

Task 1.1A

Using scapy the task was to make a sniffer that can sniff packets and then we ran it and pinged google and started receiving packets from google.

Task 1.1B

By using scapy, we captured ICMP packets from google by pinging google.com, after running the python script.

```
#!/usr/bin/python3

from scapy.all import *

#a = IP()
#a.show()

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp',prn=print_pkt)
```

```
ihl      = 5
tos      = 0x8
len      = 84
id       = 79
flags    =
frag     = 0
ttl      = 115
proto    = icmp
checksum = 0x8bdc
src      = 172.217.2.142
dst      = 10.0.2.15
options  \
###[ ICMP ]###
  type    = echo-reply
  code    = 0
  checksum = 0x91ad
  id      = 0x94a
  seq     = 0x4b
###[ Raw ]###
  load    = '\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
! "%$%&'()*+,-./01234567'

64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=61
ttl=115 time=22.4 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=62
ttl=115 time=21.7 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=63
ttl=115 time=19.9 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=64
ttl=115 time=21.7 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=65
ttl=115 time=23.7 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=66
ttl=115 time=29.9 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=67
ttl=115 time=23.8 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=68
ttl=115 time=19.5 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=69
ttl=115 time=25.7 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=70
ttl=115 time=20.1 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=71
ttl=115 time=21.4 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=72
ttl=115 time=23.4 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=73
ttl=115 time=16.8 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=74
ttl=115 time=19.6 ms
64 bytes from yyz08s14-in-f142.1e100.net (172.217.2.142): icmp_seq=75
ttl=115 time=19.6 ms
```

We captured TCP packets by using the telnet command to connect to another VM.

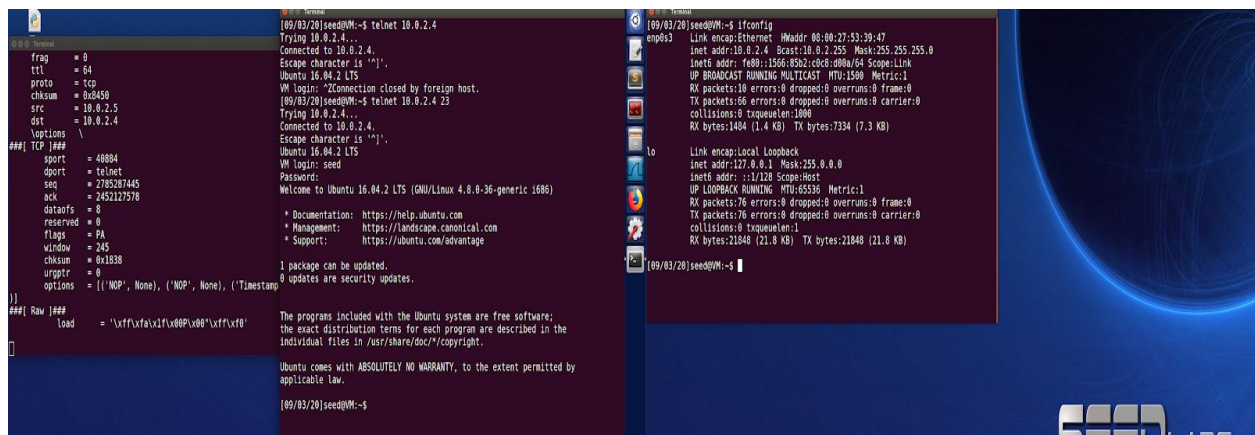
```
#!/usr/bin/python3

from scapy.all import *

#a = IP()
#a.show()

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.4 and dst port 23',prn=print_pkt)
```



```
[09/03/20]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: "2Connection closed by foreign host.
[09/03/20]seed@VM:~$ telnet 10.0.2.4 23
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic 1686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

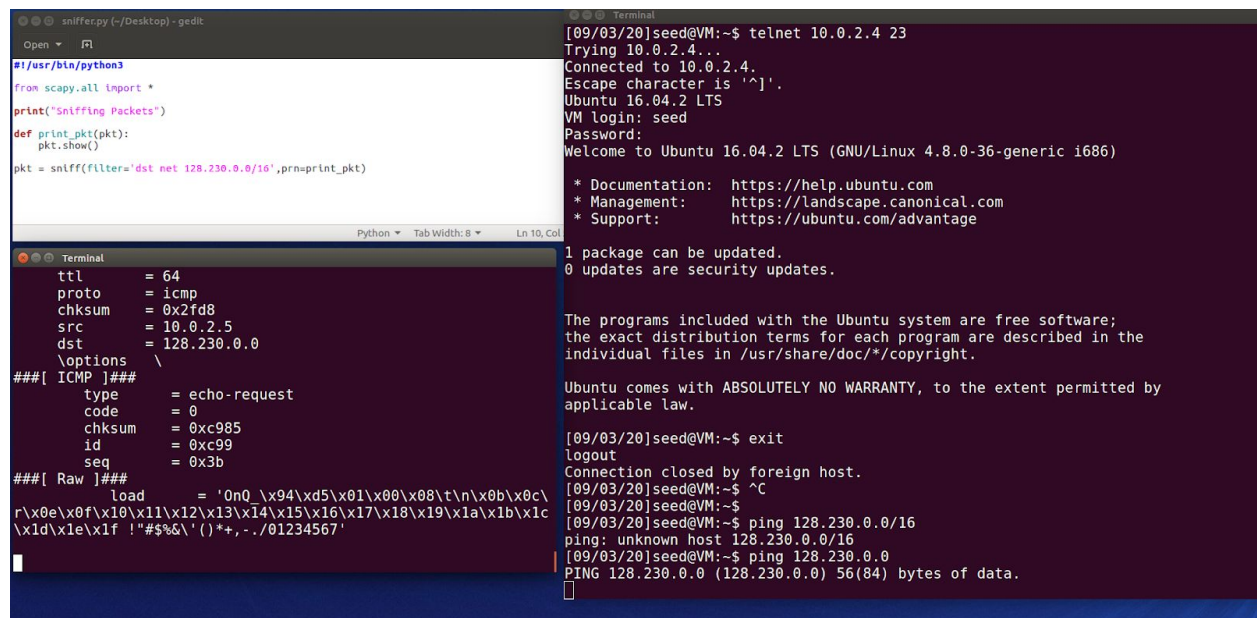
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

[09/03/20]seed@VM:~$

Link encap:Ethernet  HWaddr 08:00:27:53:39:47
inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:10 errors:0 dropped:0 overruns:0 frame:0
TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1494 (1.4 KB)  TX bytes:7334 (7.3 KB)

Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:65536  Metric:1
RX packets:76 errors:0 dropped:0 overruns:0 frame:0
TX packets:76 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:21848 (21.8 KB)  TX bytes:21848 (21.8 KB)
```

We captured packets from a particular subnet the machine is not attached to we captured from 128.230.0.0/16.



```
#!/usr/bin/python3

from scapy.all import *

print("Sniffing Packets")

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='dst net 128.230.0.0/16',prn=print_pkt)
```

```
[09/03/20]seed@VM:~$ telnet 10.0.2.4 23
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic 1686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

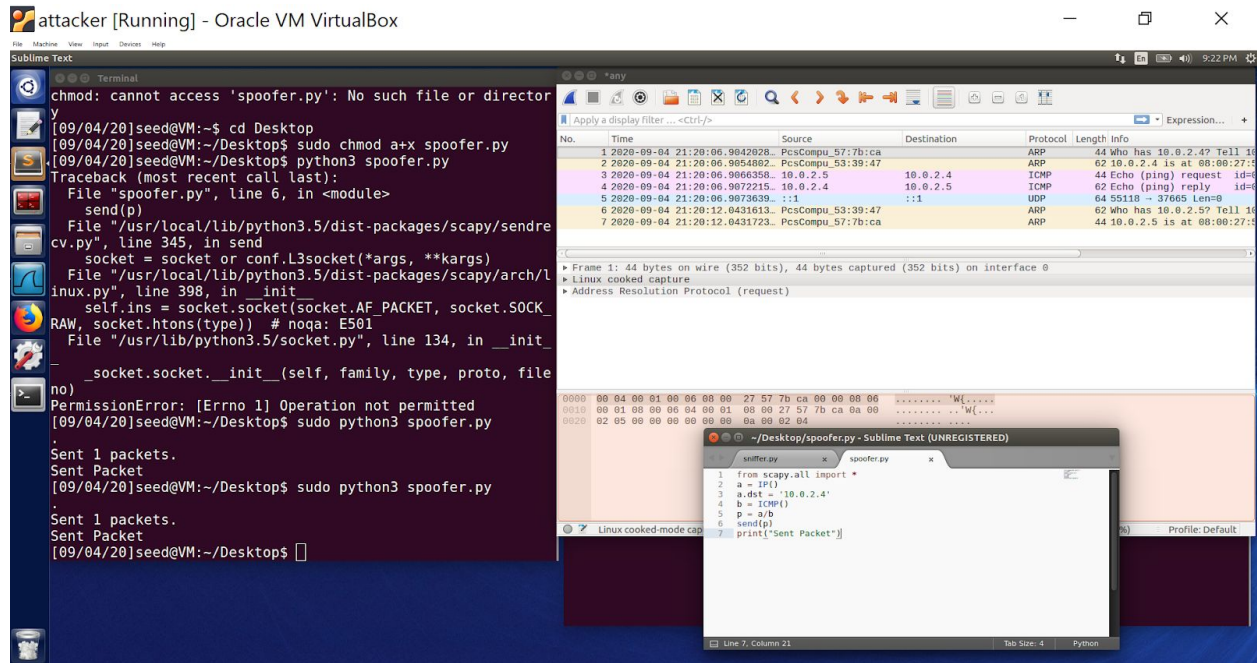
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

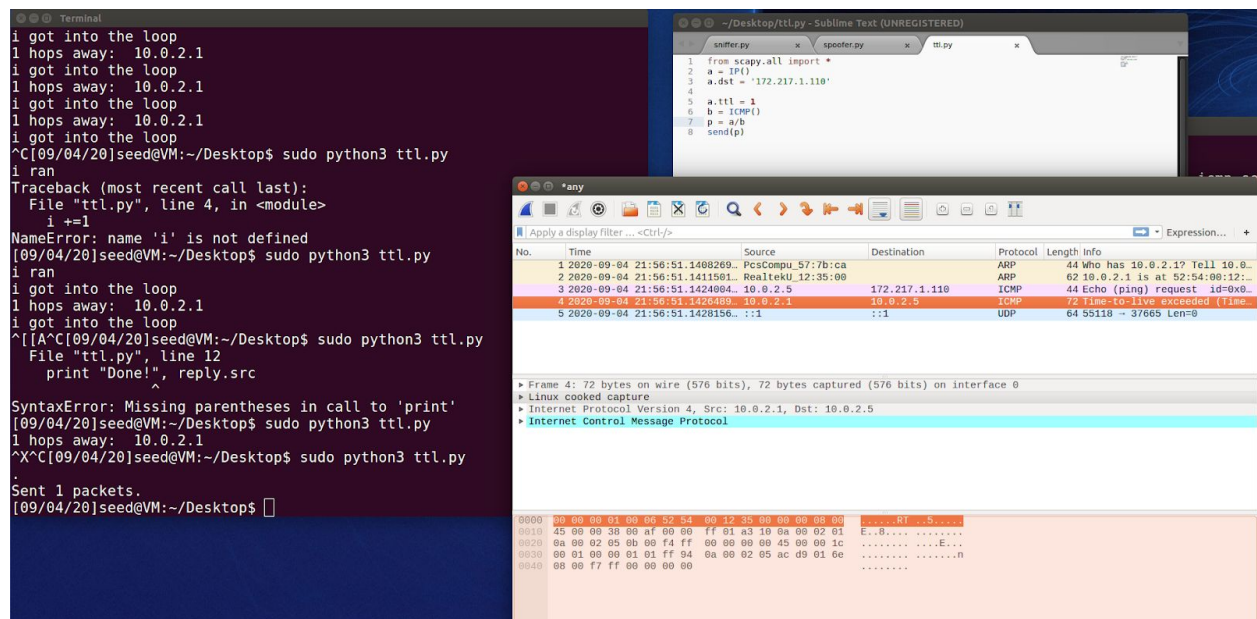
[09/03/20]seed@VM:~$ exit
logout
Connection closed by foreign host.
[09/03/20]seed@VM:~$ ^C
[09/03/20]seed@VM:~$
[09/03/20]seed@VM:~$ ping 128.230.0.0/16
ping: unknown host 128.230.0.0/16
[09/03/20]seed@VM:~$ ping 128.230.0.0
PING 128.230.0.0 (128.230.0.0) 56(84) bytes of data.
```

Task 1.2

We sent an ICMP packet using scapy to the IP of another VM on our network. Using Wireshark, we can see that the other VM received the request and sent back a reply.



Task 1.3: We found the Time-To-Live from our machine to Google by starting from 1 and incrementing the TTL until we received a response back from Google, which was 16.




```
Terminal
1 hops away: 10.0.2.1
i got into the loop
1 hops away: 10.0.2.1
i got into the loop
^C[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
i ran
Traceback (most recent call last):
  File "ttl.py", line 4, in <module>
    i +=1
NameError: name 'i' is not defined
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
i ran
i got into the loop
1 hops away: 10.0.2.1
i got into the loop
^[A^C[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
File "ttl.py", line 12
    print "Done!", reply.src
          ^
SyntaxError: Missing parentheses in call to 'print'
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
1 hops away: 10.0.2.1
^XC[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$
```

~/Desktop/ttl.py - Sublime Text (UNREGISTERED)

```
sniffer.py x spoofer.py x ttl.py x
1 from scapy.all import *
2 a = IP()
3 a.dst = '172.217.1.110'
4
5 a.ttl = 5
6 b = ICMP()
7 p = a/b
8 send(p)
```

*any

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09-04 22:54:31.700189...	:::1	:::1	UDP	64	55118 → 37665 Len=0
2	2020-09-04 22:54:32.816184...	PcsCompu_57:7b:ca		ARP	44	Who has 10.0.2.1? Tell 10.0...
3	2020-09-04 22:54:32.8175723...	RealtekU_12:35:00		ARP	62	10.0.2.1 is at 52:54:00:12:...
4	2020-09-04 22:54:32.8192622...	10.0.2.5	172.217.1.110	ICMP	44	Echo (ping) request id=0x0...
5	2020-09-04 22:54:32.8321440...	90.108.89.249	10.0.2.5	ICMP	72	Time-to-live exceeded (Time...

Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0

Linux cooked capture

Internet Protocol Version 6, Src: ::1, Dst: ::1

User Datagram Protocol, Src Port: 55118, Dst Port: 37665

0000 00 00 03 04 00 00 00 00 00 00 00 00 00 00 00 dd
0010 00 67 dd c0 00 00 11 40 00 00 00 00 00 00 00@
0020 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 01 d7 4e 93 21 00 00 00 1bN.I....

```
Terminal
Traceback (most recent call last):
  File "ttl.py", line 4, in <module>
    i +=1
NameError: name 'i' is not defined
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
i ran
i got into the loop
1 hops away: 10.0.2.1
i got into the loop
^[A^C[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
File "ttl.py", line 12
    print "Done!", reply.src
          ^
SyntaxError: Missing parentheses in call to 'print'
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
1 hops away: 10.0.2.1
^XC[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$ sudo python3 ttl.py
.
Sent 1 packets.
[09/04/20]seed@VM:~/Desktop$
```

~/Desktop/ttl.py - Sublime Text (UNREGISTERED)

```
sniffer.py x spoofer.py x ttl.py x
1 from scapy.all import *
2 a = IP()
3 a.dst = '172.217.1.110'
4
5 a.ttl = 16
6 b = ICMP()
7 p = a/b
8 send(p)
```

*any

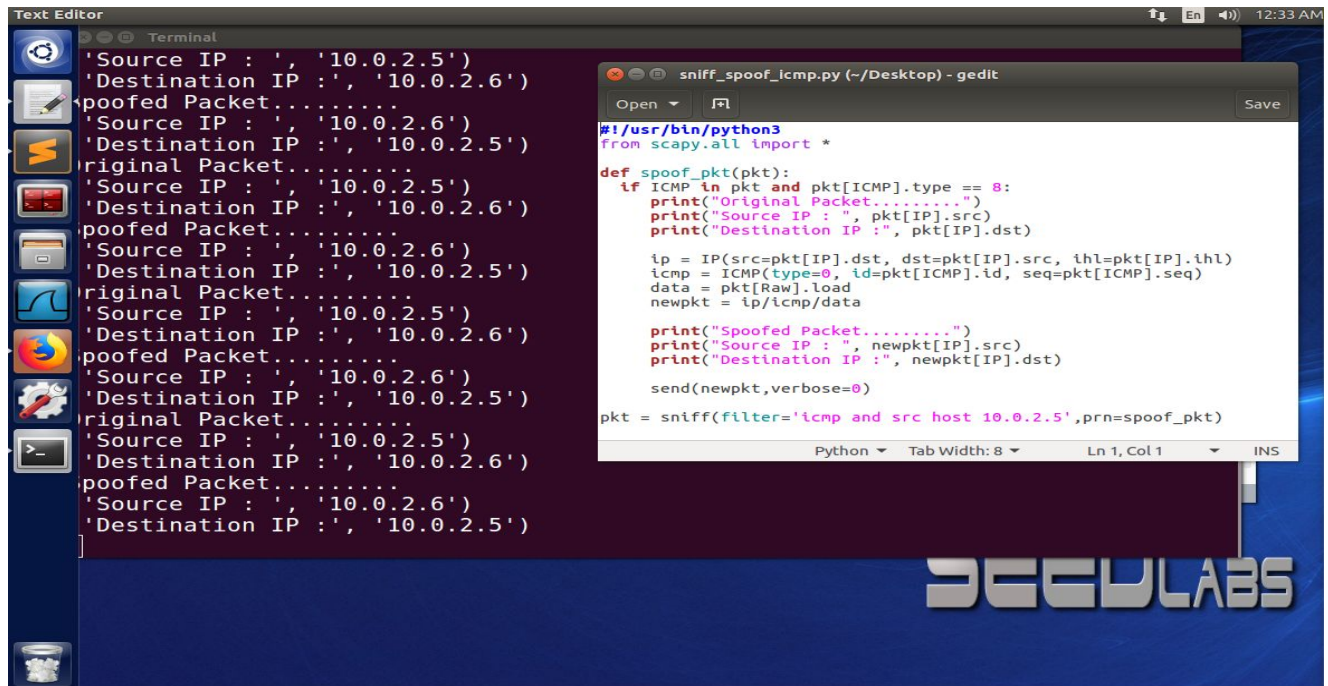
Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09-04 22:55:18.7094786...	PcsCompu_57:7b:ca		ARP	44	Who has 10.0.2.1? Tell 10.0...
2	2020-09-04 22:55:18.7099644...	RealtekU_12:35:00		ARP	62	10.0.2.1 is at 52:54:00:12:...
3	2020-09-04 22:55:18.7110790...	10.0.2.5	172.217.1.110	ICMP	44	Echo (ping) request id=0x0...
4	2020-09-04 22:55:18.7386355...	172.217.1.110	10.0.2.5	ICMP	62	Echo (ping) reply id=0x0...
5	2020-09-04 22:55:18.7388450...	:::1	:::1	UDP	64	55118 → 37665 Len=0

Task 1.4:

We have 3 machines and each with a unique ip address. Machine a would pretend to be machine c and machine b will try to ping machine c. This will work and it does work when machine c is on or off. Machine b would ping machine c but machine a would intercept the ping request by sniffing the network for packets from machine b and then spoof a packet by sending a packet back to b pretending it is c and the reply on b's end looks like its coming from c.

Machine A



```
Text Editor
Terminal
'Source IP : ', '10.0.2.5')
'Destination IP : ', '10.0.2.6')
spoofed Packet.....
'Source IP : ', '10.0.2.6')
'Destination IP : ', '10.0.2.5')
Original Packet.....
'Source IP : ', '10.0.2.5')
'Destination IP : ', '10.0.2.6')
spoofed Packet.....
'Source IP : ', '10.0.2.6')
'Destination IP : ', '10.0.2.5')
Original Packet.....
'Source IP : ', '10.0.2.5')
'Destination IP : ', '10.0.2.6')
spoofed Packet.....
'Source IP : ', '10.0.2.6')
'Destination IP : ', '10.0.2.5')
Original Packet.....
'Source IP : ', '10.0.2.5')
'Destination IP : ', '10.0.2.6')
spoofed Packet.....
'Source IP : ', '10.0.2.6')
'Destination IP : ', '10.0.2.5')

sniff_spoof_icmp.py (~/.Desktop) - gedit
#!/usr/bin/python3
from scapy.all import *

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.....")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP : ", pkt[IP].dst)

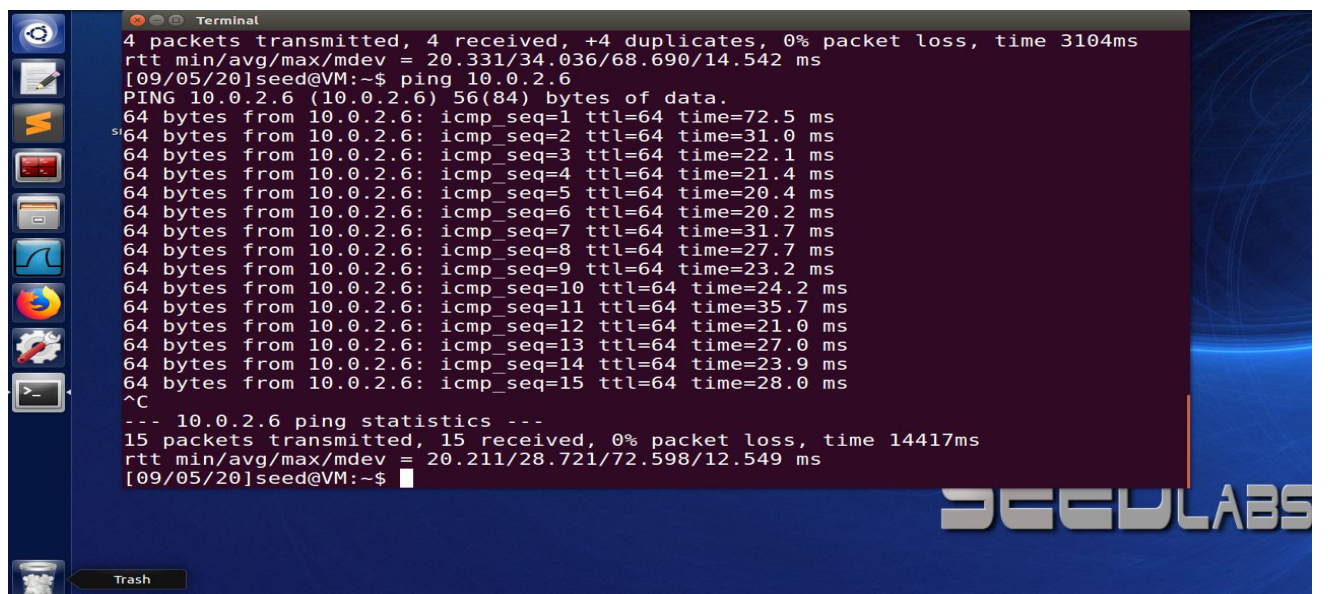
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.....")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP : ", newpkt[IP].dst)

        send(newpkt, verbose=0)

pkt = sniff(filter='icmp and src host 10.0.2.5', prn=spoof_pkt)
```

Machine B



```
Terminal
4 packets transmitted, 4 received, +4 duplicates, 0% packet loss, time 3104ms
rtt min/avg/max/mdev = 20.331/34.036/68.690/14.542 ms
[09/05/20]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=1 ttl=64 time=72.5 ms
64 bytes from 10.0.2.6: icmp_seq=2 ttl=64 time=31.0 ms
64 bytes from 10.0.2.6: icmp_seq=3 ttl=64 time=22.1 ms
64 bytes from 10.0.2.6: icmp_seq=4 ttl=64 time=21.4 ms
64 bytes from 10.0.2.6: icmp_seq=5 ttl=64 time=20.4 ms
64 bytes from 10.0.2.6: icmp_seq=6 ttl=64 time=20.2 ms
64 bytes from 10.0.2.6: icmp_seq=7 ttl=64 time=31.7 ms
64 bytes from 10.0.2.6: icmp_seq=8 ttl=64 time=27.7 ms
64 bytes from 10.0.2.6: icmp_seq=9 ttl=64 time=23.2 ms
64 bytes from 10.0.2.6: icmp_seq=10 ttl=64 time=24.2 ms
64 bytes from 10.0.2.6: icmp_seq=11 ttl=64 time=35.7 ms
64 bytes from 10.0.2.6: icmp_seq=12 ttl=64 time=21.0 ms
64 bytes from 10.0.2.6: icmp_seq=13 ttl=64 time=27.0 ms
64 bytes from 10.0.2.6: icmp_seq=14 ttl=64 time=23.9 ms
64 bytes from 10.0.2.6: icmp_seq=15 ttl=64 time=28.0 ms
^C
--- 10.0.2.6 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14417ms
rtt min/avg/max/mdev = 20.211/28.721/72.598/12.549 ms
[09/05/20]seed@VM:~$
```


Task 2.1

Using the C code provided, we captured ICMP packets by pinging google.com.

```
[09/08/20]seed@VM:~$ sudo ./a.out
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
[09/08/20]seed@VM:~$ ping google.com
PING google.com (172.217.15.206) 56(84) bytes of data.
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=1 ttl=114 tim
e=31.4 ms
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=2 ttl=114 tim
e=28.7 ms
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=3 ttl=114 tim
e=36.3 ms
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=4 ttl=114 tim
e=29.8 ms
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=5 ttl=114 tim
e=32.8 ms
64 bytes from mia09s20-in-f14.1e100.net (172.217.15.206): icmp_seq=6 ttl=114 tim
e=28.1 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5027ms
rtt min/avg/max/mdev = 28.168/31.218/36.365/2.793 ms
[09/08/20]seed@VM:~$
```

Task 2.1A:

Captured packets with ICMP, UDP and TCP protocols by removing the filter. We found the source and the destination address by browsing the seed labs website.

[illegible]

Question 1.

First a sniffer program finds a capture device to capture on and then returns the network number and mask for device then it sniffs then returns the device you're capturing on and compiles filter expression and and the end closes the session.

Question 2.

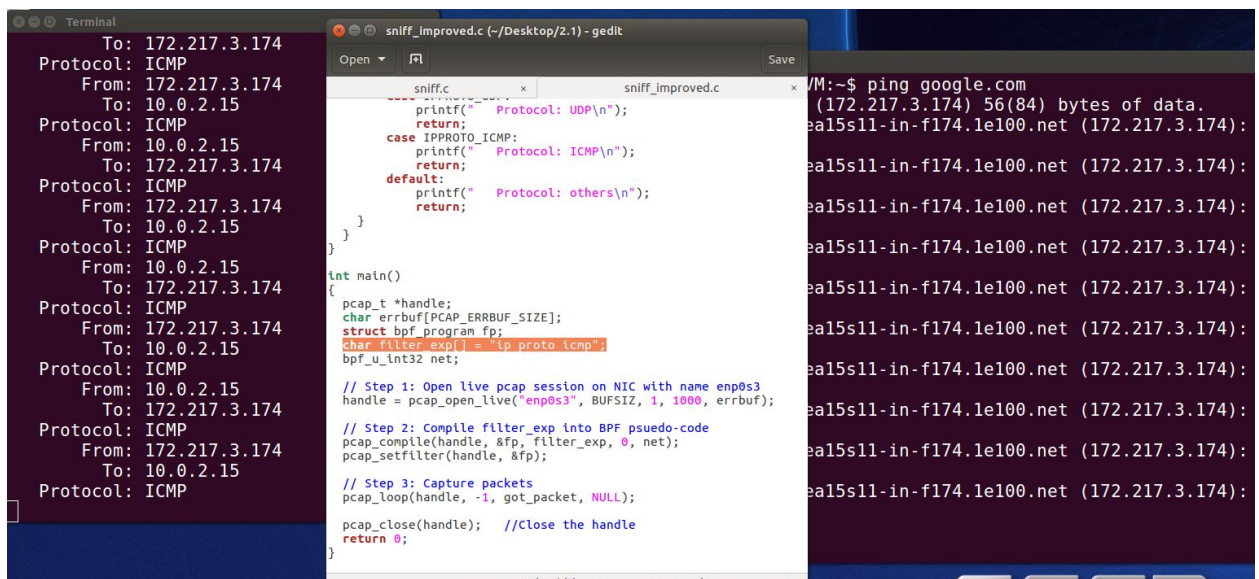
Because to sniff your network device needs to be accessed and only root can do that.

Question 3.

Promiscuous mode allows for network sniffer to get all traffic from a network and not only the traffic only intended to that network controller.

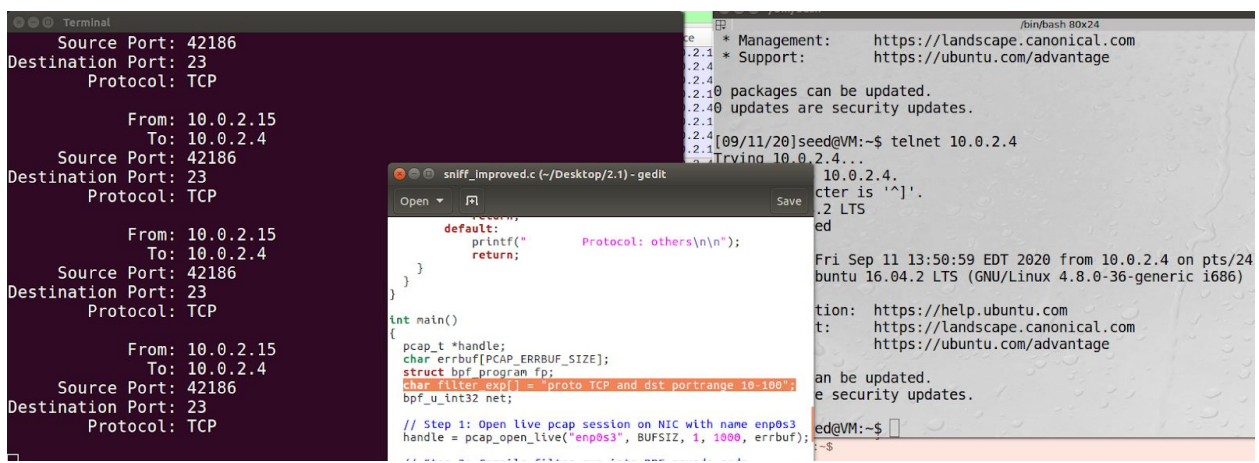
Task 2.1B:

- We set the filter to catch ICMP packets and we did by pinging google.com



The screenshot shows a terminal window on the left displaying captured ICMP traffic. The traffic consists of multiple ping requests from 10.0.2.15 to 172.217.3.174 (google.com). In the center, a code editor shows the source code for 'sniff_improved.c'. The code includes a filter expression 'ip proto icmp' and steps for opening a pcap session, compiling the filter, and capturing packets. On the right, another terminal window shows the command 'ping google.com' being executed, resulting in a successful ping with 56(84) bytes of data.

- We set the filter to catch TCP packets and the destination port between 10 and 100. Then we used the telnet command to connect to another VM and captured packets with the destination port between 10 and 100.



The screenshot shows a terminal window on the left displaying captured TCP traffic. The traffic consists of multiple telnet connections from 10.0.2.15 to 10.0.2.4. In the center, a code editor shows the source code for 'sniff_improved.c'. The code includes a filter expression 'proto TCP and dst portrange 10-100' and steps for opening a pcap session, compiling the filter, and capturing packets. On the right, another terminal window shows the command 'telnet 10.0.2.4' being executed, resulting in a connection to the target VM.

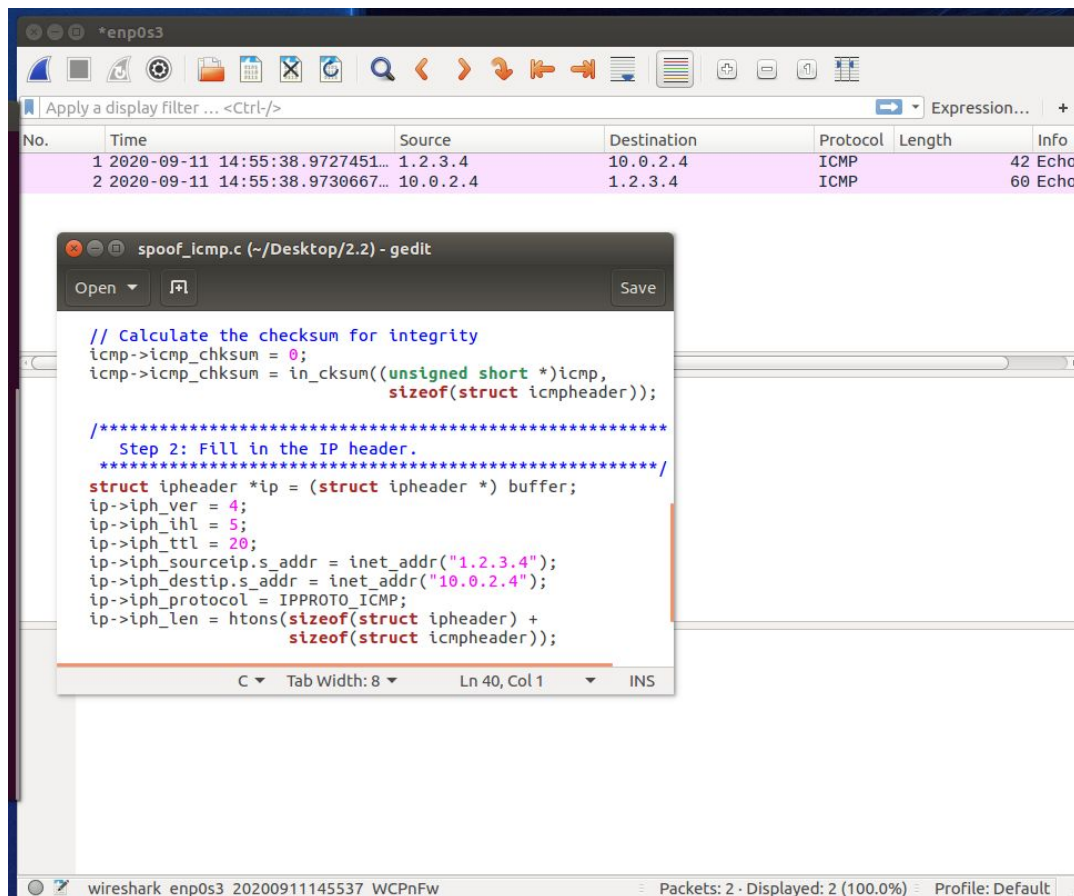
Task 2.1C:

We added the code to our sniffing program to capture data from the packets. We were able to capture the password when we connected to another VM using telnet.

```
Terminal
Srce Port: 42208
Dest Port: 23
Protocol: TCP
.(.>...g.....z.....K.d
Sniffed Packet....
    From: 10.0.2.5
    To: 10.0.2.6
Srce Port: 42208
Dest Port: 23
Protocol: TCP
.(.?...g.....L.e
Sniffed Packet....
    From: 10.0.2.5
    To: 10.0.2.6
Srce Port: 42208
Dest Port: 23
Protocol: TCP
.(.@...g.....A..L6e
Sniffed Packet....
    From: 10.0.2.5
    To: 10.0.2.6
Srce Port: 42208
Dest Port: 23
Protocol: TCP
.(.A...g.....S.....t..L^s
Sniffed Packet....
    From: 10.0.2.5
    To: 10.0.2.6
```


Task 2.2A/2.1B

We sent a spoofed ICMP packet by changing the source IP to 1.2.3.4 and used the IP for the other VM. Using Wireshark, we can see that it successfully sent the packet and received a reply.



Question 4:

The total size of an IPv4 packet is 65535 bytes; you can't make the packet bigger, can't manipulate the header for udp/tcp.

Question 5:

The kernel builds the packet including the checksum for the data. But in raw socket programming you have to calculate the checksum for every data packet sent.

Question 6:

Raw packets need to use protocols which are part of the kernel and the usermode won't be able to access some of those protocols needed in raw packet creation therefore root is needed.