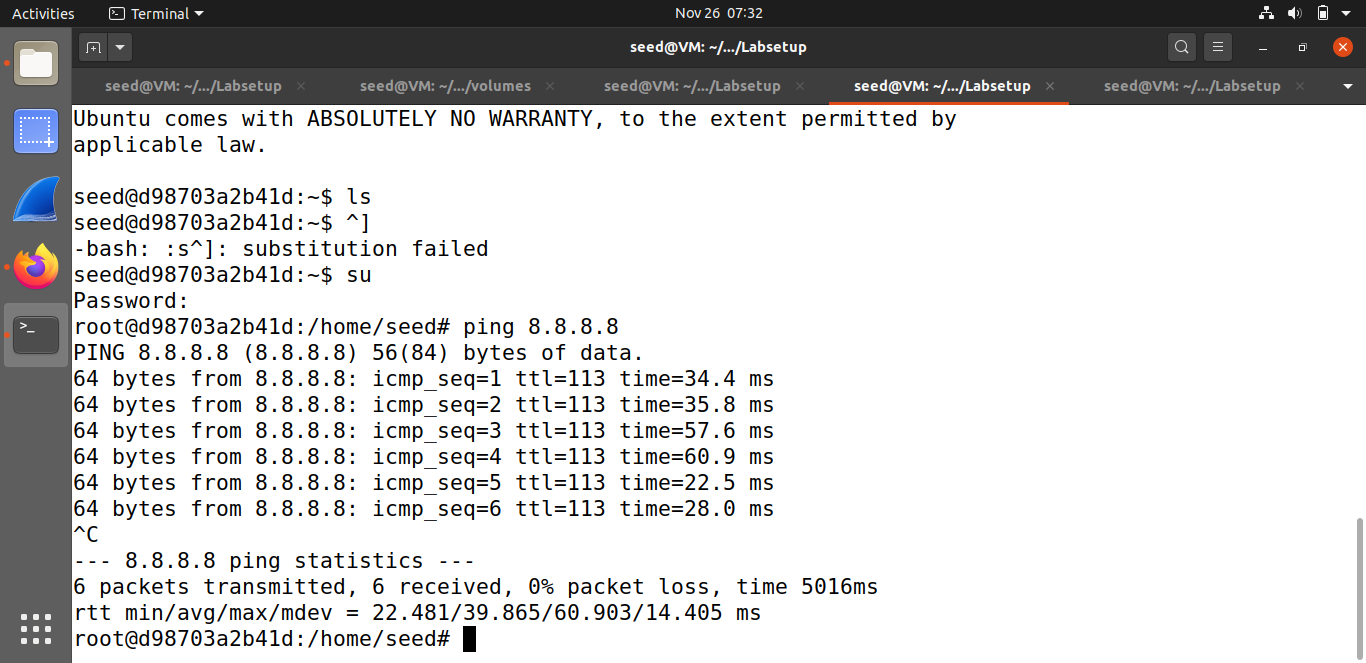
**Task#1.1B:**

* **Capture only the ICMP packet.**

I used the same filter like the last code which accepts the syntax of Berkeley Packet Filter. I have used the method ‘print\_pkt()’ but I wanted to print details returned. I sent a ping an IP of (8.8.8.8). This will generate an ICMP echo request packet. If it is alive, the program will receive an echo reply, and print out the response.

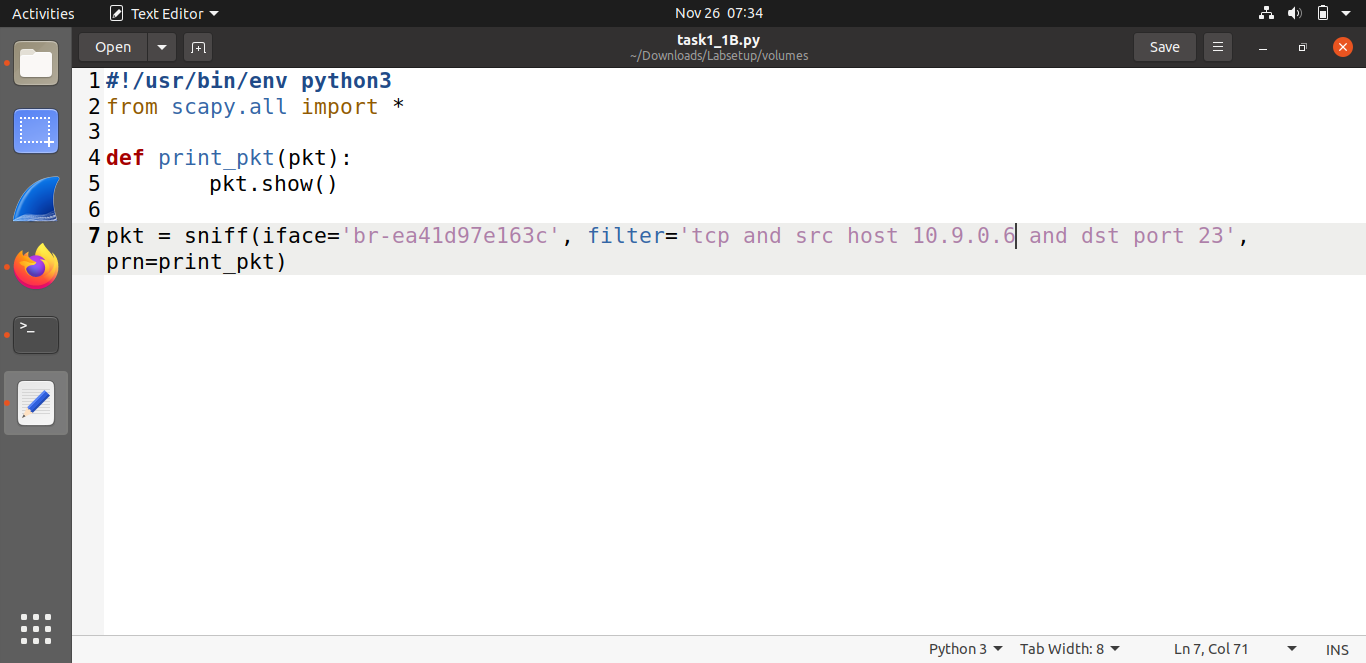


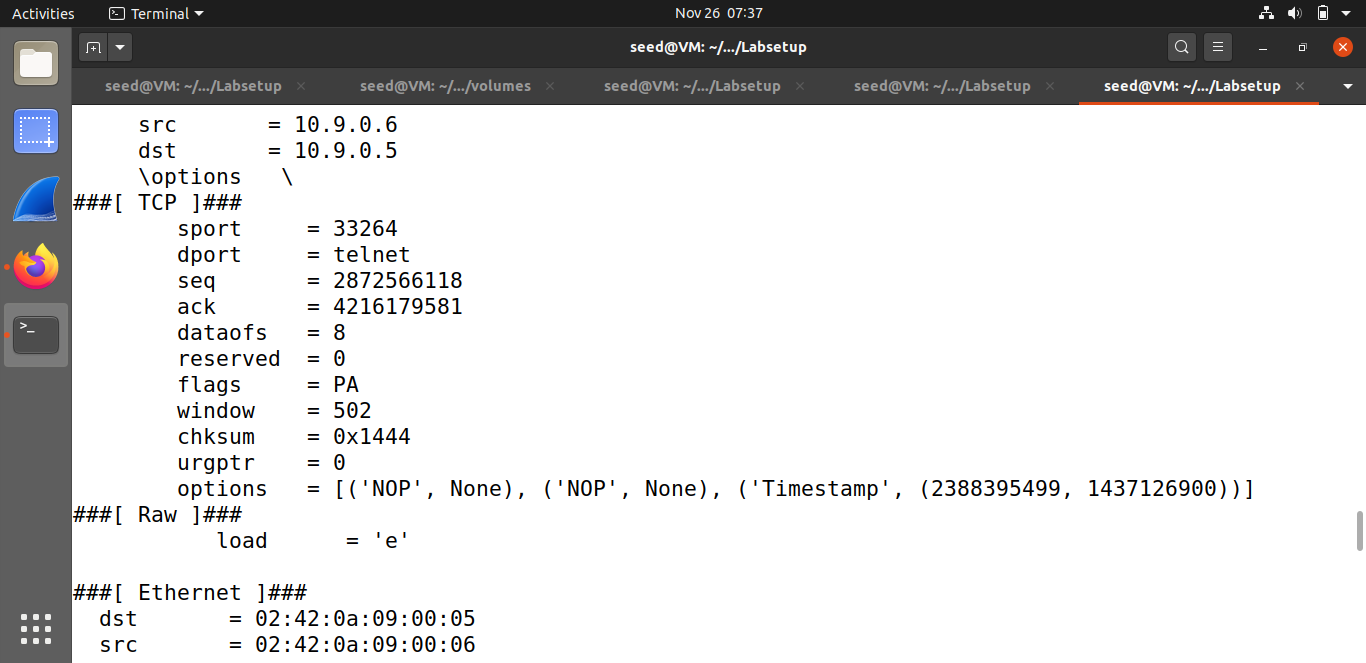


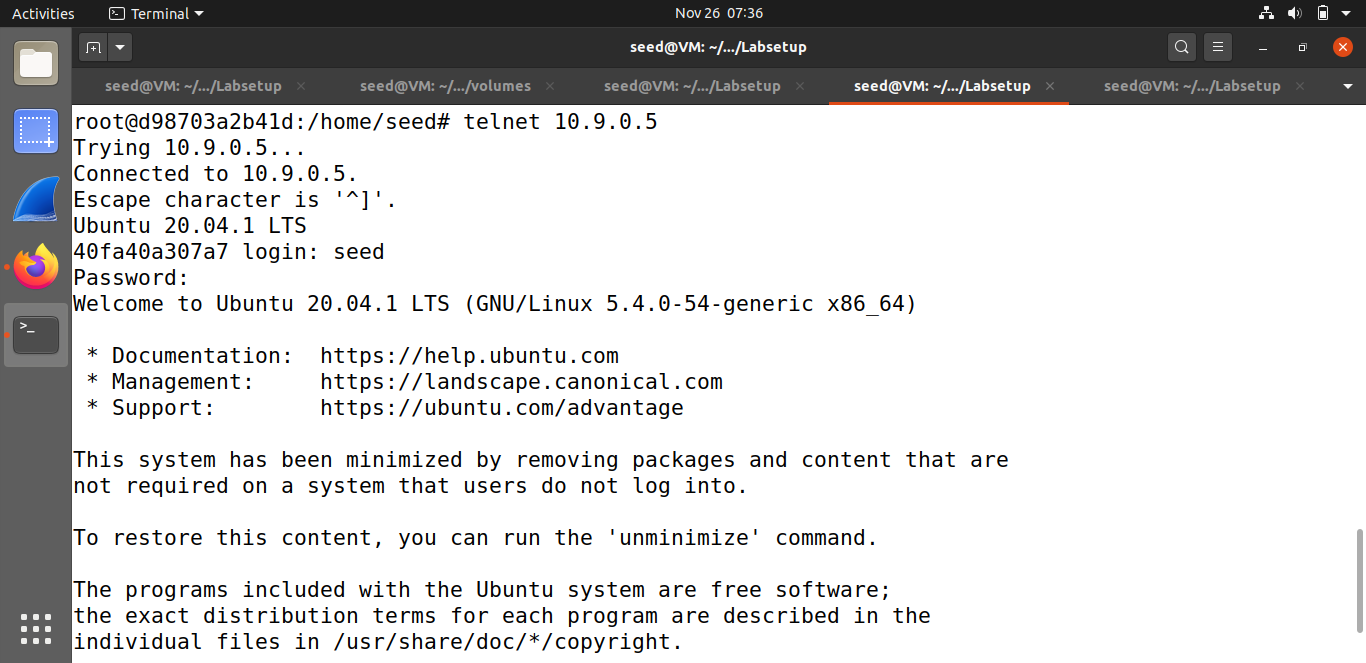


* **Capture any TCP packet that comes from a particular IP and with a destination port number 23.**

I filtered this time with 'tcp port 23 and src host 10.0.2.4' . The syntax I used for this filter is from BPF syntax website. My default VM’s IP is ‘10.9.0.6’ and I sent it to ‘10.0.2.5’ which is another terminal that I used. This way, I sent a command ‘telnet 10.0.2.5’ from my one terminal to the second one, and the program sniffed the TCP packets. This protocol is used to establish a connection to TCP port number 23 hence the use of telnet.



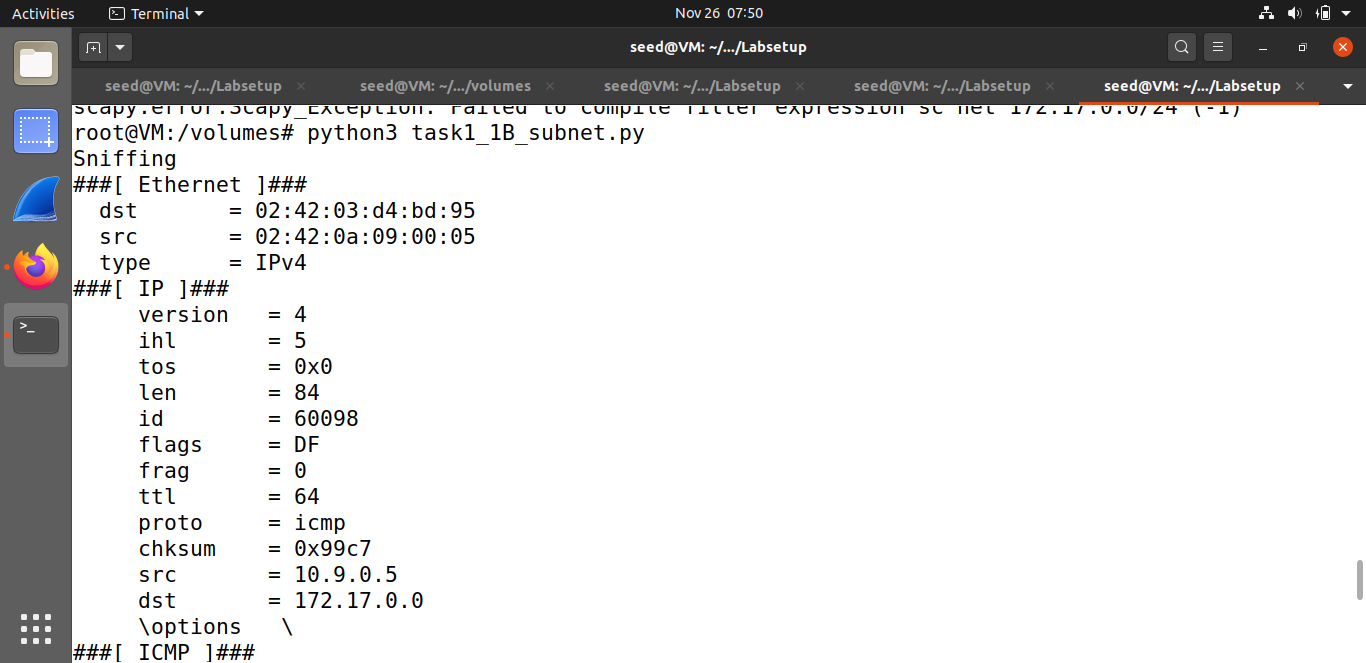




* **Capture packets comes from or to go to a particular subnet.**

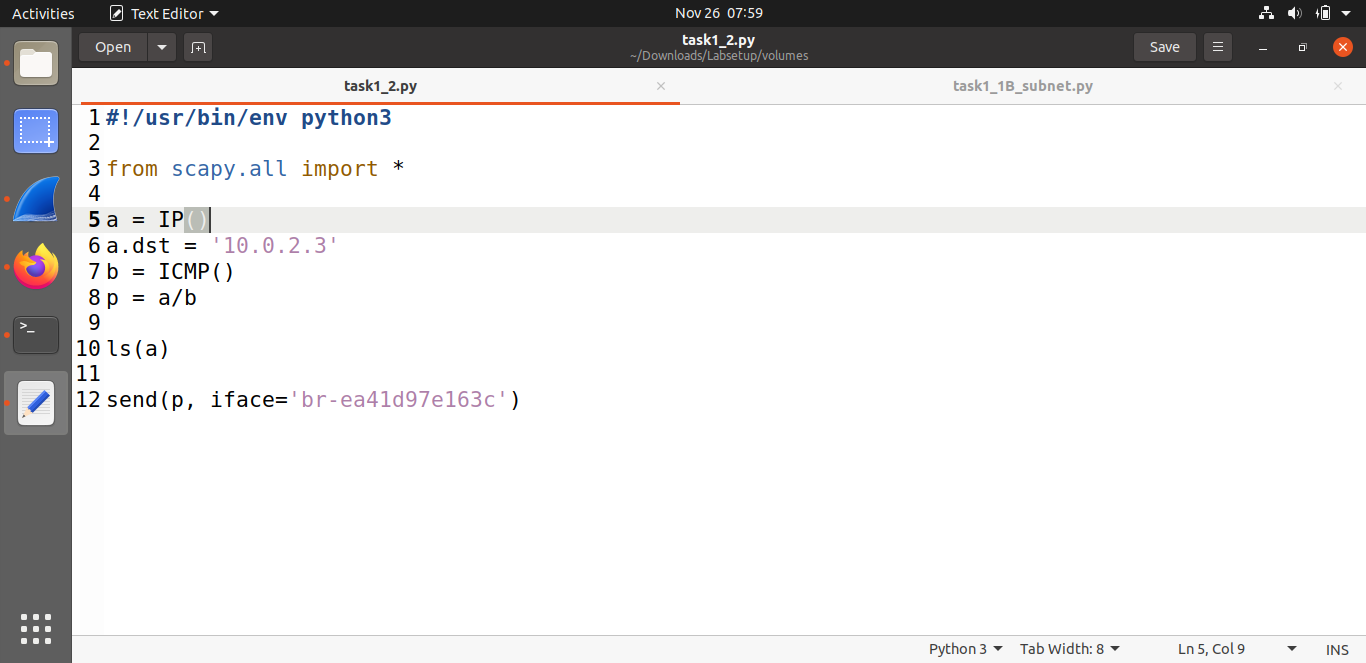
I picked the filter using Berkeley Packet Filter syntax: 'dst net 172.17.0.1/24' . A packet was sent to a particular subnet and the program sniffed only packets that were sent from source ‘10.0.2.5’ to destination IP from the other subnet.

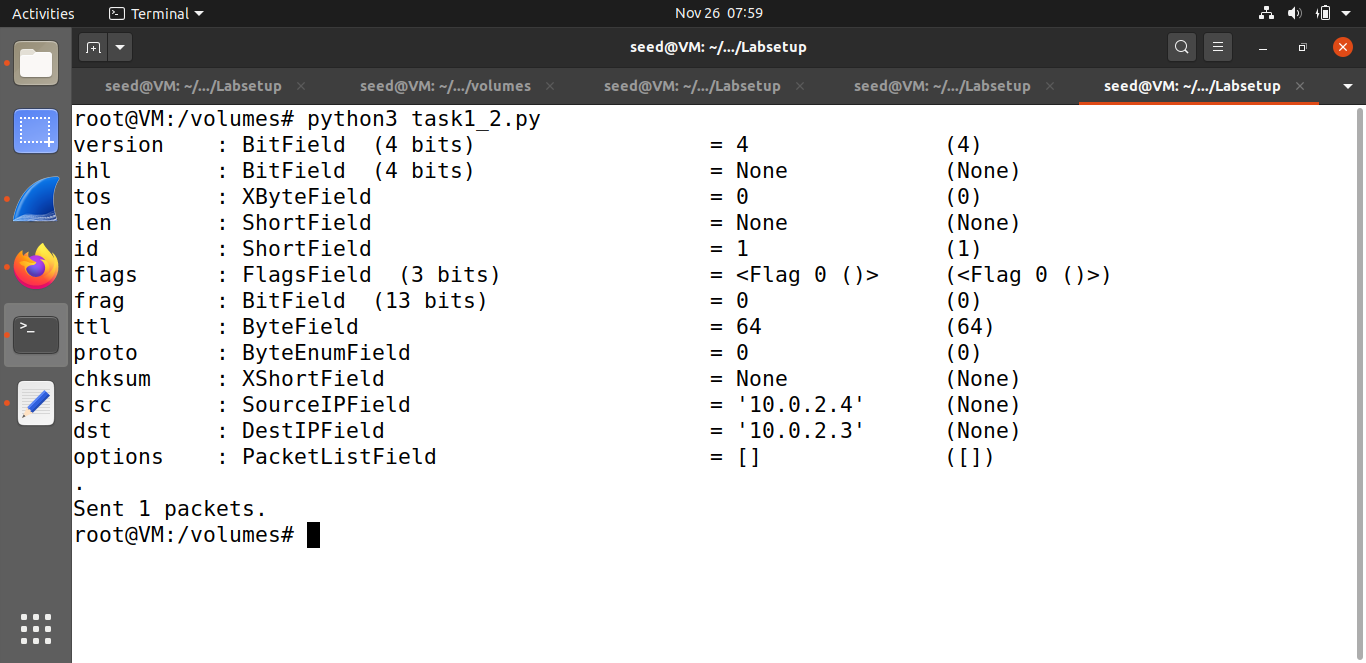
****

****

**Task#1.2:**

When a normal user sends a packet, the OS usually does not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers). The OS will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc. However, if users have the root privilege, they can set any arbitrary field in the packet headers. This is called packet spoofing. I spoofed ICMP echo request packet, and sent it to another terminal on the same subnet. Using scapy library, the source IP was overwritten with our own IP and sent the packet to destination 10.0.2.3; the packet was received by 10.0.2.4 and sent an echo reply back.

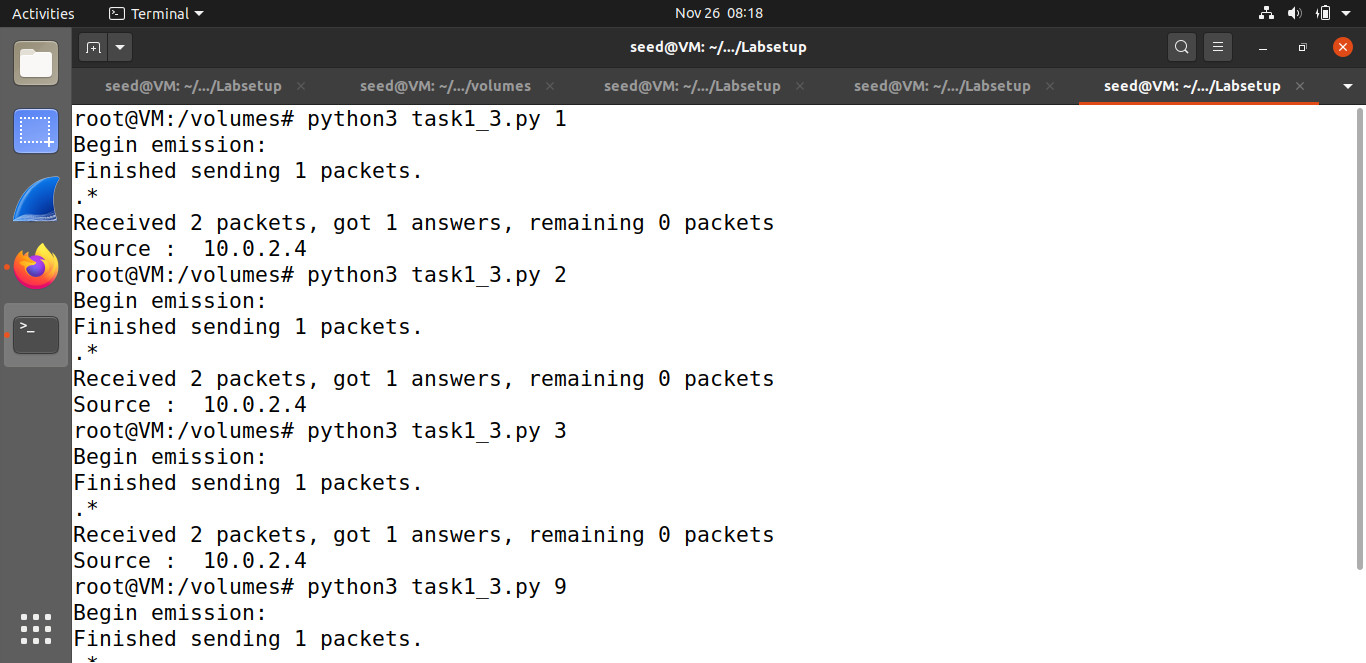
****

****

**Task#1.3:**

Traceroute is a computer network diagnostic command for displaying possible routes and measuring transit delays of packets across an Internet Protocol network. This program figures out how many routers (hops) it takes to send out that packet all the way to the IP address destination. Each line at the display is a different router. The time- to-live is used to return an error of each hop till the destination, this way we can print each IP router till it stops. In this case we reached 15 different routers, 5 of them are timed out. A “Request timed out” message at the beginning/middle of a traceroute is very common and can be ignored. This is typically a device that doesn’t respond to ICMP or traceroute requests.

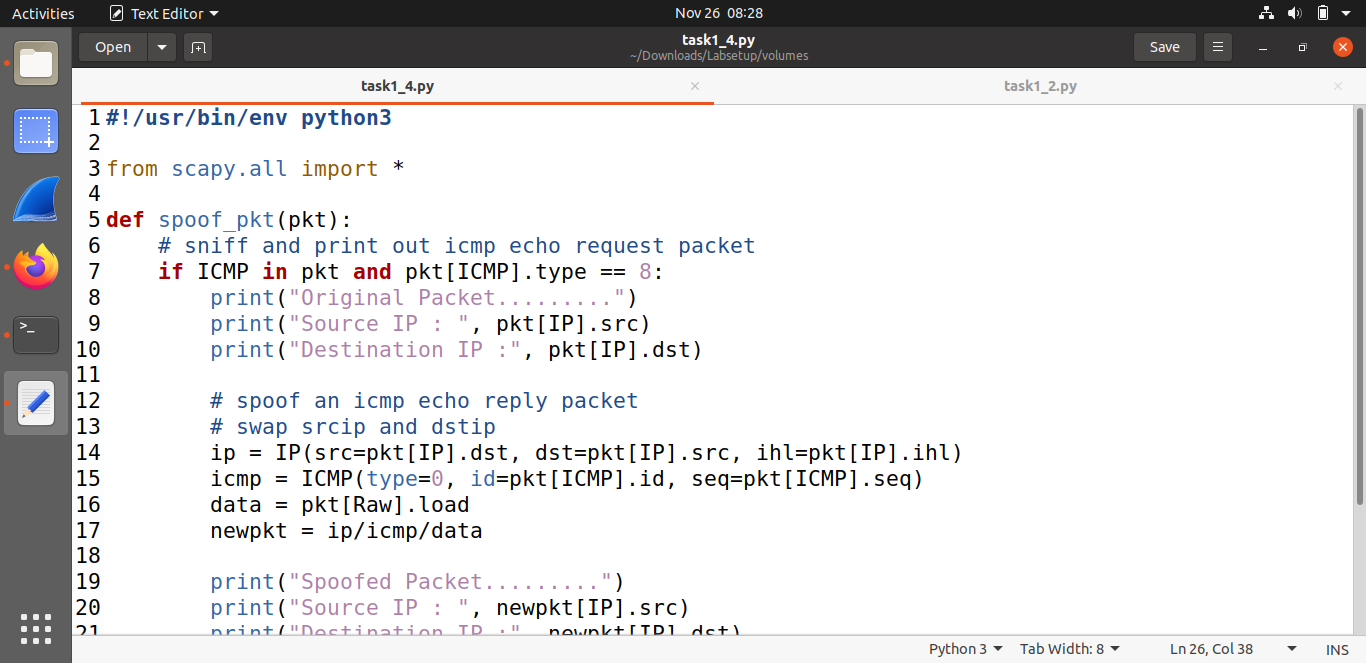
****

****

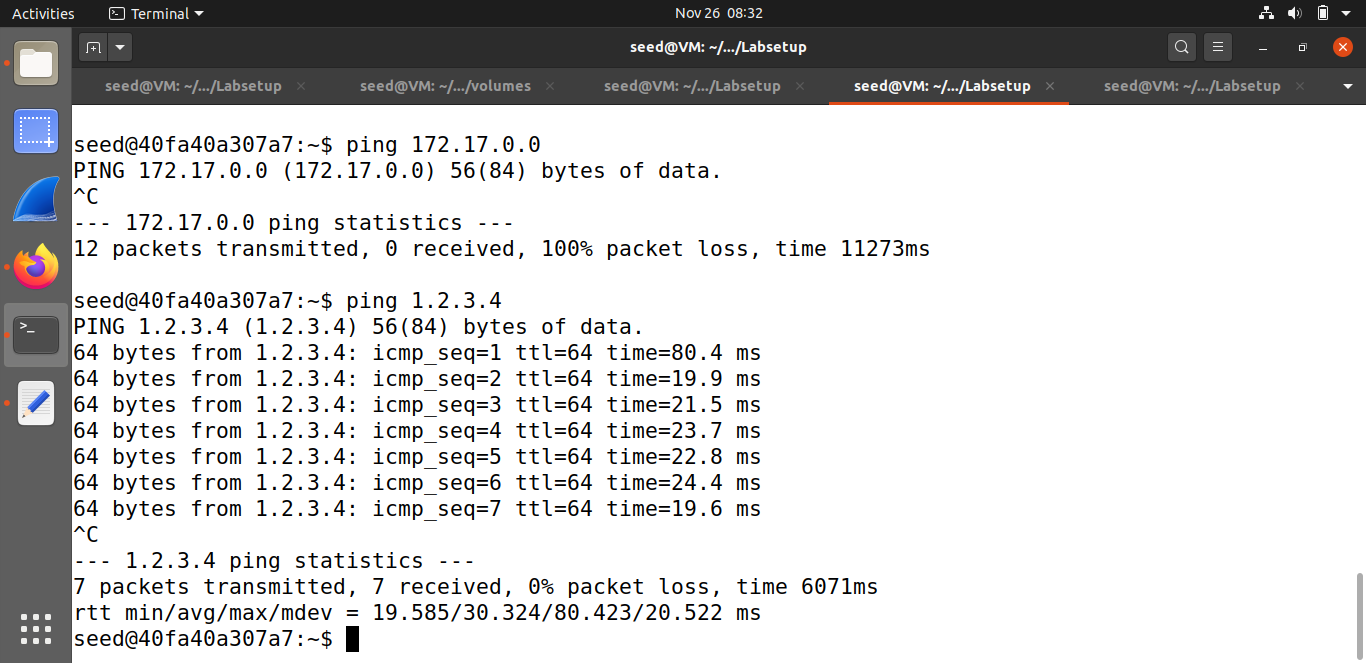
**Task#1.4:**

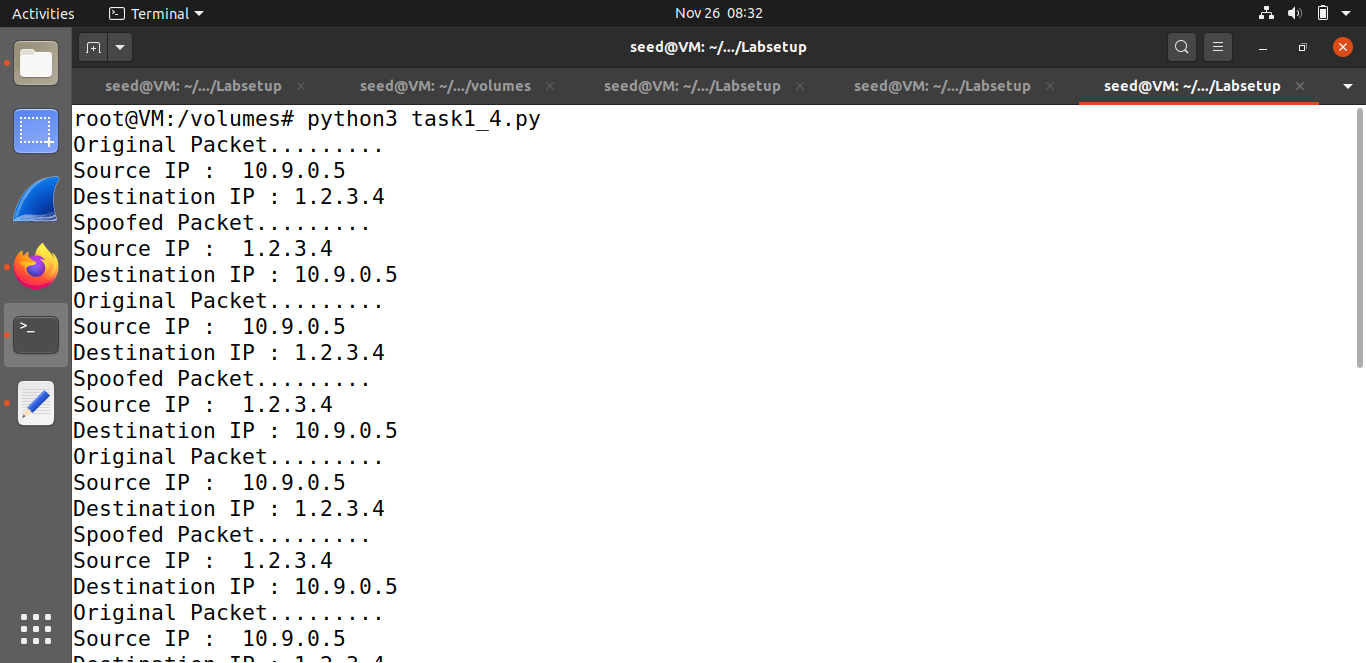
ARP protocol is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. Such a request consists of special packets commonly known as ‘who-has’ packets. ‘Who-has’ packets are packets that the IP system broadcasts to all devices on a network (or VLAN), to determine the owner of a specific IP address. The ‘if’ block is checking if an ICMP is a request. If it’s true, the reply packet will be based on details derived from the original packet, but it will flip dst and src so whenever it sees an ICMP echo request, regardless of what the target IP address is, the program should immediately send out an echo reply using this packet spoofing technique. The pkt[Raw].load is used to store the original packet data payload this way it Will return properly to the sender.

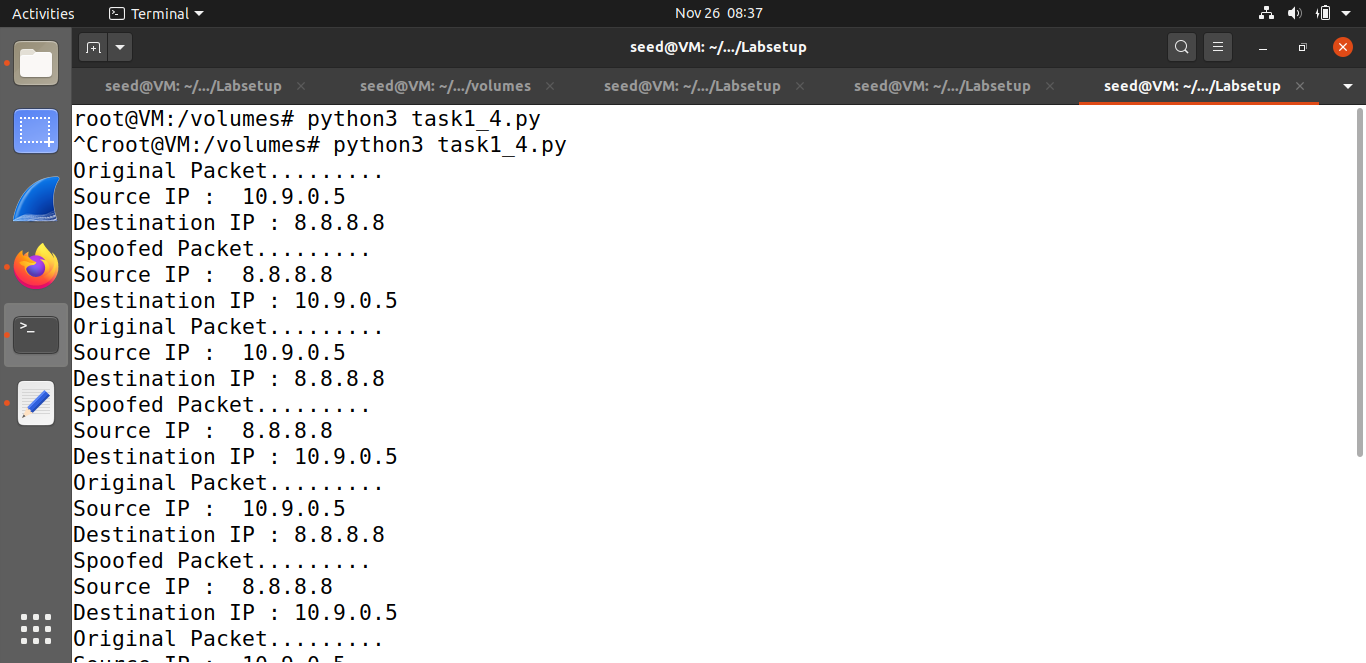
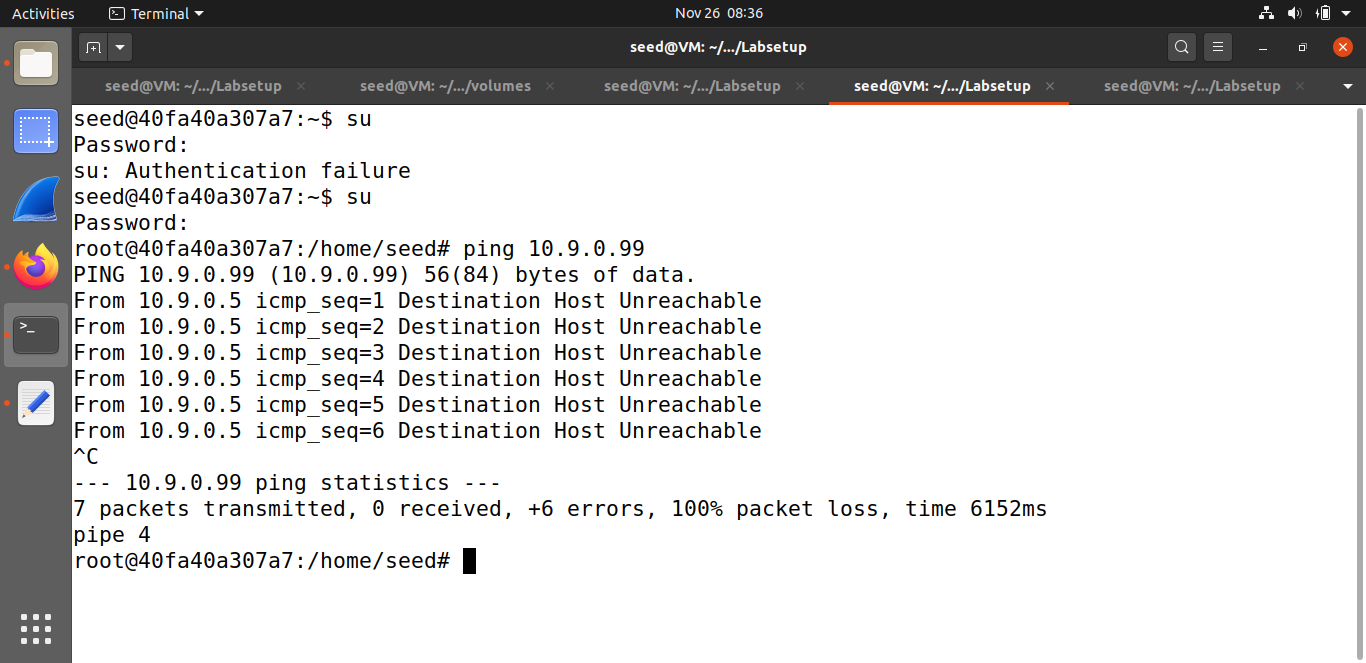
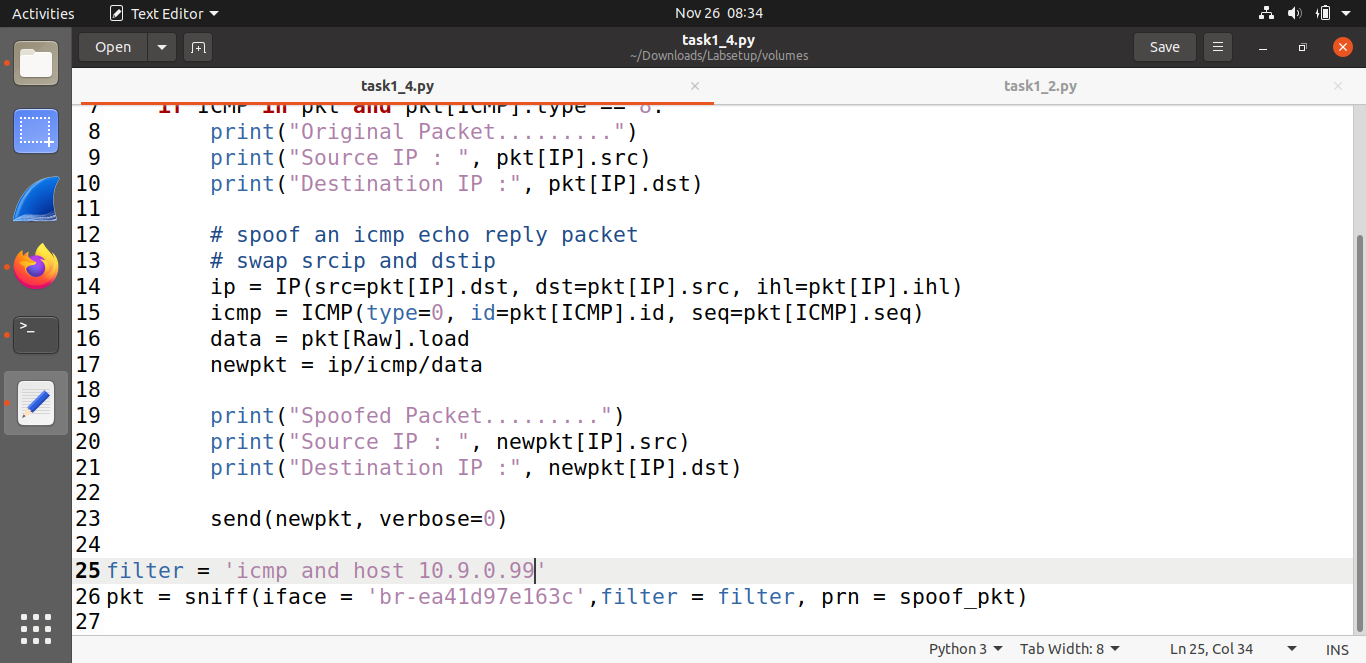
1. Terminal 1 sends a ping to ‘172.17.0.0’ which is a non-existing host on the Internet. Without the program we will get a 100% packet loss because it will never return to the source. The attacker (my program on terminal 2) returns with an answer to it and the ICMP packet reply is coming back to VM2.

****

1. VM2 sends a ping to ‘10.9.0.99’ which is a non-existing host on the LAN. In this scenario we’re getting the same concept with the same idea of the ARP protocol. Even though the host doesn't even exist. The program from VM1 will send back an ICMP response packet.

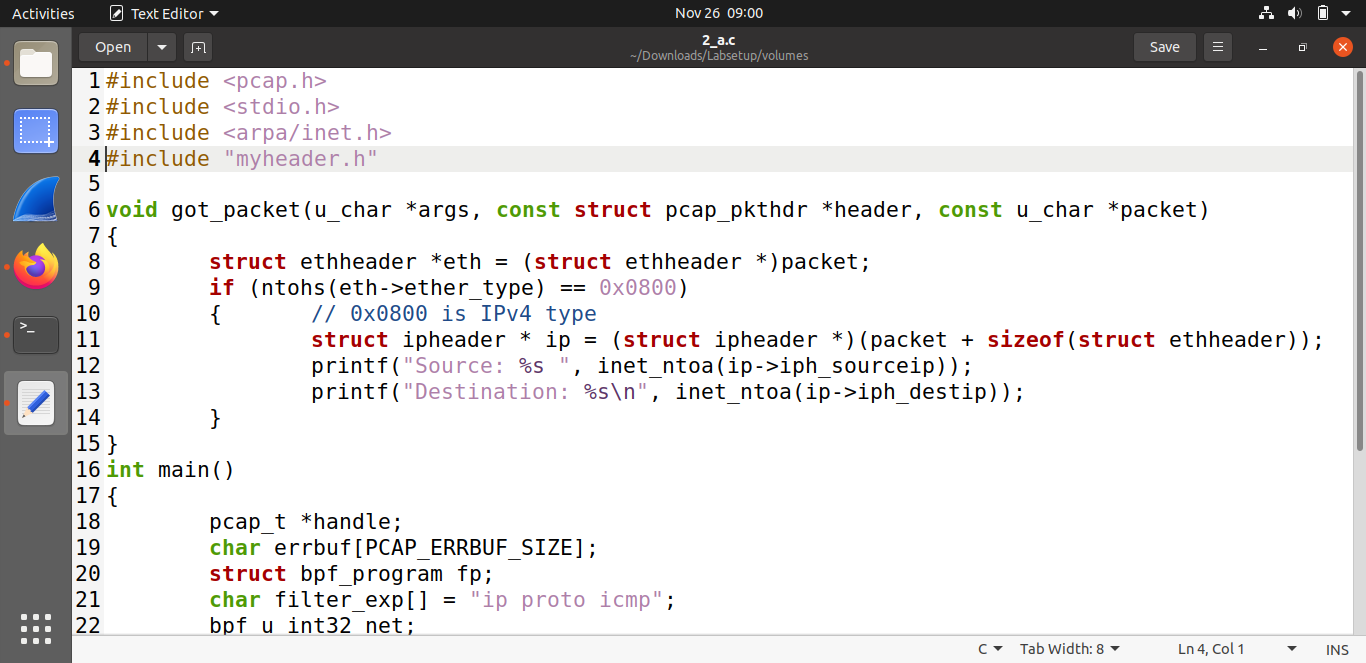
****

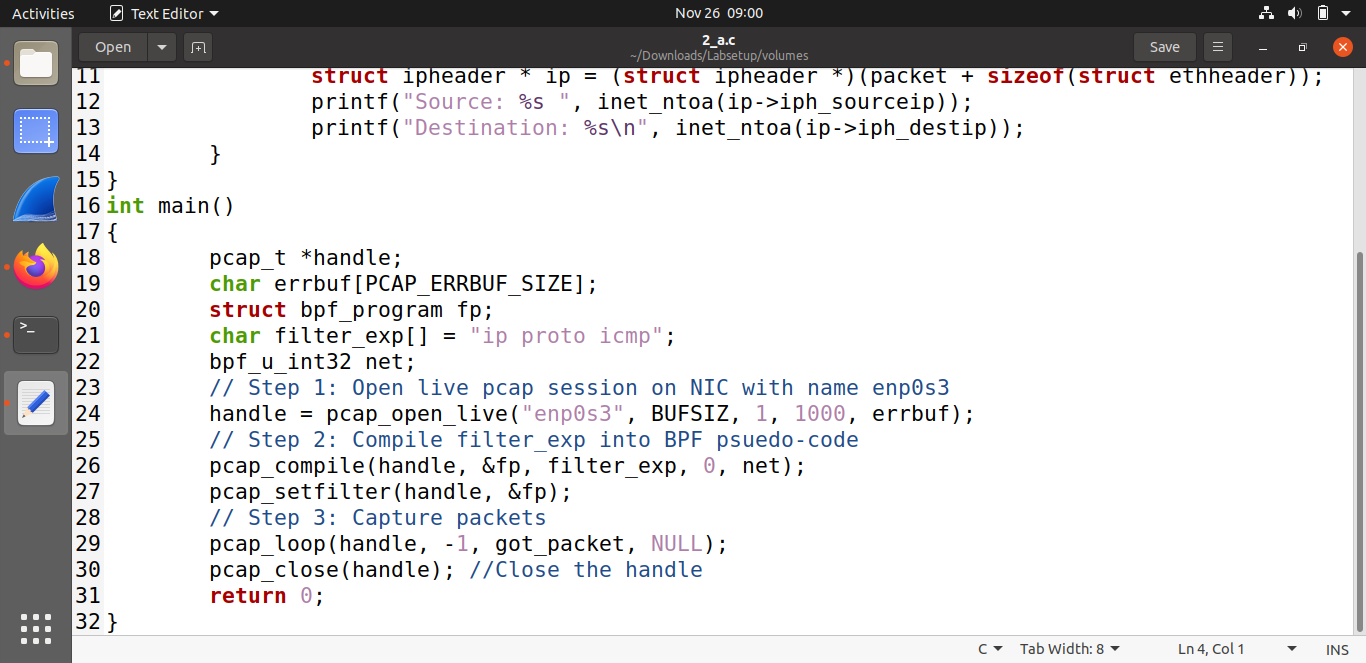
****

1. Terminal 2 sends a ping to ‘8.8.8.8’ which is an existing host on the Internet. This scenario is dierent from the others because it really exists on the net. So in this case we’re getting duplicate responses, that’s because the real destination is responding to the source, but my program is also responding to the source. ****

**Task#2.1:**

PCAP is an application programming interface (API) for capturing network traffic. I created a sniffer program using pcap library for capturing network traffic and displays the source and the destination IP addresses. I used the same filter syntax of BPF for filtering only ICMP packets. When the program captures a packet. It checks if the header is IPv4 type and if it’s true, it will print the source and destination of that IP header packet. I sent a ping with IP and the program sniffed it and printed the correct source and destination.



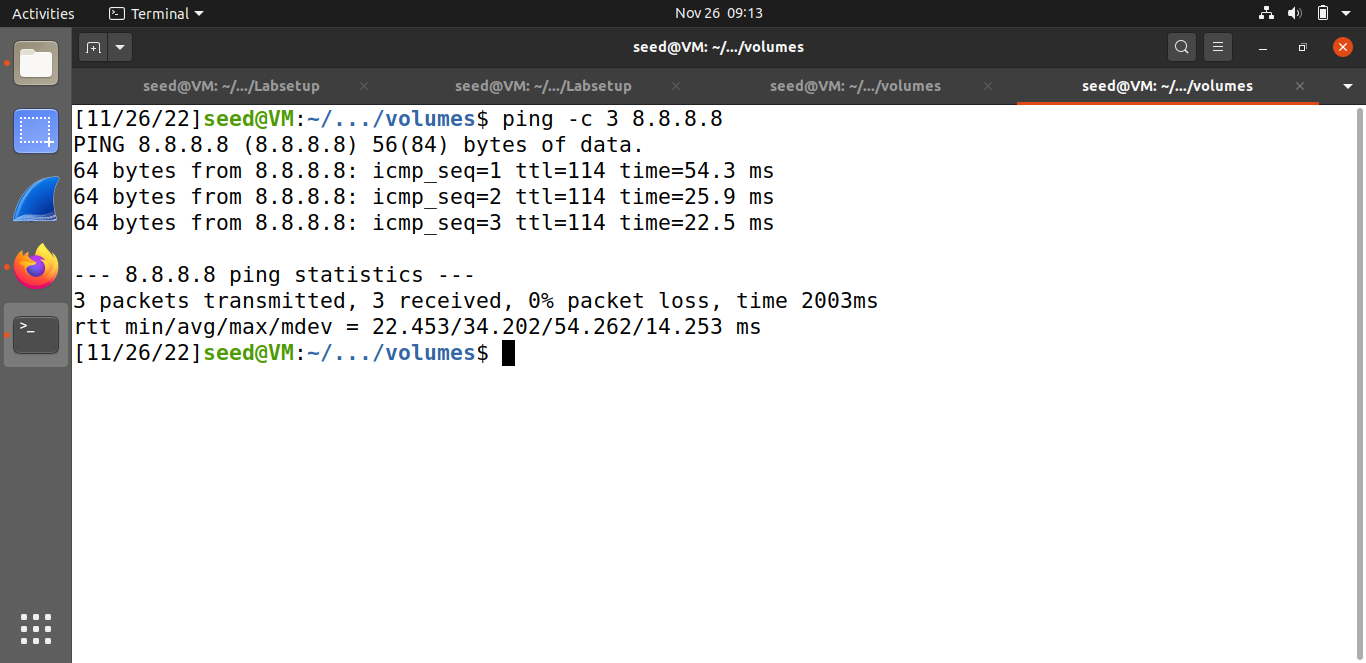
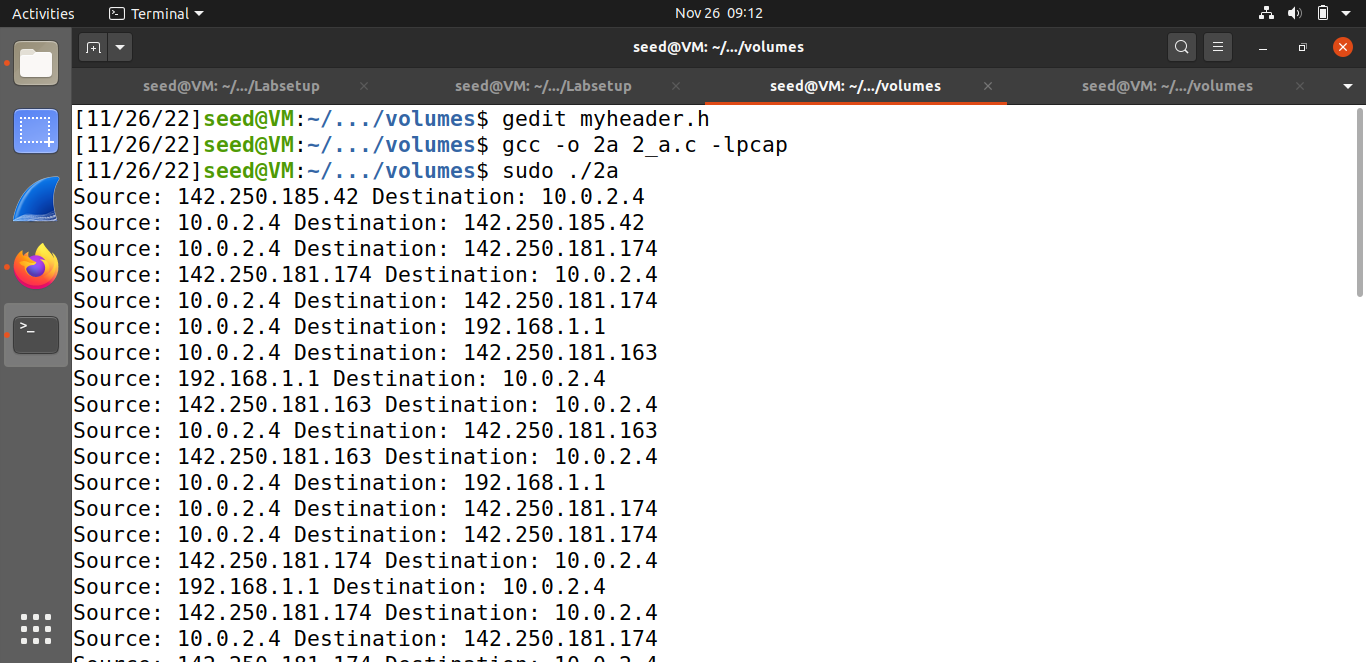


**Task#2.1A:**

**Solution Q1:** First step, we open a live pcap session on NIC with name enp0s3, this operation is done by the ‘pcap\_open\_live’ (a function from the pcap library). This function lets us see the whole network traffic in the interface and binds the socket. Second step, we are setting the filter by using the following methods: pcap\_compile() is used to compile the string str into a filter program pcap\_setfilter() is used to specify a filter program. Third step, we capture the packets in a loop and process the captured packets using the ‘pcap\_loop’ function, the -1 means an infinity loop.

**Solution Q2:** A root privilege is required to set up the card in promiscuous mode and raw socket, this way we can see the whole network traffic in the interface. If we run the program without a root user, where the pcap\_open\_live function fails to access the device and so it will cause an error to the whole program.

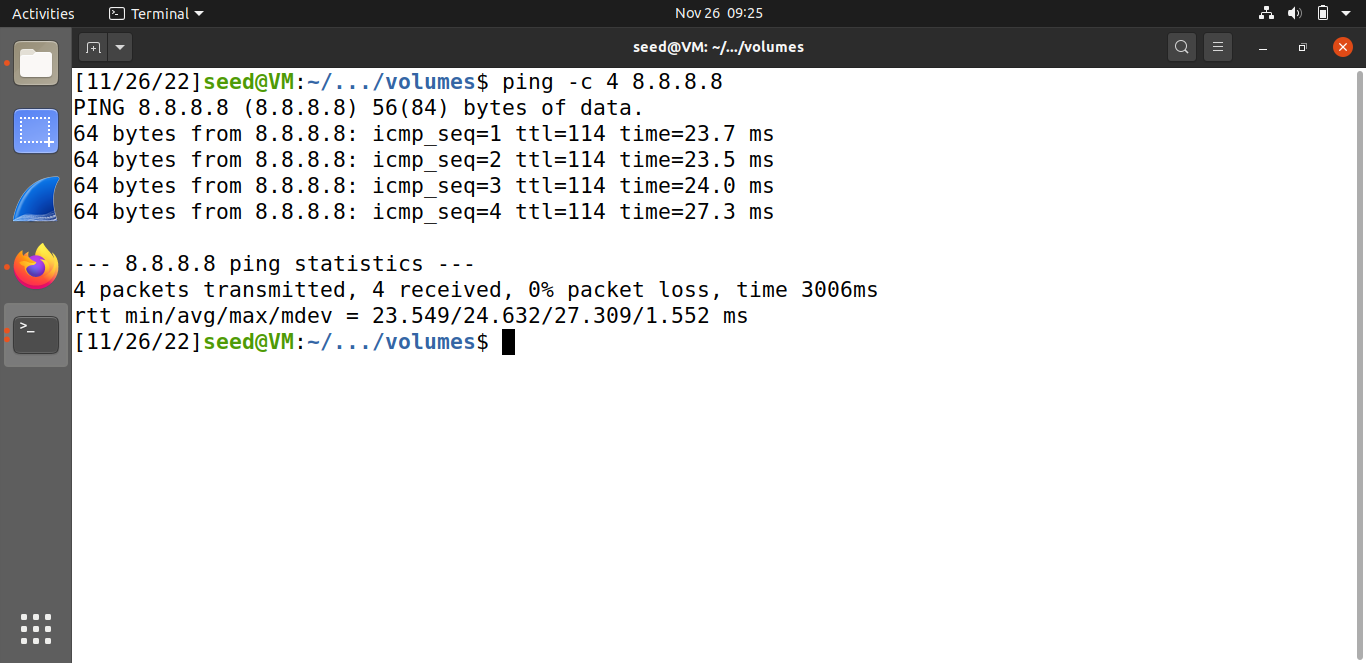
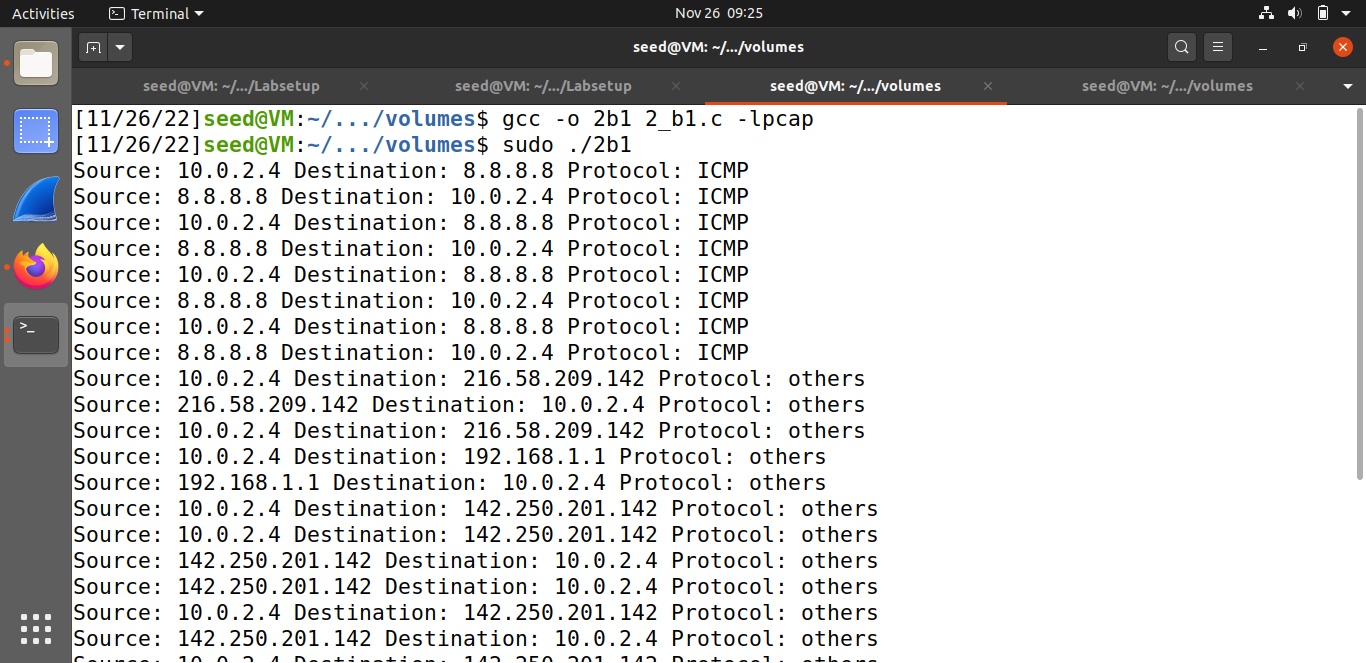
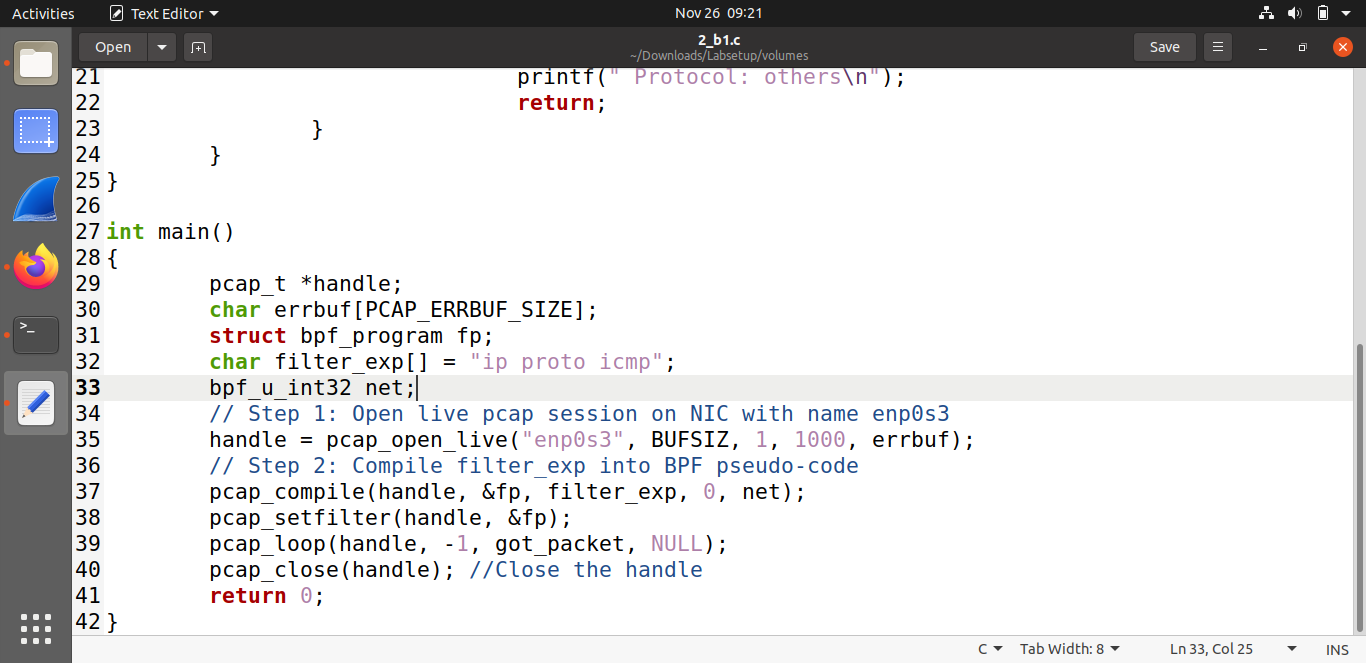
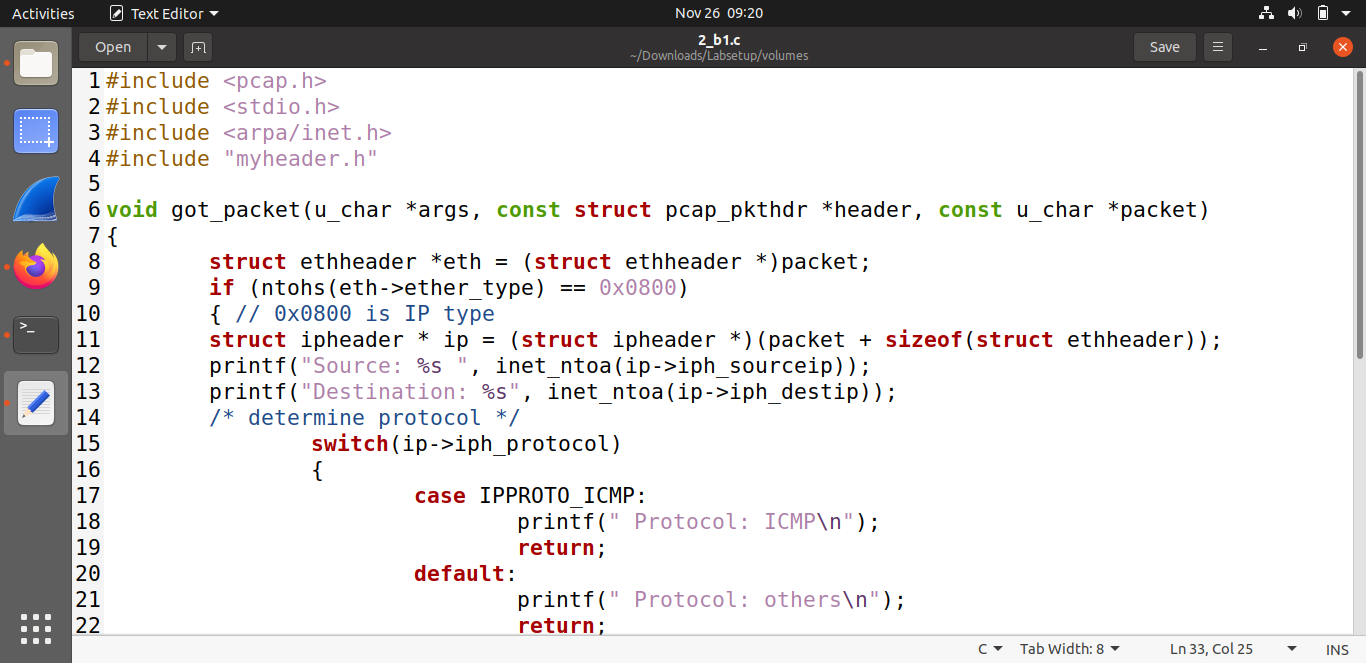
**Solution Q3:** The promiscuous mode is a part of the chip in my NIC card, which is within the computer, activated using the ‘pcap\_open\_live’ function. If you change the third param of the ‘pcap\_open\_live’ function to 0 = OFF and anything other than 0 will be ON. If I’ll turn the promiscuous mode OFF, a host is sniffing only traffic that is directly related to it. Only traffic to, from, or routed through the host will be picked up by the sniffer. On the other hand, if I turn the promiscuous mode ON, it sniffs all traffic on the wire and you will get all packets your device sees, whether they are intended for you or not.

****

**Task#2.1B:**

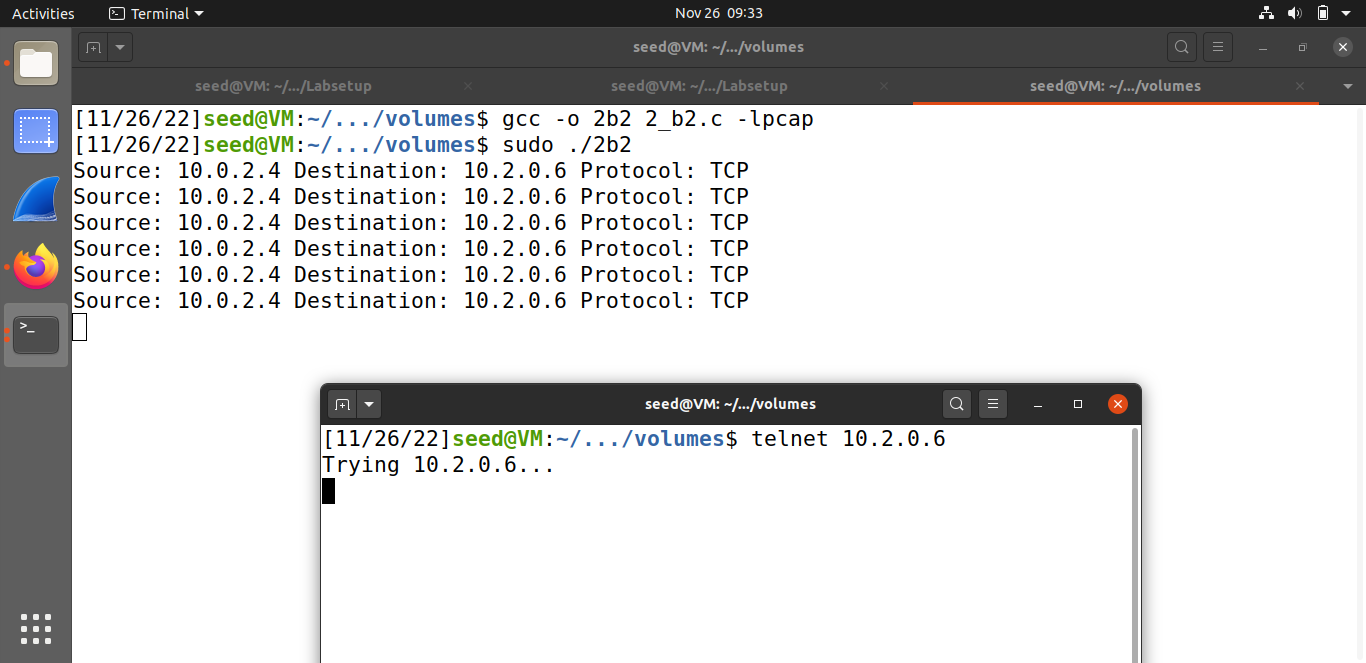
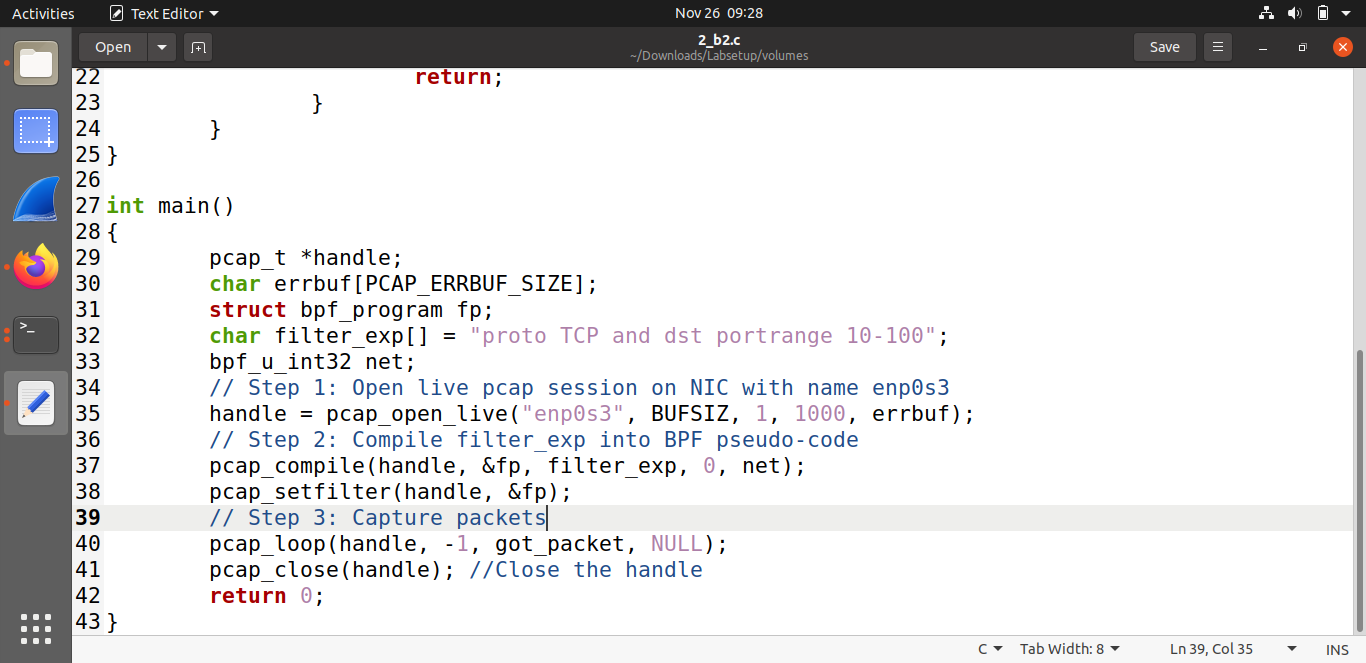
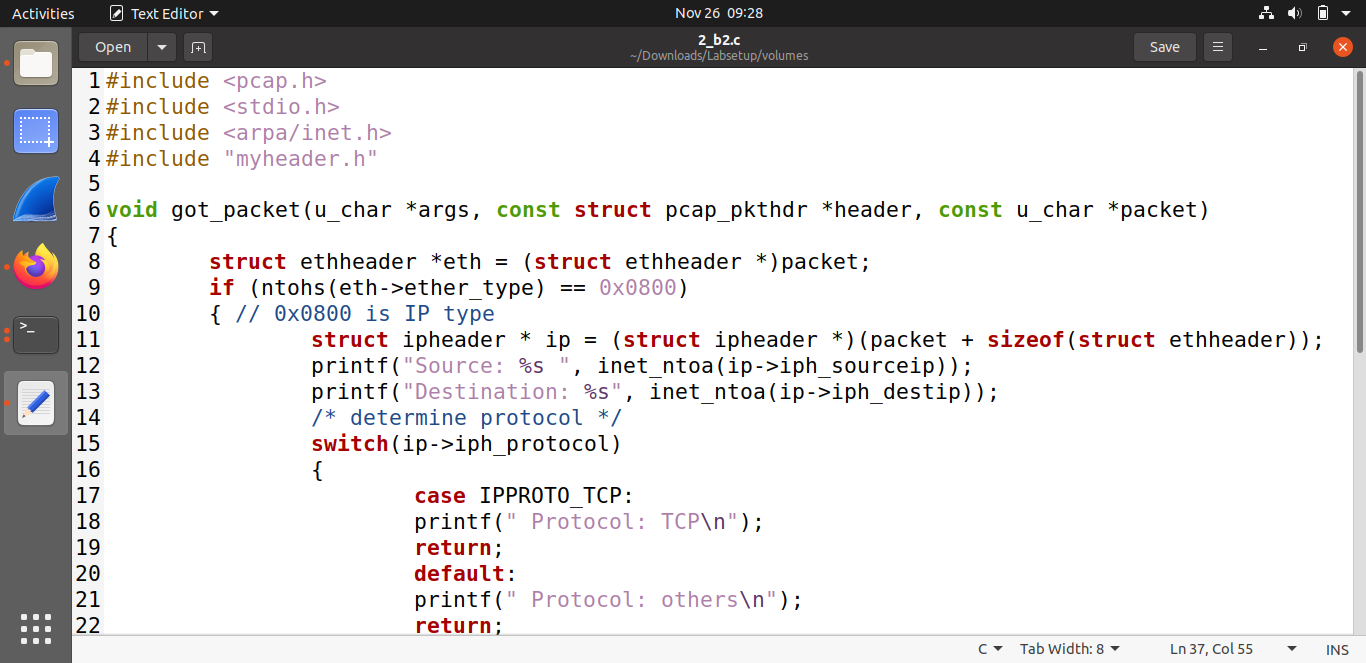
I took the previous code and added some features to it. The pcap filter which is based on the BPF syntax is now: "ip proto icmp". My current IP address is 10.0.2.4. The program checks if it’s IPv4 type and if it’s true, it’s also checks if the protocol is ICMP - In this case we’ll also print that it’s an ICMP protocol type. 2. About the question : I used another VM (10.0.2.6 - the victim) which sent a ping to ‘8.8.8.8’. The attacker (10.0.2.4) sniffed the packet and displayed it.

* **Capture the ICMP packets between two specific hosts.**

****

* **Capture the TCP packets with a destination port number in the range from 10 to 100.**

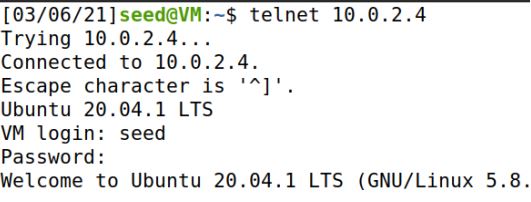
My pcap filter is: "proto TCP and dst port range 10-100. The program captures the packet and checks if the header is IPv4 type. If it’s true, it also checks if the protocol type is TCP, and if it’s also true it will print it. I used telnet to capture the TCP packet and sent it to another alive terminal. The program captures the packet and displays it. The syntax used for this filter is from BPF syntax.

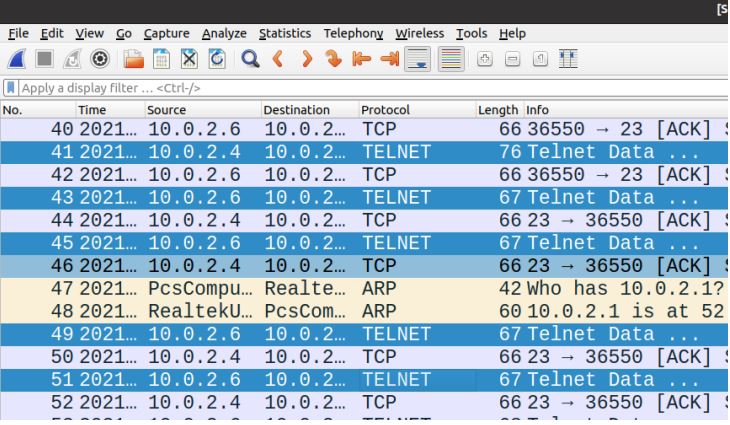
****

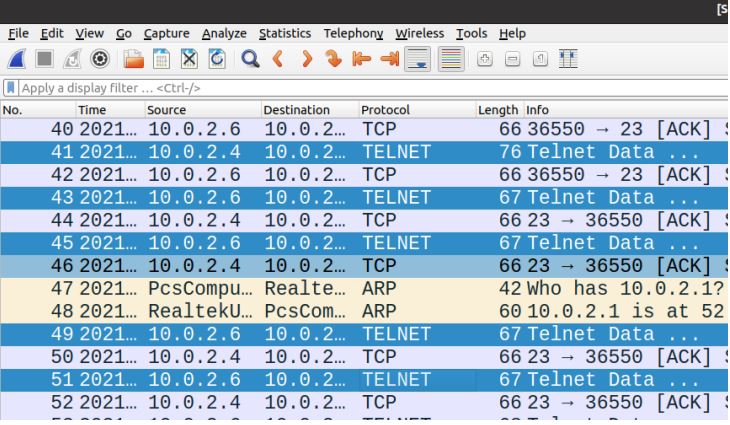
**Task#2.1C:**

My pcap filter is: "tcp port telnet". The syntax used for this filter is from BPF syntax website. The program was set to sniff the tcp packets of telnet and when executed and performed a telnet from machine 10.0.2.4 to 10.0.2.6; the data was captured which includes password. The program is running and listening to the tcp packets. As telnet is a tcp program, the packets are captured, and the payload was displayed and in a clear text. I marked the password in red as we can see in the screenshots down below.



****

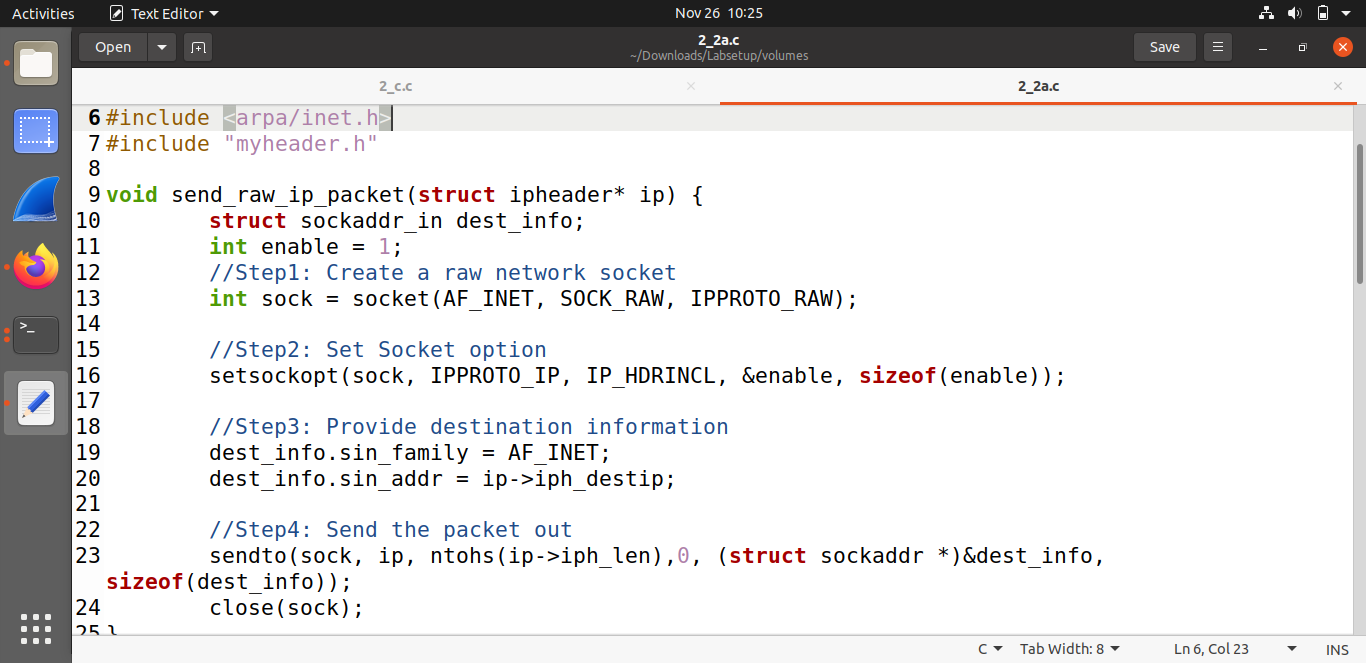
****

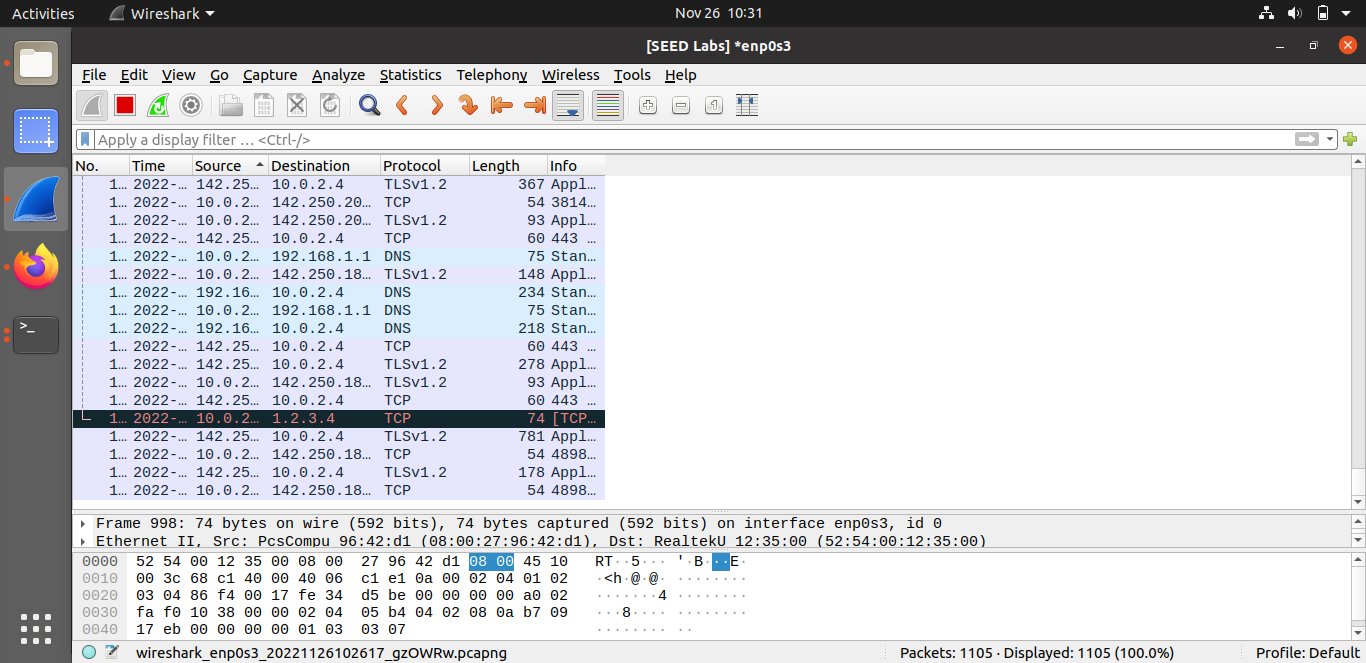
****

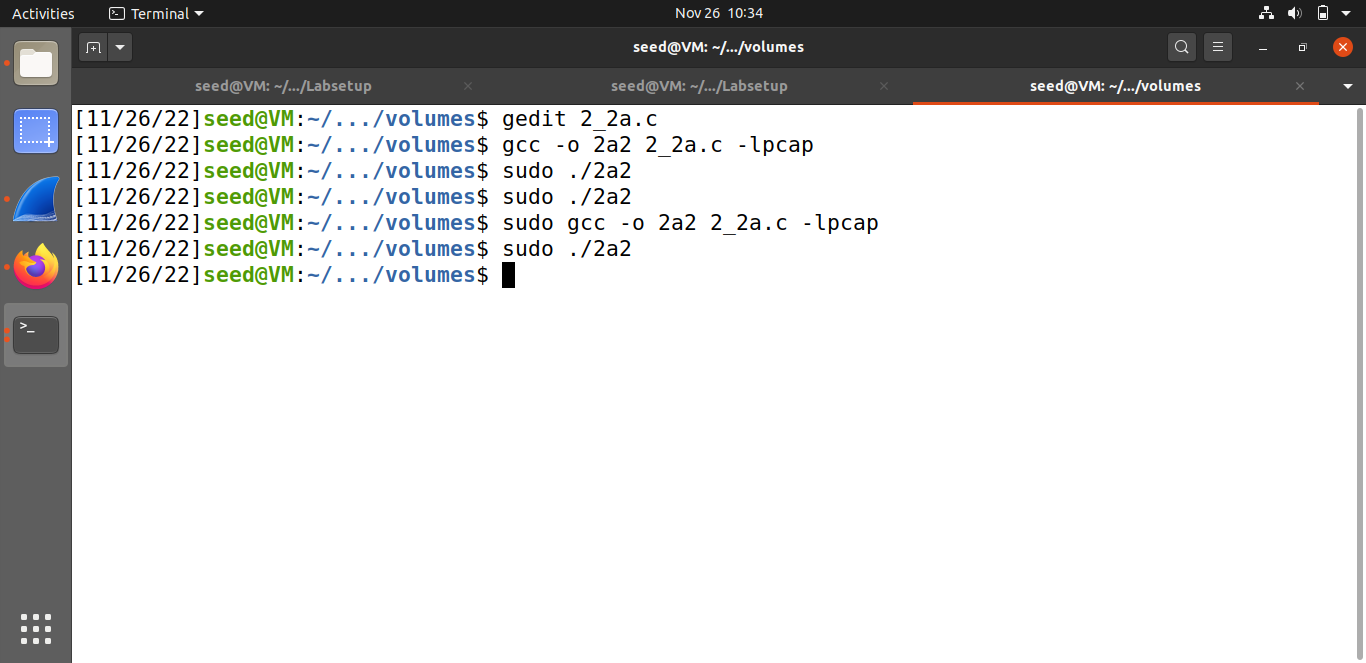
**Task#2.2A:**

The program spoofing between another active VM (10.0.2.6) to 1.2.3.4 . I took the example from the task info and added to it a header that contains a UDP protocol and sent it to destination 10.0.2.6 (from - 1.2.3.4) but faking it. The program was created with a pcap library and modified the IP headers to use the source IP as 1.2.3.4 and destination as victim IP (10.0.2.6). When executed the packet was created with 1.2.3.4 and sent to the victim. I worked with 2 terminals. I created the spoof program using pcap library and when executed the spoofing machine (10.0.2.4) sent a packet to the victim machine (10.0.2.6) with a fake IP address (1.2.3.4)







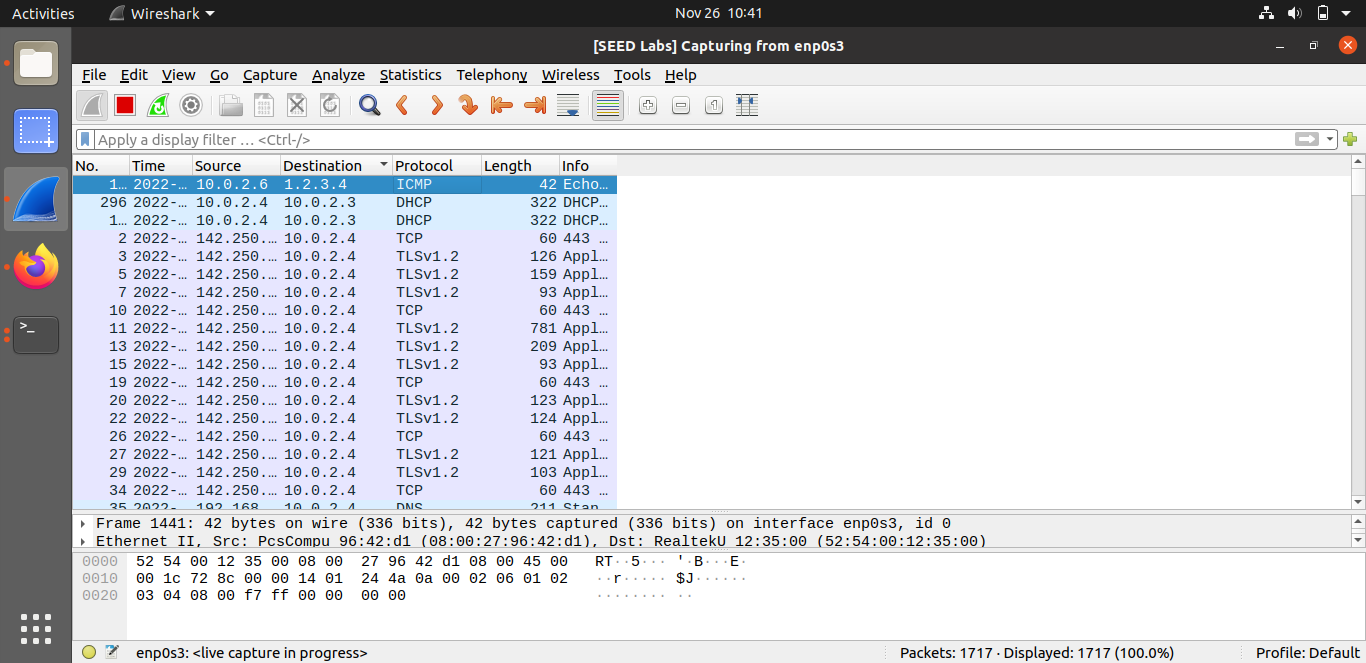


**Task#2.2B:**

Though the ICMP request originated from 10.0.2.4, the attacker created the packet with a spoofed IP (victim’s). So, the remote server once received the ICMP packet, it responded back to the source IP that is present in the packet instead of sending to the attacker. Thus, the attacker spoofed an ICMP Echo request. Created a spoof ICMP request from attacker machine with source IP as victim (10.0.2.6) and sent to remote server (1.2.3.4); the remote server responded to ICMP request and sent it to the victim (10.0.2.6).



****

****

**Solution Q4:** Yes, the IP packet length field can be any arbitrary value. But the packet's total length is overwritten to its original size when it’s sent.

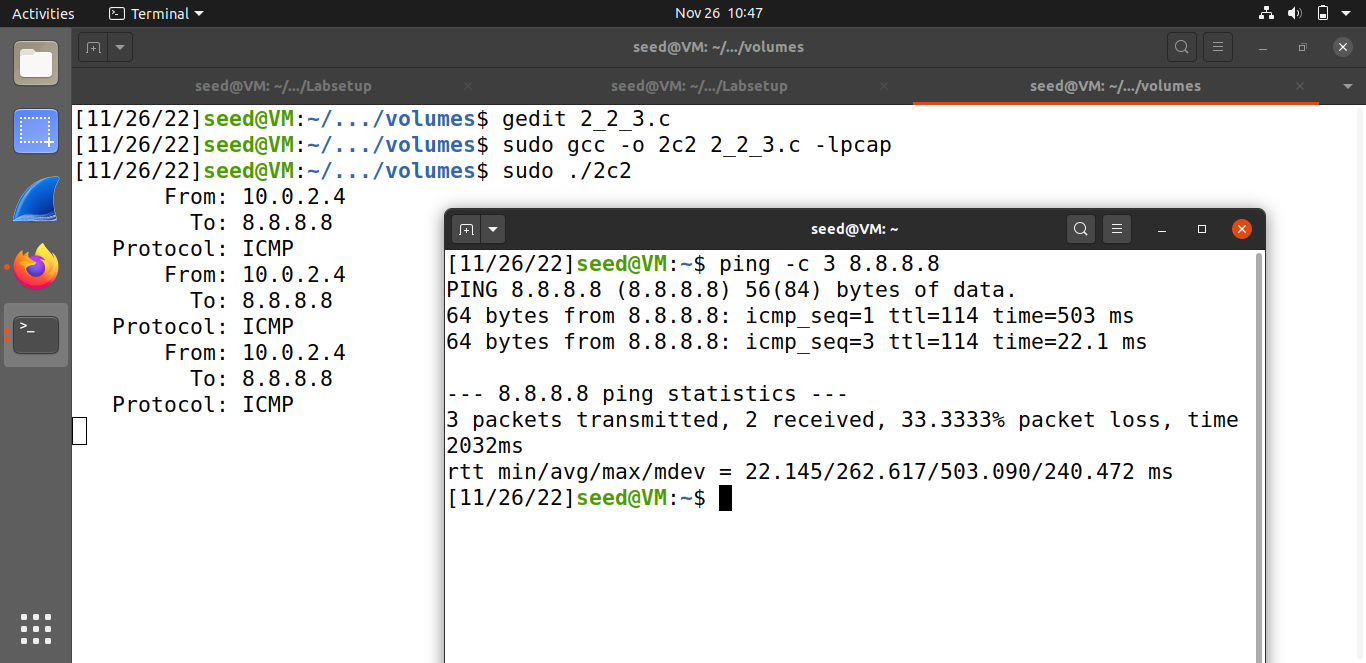
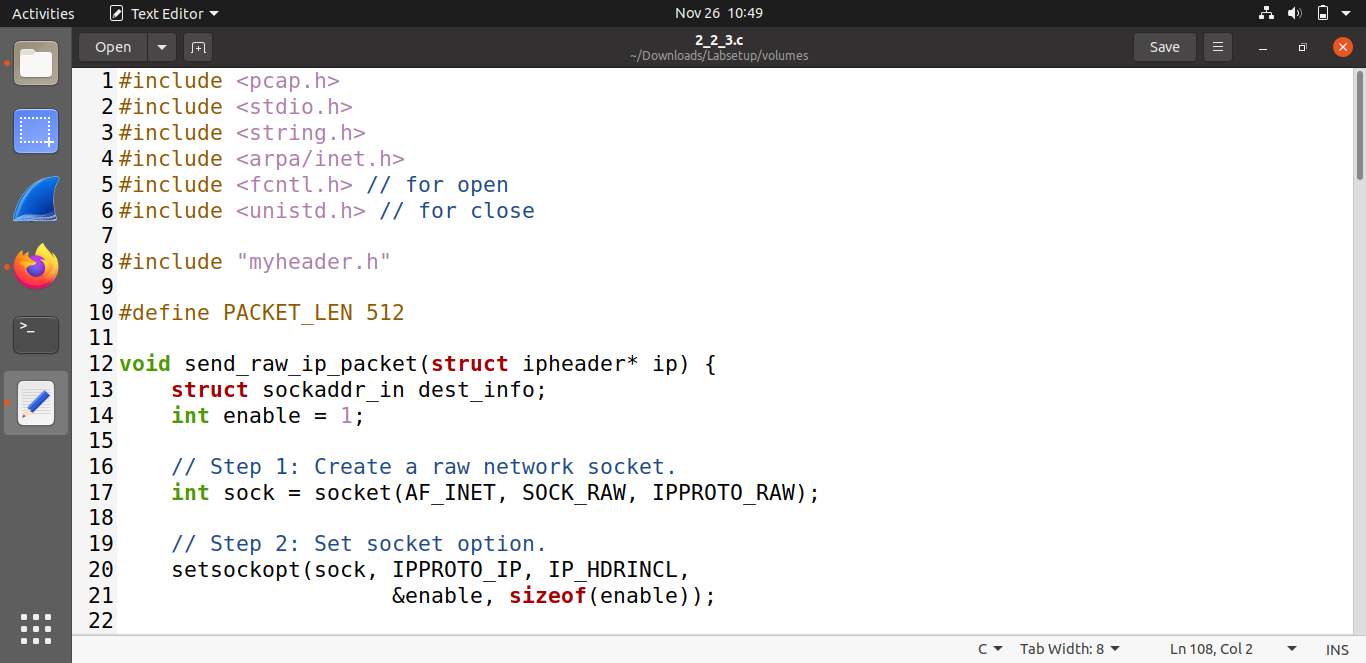
**Solution Q5:** When using the raw sockets, you can tell the kernel to calculate the checksum for the IP header. In IP header fields it’s actually the default option, ip\_check = 0 will let the kernel do it unless you change it to a different value but then you’ll have to use a checksum method.

**Solution Q6:** Root privileges are necessary to run programs that implement raw sockets. non-privileges users do not have the permissions to change all the fields in the protocol headers. Root privileges users can set any field in the packet headers and to access the sockets and put the interface card in promiscuous mode. If we run the program without the root privilege, it will fail at socket setup.

**Task#2.3:**

The attacker machine was in promiscuous mode and then when we executed our spoofing program, the NIC captured all the packets that reached and the program then processed in such a way, it modified the destination as source and source as destination. Once the packet is created it sends the packet out and the victim has received it. Thus, we spoofed the ICMP echo request.



****