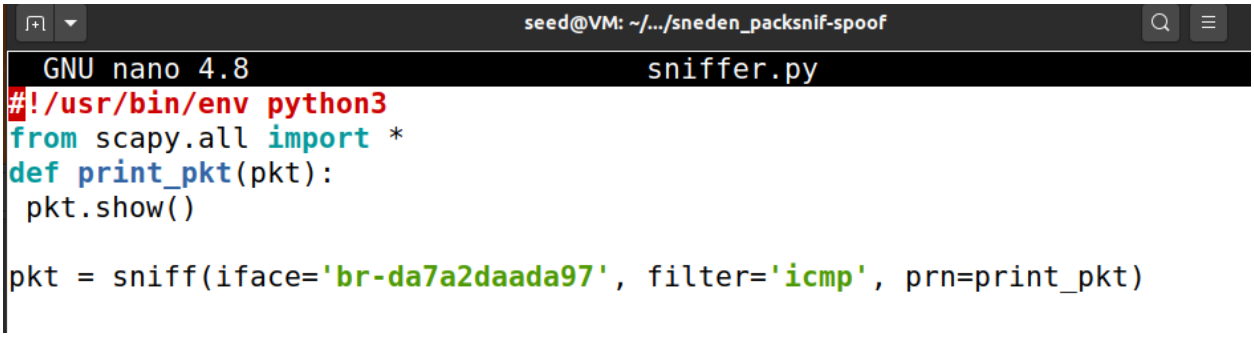
CSE 644 Internet Security Lab-1 (Sniffing & Spoofing packets)

Sneden Rebello

Task 1.1A - Run the Scapy program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

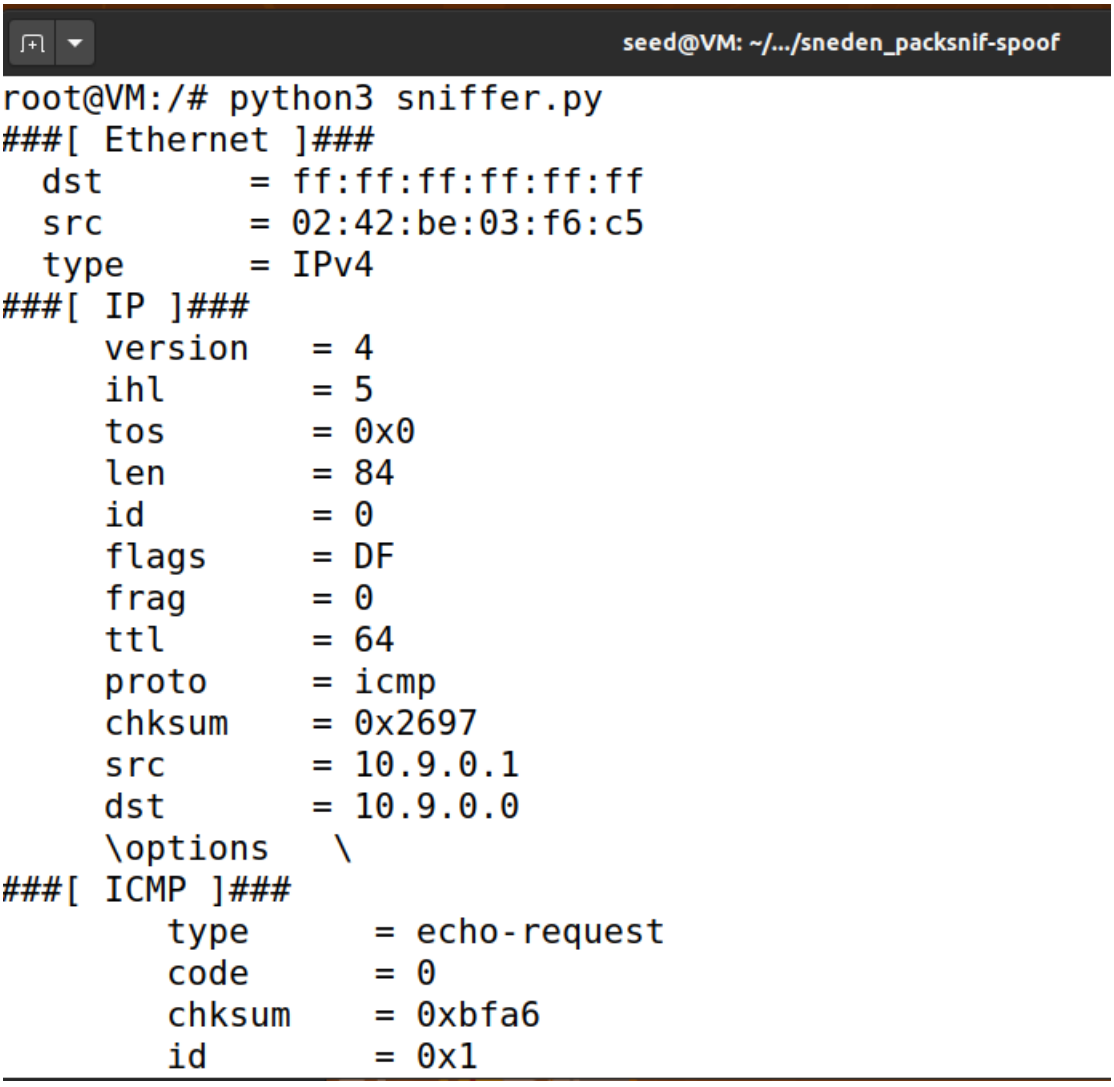
The sniffer.py code is as shown below. I obtained the interface to sniff on by running 'ifconfig' on the attacker container.



Using the docker host A container, login using root, run the scapy code that captures ICMP packets and displays them. Using the ping utility, we generate ICMP packets from another terminal. In this case I did a ping to IP 10.9.0.0 .

```
[02/06/22]seed@VM:~/.../sneden_packsnif-spoof$ ping 10.9.0.0
ping: Do you want to ping broadcast? Then -b. If not, check your local firewall
rules
[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
```

The packet's content, including Ethernet headers, IP headers, ICMP headers, and raw payload, is then shown by the running program:



Now in this case, we execute the same code without root access by doing ‘su seed’ in the root container to change user. We get an error message stating that the operation is not permitted. It happens when the sniff function tries to initialize a raw socket. Promiscuous mode is enabled via raw sockets. However, the software requires root capabilities to enable promiscuous mode. As a result, we'll require root privileges to sniff the raw socket in promiscuous mode.

```
[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
^Z
[1]+  Stopped                 ping -b 10.9.0.0
[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
```

After doing a ping to 10.9.0.0, we notice the error.

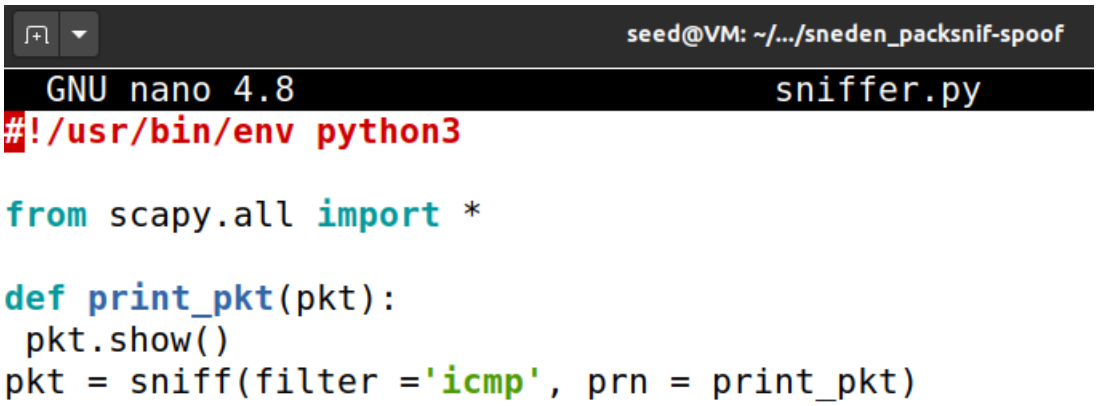
```
seed@VM:/$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 8, in <module>
    pkt = sniff(iface='br-0d69bb96f23d', filter='icmp', prn=print_
pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
 line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
 line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py
", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, so
cket.htons(type))  # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
seed@VM:/$
```

<u>Task 1.1 b</u> - Please set the following filters and demonstrate your sniffer program again

- **Capture only the ICMP packet**
- **Capture any TCP packet that comes from a particular IP and with a destination port number 23.**
- **Capture packets comes from or to go to a particular subnet.**

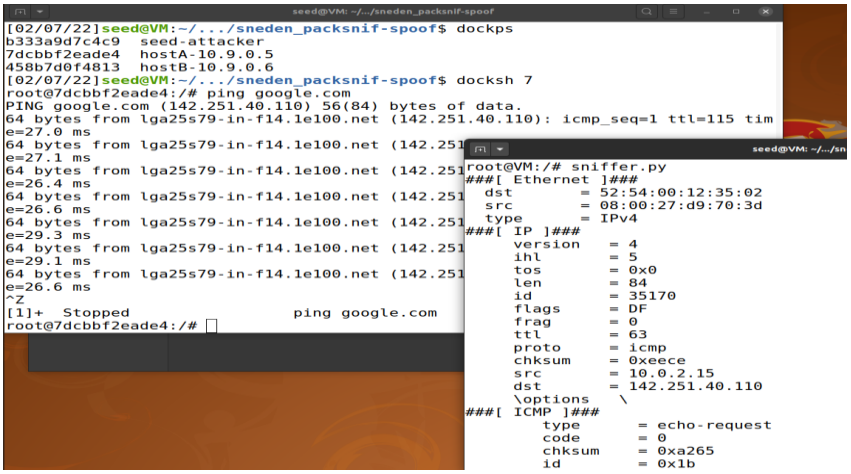
1.To capture ICMP packets : The sniffer python code is as below,

Reference source - https://biot.com/capstats/bpf.html



We run the above code and then ping any address, in this case 'ping google.com' We can see that the code sniffs the ICMP packets on the network and shows the information included in the packet as soon as we start the ping.

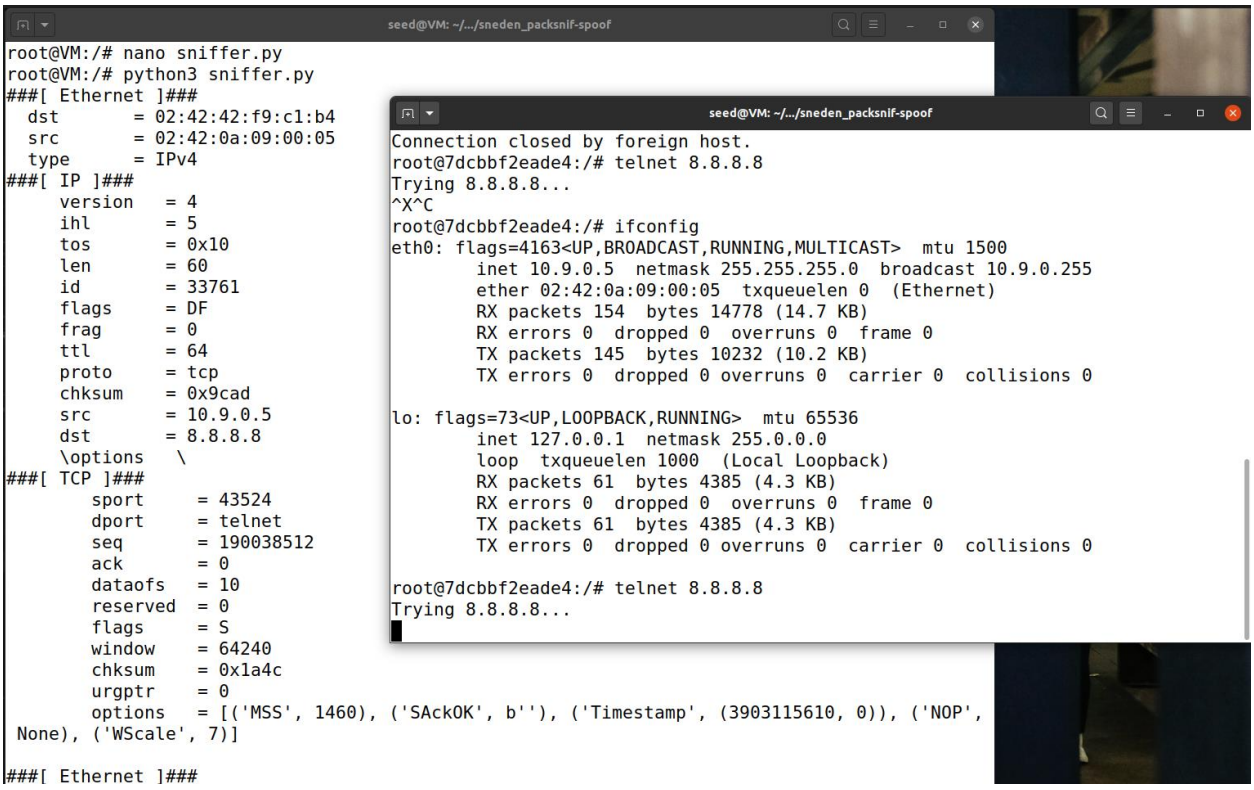
The below image shows the captured packets using the sniffing program. Only the ICMP packets are captured.



2. Capture TCP packets on 10.9.0.5 and port 23 :

I will be sniffing TCP packets on 10.9.0.5 (Host A) on port 23.

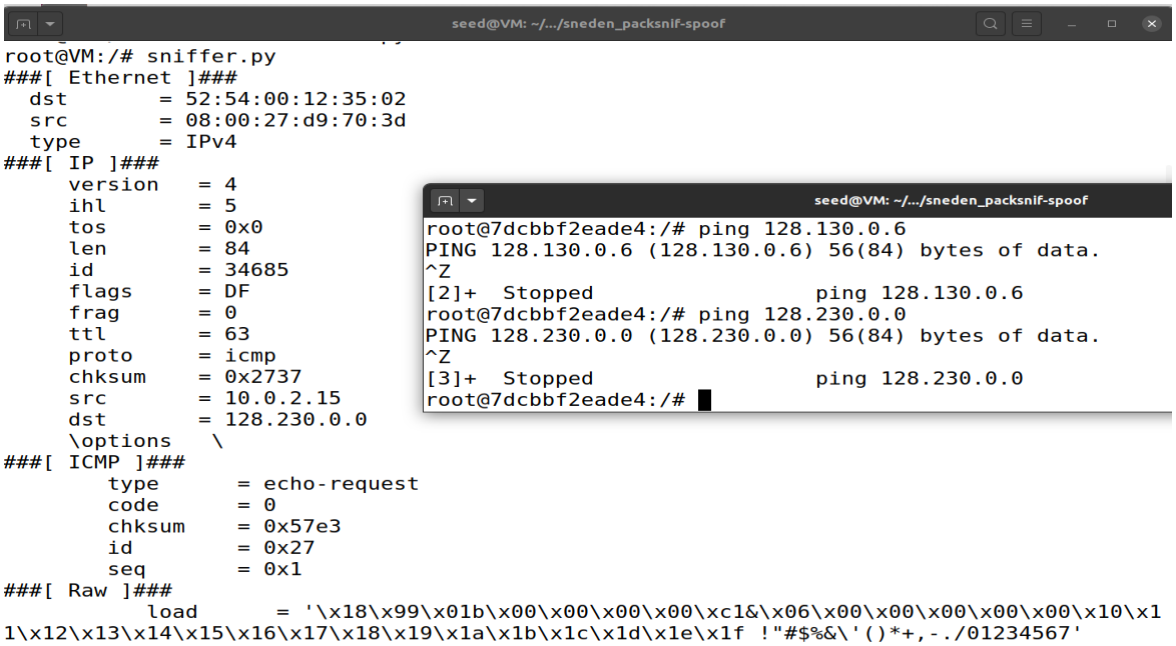
On running the sniffer program and running the telnet service (port 23) to google server 8.8.8.8 from the host A machine. I was able to sniff the TCP packets.



3. Capture packets on 128.230.0.0/16 subnet.

Running the following sniffer code, sniffs packets on the particular subnet included.

At first, I try to sniff on a different subnet to see if the code captures packets. I tried this on 128.130.0.6. The sniffer program could not sniff the packets after the ping. Then I tried pinging the subnet on the filter 128.230.0.0 and now the sniffer program was able to sniff the packets a shown below.



<u>Task 1.2 -</u> The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver.

Below is the code written on python console to spoof an ICMP echo request actually from 10.9.0.1 with any arbitrary source IP address – ie. 8.8.8.8.

*>>> from scapy.all import **

>>> a = IP(src='8.8.8.8', dst='10.9.0.5')

>>> b = ICMP()

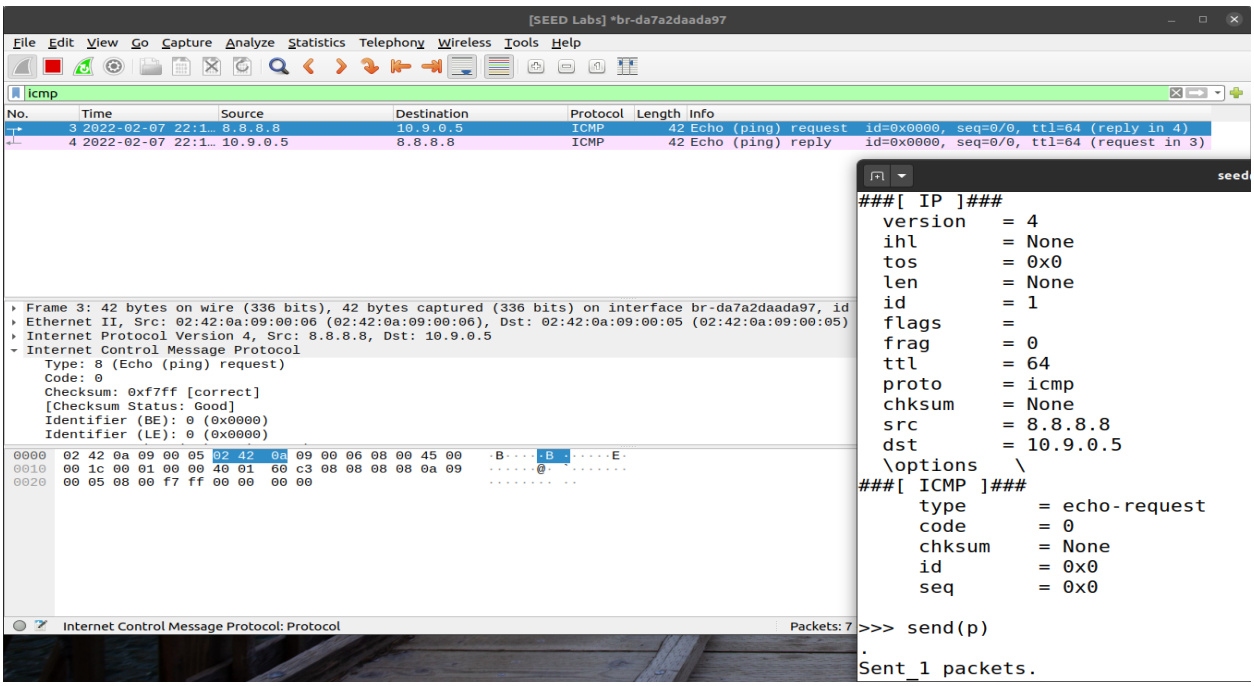
>>> p = a/b

>>>p.show()

>>> send(p)

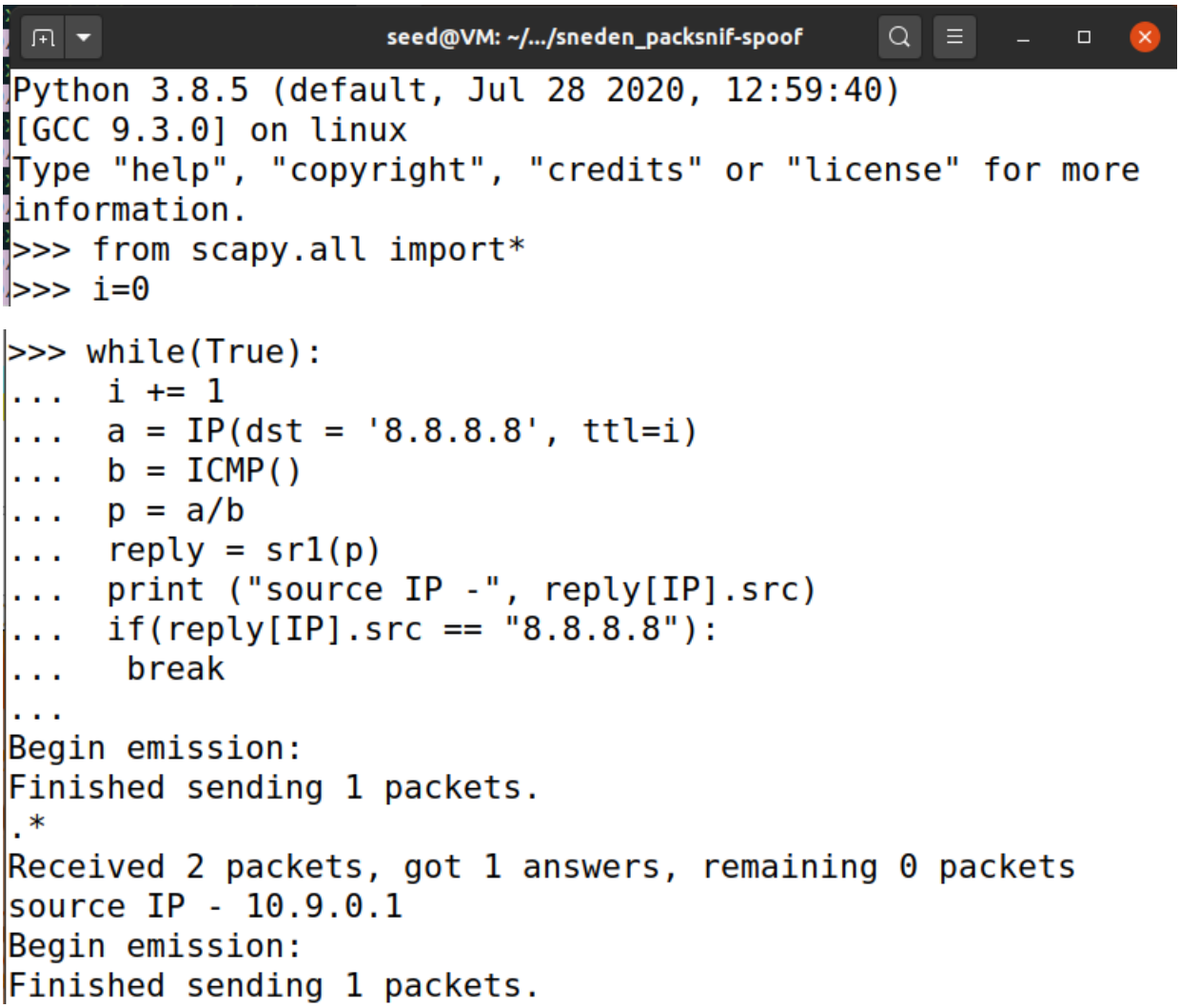
Sent 1 packets.

Now start Wireshark, to check if the spoofed packet is received. As seen here, the packet is received as we can see that the receiver accepted the request and sent back a reply.



<u>Task 1.3:</u> The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. This is basically what is implemented by the traceroute tool. In this task, we will write our own tool.

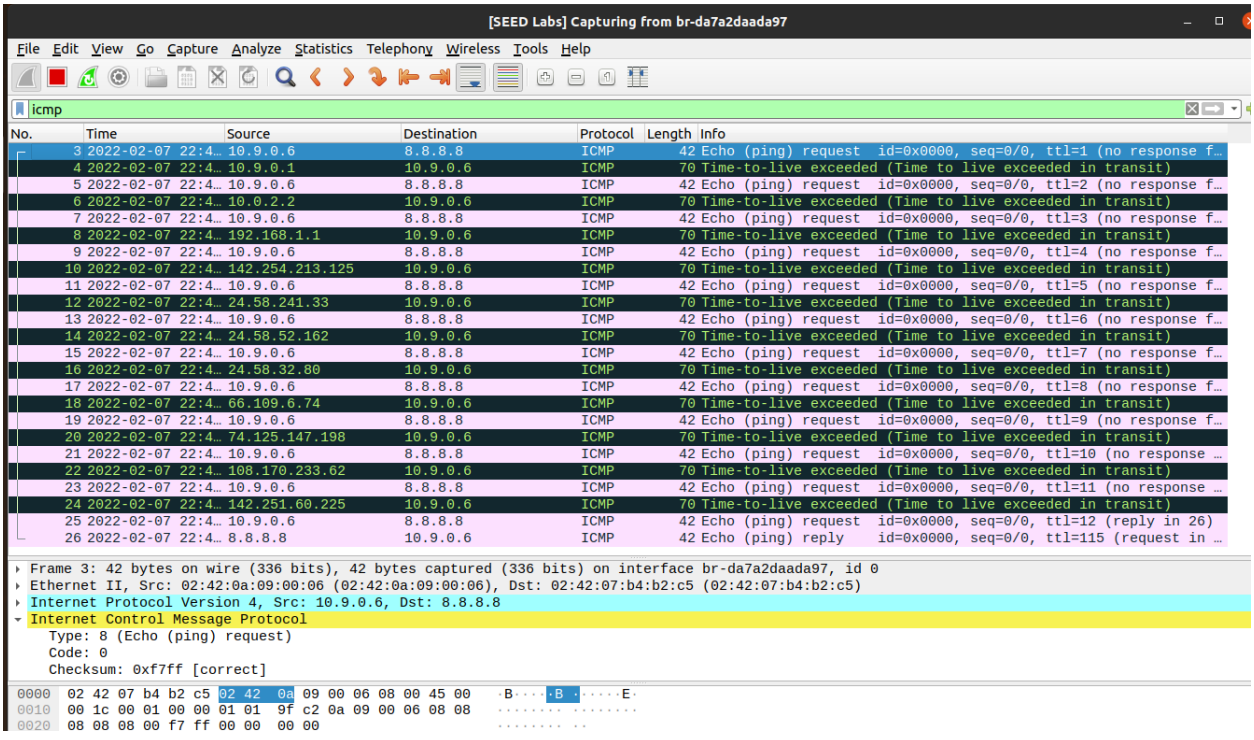
The scapy code for implementing the traceroute functionality is as follows. The code will print out the distance to the destination IP, which in this case is google's server 8.8.8.8.



On running the code, and printing the distance = 'i', we are able to obtain a status of the route of the packet to reach 8.8.8.8 from the source 10.9.0.1 including all the routers in the route taken that dropped the packet with their IP address included.

```
Received 1 packets, got 1 answers, remaining 0 packets
source IP - 108.170.233.62
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
source IP - 142.251.60.225
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
source IP - 8.8.8.8
Distance :  12
root@458b7d0f4813:/# nano spoof.py
```

Using wireshark I was able to actually see on a graphical interface a trace of the packet from source to destination after passing through different routers. In this case, the distance between the source and the destination came out to be 12 hops.



Task 1.4: **In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and then-spoof program. From the user container, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on the VM, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the ping program will always receive a reply, indicating that X is alive. You need to use Scapy to do this task.**

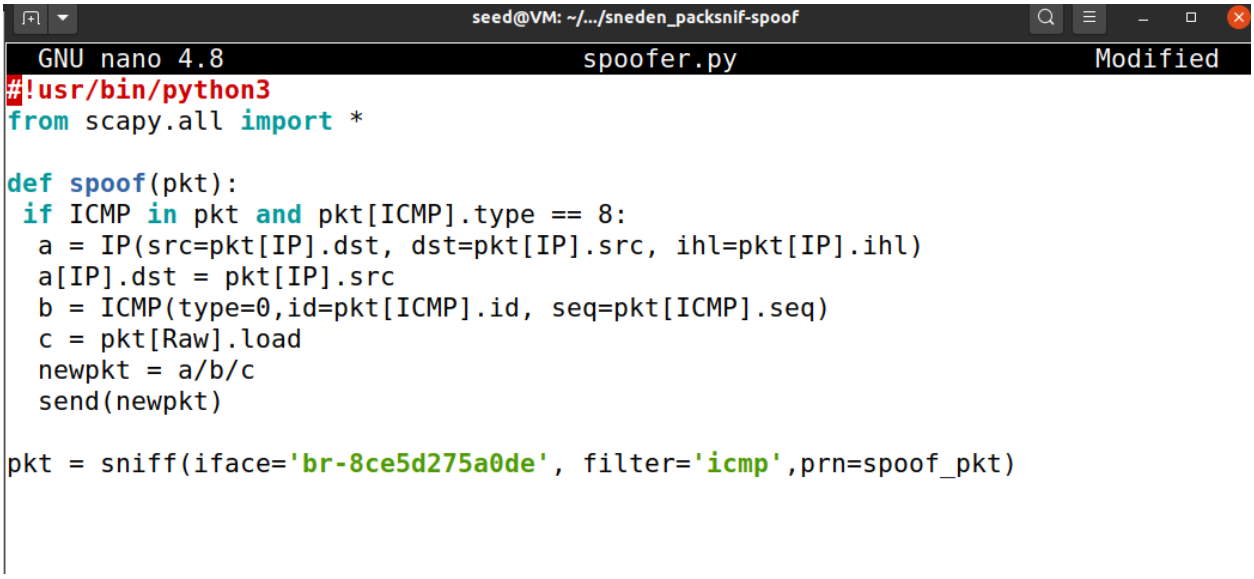
You should ping the following three IP addresses

ping 1.2.3.4 # a non-existing host on the Internet

ping 10.9.0.99 # a non-existing host on the LAN

ping 8.8.8.8 # an existing host on the Internet

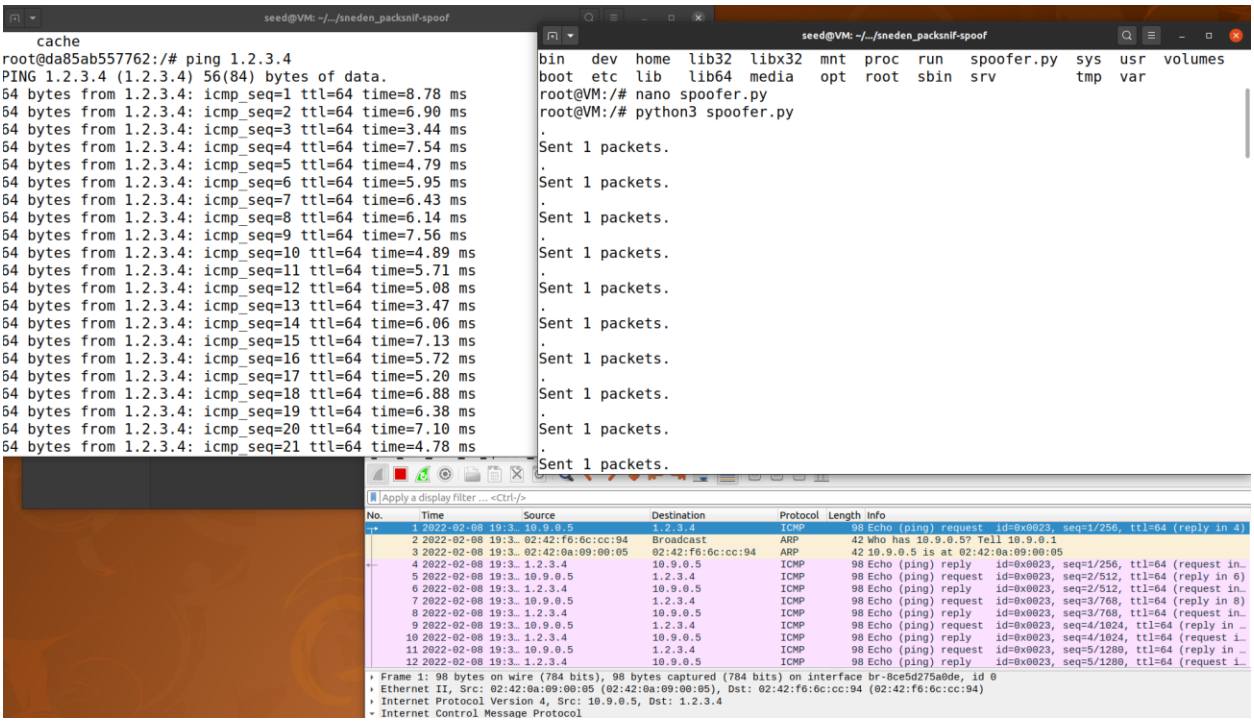
The sniffing and spoofing code is implemented in the below code.



The program sniffs ICMP packets and generates and sends a faked ICMP echo reply if it is an ICMP echo request (type 8).

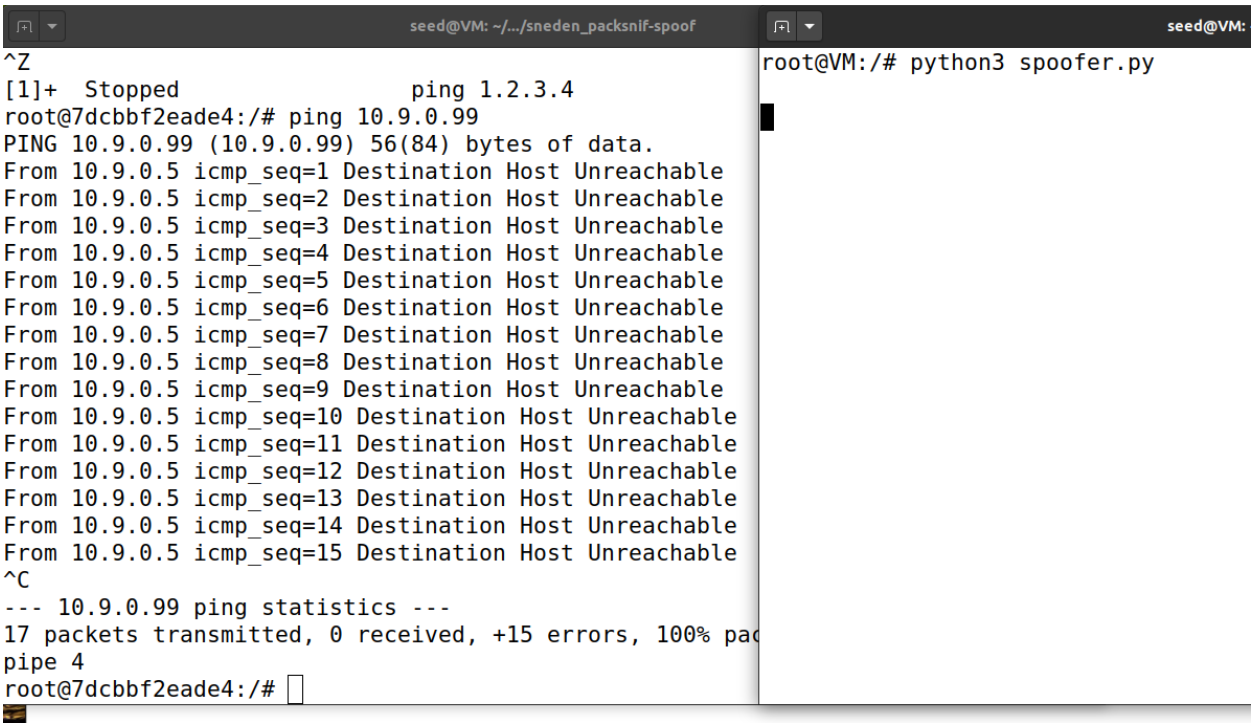
Ping 1.2.3.4

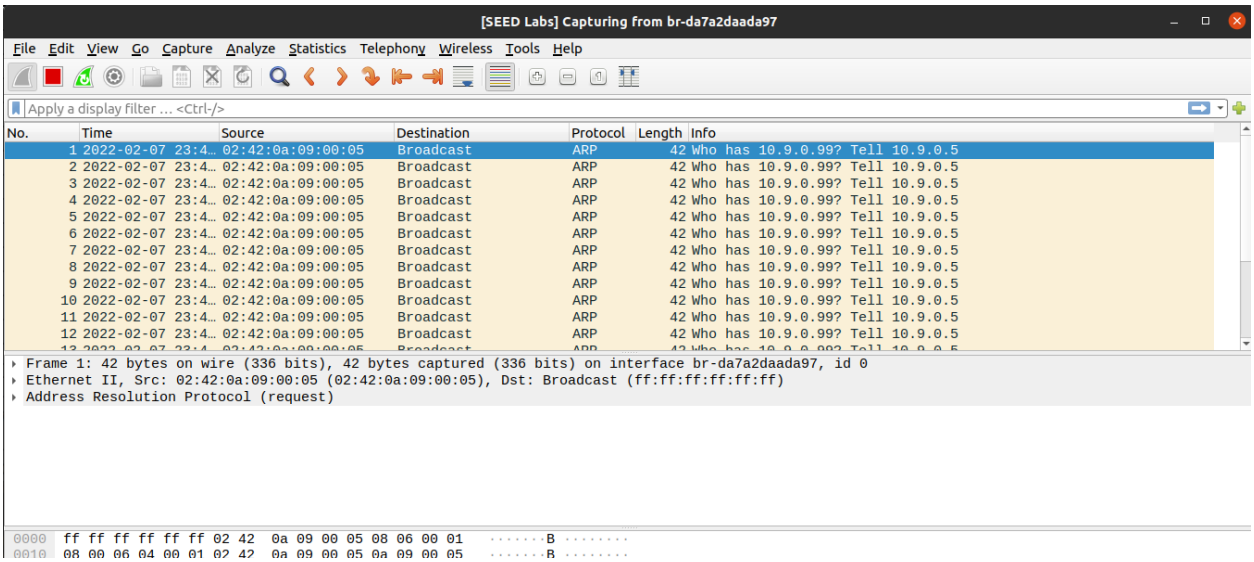
We run the spoofer.py program and then use the Host A (10.9.0.5) to ping a non-existing host on the Internet – 1.2.3.4 and find that the ping is successful due to the spoofed echo return, giving the impression that the host is reachable.



Ping 10.9.0.99

We run the spoofer.py program and then use the Host A (10.9.0.5) to ping a non-existing host on the same LAN – 10.9.0.99 and find that the ping is not successful, and Host is unreachable. This happens because the host does not exist, and a router would not give a reply. The ARP protocol keeps broadcasting for information about 10.9.0.99 with no reply.





Ping 8.8.8.8

We run the spoofer.py program and then use the Host A (10.9.0.5) to ping an existing host, google server 8.8.8.8 on the Internet and find that the ping is successful due to the host actually existing and receives a genuine response after every request from the router.