Rakshith Nallahally Shivanna
928670713
rnallaha@syr.edu
Internet Security (CIS644)
Lab Report 1: Sniffing and Spoofing
Date 01/28/2020

Task 1.1A.The above program sniffs packets. For each captured packet, the callback function print_pkt () will be invoked; this function will print out some of the information about the packet. Run the 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 code below uses sniff method from scapy to sniff packets, I am saving it in the file task1.1_sniff.py.

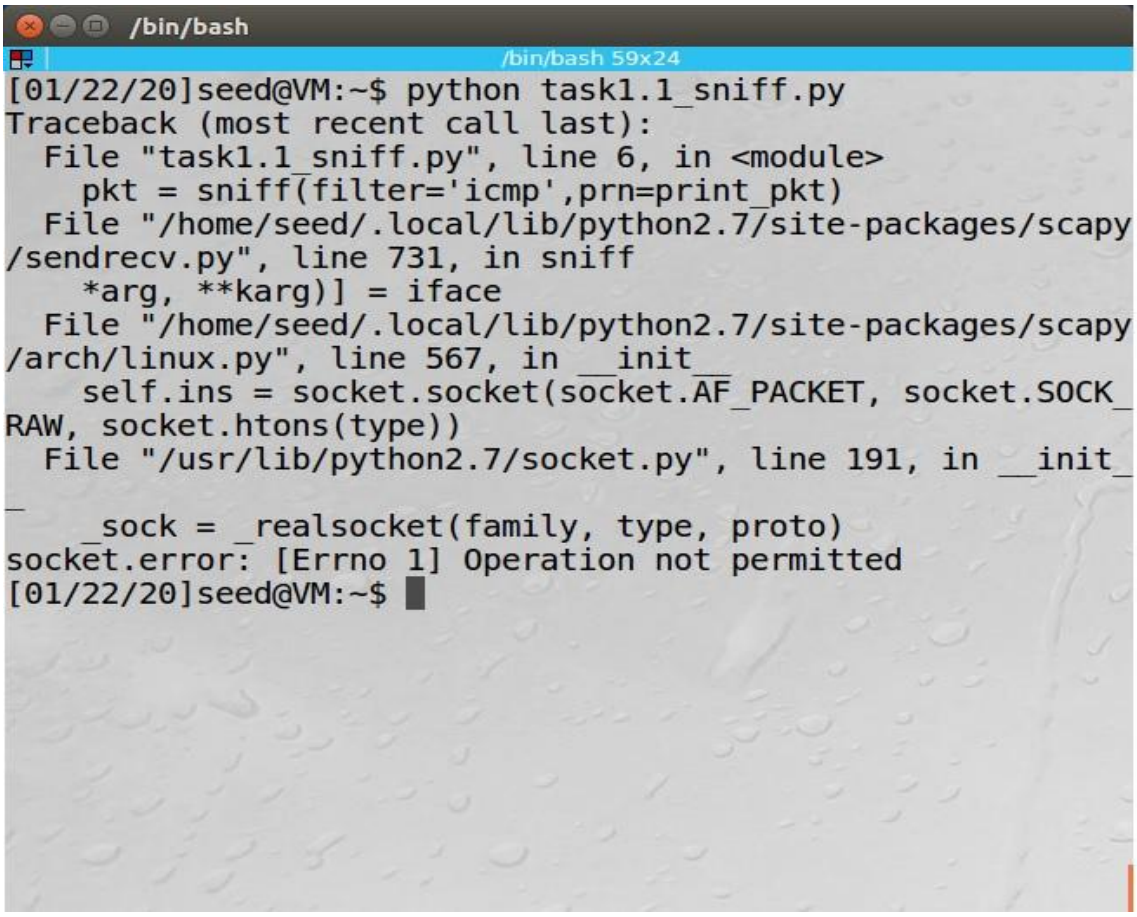
```
[01/22/20]seed@VM:~$ cat task1.1_sniff.py
#!/usr/bin/python
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='icmp',prn=print_pkt)
[01/22/20]seed@VM:~$
[01/22/20]seed@VM:~$
[01/22/20]seed@VM:~$
```

In order to demonstrate that task1.1_sniff.py can capture ICMP packets, I am trying to ping my VM from another VM in my LAN, I am executing this code as super user and the pkt.show() is printing header fields of the packet. This demonstrates that my filter is able to sniff packets.

```
/bin/bash 56x26
^C[01/28/20]seed@VM:~$ clear
[3;J
[01/28/20]seed@VM:~$ sudo python task1.1_sniff.py
###[ Ethernet ]###
  dst       = 08:00:27:cb:0d:d0
  src       = 08:00:27:3b:2b:b3
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 2528
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x18b1
     src       = 10.0.2.10
     dst       = 10.0.2.15
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x9e08
        id        = 0x11bc
```

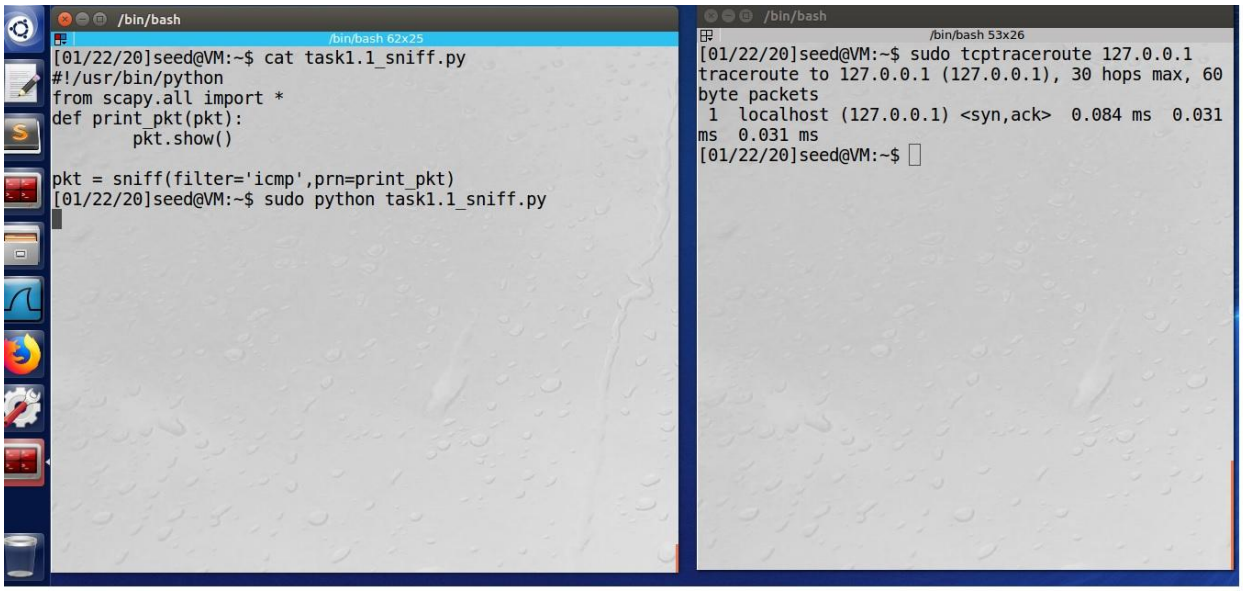
Step 3: I tried to run the same code using user mode, but the code does not execute because scapy does not work on user mode, scapy requires privileges of super user mode. The error in the below image shows that scapy cannot execute _realsocket() which is one of its internal libraries, in order to establish a socket connection through this function we need root permission or super user mode.

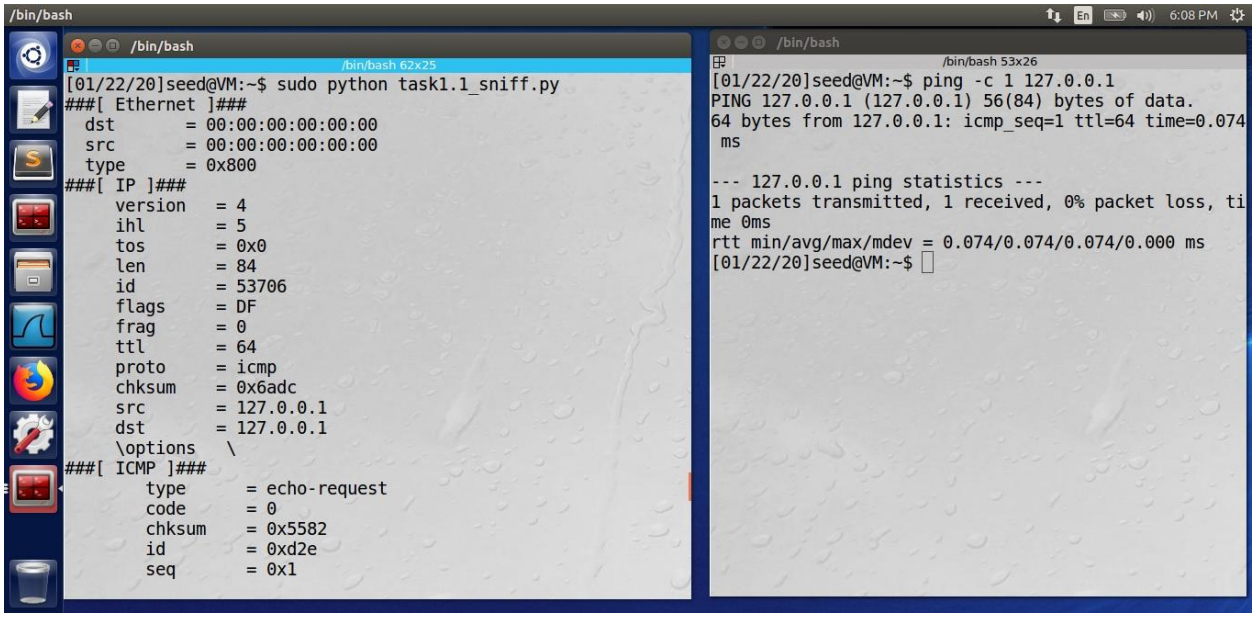


Task 1.1B.Usually, when we sniff packets, we are only interested certain types of packets. We can do that by setting filters in sniffing. Scapy's filter use the BPF (Berkeley Packet Filter) syntax; you can find the BPF manual from the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

• Capture only the ICMP packet

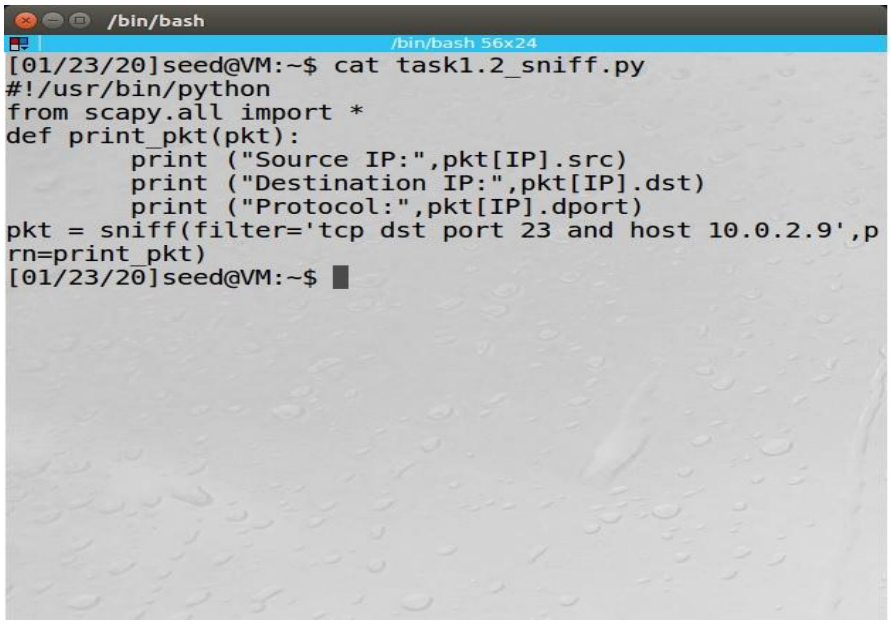
In order to demonstrate that my script is sniffing only ICMP packets I am trying to do a tcptraceroute on my loopback address, since I have set the filter to ICMP we are unable to sniff TCP traffic. Whereas when I do a normal ping to the loopback my sniffer is successfully able to print the packet information.

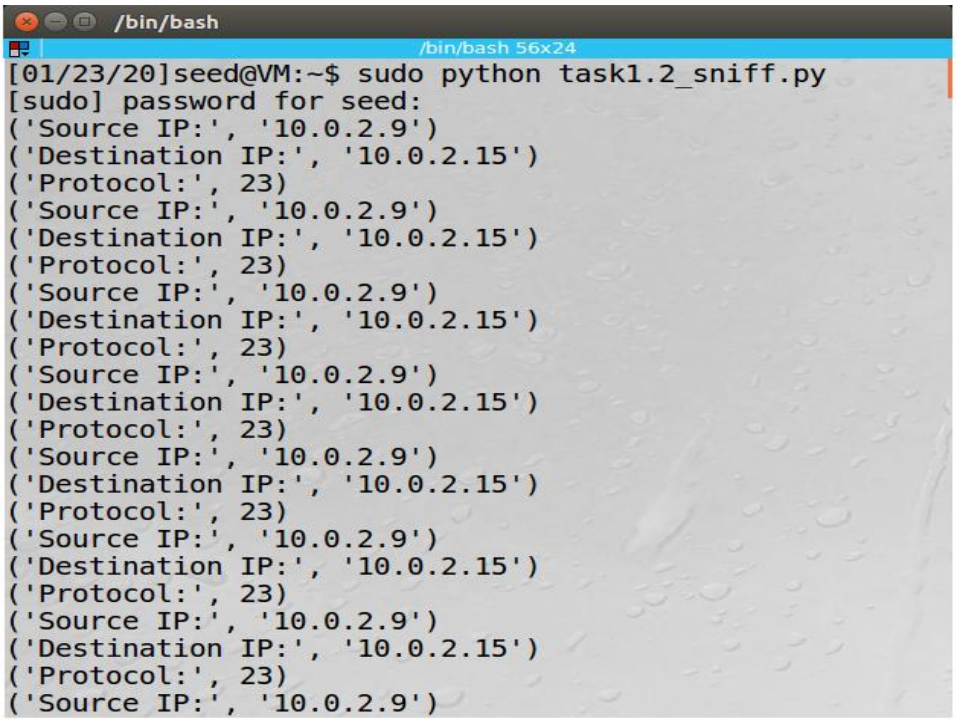




• Capture any TCP packet that comes from a particular IP and with a destination port number 23.

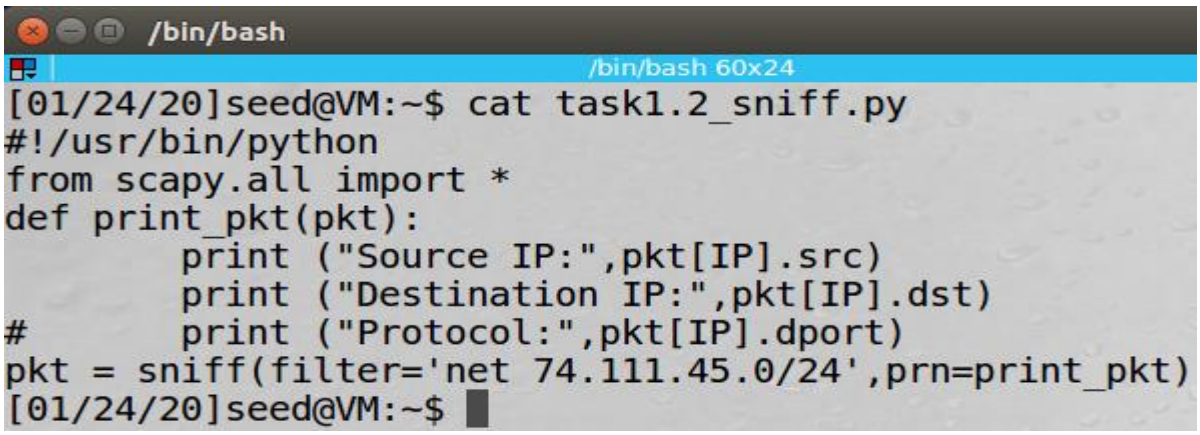
I have modified my sniffer script to capture TCP traffic on dst port 23, here I am using a different VM to do tcptraceroute towards IP address of the VM where I am running the script, as expected we can see that we are successfully able to sniff traffic from a particular host on port 23.

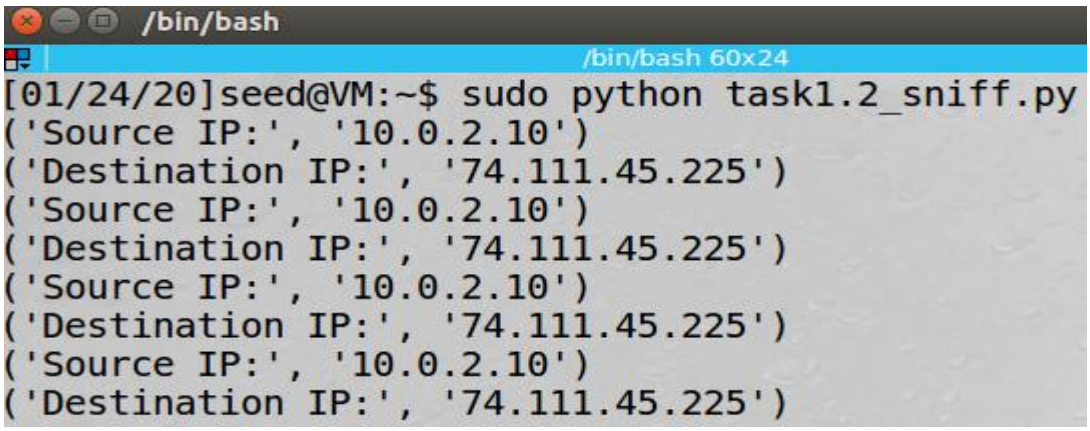




• Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as `128.230.0.0/16`; you should not pick the subnet that your VM is attached to.

Here I am trying to capture traffic from a particular subnet 72.111.45.0/24, which is the subnet provided by my wifi provider. I am successfully able to reach IP address from this subnet using my VM.

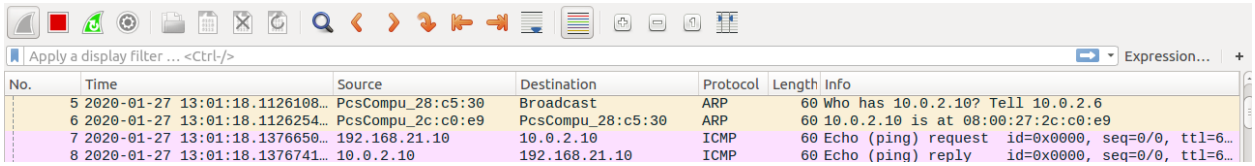




Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.

I have modified the code, I am trying to spoof a non existing source IP 192.168.21.10 towards an IP address of my VM 10.0.2.10, I was successfully able to send the packet and this can be demonstrated by the wireshark output.

```
[01/27/20]seed@VM:~$ cat spoof_1.2.py
from scapy.all import *
a=IP()
a.src='192.168.21.10'
a.dst='10.0.2.10'
b=ICMP()
p=a/b
send(p)
[01/27/20]seed@VM:~$
```

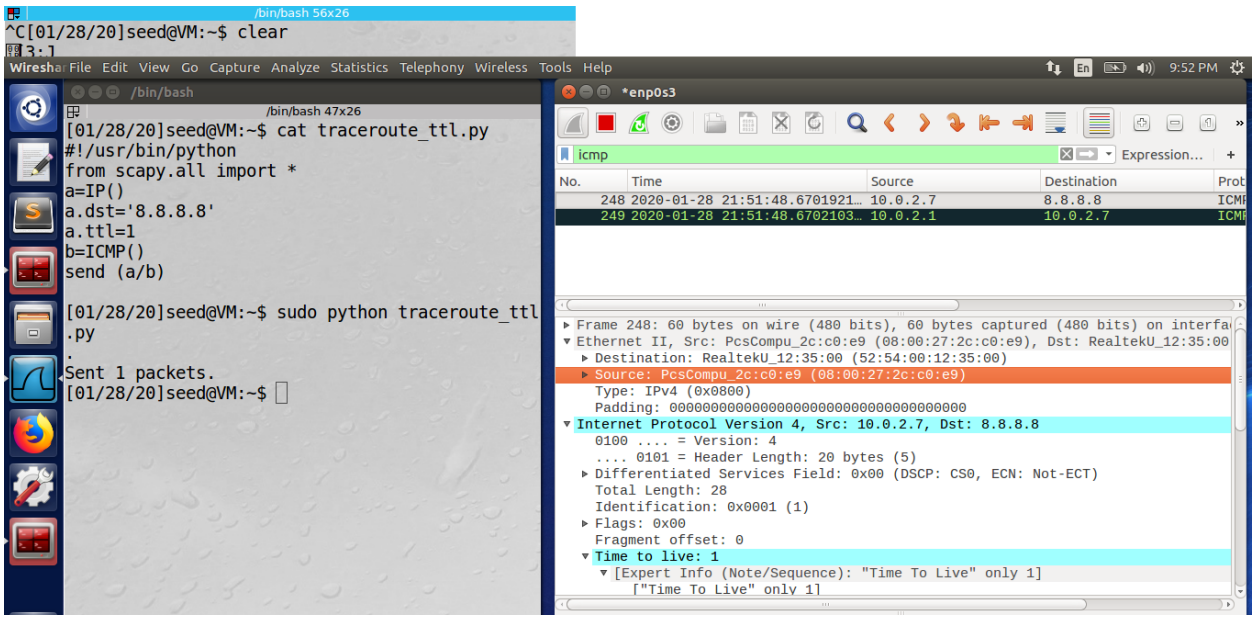


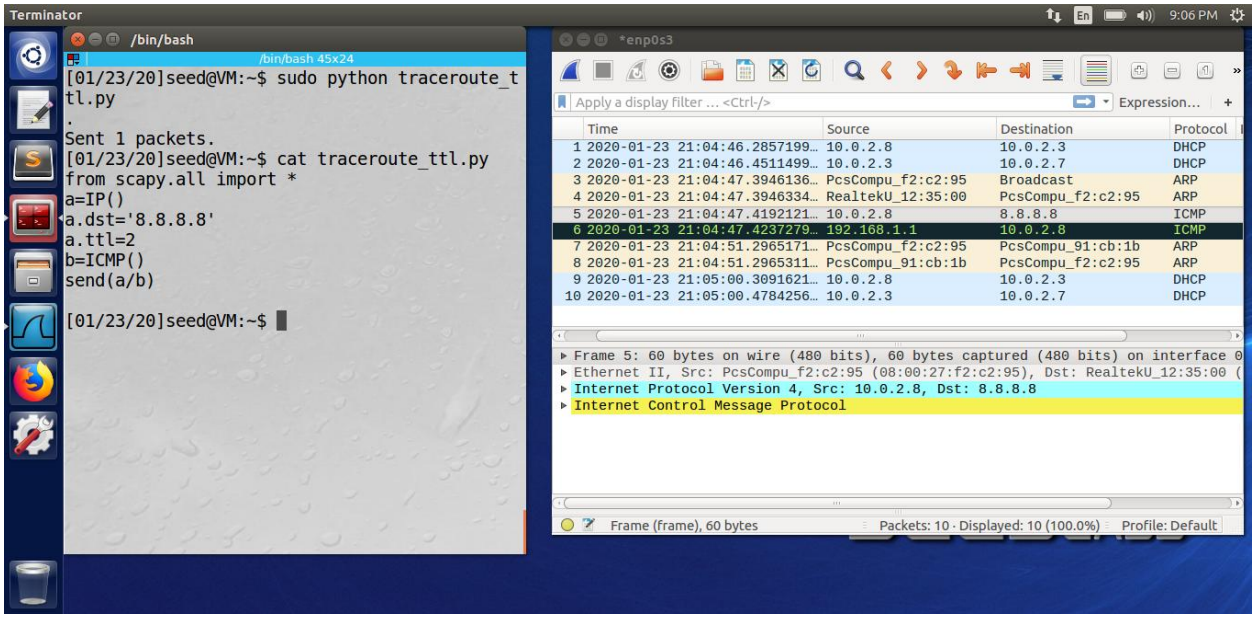

```
[01/27/20]seed@VM:~$ sudo python spoof_1.2.py
[sudo] password for seed:
.
Sent 1 packets.
[01/27/20]seed@VM:~$
```

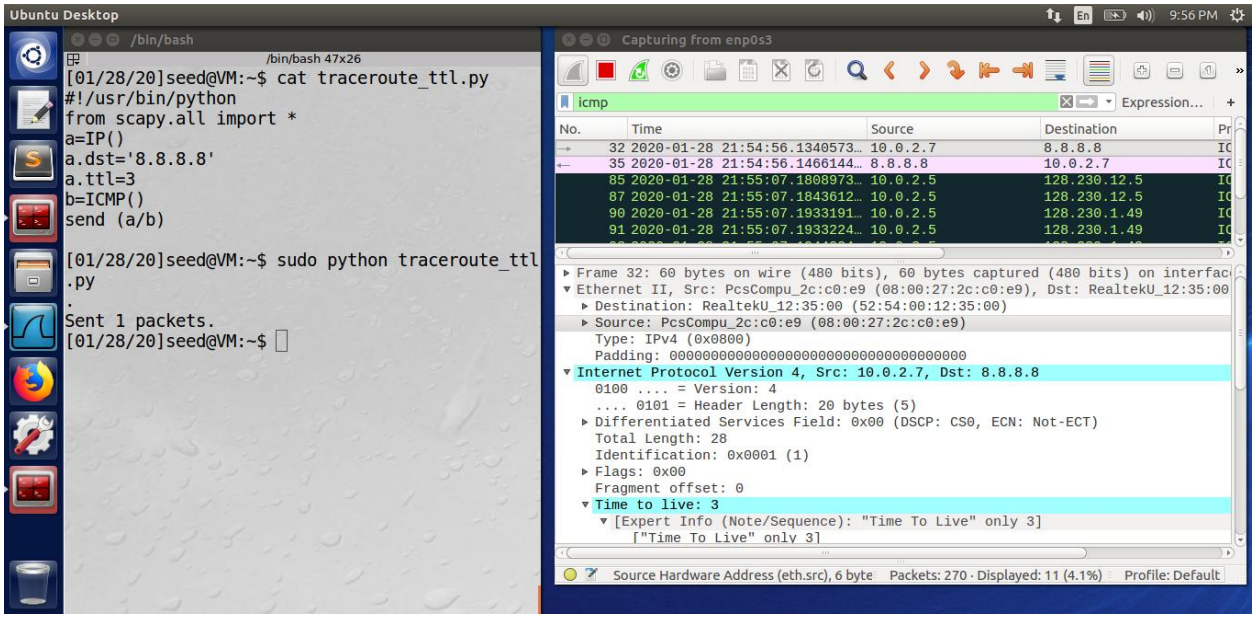
2.3 Task 1.3: Traceroute

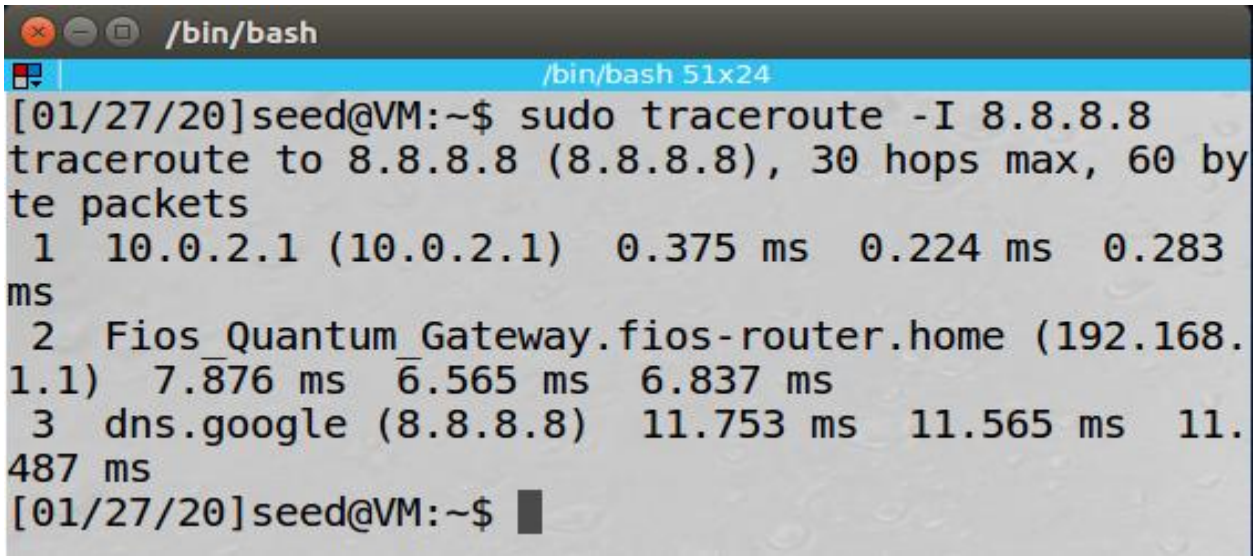
If you are an experienced Python programmer, you can write your tool to perform the entire procedure automatically. If you are new to Python programming, you can do it by manually changing the TTL field in each round, and record the IP address based on your observation from Wireshark. Either way is acceptable, as long as you get the result.

Here I am trying to send ICMP packet to destination 8.8.8.8 with ttl value's 1,2 and 3, as you can see from the wireshark output packet only the IP where the ttl expires replies. Increase in ttl value will increase in number of hops.



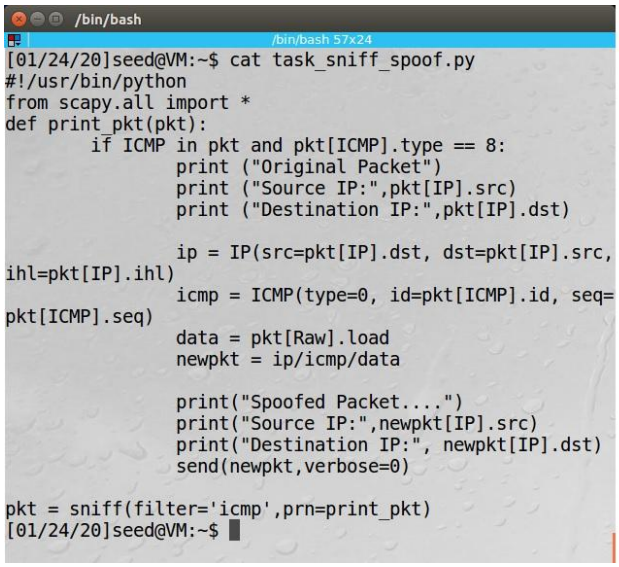


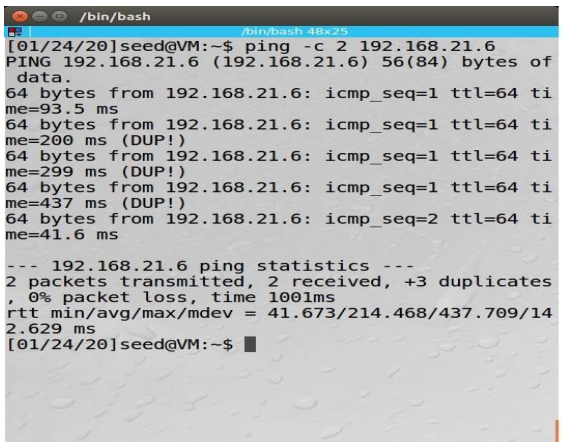

Ubuntu Desktop
9:56 PM
/bin/bash
/bin/bash 47x26
[01/28/20]seed@VM:~$ cat traceroute_ttl.py
#!/usr/bin/python
from scapy.all import *
a=IP()
a.dst='8.8.8.8'
a.ttl=3
b=ICMP()
send (a/b)
[01/28/20]seed@VM:~$ sudo python traceroute_ttl
.py
.
Sent 1 packets.
[01/28/20]seed@VM:~$
Capturing from enp0s3
icmp
Expression...
No. Time Source Destination
32 2020-01-28 21:54:56.1340573… 10.0.2.7 8.8.8.8
35 2020-01-28 21:54:56.1466144… 8.8.8.8 10.0.2.7
85 2020-01-28 21:55:07.1808973… 10.0.2.5 128.230.12.5
87 2020-01-28 21:55:07.1843612… 10.0.2.5 128.230.12.5
90 2020-01-28 21:55:07.1933191… 10.0.2.5 128.230.1.49
91 2020-01-28 21:55:07.1933224… 10.0.2.5 128.230.1.49
Frame 32: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interfac
Ethernet II, Src: PcsCompu_2c:c0:e9 (08:00:27:2c:c0:e9), Dst: RealtekU_12:35:00
Destination: RealtekU_12:35:00 (52:54:00:12:35:00)
Source: PcsCompu_2c:c0:e9 (08:00:27:2c:c0:e9)
Type: IPv4 (0x0800)
Padding: 000000000000000000000000000000000000
Internet Protocol Version 4, Src: 10.0.2.7, Dst: 8.8.8.8
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 28
Identification: 0x0001 (1)
Flags: 0x00
Fragment offset: 0
Time to live: 3
[Expert Info (Note/Sequence): "Time To Live" only 3]
["Time To Live" only 3]
Source Hardware Address (eth.src), 6 byte
Packets: 270 · Displayed: 11 (4.1%)
Profile: Default

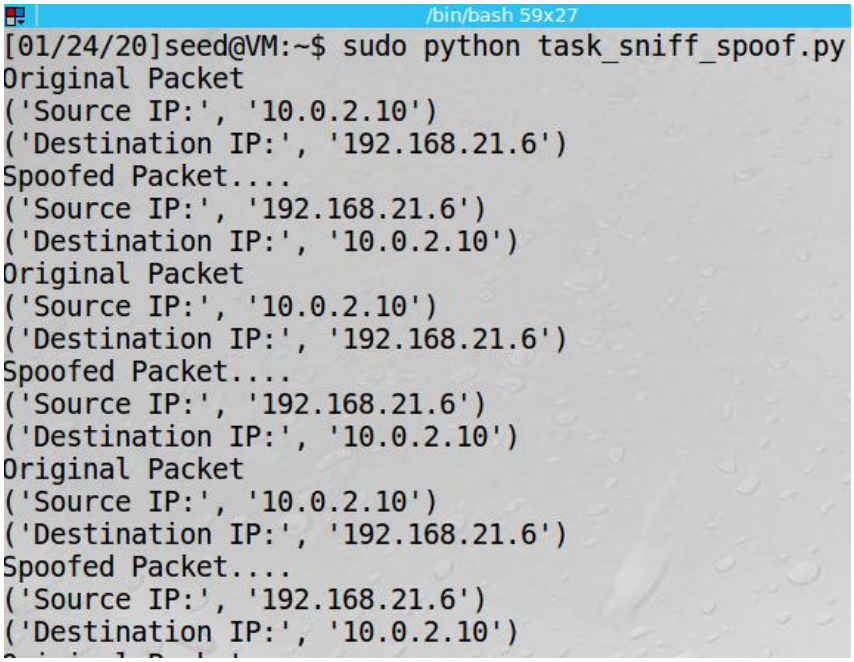

/bin/bash
/bin/bash 51x24
[01/27/20]seed@VM:~$ sudo traceroute -I 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 by
te packets
1 10.0.2.1 (10.0.2.1) 0.375 ms 0.224 ms 0.283
ms
2 Fios_Quantum_Gateway.fios-router.home (192.168.
1.1) 7.876 ms 6.565 ms 6.837 ms
3 dns.google (8.8.8.8) 11.753 ms 11.565 ms 11.
487 ms
[01/27/20]seed@VM:~$

2.4 Sniffing and-then Spoofing In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two VMs on the same LAN. From VM A, 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 VM B, 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, regard-less 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. In your report, you need to provide evidence to demonstrate that your technique works.

Here I am trying to sniff the network for any ICMP packets using sniff, then as soon as I see an ECHO request I send out spoofed packets by capturing the ICMP packets and fetching the source and IP addresses from its header. 192.168.21.6 is an unreachable IP from this VM, I am trying to ping the ICMP, even though IP is not pingable we can still record the responses, it gives us a false understanding that the IP might still be reachable from this network.






/bin/bash 59x27
[01/24/20]seed@VM:~$ sudo python task_sniff_spoof.py
Original Packet
('Source IP:', '10.0.2.10')
('Destination IP:', '192.168.21.6')
Spoofed Packet....
('Source IP:', '192.168.21.6')
('Destination IP:', '10.0.2.10')
Original Packet
('Source IP:', '10.0.2.10')
('Destination IP:', '192.168.21.6')
Spoofed Packet....
('Source IP:', '192.168.21.6')
('Destination IP:', '10.0.2.10')
Original Packet
('Source IP:', '10.0.2.10')
('Destination IP:', '192.168.21.6')
Spoofed Packet....
('Source IP:', '192.168.21.6')
('Destination IP:', '10.0.2.10')

Task 2.1A: Understanding How a Sniffer Works In this task, students need to write a sniffer program to print out the source and destination IP addresses of each captured packet. Students can type in the above code or download the sample code from the SEED book's website.

Students should provide screenshots as evidences to show that their sniffer pro-gram can run successfully and produces expected results. In addition, please answer the following questions:

• Capture the ICMP packets between two specific hosts.

In the code below, I have tried to capture ICMP packets between a particular source and destination. Source is VM1 and destination is VM2. The code is written using pcap library C. As we can see from the output we are successfully able to capture the packets between the two IPs.

```
printf ("Got a packet \n");
printf ("Source IP: %s\n", inet_ntoa(ip->iph_sourcei
p));
printf ("Destination IP: %s\n", inet_ntoa(ip->iph_de
stip));
}
}


int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "ip proto icmp and src host 10.0
.2.6 and dst host 10.0.2.15";
bpf_u_int32 net;
handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, e
rrbuf);
pcap_compile(handle, &fp, filter_exp, 0, net);
pcap_setfilter(handle, &fp);
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}
```

```
/bin/bash 52x26
[01/26/20]seed@VM:~$ sudo ./sniff
[sudo] password for seed:
Got a packet
Source IP: 10.0.2.9
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.9
Got a packet
Source IP: 10.0.2.9
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.9
Got a packet
Source IP: 10.0.2.9
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.9
cGot a packet
Source IP: 10.0.2.9
Destination IP: 10.0.2.15
```

•Question 1.Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.

The process is simple:

To begin with we open a session with one of our interfaces or devices connected to our machine to start sniffing traffic, this can be achieved by library call pcap_open_live. We specify interface / device name, number of bytes to be captured, mode to be operated through this API, prototype is mentioned below.

```
pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms,
            char *ebuf)
```

Next part is to specify the filter command, by providing suitable filter string and the session handler so that we can start sniffing traffic as per our requirement. This can be done by library call pcap_compile, finally we apply our filter using the call `pcap_setfilter`.

```
int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize,
            bpf_u_int32 netmask)
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
```

Library call pcap_loop is used to execute call back function everytime a packet is captured, in our case we are printing a string everytime a packet is captured as per our code.

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
```

•Question 2.Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

We cannot connect to our interfaces / devices without root permission, so program fails in the initial stage where we are trying to establish a session. The program will compile without errors but we get segmentation fault when we run the executable.

•Question 3.Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this.

By default, promiscuous mode is ON, this can be set while creating the pcap session with interface, by enabling the promisc bit to 1 in pcap_open_live library. To turn promiscuous mode off we have to set the promisc bit to 0. I also disabled promiscuous mode in all the interfaces of both machines, but I was still able to see traffic.

Command I used to turn promiscuous mode off.

[01/27/20]seed@VM:~$ sudo ip link set enp0s3 promisc off

[01/27/20]seed@VM:~$ sudo ifconfig enp0s3 -promisc

```
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "ip proto icmp";
bpf_u_int32 net;
handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, e
rrbuf);
pcap_compile(handle, &fp, filter_exp, 1, net);
pcap_setfilter(handle, &fp);
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}
```

Task 2.1B: Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings. You can find online manuals for pcap filters. In your lab reports, you need to include screenshots to show the results after applying each of these filters.

• Capture the ICMP packets between two specific hosts (done in previous task)

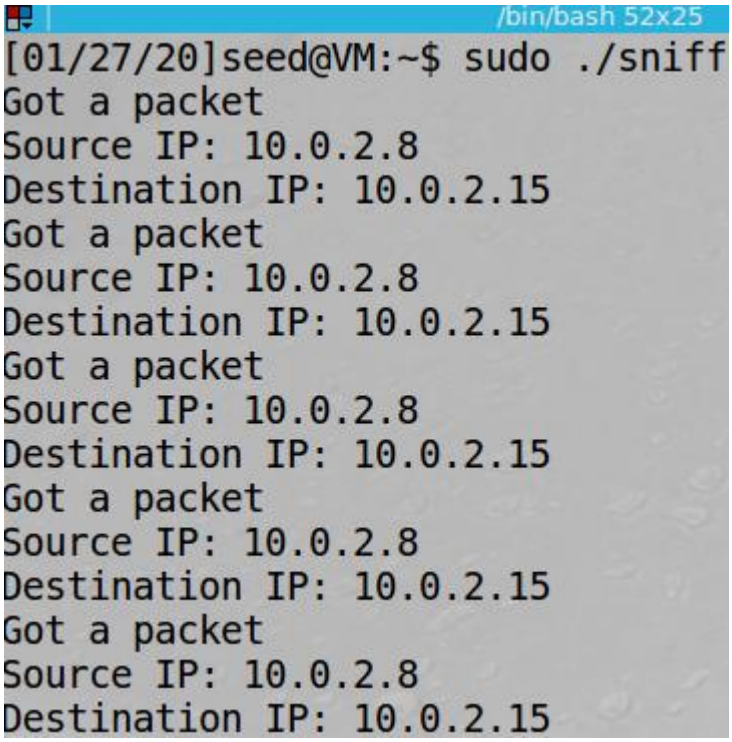
• Capture the TCP packets with a destination port number in the range from 10 to 100.

All we have to do is modify the filter expression to the previous code, to capture traffic between port 10-100 we should use filter tcp dst portrange 10-100. I am doing a tcptraceroute on port 44 from another VM, Sniff is capturing packets for destination port 44, this can be validated through the wireshark ouput.

```
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "tcp dst portrange 10-100";
bpf_u_int32 net;
handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, e
rrbuf);
pcap_compile(handle, &fp, filter_exp, 1, net);
pcap_setfilter(handle, &fp);
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}
```

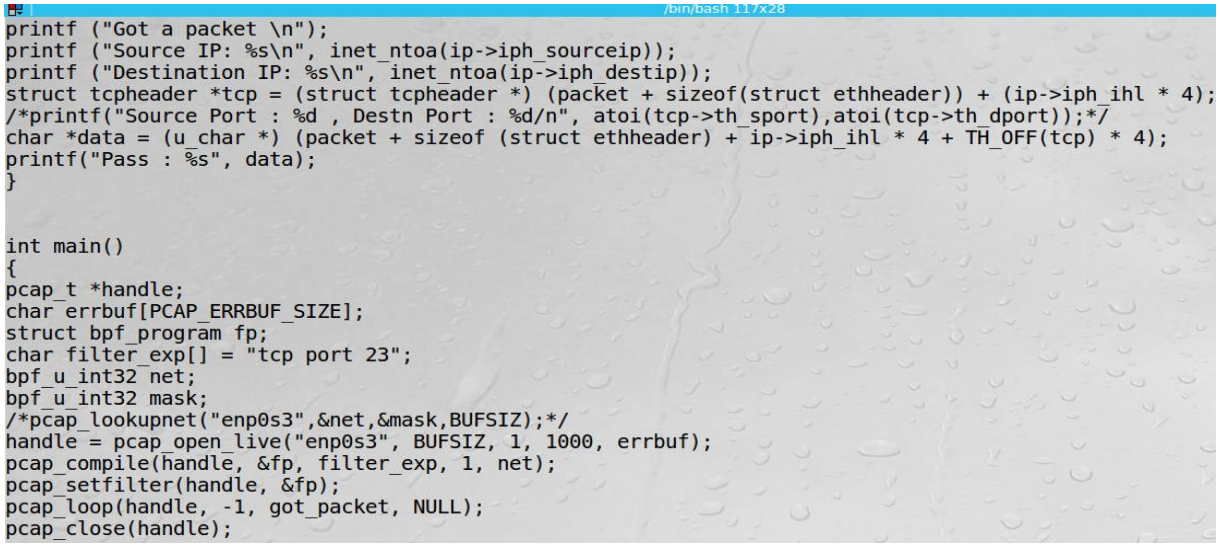
```
/bin/bash 53x24
[01/27/20]seed@VM:~$ sudo tcptraceroute 10.0.2.15 44
traceroute to 10.0.2.15 (10.0.2.15), 30 hops max, 60
byte packets
 1  10.0.2.15 (10.0.2.15) <rst,ack>  1.204 ms  1.065
ms  1.757 ms
```

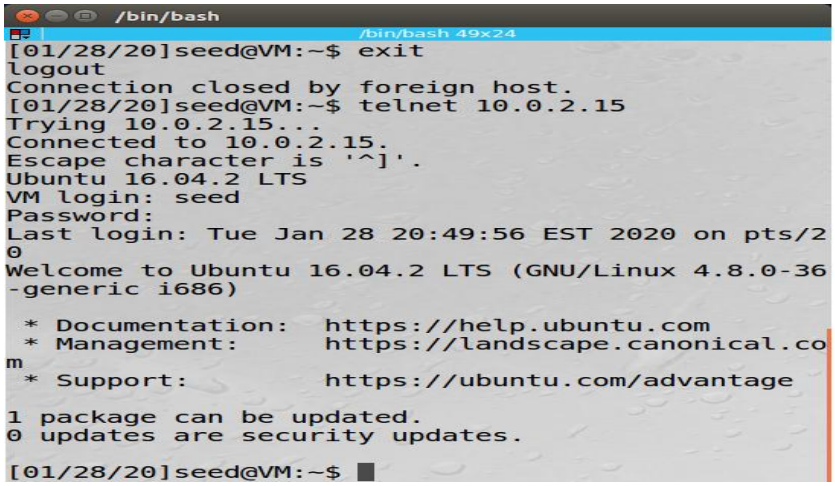
No.	Time	Source	Destination	Protocol	Length	Info
1	2020-01-27 18:05:03.7584033…	10.0.2.8	10.0.2.15	TCP	74	60839 → 44 [SYN] Seq=4274801107 Win=5840 Len=0…
2	2020-01-27 18:05:03.7584222…	10.0.2.8	10.0.2.15	TCP	74	44399 → 44 [SYN] Seq=2949470929 Win=5840 Len=0…
3	2020-01-27 18:05:03.7584284…	10.0.2.8	10.0.2.15	TCP	74	47251 → 44 [SYN] Seq=4147198988 Win=5840 Len=0…
4	2020-01-27 18:05:03.7584327…	10.0.2.8	10.0.2.15	TCP	74	40081 → 44 [SYN] Seq=152610498 Win=5840 Len=0 …
5	2020-01-27 18:05:03.7584369…	10.0.2.15	10.0.2.8	TCP	60	44 → 60839 [RST, ACK] Seq=0 Ack=4274801108 Win…
6	2020-01-27 18:05:03.7584404…	10.0.2.15	10.0.2.8	TCP	60	44 → 44399 [RST, ACK] Seq=0 Ack=2949470930 Win…
7	2020-01-27 18:05:03.7584437…	10.0.2.15	10.0.2.8	TCP	60	44 → 47251 [RST, ACK] Seq=0 Ack=4147198989 Win…
8	2020-01-27 18:05:03.7584469…	10.0.2.15	10.0.2.8	TCP	60	44 → 40081 [RST, ACK] Seq=0 Ack=152610499 Win=…
9	2020-01-27 18:05:03.7584509…	10.0.2.8	10.0.2.15	TCP	74	51119 → 44 [SYN] Seq=2977847783 Win=5840 Len=0…
10	2020-01-27 18:05:03.7594349…	10.0.2.8	10.0.2.15	TCP	74	55009 → 44 [SYN] Seq=823720716 Win=5840 Len=0 …
11	2020-01-27 18:05:03.7596134…	10.0.2.15	10.0.2.8	TCP	60	44 → 51119 [RST, ACK] Seq=0 Ack=2977847784 Win…
12	2020-01-27 18:05:03.7596196…	10.0.2.8	10.0.2.15	TCP	74	42177 → 44 [SYN] Seq=2344002327 Win=5840 Len=0…
13	2020-01-27 18:05:03.7596245…	10.0.2.8	10.0.2.15	TCP	74	51729 → 44 [SYN] Seq=1695364516 Win=5840 Len=0…
14	2020-01-27 18:05:03.7596290…	10.0.2.15	10.0.2.8	TCP	60	44 → 55009 [RST, ACK] Seq=0 Ack=823720717 Win=…

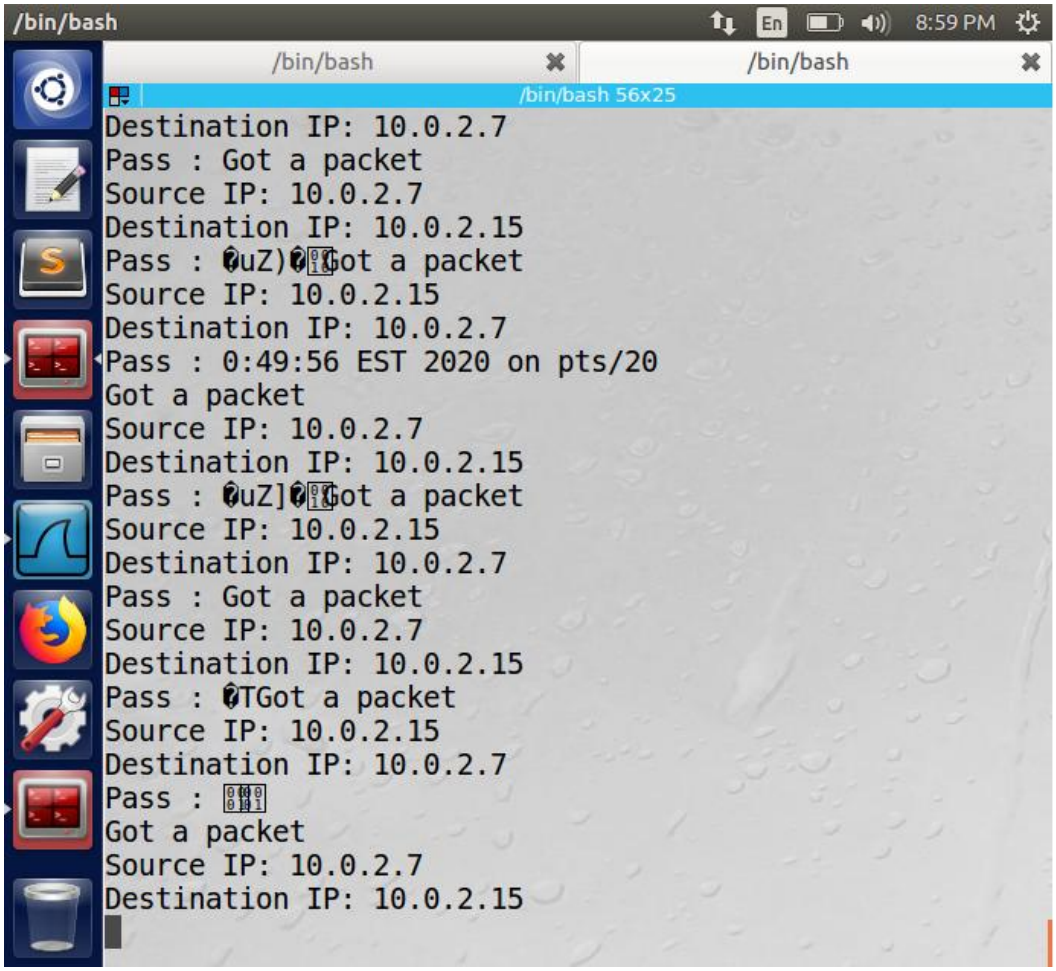

/bin/bash 52x25
[01/27/20]seed@VM:~$ sudo ./sniff
Got a packet
Source IP: 10.0.2.8
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.8
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.8
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.8
Destination IP: 10.0.2.15
Got a packet
Source IP: 10.0.2.8
Destination IP: 10.0.2.15

Task 2.1C: Sniffing Passwords. Please show how you can use your sniffer program to capture the pass-word when somebody is using `telnet` on the network that you are monitoring. You may need to modify your sniffer code to print out the data part of a captured TCP packet (`telnet` uses TCP). It is acceptable if you print out the entire data part, and then manually mark where the password (or part of it) is.

I have modified the sniffer program to capture data on TCP port 23. TH_OFF() in the TCP header provides the start of the offset of the payload, we then get start of the memory through packet, followed by sizeof (struct ethheader) which gives Ethernet header size, then ip header length and tcp offset is calculated in bytes by multiplying by 4. I am trying to store all these data in a char buffer and trying to print it as a string. Unfortunately, the output is not clear due to char buffer and printing characters as %s is not good idea plus we have to consider the screen resolution to print these values, but we can see some legible data from the machine we are telnetting from.



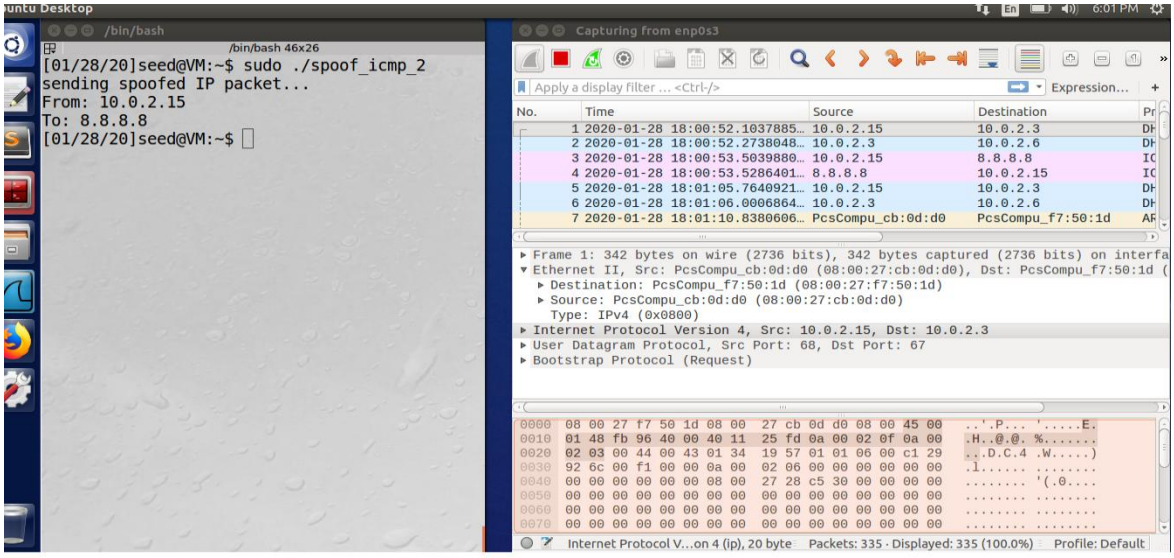



/bin/bash
En
8:59 PM
/bin/bash
/bin/bash
/bin/bash 56x25
Destination IP: 10.0.2.7
Pass : Got a packet
Source IP: 10.0.2.7
Destination IP: 10.0.2.15
Pass : uZ)Got a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.7
Pass : 0:49:56 EST 2020 on pts/20
Got a packet
Source IP: 10.0.2.7
Destination IP: 10.0.2.15
Pass : uZ]Got a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.7
Pass : Got a packet
Source IP: 10.0.2.7
Destination IP: 10.0.2.15
Pass : TGot a packet
Source IP: 10.0.2.15
Destination IP: 10.0.2.7
Pass :
Got a packet
Source IP: 10.0.2.7
Destination IP: 10.0.2.15

Task 2.2A: Write a spoofing program. Please write your own packet spoofing program in C. You need to provide evidences (e.g., Wireshark packet trace) to show that your program successfully sends out spoofed IP packets.

Task 2.2B: Spoof an ICMP Echo Request. Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alive). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.

I am showing both tasks in this module.




```
printf ("From: %s\n", inet_ntoa(ip->iph_sourceip));
printf ("To: %s\n", inet_ntoa(ip->iph_destip));
}
close(sock);
}

int main()
{
char buffer[1500];
memset(buffer,0,1500);
struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
icmp->icmp_type=8;
icmp->icmp_chksum=0;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
struct ipheader *ip = (struct ipheader *) buffer;
ip->iph_ver=4;
ip->iph_ihl=5;
ip->iph_ttl=20;
ip->iph_sourceip.s_addr=inet_addr("10.0.2.15");
ip->iph_destip.s_addr=inet_addr("8.8.8.8");
ip->iph_protocol=IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader ));
send_raw_ip_packet(ip);
return 0;
```

I am spoofing an ICMP request on behalf of 10.0.2.15 towards 8.8.8.8 (google DNS) which is a remote machine in the Internet, and I am doing this using raw sockets by manually defining values, as seen in the wireshark ouput, we are getting a response from 8.8.8.8.

Questions. Please answer the following questions.

•Question 4.Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

No, we cannot set the packet length to any arbitrary value, while spoofing a lot of factors depend on packet length for computation, an arbitrary packet length might cause memory issues and could result in segmentation fault.

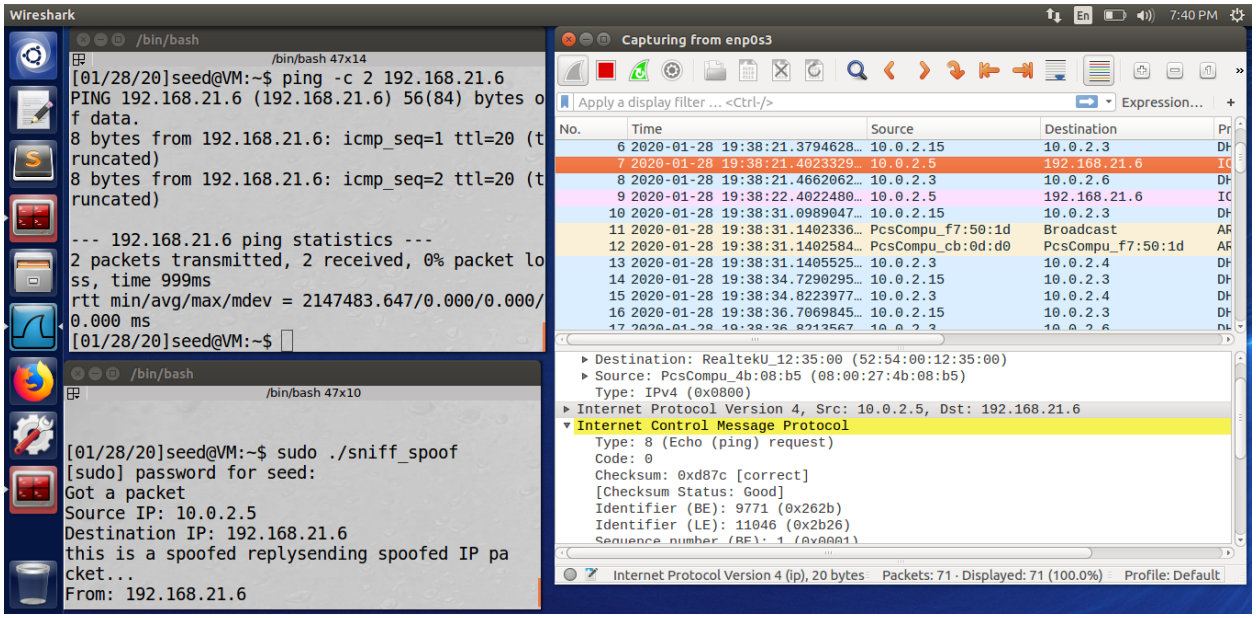
•Question 5.Using the raw socket programming, do you have to calculate the checksum for the IP header?

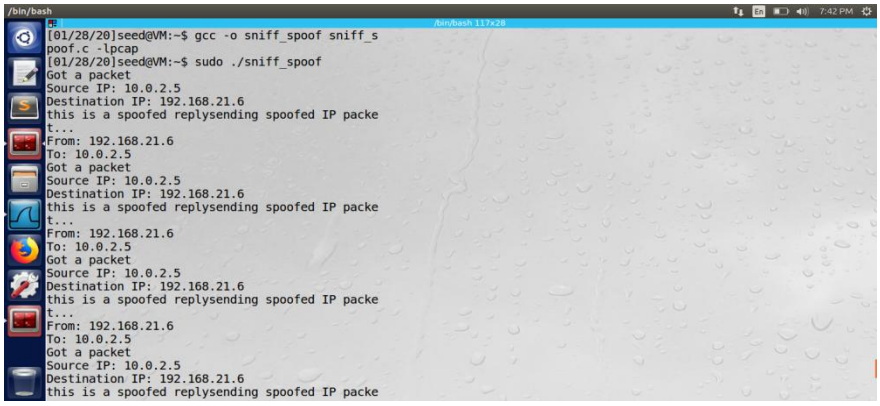
Yes, computation of checksum is necessary in socket programming, if checksum is not computed or is incorrect, it will result in malformed packet, and we will not receive a response from destination.

•Question 6.Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

On a moral side raw sockets have the capability to bypass protocols like tcp, if access falls to an unauthorized user then sensitive information can be processed, to prevent this from happening there is a requirement that only users with root privilege can run these programs, without root access kernel will not allow users or programs to bind to the socket, if programs are still run without root access, then it will result in segmentation fault.

3.3 Task 2.3: Sniff and then Spoof In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two VMs on the same LAN. From VM A, 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 VM B, 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, regard-less of whether machine X is alive or not, the `ping` program will always receive a reply, indicating that X is alive. You need to write such a program in C, and include screenshots in your report to show that your program works. Please also attach the code (with adequate amount of comments) in your report.





In this task, I have included code of both sniff and spoof tasks, I am trying to ping an unreachable IP 192.168.21.6 from VM 10.0.2.5, my code is able to sniff the ICMP request and send out spoofed ICMP packets. I have included code of few methods for reference.

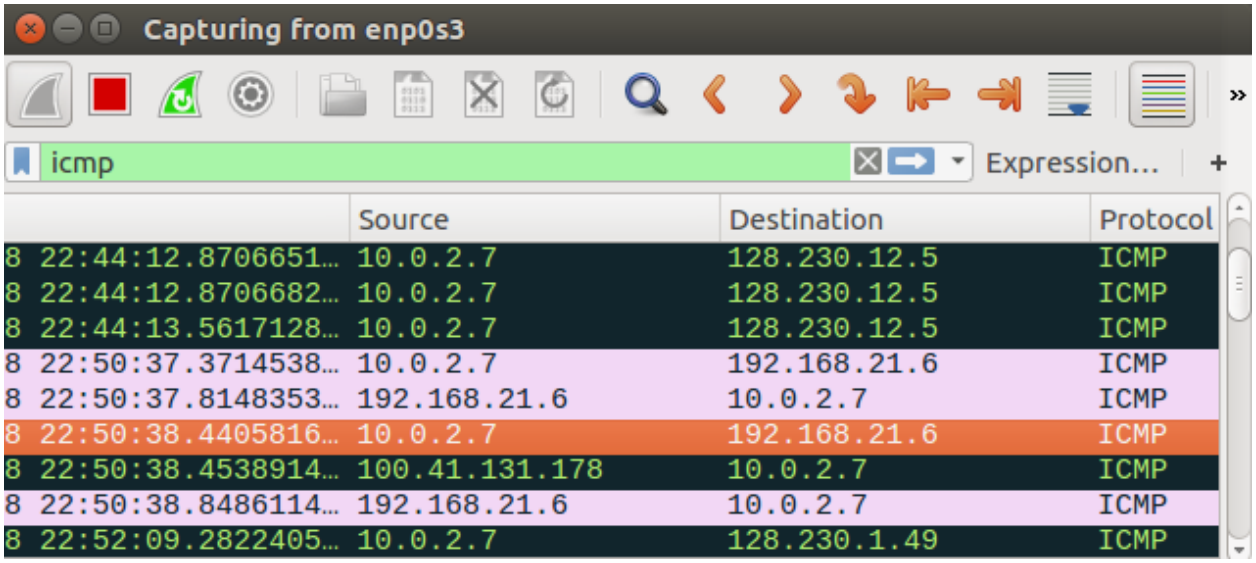
```
void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{
struct ethheader *eth = (struct ethheader *)packet;
struct ipheader *ip = (struct ipheader *) (packet + sizeof(struct ethheader));
printf ("Got a packet \n");
printf ("Source IP: %s\n", inet_ntoa(ip->iph_sourceip));
printf ("Destination IP: %s\n", inet_ntoa(ip->iph_destip));
if(ip->iph_protocol == IPPROTO_ICMP)
{
spoof_icmp(ip);
}
}
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "icmp and src host 10.0.2.5";
bpf_u_int32 net;
bpf_u_int32 mask;
handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);
pcap_compile(handle, &fp, filter_exp, 1, net);
pcap_setfilter(handle, &fp);
pcap_loop(handle, -1, got_packet, NULL);
```

```
int spoof_icmp(struct ipheader *ip)
{
char buffer[1500];
memset(buffer,0,1500);
struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
memset((char*)buffer,0,1500);
memcpy((char *)buffer,ip,ntohs(ip->iph_len));
struct ipheader * newip = (struct ipheader *) buffer;
printf ("this is a spoofed reply");
icmp->icmp_type=0;
icmp->icmp_chksum=0;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
newip->iph_ver=4;
newip->iph_ihl=5;
newip->iph_ttl=20;
newip->iph_sourceip=ip->iph_destip;
newip->iph_destip=ip->iph_sourceip;
newip->iph_protocol=IPPROTO_ICMP;
newip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader ));
send_raw_ip_packet(newip);
return 0;
}
```

Snapshots when tried from different VM .

```
[01/28/20]seed@VM:~$
[01/28/20]seed@VM:~$ ping  -c 2 192.168.21.6
PING 192.168.21.6 (192.168.21.6) 56(84) bytes of
data.
8 bytes from 192.168.21.6: icmp_seq=1 ttl=20 (tru
ncated)
From 100.41.131.178 icmp_seq=2 Destination Net Un
reachable

--- 192.168.21.6 ping statistics ---
2 packets transmitted, 1 received, +1 errors, 50%
 packet loss, time 1069ms
rtt min/avg/max/mdev = 2147483.647/0.000/0.000/0.
000 ms
[01/28/20]seed@VM:~$
```



```
[01/28/20]seed@VM:~$ sudo ./sniff_spoof
Got a packet
Source IP: 10.0.2.7
Destination IP: 192.168.21.6
this is a spoofed replysending spoofed IP packet...
From: 192.168.21.6
To: 10.0.2.7
Got a packet
Source IP: 10.0.2.7
Destination IP: 192.168.21.6
this is a spoofed replysending spoofed IP packet...
```

```
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "icmp and src host 10.0.2.7";
bpf_u_int32 net;
bpf_u_int32 mask;
handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf
);
pcap_compile(handle, &fp, filter_exp, 1, net);
pcap_setfilter(handle, &fp);
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}
```