Lab2 Packet Sniffing and spoofing

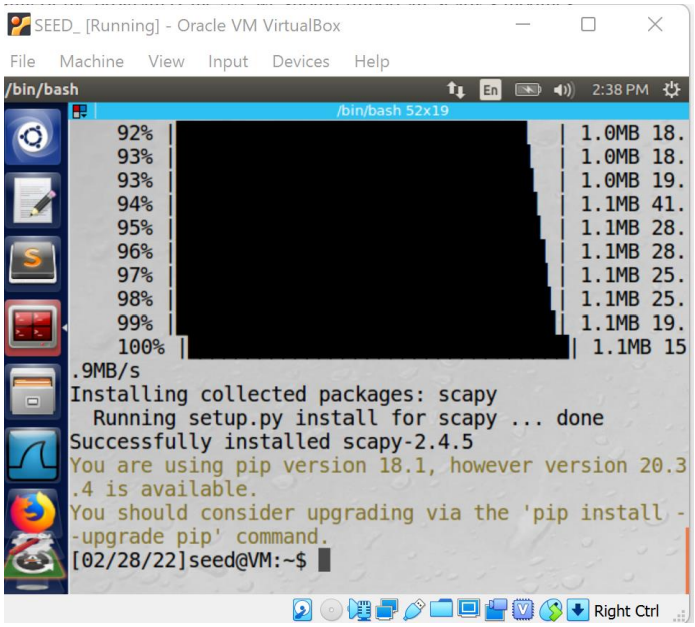
I. Lab set up

a. Set up two virtual machines (SEEDLab image) on VirtualBox

i. Attacker IP: 10.0.2.15

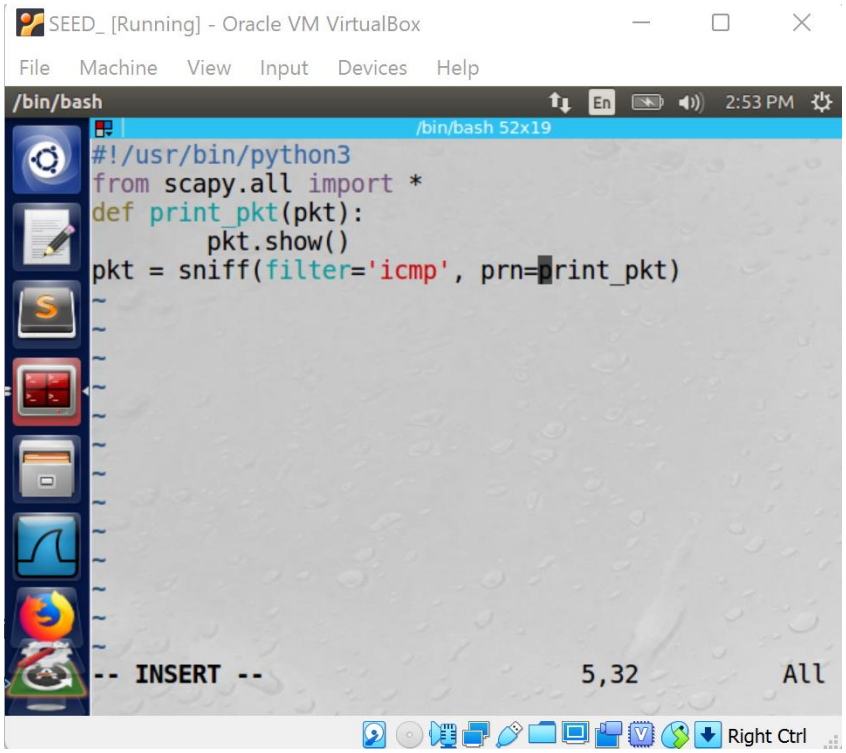
ii. Victim IP: 10.0.2.4

b. Install scapy by ‘sudo pip3 install scapy’

c. 

II. Task 1.1A

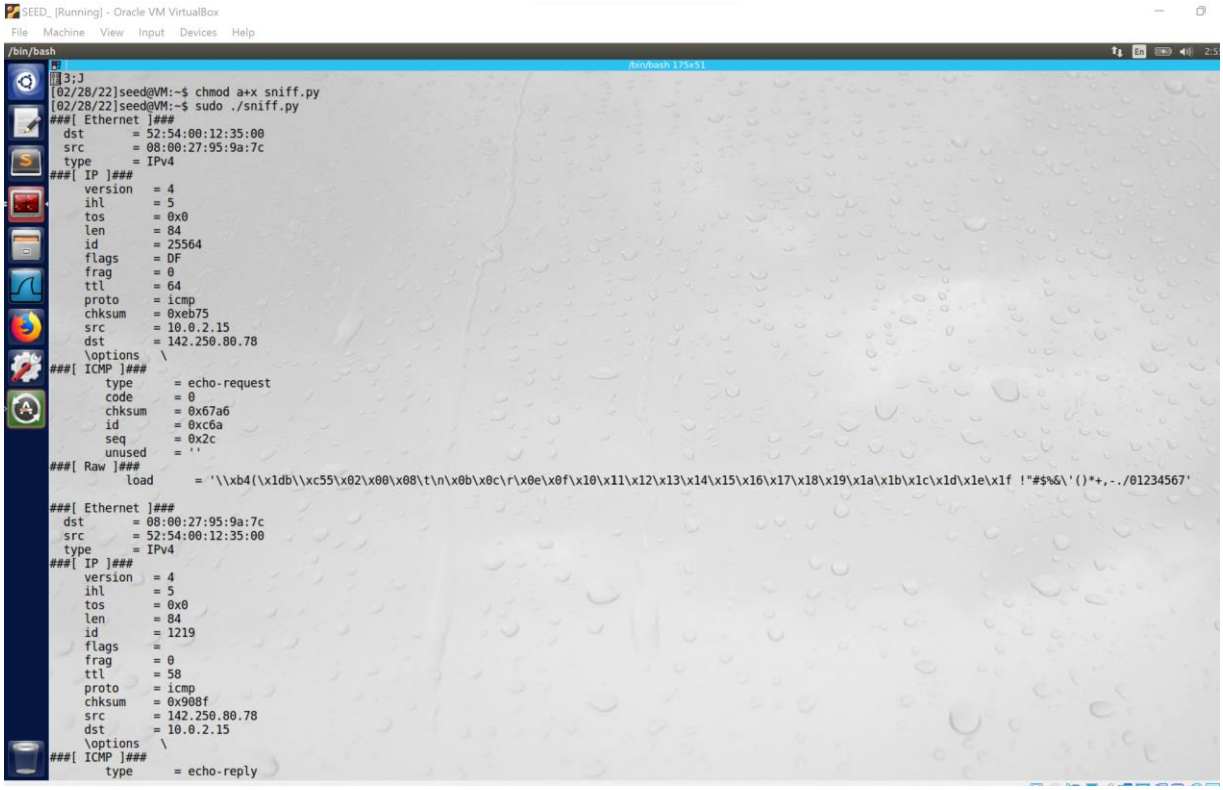
a. Use the skeleton code provided to create a sniff.py

b. 

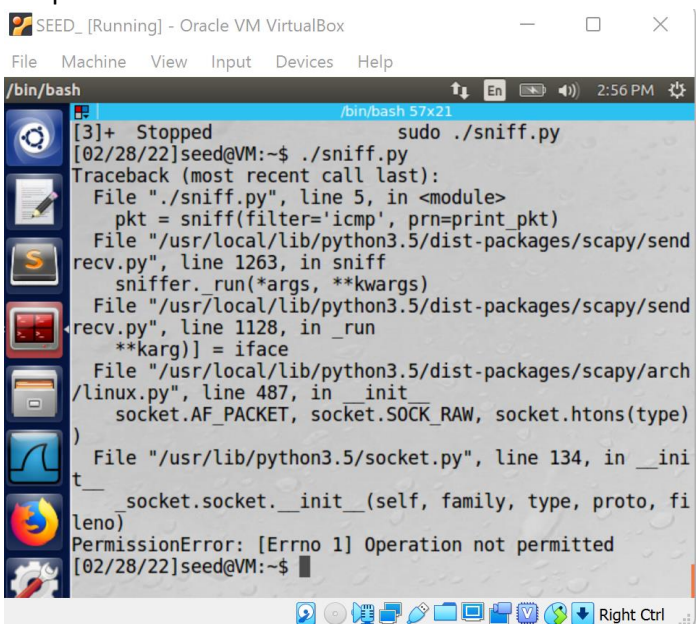
c. To create some traffic on the machine, ping a random website, here I pinged youtube.com

d. 

e. Make sniff.py executable by running ‘chmod a+x sniff.py’ and run the file in root by ‘sudo ./sniff.py’. We can see echo request and echo reply ICMP packets generated when pinging the website.

f. 

g. Now run sniff.py without root by './sniff.py', we see a permission error saying operation not permitted.

h. 

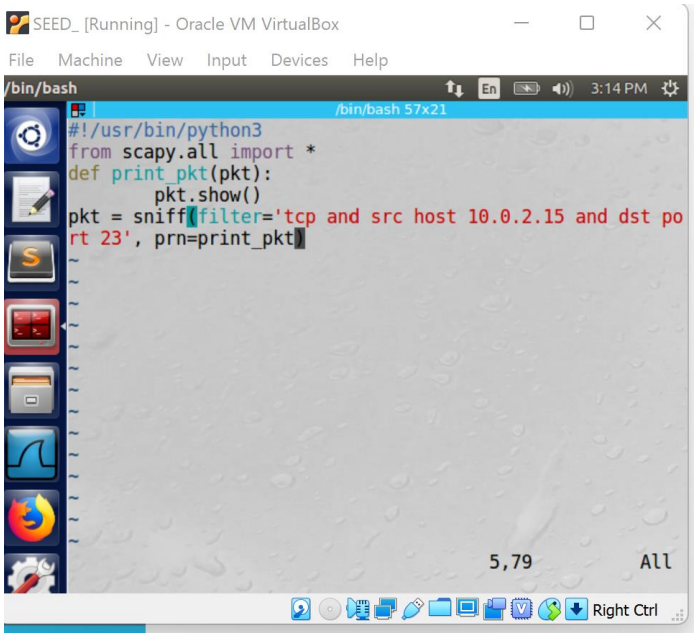
III. Observations of Task 1.1A

a. Line 5 in sniff.py, the function sniff() from scapy needs root permission to operate. A get-around is to pip install scapy-unroot packet. Otherwise, root privileges are needed for scapy.

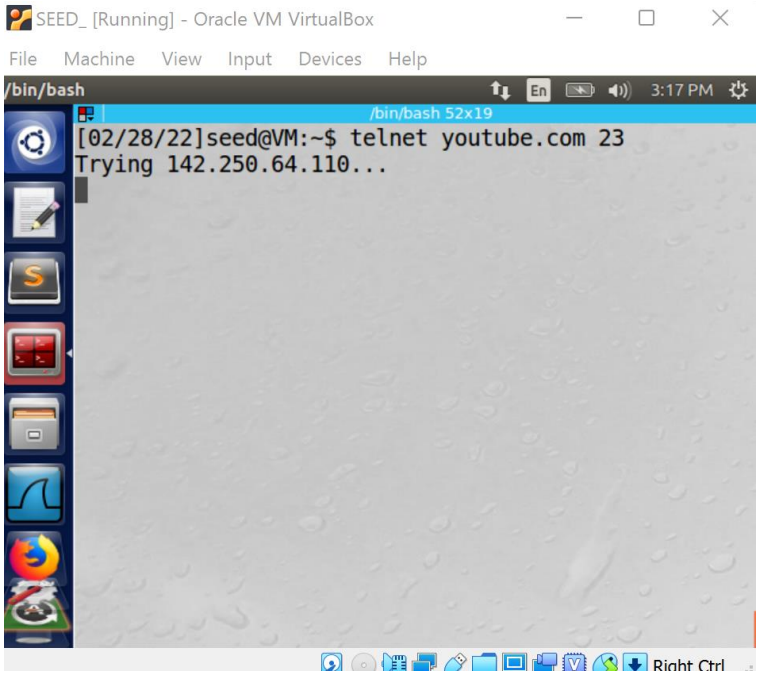
IV. Task 1.1B

a. Capture only ICMP packets (this was set in the skeleton code, see II.(e) for output).

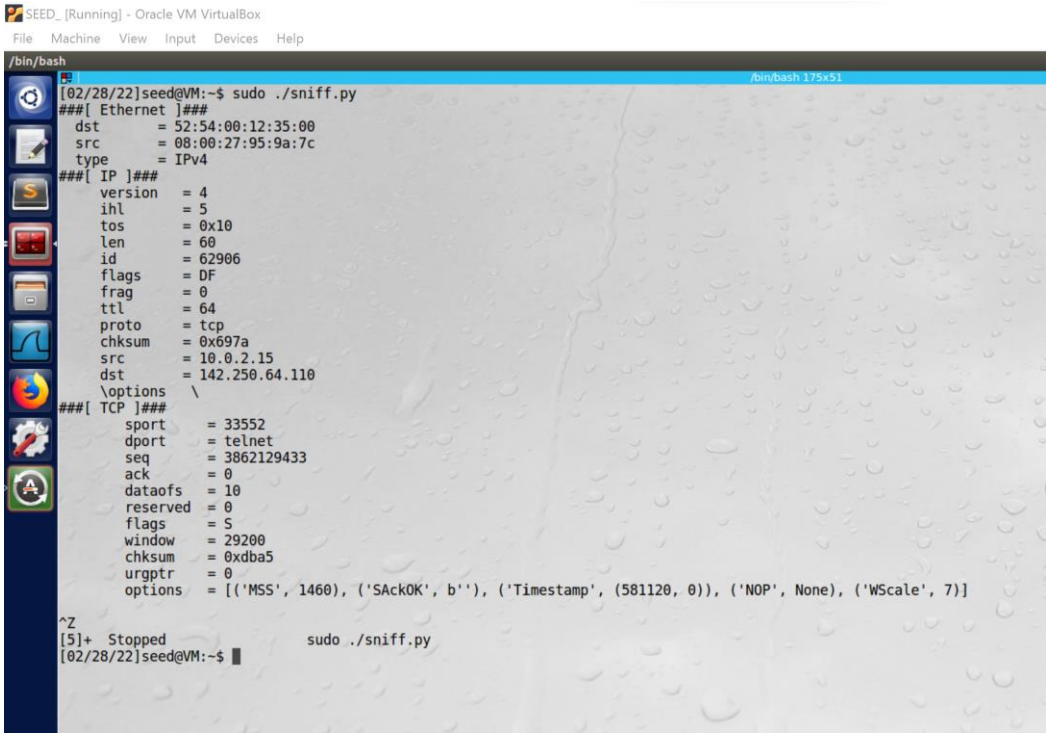
b. To capture any TCP packet that comes from a particular IP and with a destination port number 23, we need to tweak the sniff.py. Source host IP is the IP of the current machine, which is 10.0.2.15.

c. 

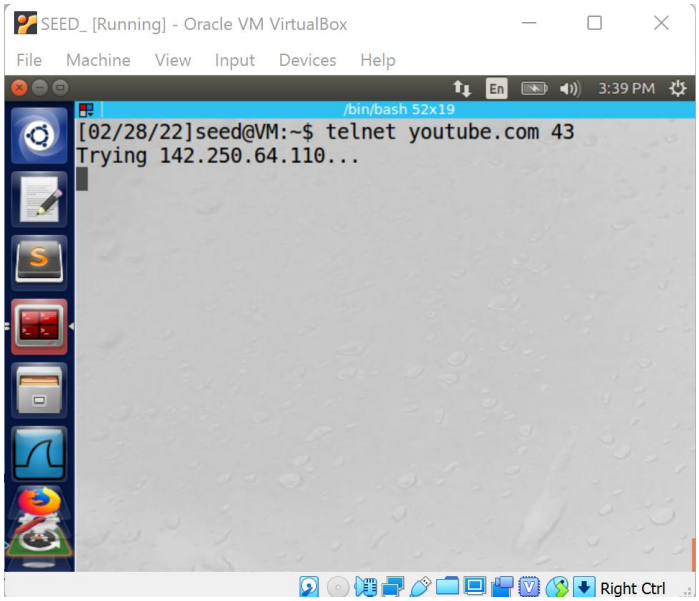
d. To generate some TCP packets, I choose to telnet youtube.com at port 23.

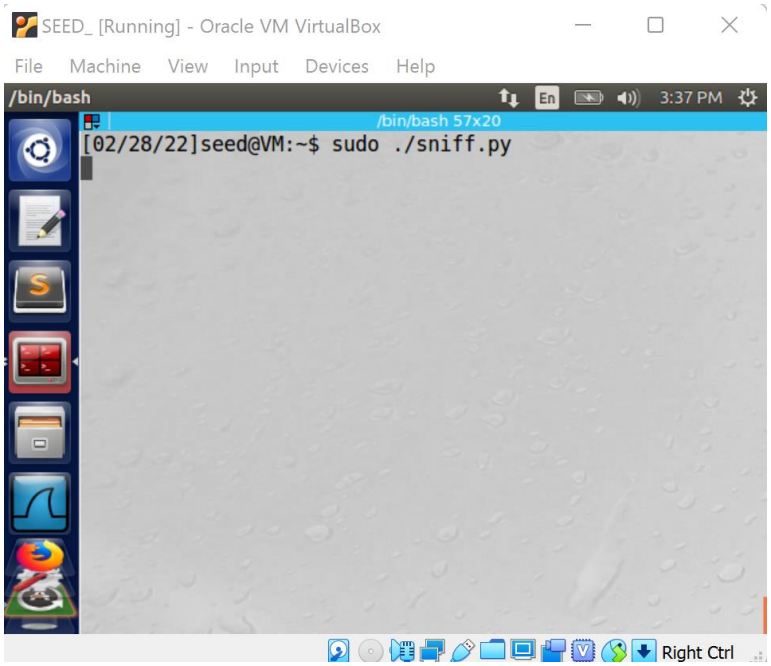
e. 

f. Then run sniff.py with root, the information of the captured TCP packet is printed out. Noted here only packets from 10.0.2.15 to port 23 was captured.

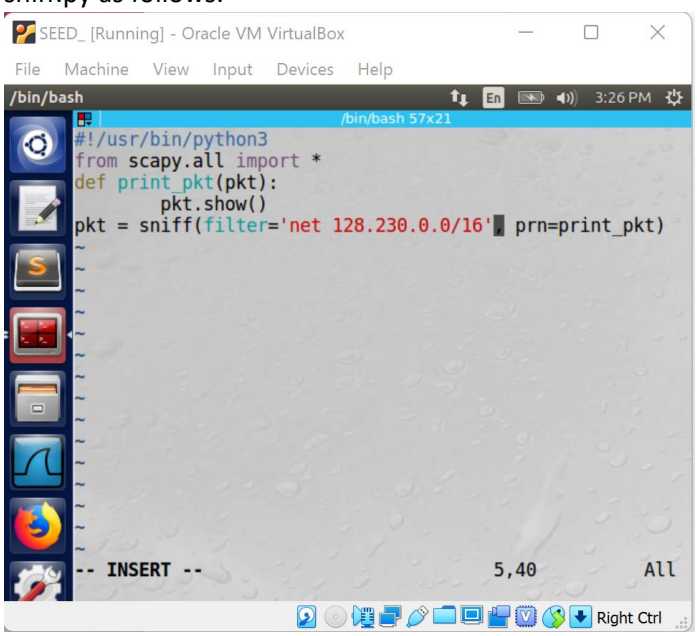
g. 

h. If create a telnet connection by telnet youtube.com <other ports #>, no packets would be captured.

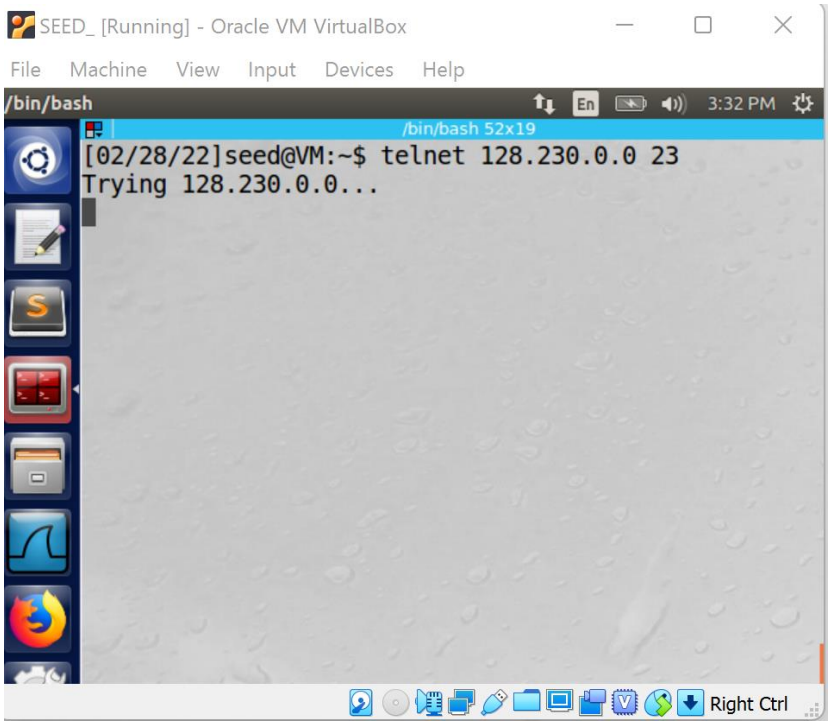
i. 

j. 

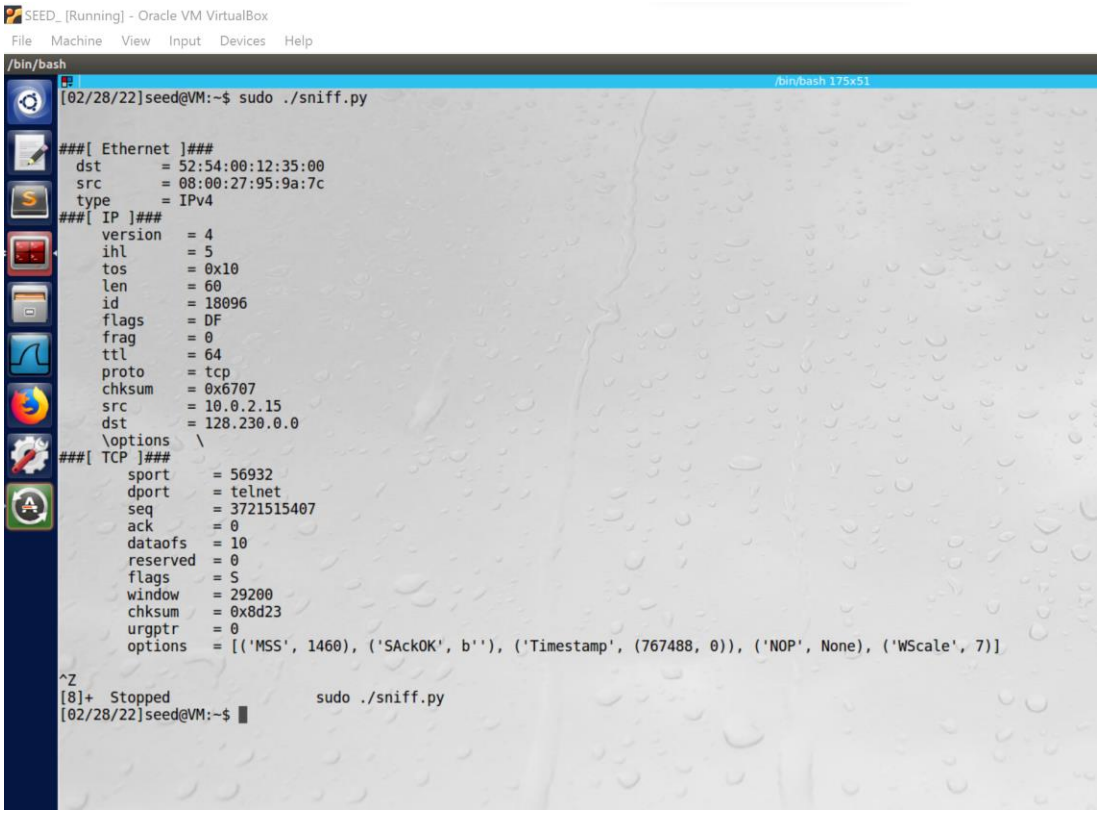
k. To capture packets comes from or to go to a particular subnet (128.230.0.0/16), tweak sniff.py as follows.

l. 

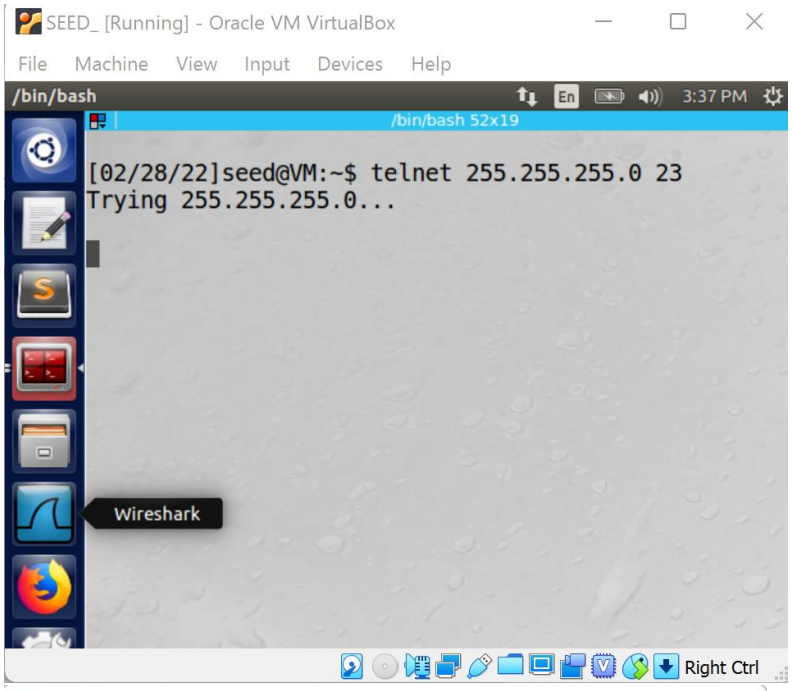
m. On terminal, we telnet this subnet by ‘telnet 128.230.0.0/16 23’

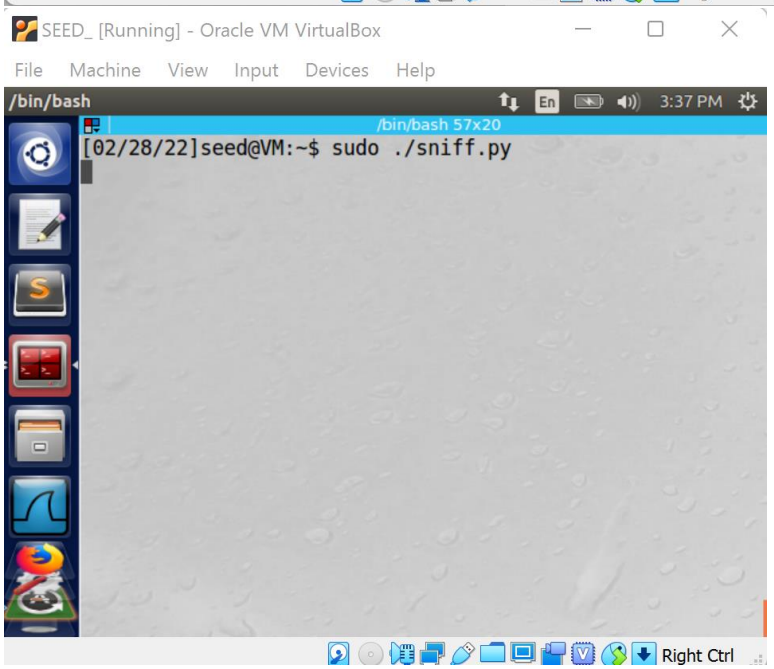
n. 

o. Run sniff.py with root, the information of the packets goes or comes from the subnet specified is printed out.

p. 

q. If we telnet another subnet, no packets would be captured by this filter.

r. 

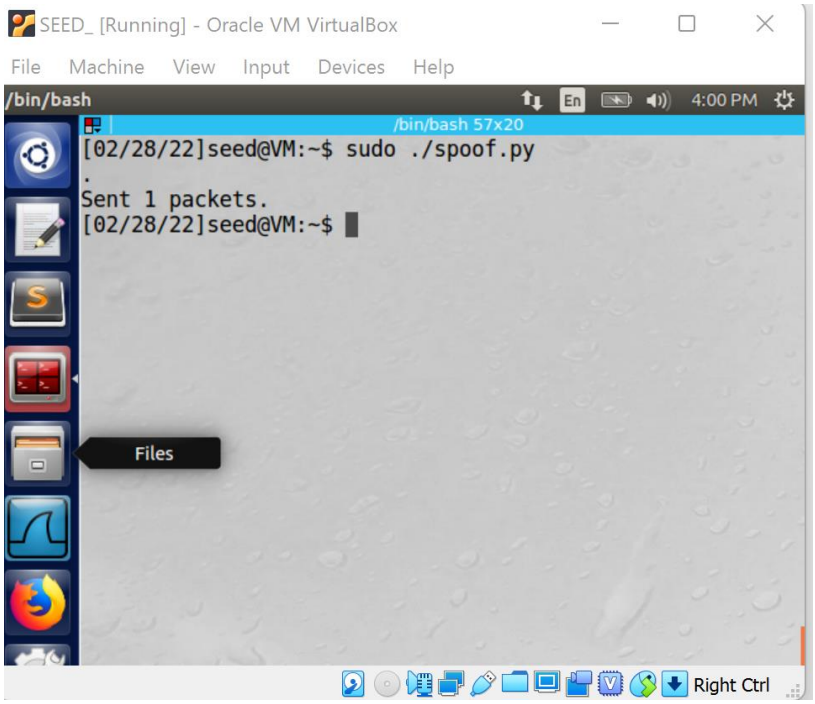
s. 

V. Task 1.2

a. Use the skeleton code to create spoof.py, in it, create a source IP that is arbitrary. Here I choose 10.0.0.0 as arbitrary IP. Insert a line ‘a.src=’10.0.0.0’’

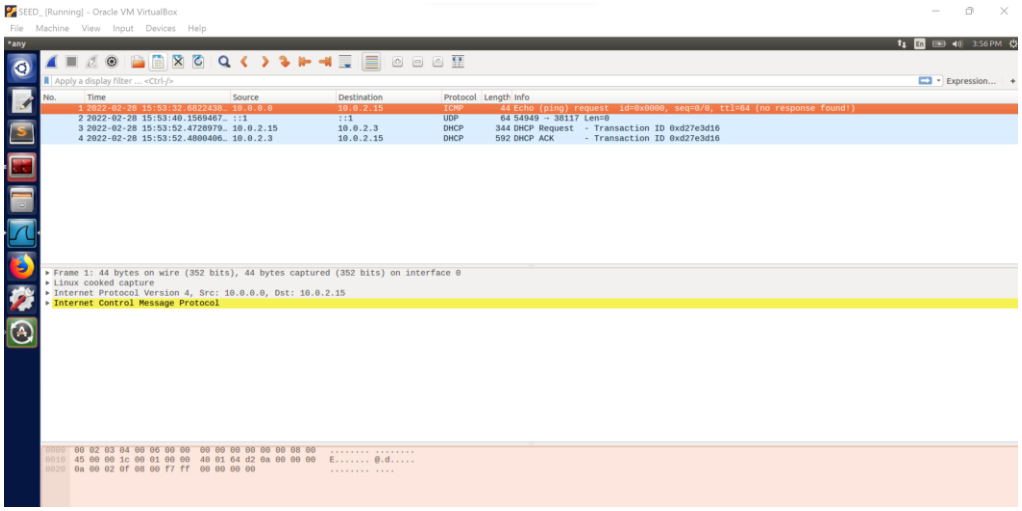
b. 

c. Make spoof.py executable by running 'chmod a+x spoof.py' and run it with root privileges. It outputs 'Sent 1 packet'.

d. 

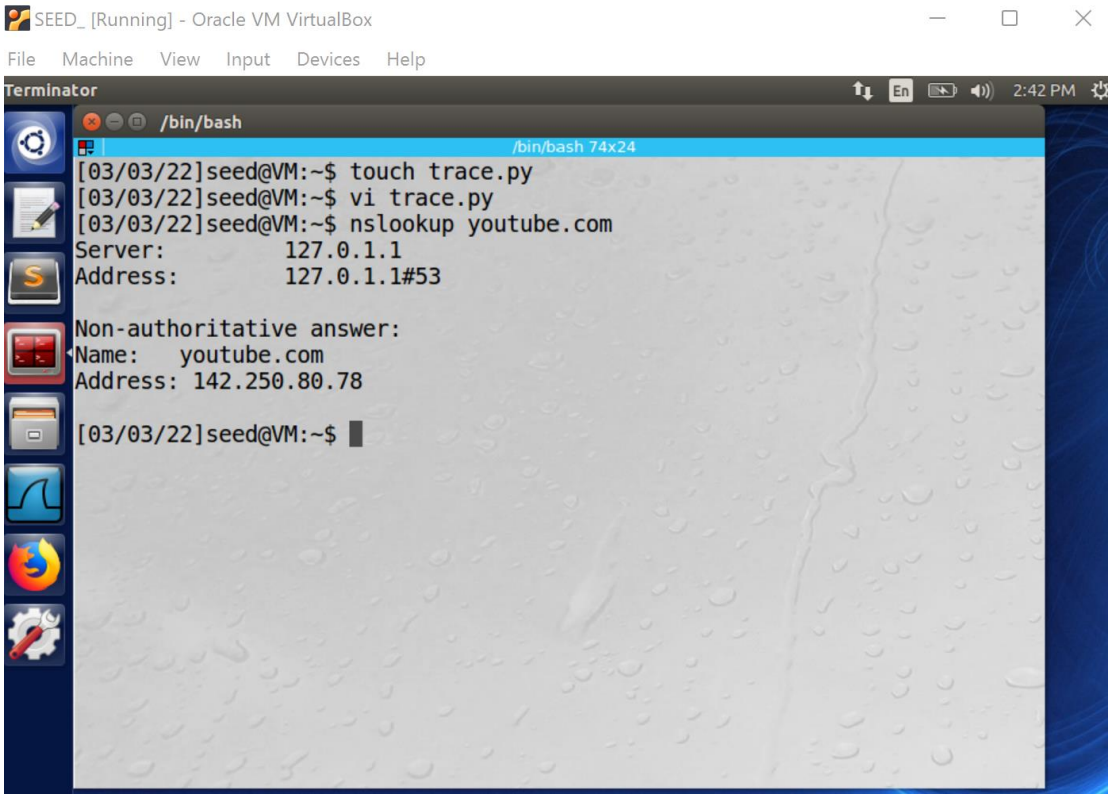
e. At the same time, open Wireshark to capture the packet, we can see an ICMP echo request packets were captured, with source IP of 10.0.0.0 (arbitrary IP specified in

script) and destination IP of 10.0.2.15. The spoof of an ICMP echo request packet with an arbitrary source IP address was successful.

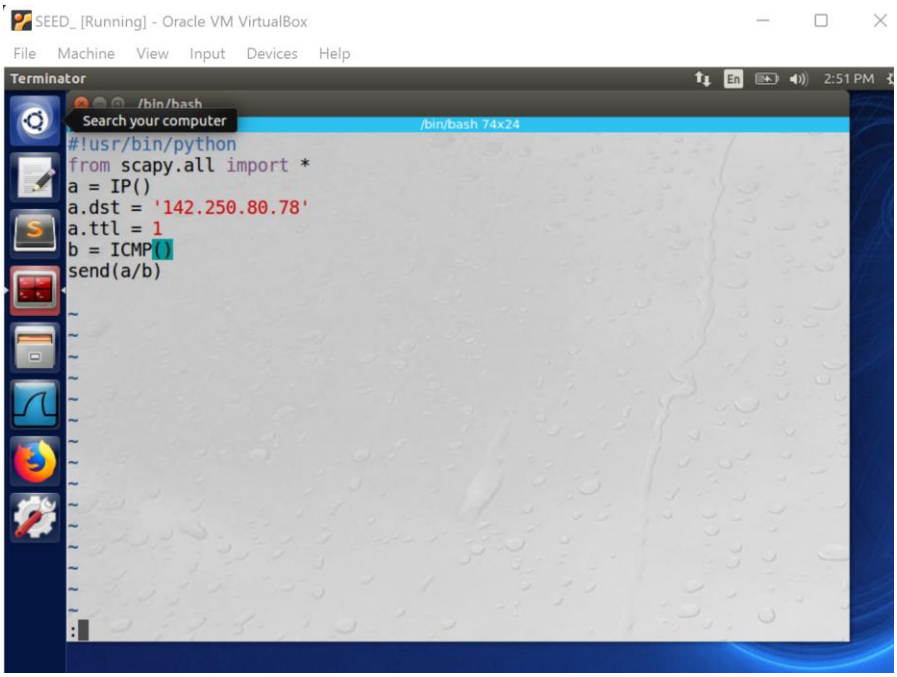
f. 

VI. Task 1.3

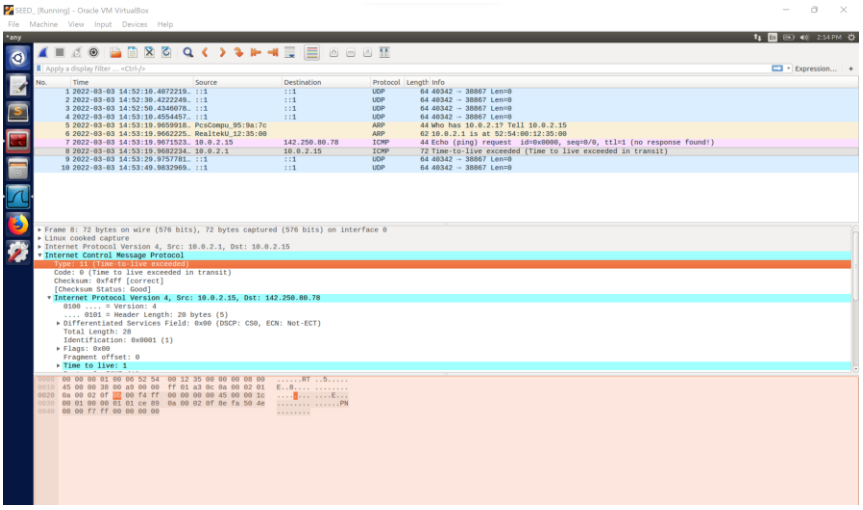
a. In this task, I will try to send a packet to youtube.com. Use nslookup to find the IP address of youtube.com.

b. 

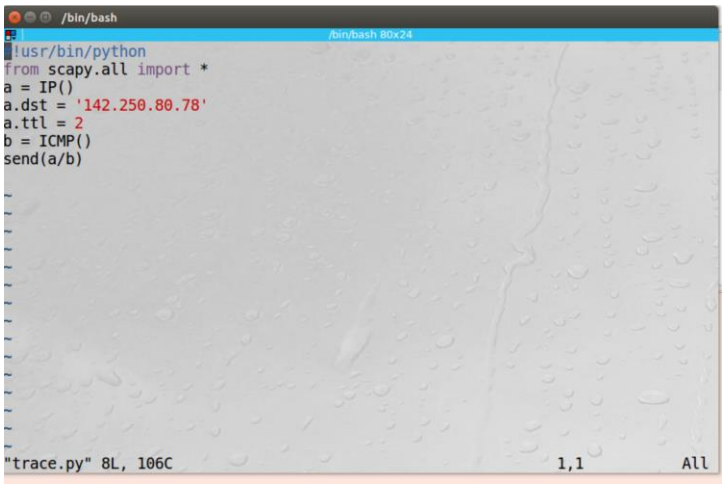
c. Create trace.py using the skeleton code provided. Set ttl field to 1 at the first run. Set destination IP address as youtube's IP Address.

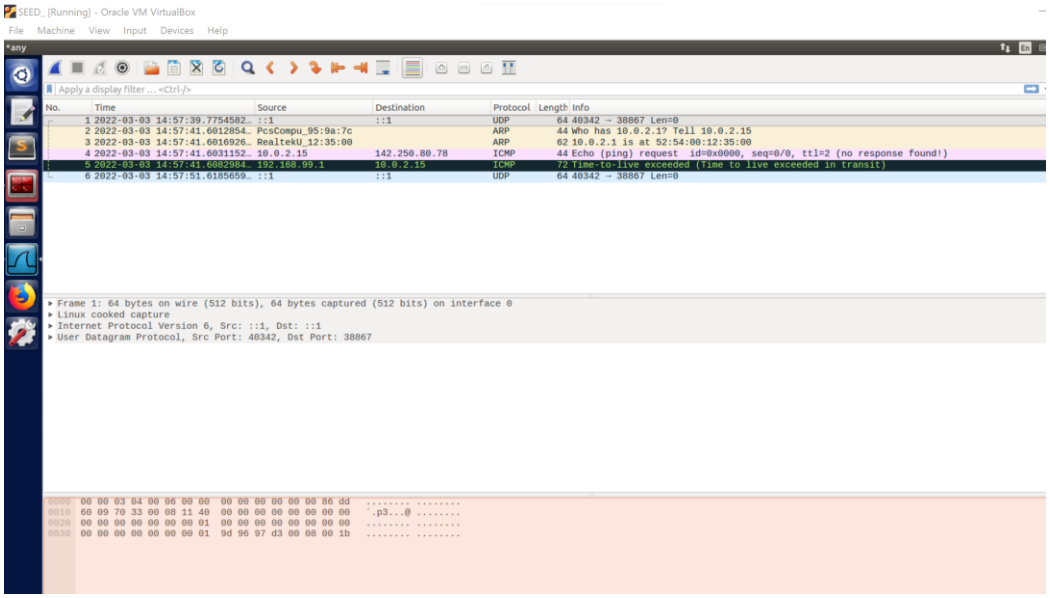
d. 

e. Run trace.py and capture packets on Wireshark in the meanwhile. We can see that time-to-live exceeded since ttl was set to 1. We can get the IP address of the first router at the last ICMP packet captured.

f. 

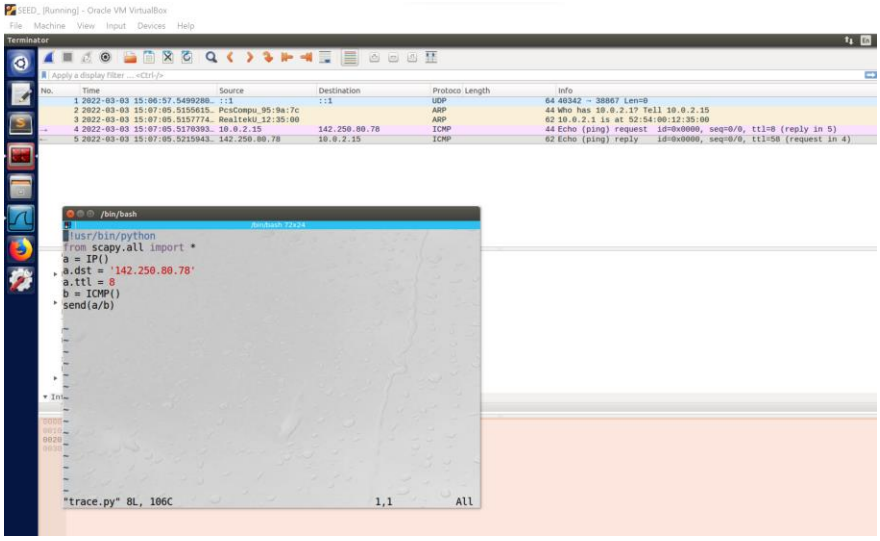
g. Increase ttl field value by 1 every time if time-to-live exceeded. The IP address of the next router can be found by looking at the last ICMP packet information.

h. 

i. 

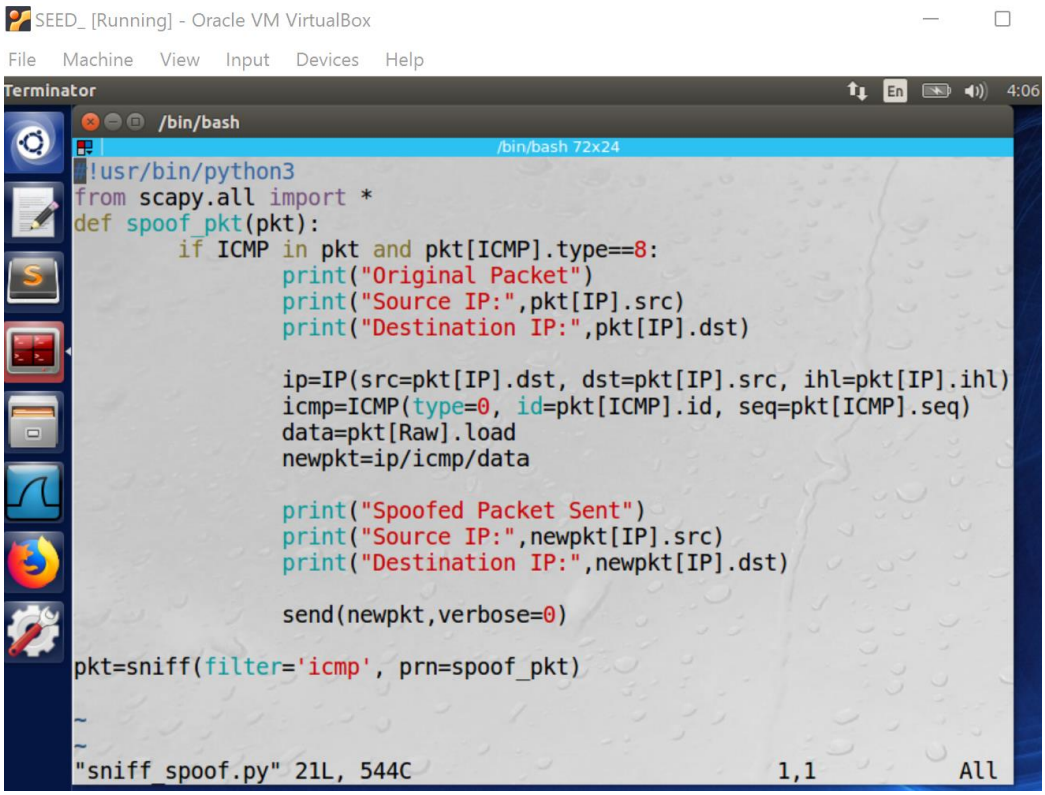
j. Screenshots of steps in between are omitted; the IP address of each router is documented as:
   i. 10.0.2.1
   ii. 192.168.99.1
   iii. 64.18.144.1
   iv. 206.82.104.232
   v. 206.82.104.202
   vi. 108.170.248.33
   vii. 142.251.65.101
   viii. 142.250.80.78 (destination IP)

k. The packet takes 8 hops to reach the destination IP address, when ttl is set to 8, on Wireshark an ICMP echo reply from youtube.com's IP address is received. Indicating that the packet reaches the destination.

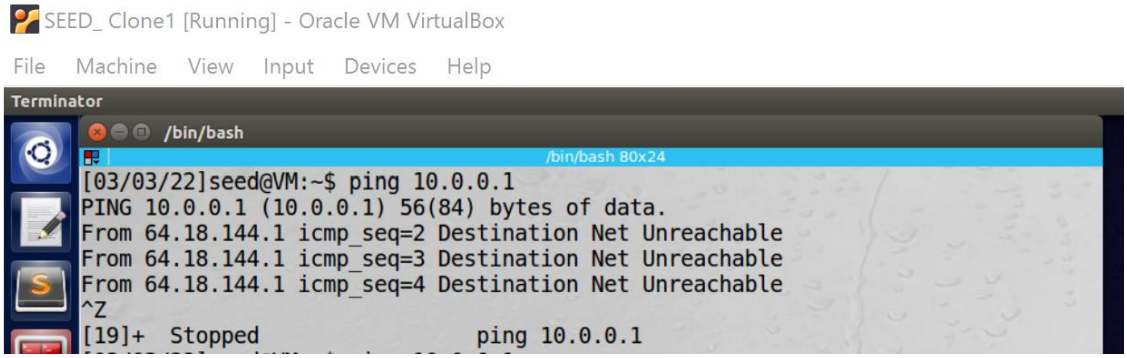
l. 

VII. Task 1.4

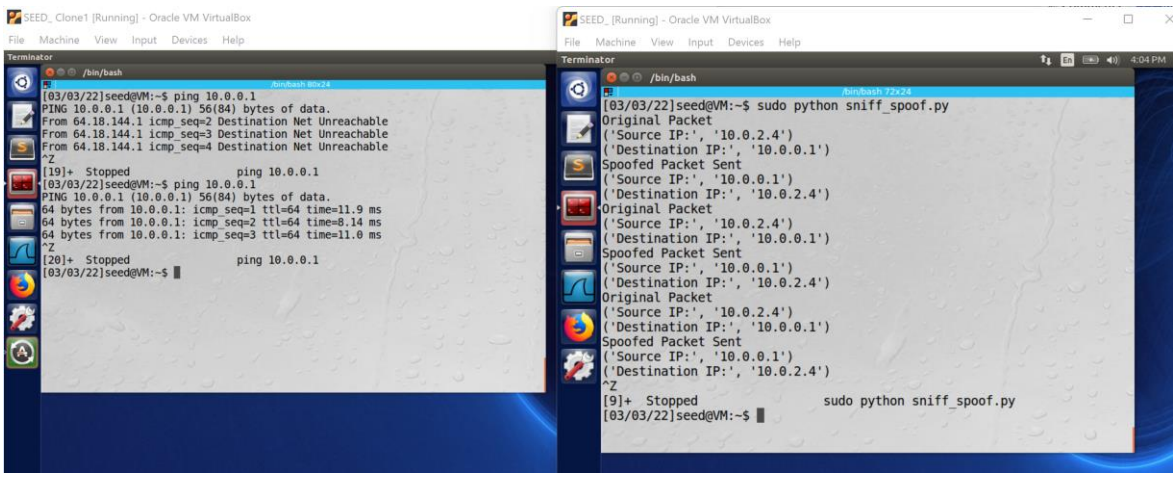
a. Create sniff_spoof.py using the code of *handsonsecurity.net, Network Security, Chapter 15 slides, pp 28 by Wenliang Du.*

b. 

c. On the victim machine, we ping a random IP, here I choose 10.0.0.1, and it is not reachable when pinging it.

d. 

e. On attacker machine, make sniff_spoof.py executable and then run it, while running it, ping 10.0.0.1 again on the victim machine, we can see it is receiving replies from 10.0.0.1 which is not alive.

f. 

g. Stop the program on attacker machine and try to ping 10.0.0.1 again, the destination net unreachable message would show up again, indicating that the sniff_spoof.py successfully faked the echo request.

h.