**Lab 2**

**Task 1.1 - Sniffing Packets**

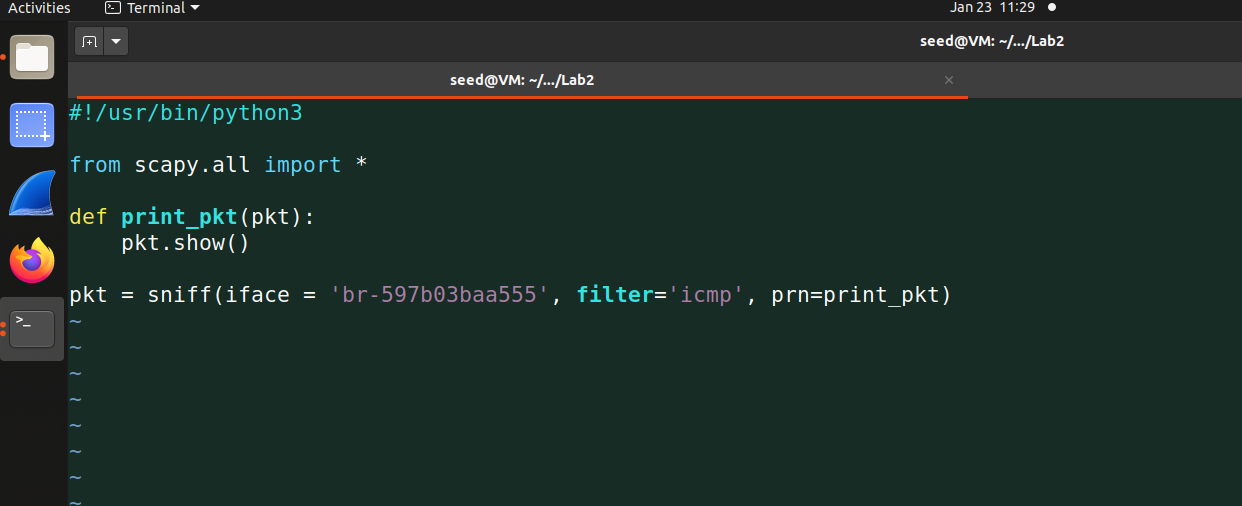
**Task 1.1A**

Before sniffing packets, we need to find the interface name.

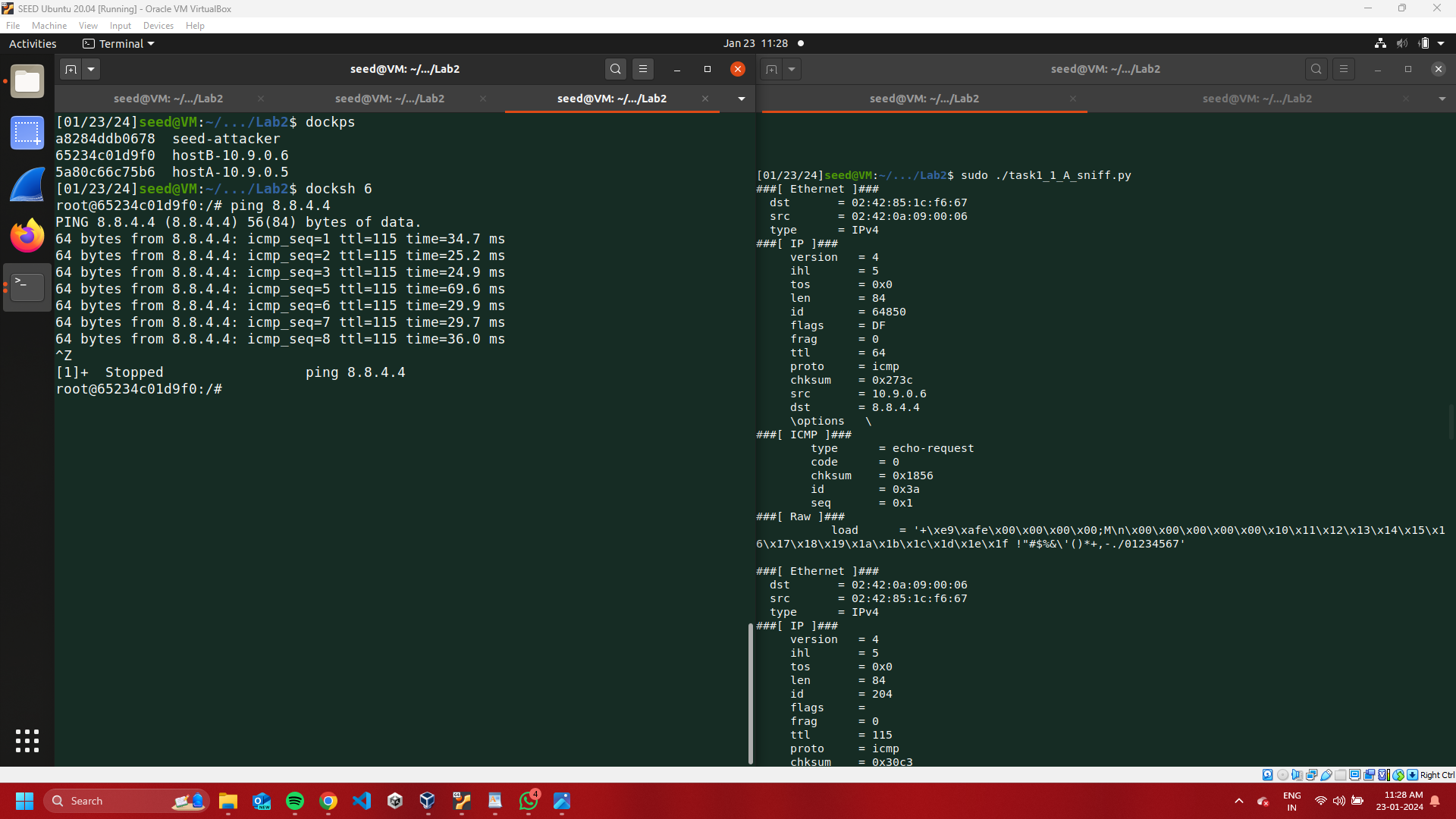
I started my docker containers and then using **ifconfig** command I found my interface name.



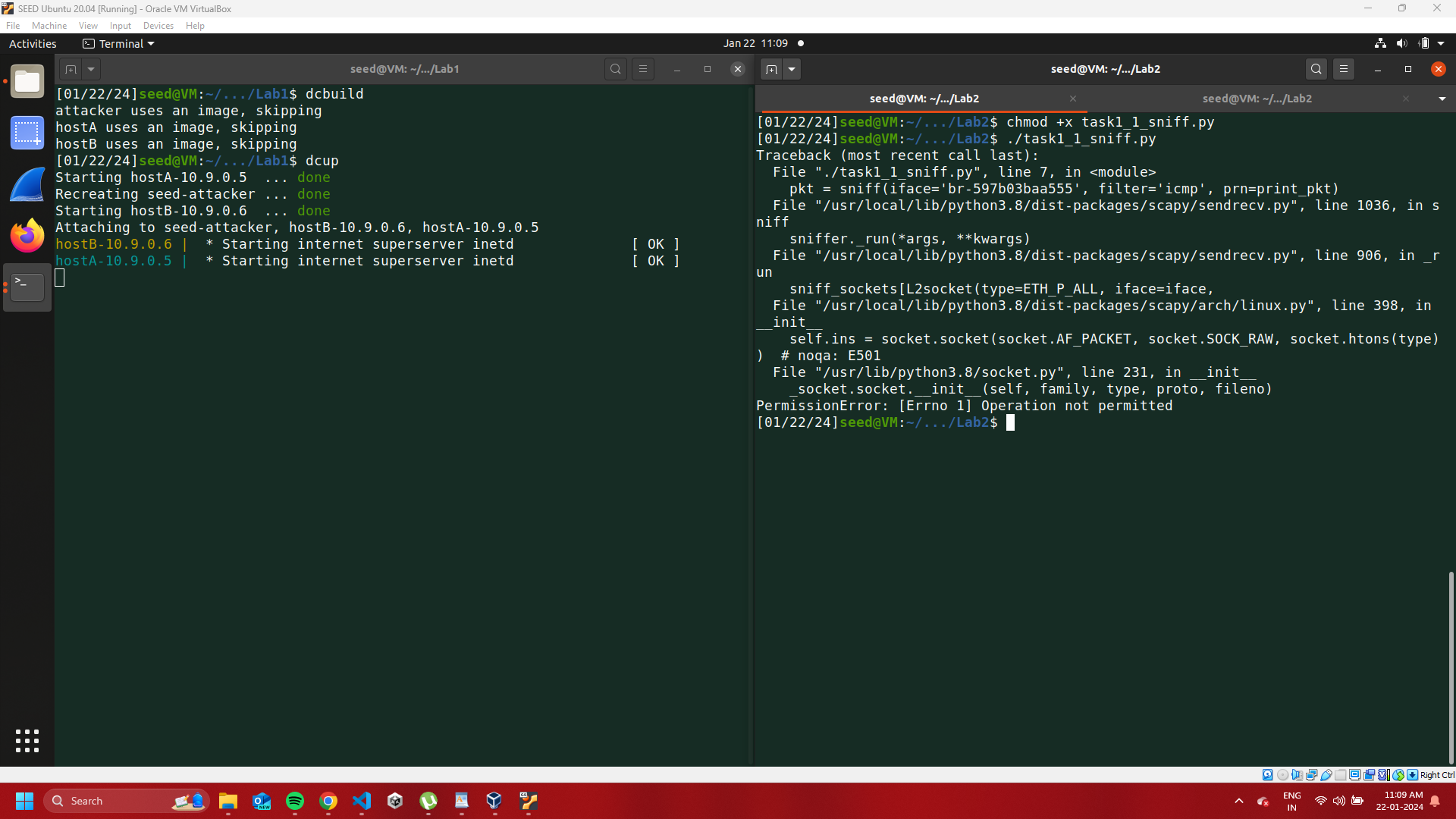
**Code**



First I executed the code with root privileges and my code ran successfully by giving the below output. I used the ping command to test the code. The ping command sends a request to a given IP and then the response from that IP will be recorded on the interface I provided and I will get the packet details on my screen.



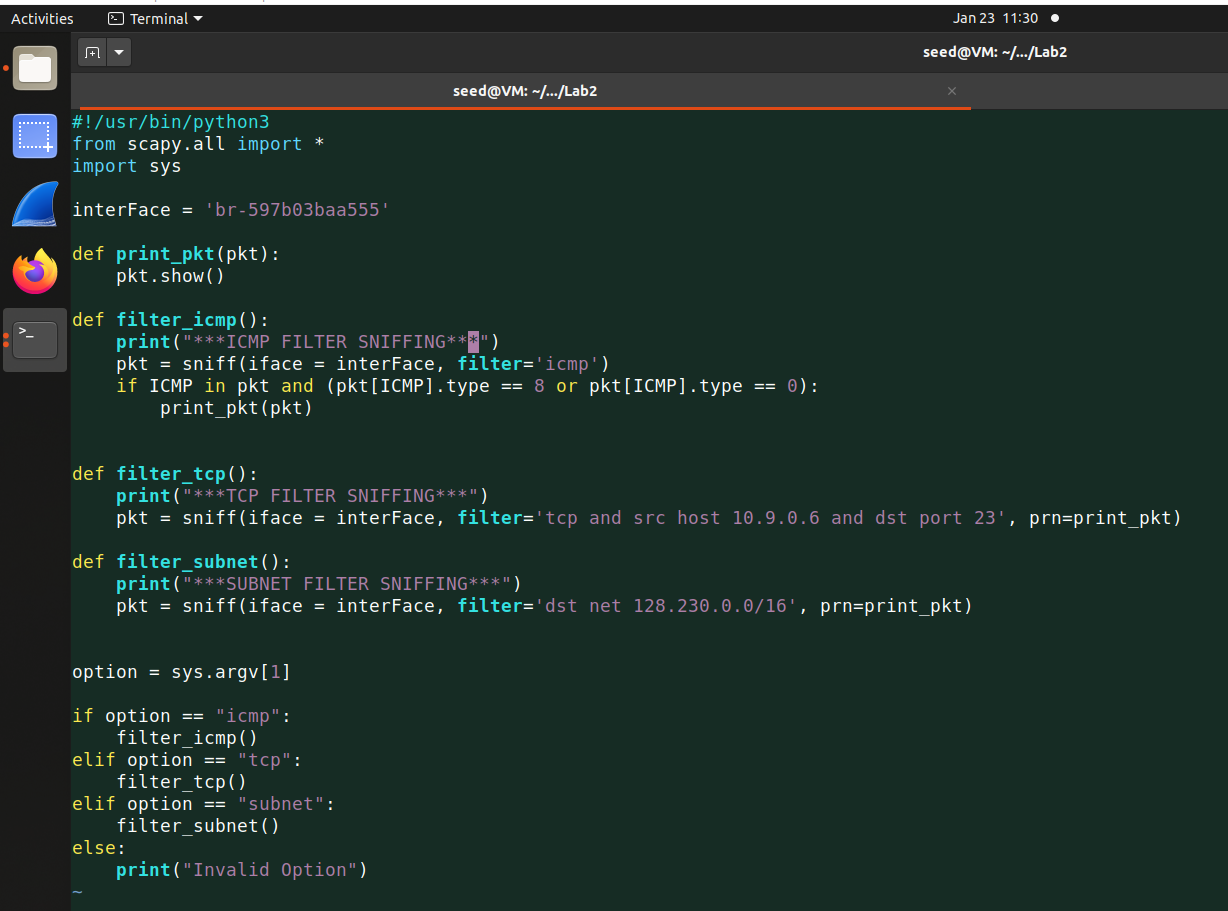
Second, This time I tried to execute code without using root privileges, and as expected that didn’t work because we need root privileges to run the code as all operations are on the operating system level.



**Task 1.1B**

Below is my code for the next task, I created three different functions for each sub-task like ICMP, TCP, and subnet sniffing. While running the code pass the option as a command line argument to run that code.

**Code**

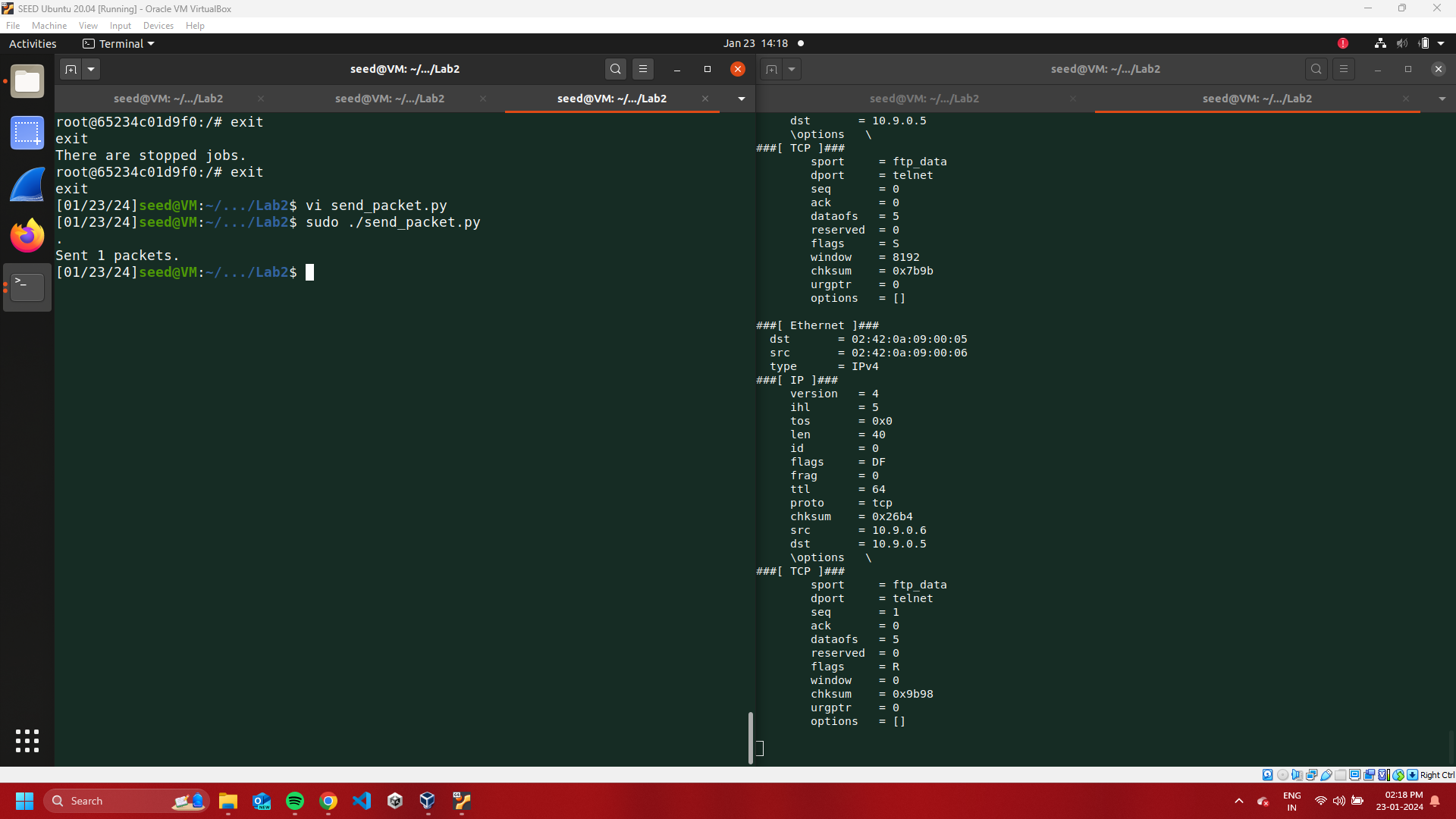


**Capture only the ICMP packet** - capture the ICMP packet is the same as a previous task, I login into one of the containers and used the ping command to IP 8.8.4.4 and my code captured the result in the right window.



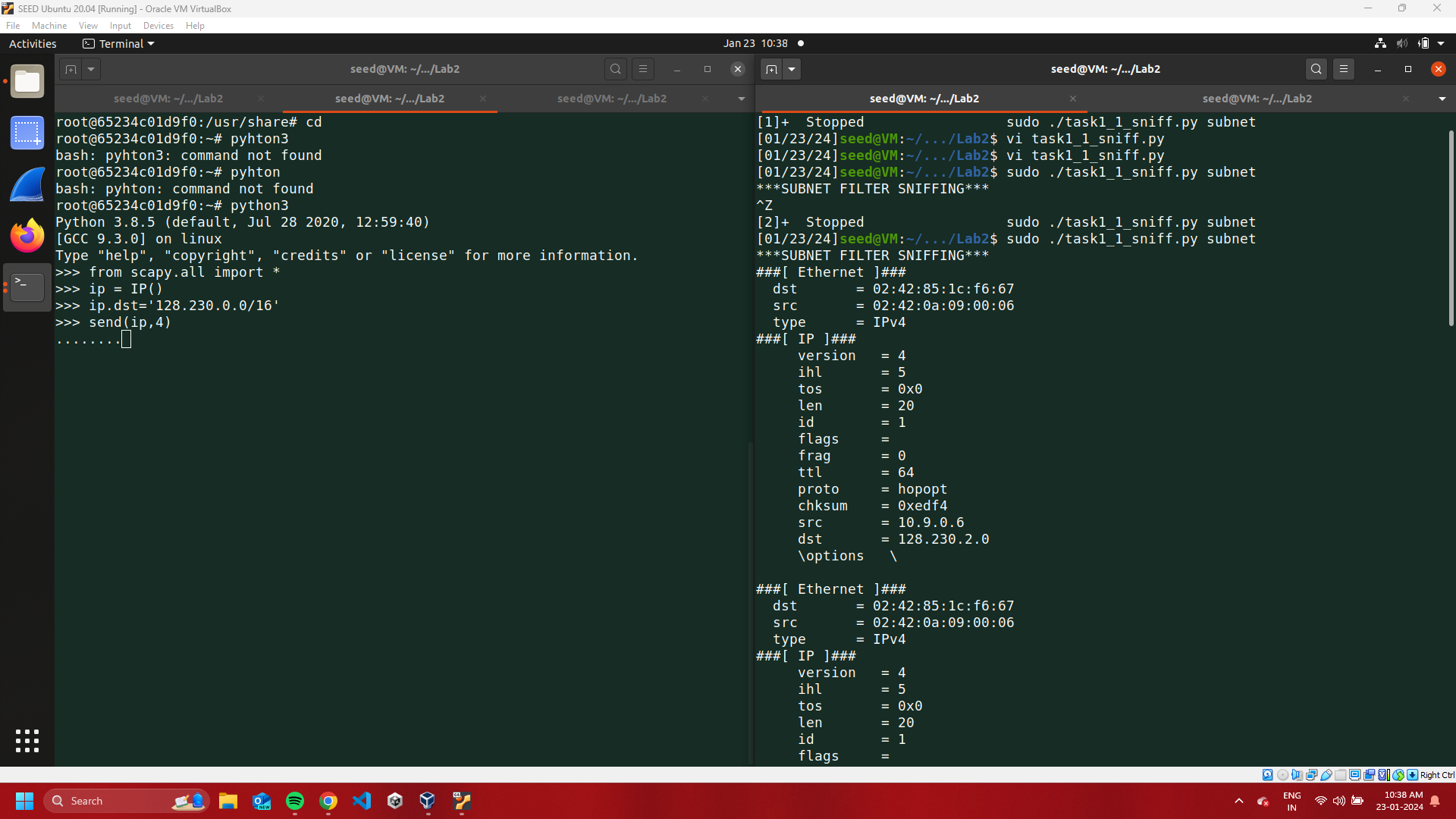
**Capture any TCP packet that comes from a particular IP and witha destination port number 23**

For this one as per BPF syntax, I used ***tcp and src host 10.9.0.6 and port 23*** as filters for sniffing. I wrote another code to send a packet to the above IP and port, executed that code, and received the result.



**Capture packets come from or go to a particular subnet**

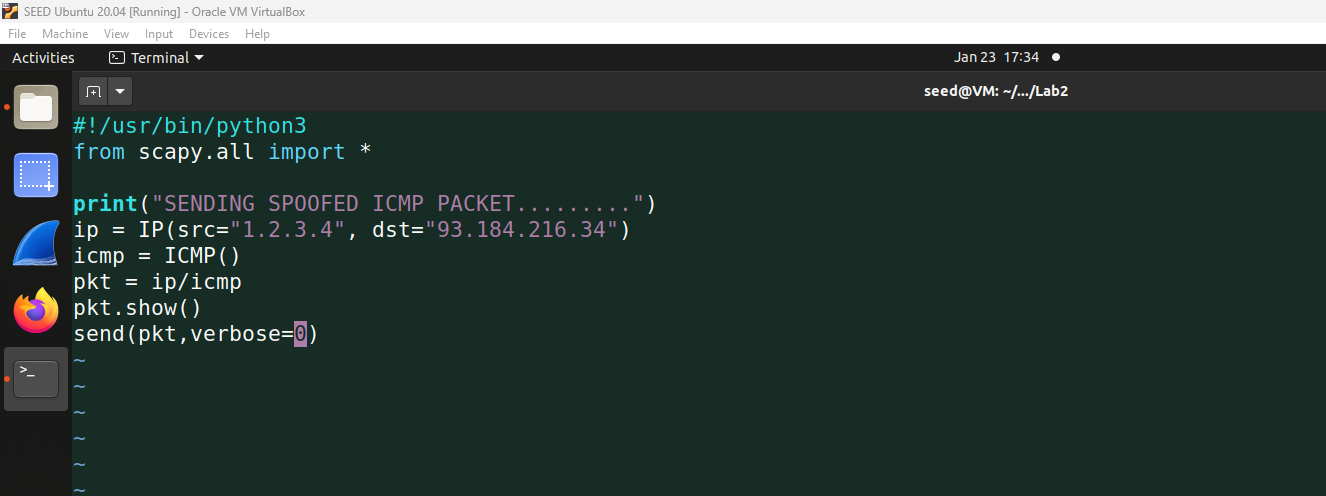
Used BPF filter - ***dst net 128.230.0.0/16*** and sent packet by login to one of the containers.

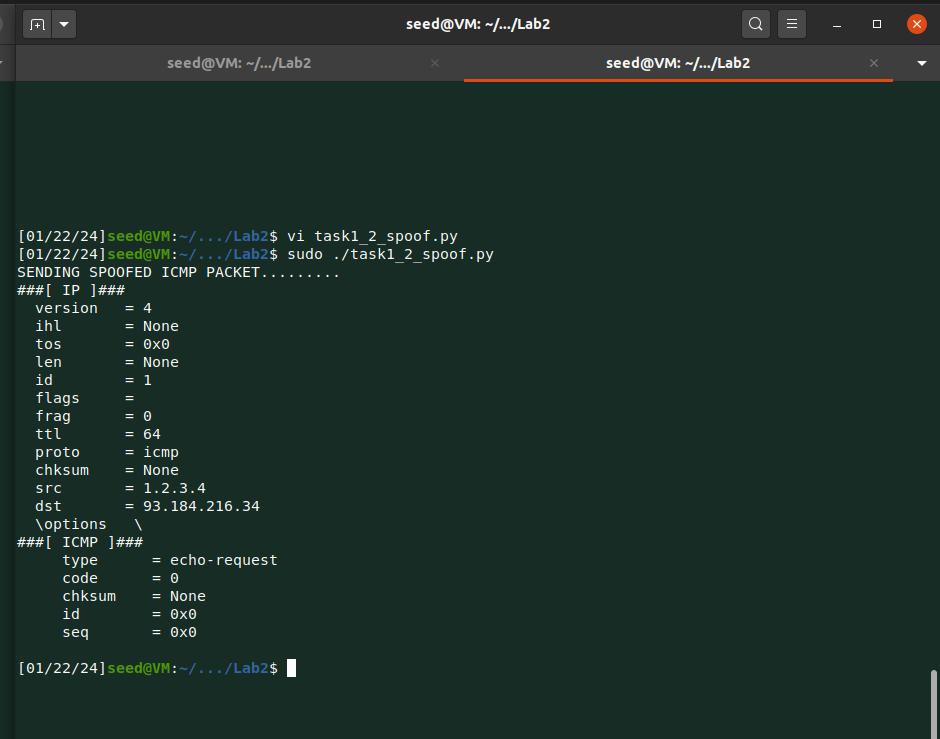


**Task 1.2**

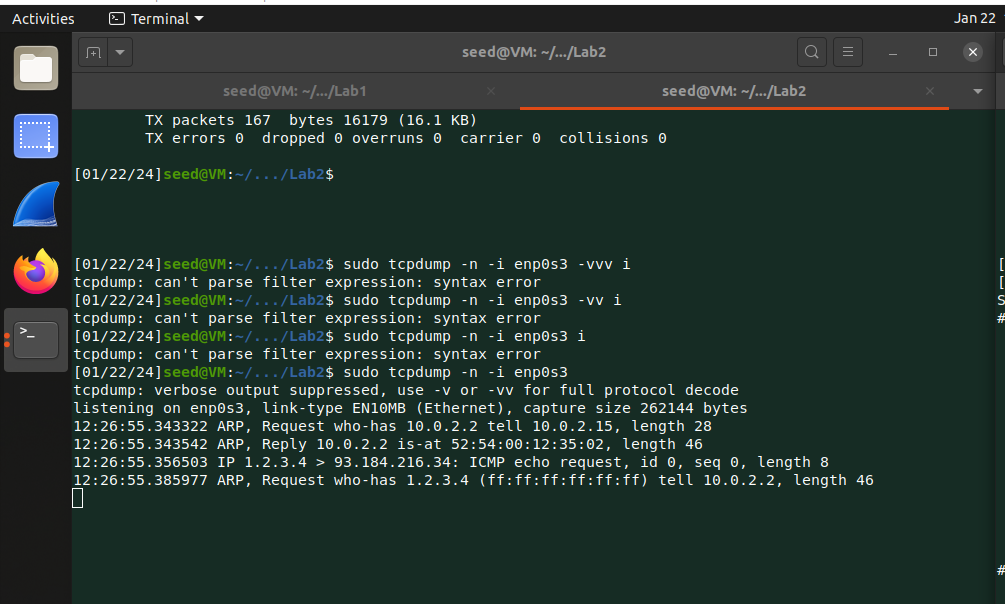
To spoof an ICMP packet, we have to send a packet to a valid IP from the host IP meaning that the destination will not know whether the request is real or not !! I used **tcpdump** to check my request response.

**Code**





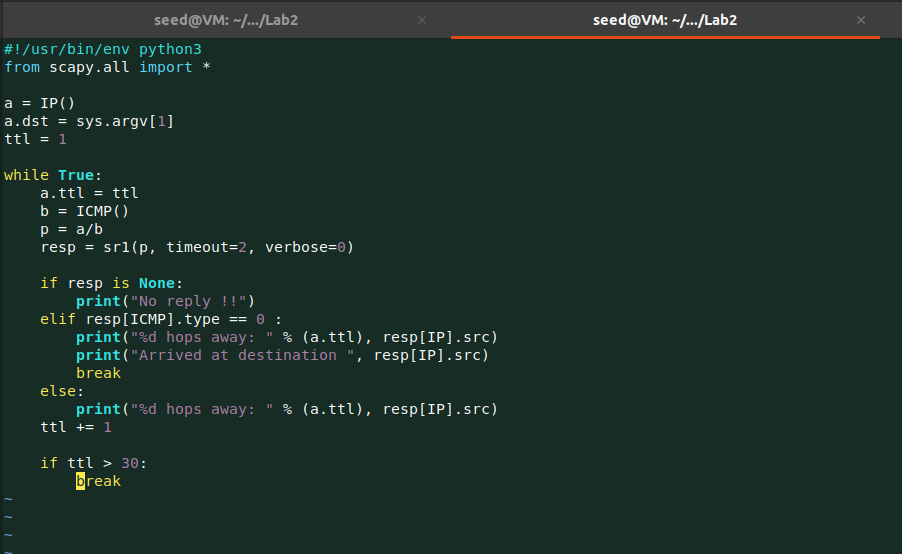
You can see the last line below, which ARP requests from the destination IP to confirm who has the 1.2.3.4 IP address.

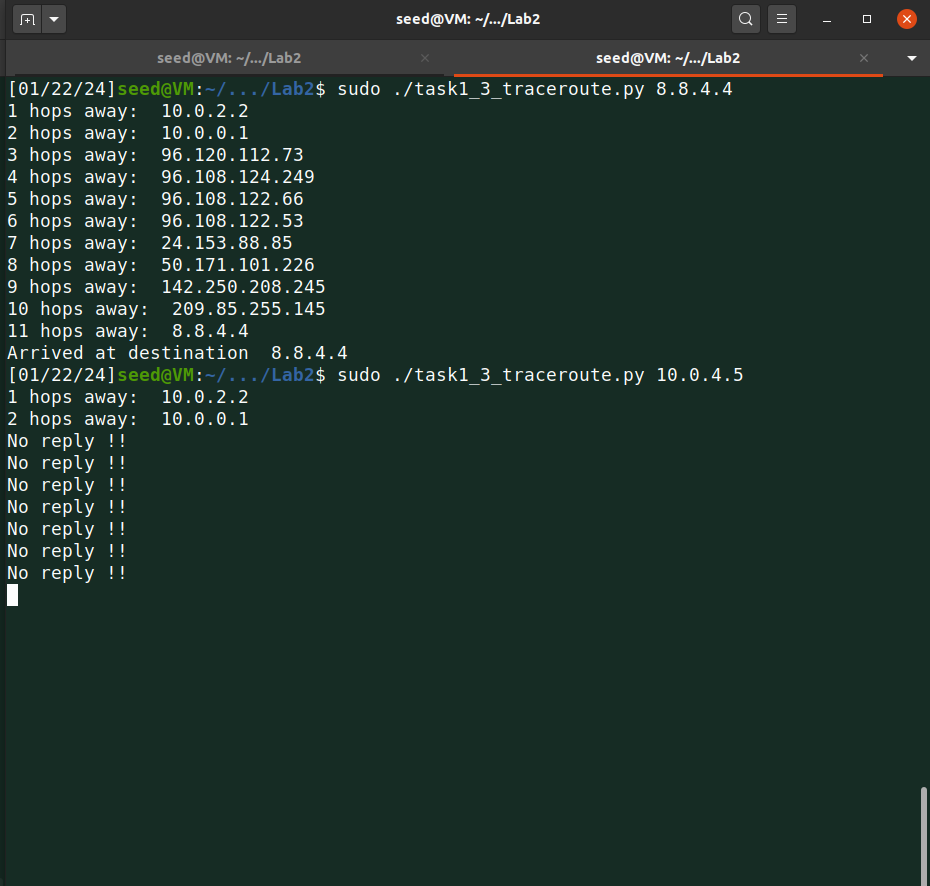


**Task 1.3**

I referred to the code shown in the lecture, to find the distance between my machine and to destination IP address machine. **TTL** is the time-to-live field if it is 1 at the start and if I didn’t use the below while loop code then after one router/hope this field value has become 0. We need TTL until we reach the destination so using loop increases the TTL by 1 send request again.

**Code**





**Task 1.4**

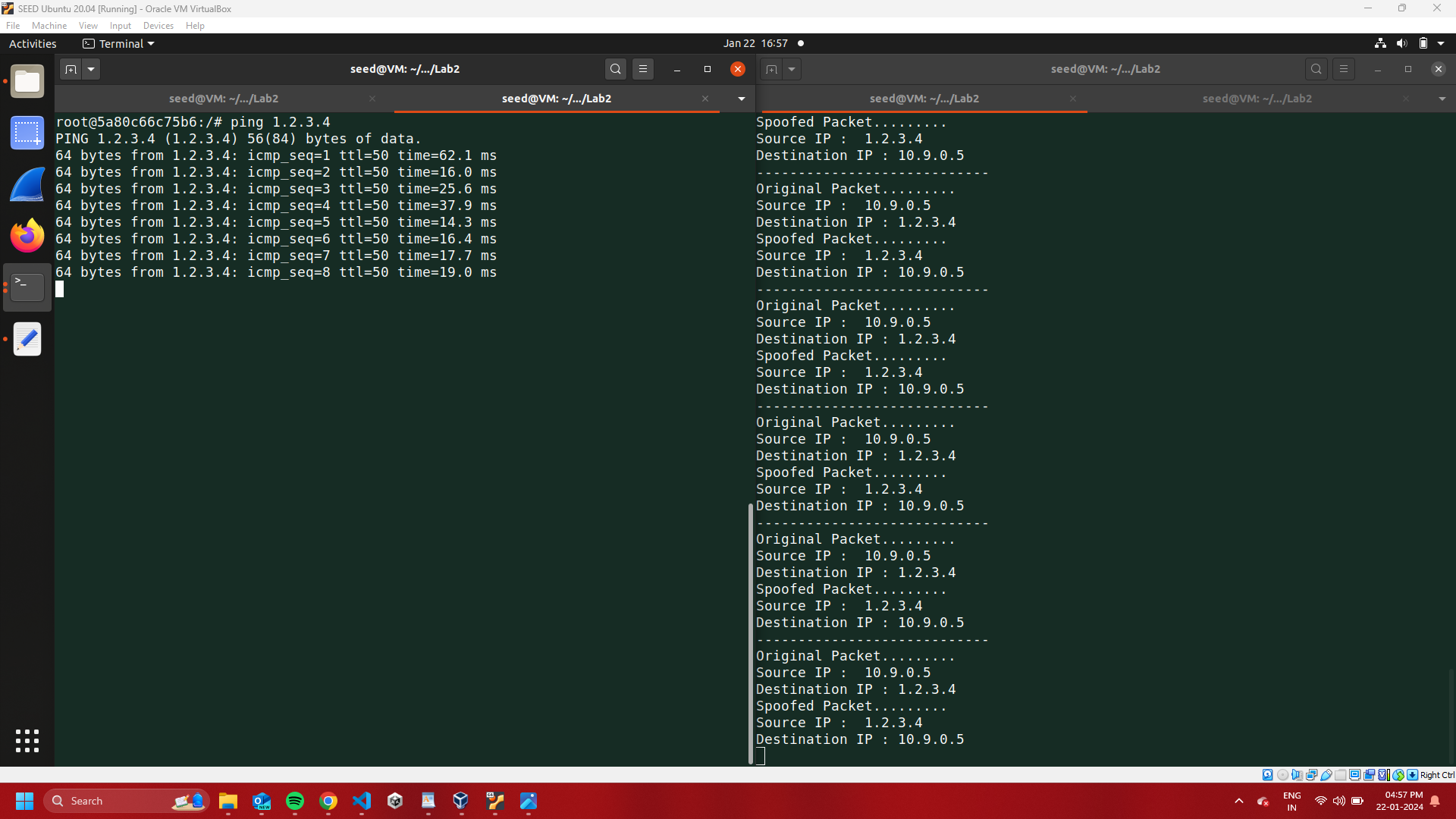
In this task, we are sniffing into a packet, and then using that source and destination address we are sending a fake reply back to the source.

We used three different IP addresses to test.

* 1.2.3.4 - non-existing host on the Internet

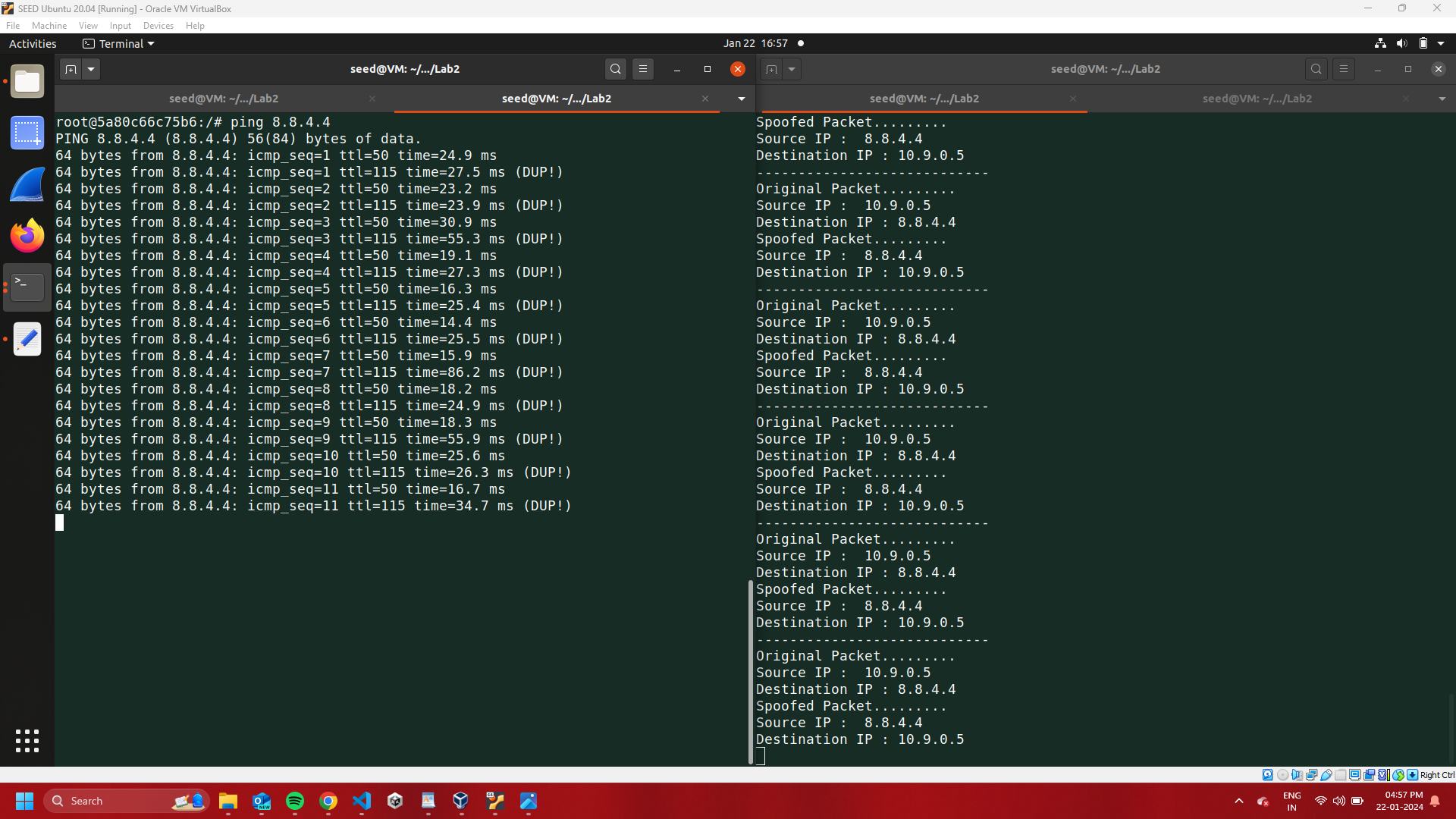
For 1.2.3.4 as we know it is non-existent but still we are getting replies from that IP, the reason is there is a mapping for this address in the routing table. See the last screenshot I used ***ip route get 1.2.3.4*** and found that there is a route via 10.9.0.1.

In the right window, you can see I changed source to destination and destination to source to send a fake reply.



* 8.8.4.4 - existing host on the internet

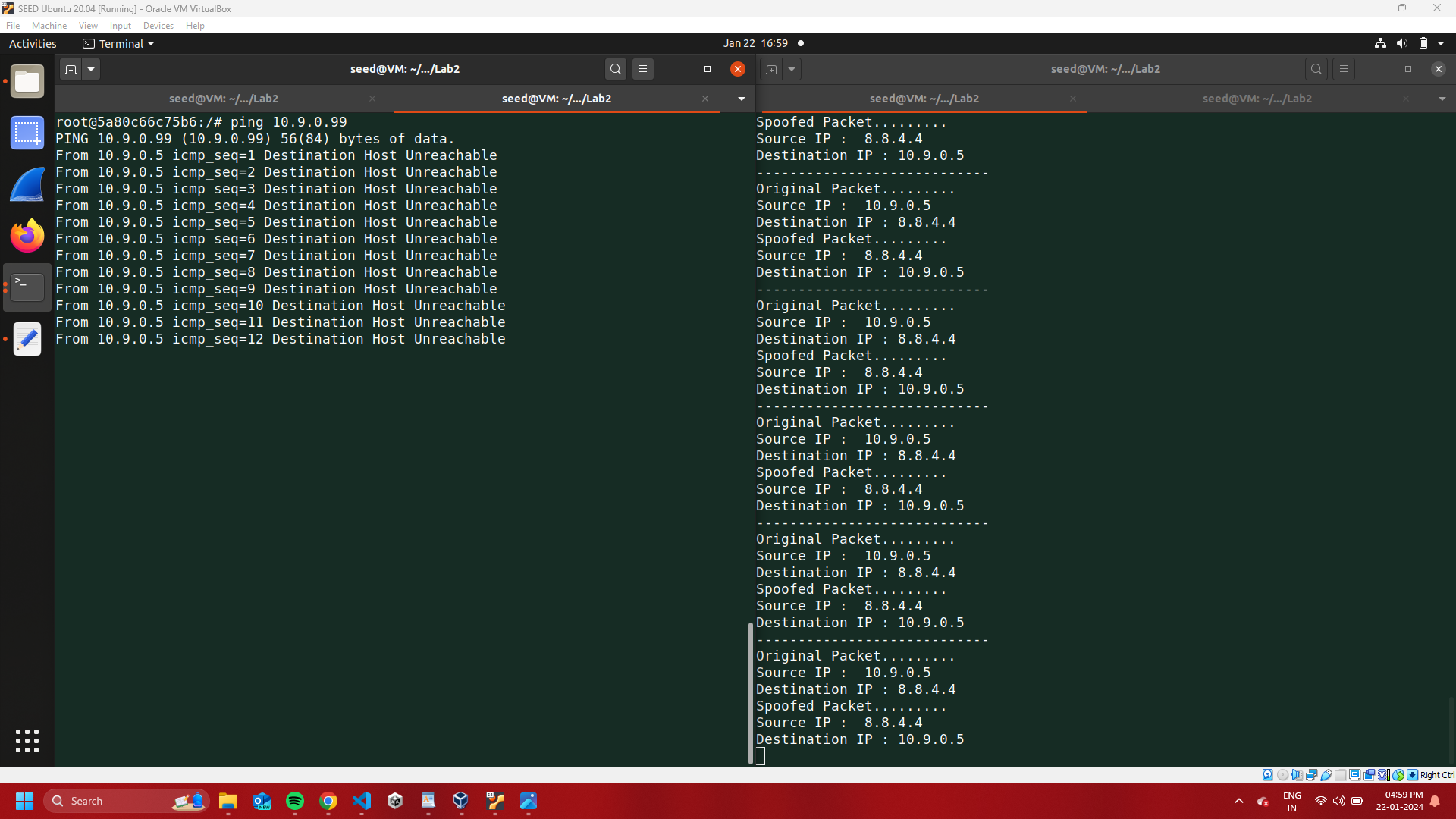
For this, as you see we are getting ***DUP!*** For some of the replies, this is because of our container. The real machine is also sending us the reply and my code which is running at the same time also sending the spoof reply.



* 10.9.0.99 - non-existing host on the LAN

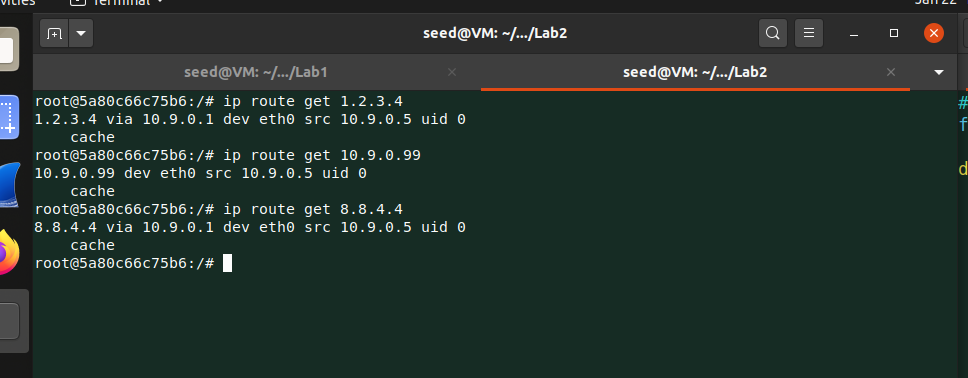
This IP doesn’t exist and when I send a spoof reply to this IP, I get an error that Destination Host Unreachable.

As for successful connection machine always sends an ARP request to the destination to confirm his MAC address but in this case, there is no valid IP and MAC address. So no one is sending ARP reply back to the source and that’s why we are getting a host unreachable error.



Used IP route get for each IP to find there is a route exists or not.

Except last one other two have routes that’s why the code worked for those.



**Code**

