References
https://linuxize.com/post/linux-ping-command/
https://www.tcpdump.org/manpages/pcap_open_offline.3pcap.html **- Task 1**
http://www.cplusplus.com/forum/unices/168391/ **- for the checksum function**
https://people.engr.tamu.edu/guofei/csce465/rawip.txt **- for task 2/3**
http://www.tenouk.com/ Module43a.html **- 2/3**

HW1

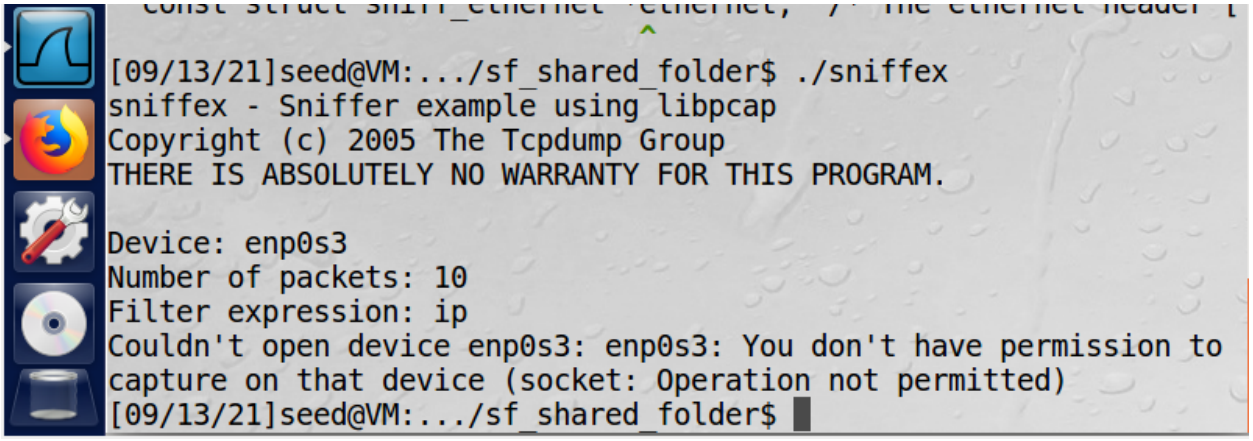
*All code is located in the zip file

Sniffex.c is for task 1
Spoofing.c is for task 2
Sniffandspoof.c for task 3

Task 1: Writing a Packet Sniffing Program
Compiling and Running sniffex.c:
Problem 1:

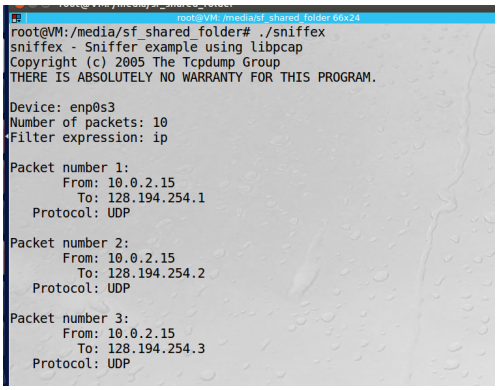
1. First, find the capture device by using pcap_lookup command
2. Second, is the pcap_open_live command which creates a handle to open the device we are watching.
3. pcapc_datalink checks to see if the data is being read
4. Pcap_compile, compiles and checks to see if the filter expression is correct
5. Pcap_setfiler applies the filter expression
6. Pcap_loop sets the callback function to loop through the # of packets set.

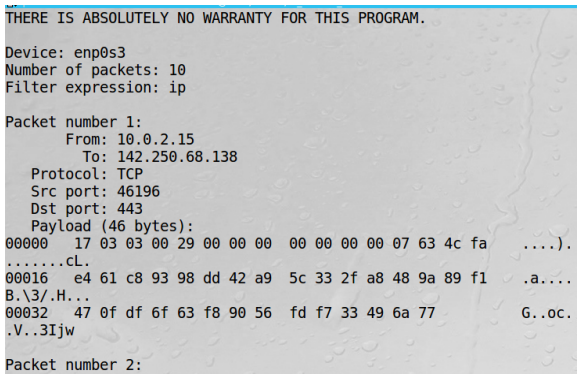
Problem 2:



Root privilege gives you access to all commands and files. So without root privileges, we wouldn't be able to access the socket which is what we see in the first picture when I compiled and ran the code. The program fails at the pcap_open_live call, this is when the program is opening the device it will sniff on.

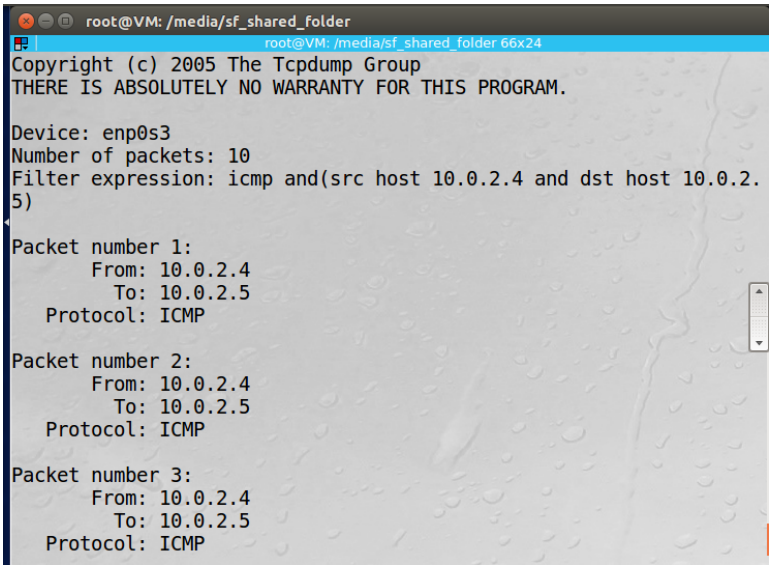
Problem 3:





When promiscuous mode is on, the program displays a limited amount of information about the packet as seen in the first picture. When it is off the program displays that same information as before as well as the payload with contains the data. To turn promiscuous mode on and off, you change the promisc int in the pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf) call. The third value is the promiscuous integer which determines when promiscuous mode is is on or off. So 1 is on and 0 is off.

Problem 4

- ICMP packets between two specific hosts
  - 

- ```
  Packet number 4:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

  Packet number 5:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

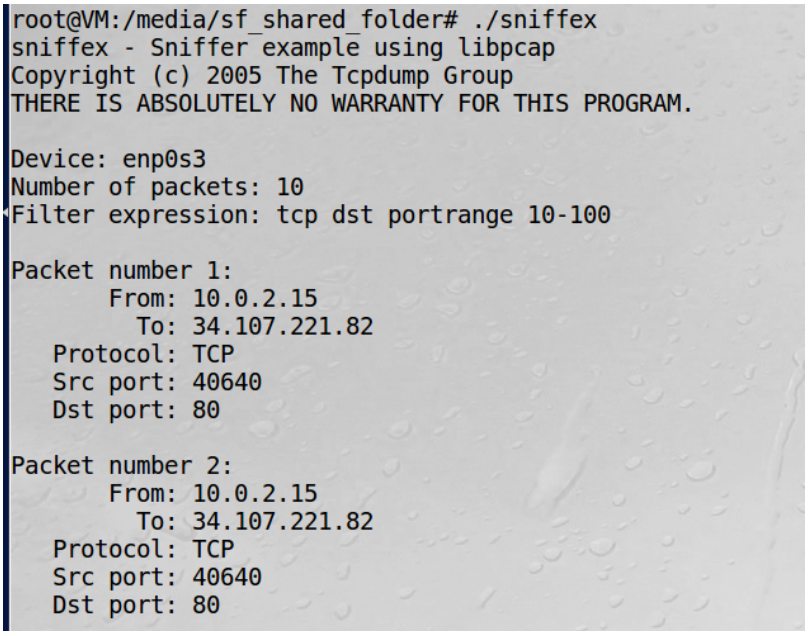
  Packet number 6:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

  Packet number 7:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP
  ```
- ```
  Packet number 8:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

  Packet number 9:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

  Packet number 10:
        From: 10.0.2.4
          To: 10.0.2.5
    Protocol: ICMP

  Capture complete.
  root@VM:/media/sf_shared_folder#
  ```
- The picture below shows me using the ping command to intercept the packets
- ```
  [09/16/21]seed@VM:.../sf_shared_folder$ ping 10.0.2.4
  PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
  64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=1.40 ms
  64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=1.68 ms
  64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.931 ms
  64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=1.17 ms
  64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=0.887 ms
  64 bytes from 10.0.2.4: icmp_seq=6 ttl=64 time=0.979 ms
  64 bytes from 10.0.2.4: icmp_seq=7 ttl=64 time=1.03 ms
  64 bytes from 10.0.2.4: icmp_seq=8 ttl=64 time=1.13 ms
  64 bytes from 10.0.2.4: icmp_seq=9 ttl=64 time=1.64 ms
  64 bytes from 10.0.2.4: icmp_seq=10 ttl=64 time=1.91 ms
  64 bytes from 10.0.2.4: icmp_seq=11 ttl=64 time=2.53 ms
  64 bytes from 10.0.2.4: icmp_seq=12 ttl=64 time=1.55 ms
  ^C
  --- 10.0.2.4 ping statistics ---
  12 packets transmitted, 12 received, 0% packet loss, time 11017ms
  rtt min/avg/max/mdev = 0.887/1.406/2.534/0.469 ms
  [09/16/21]seed@VM:.../sf_shared_folder$
  ```
  ```

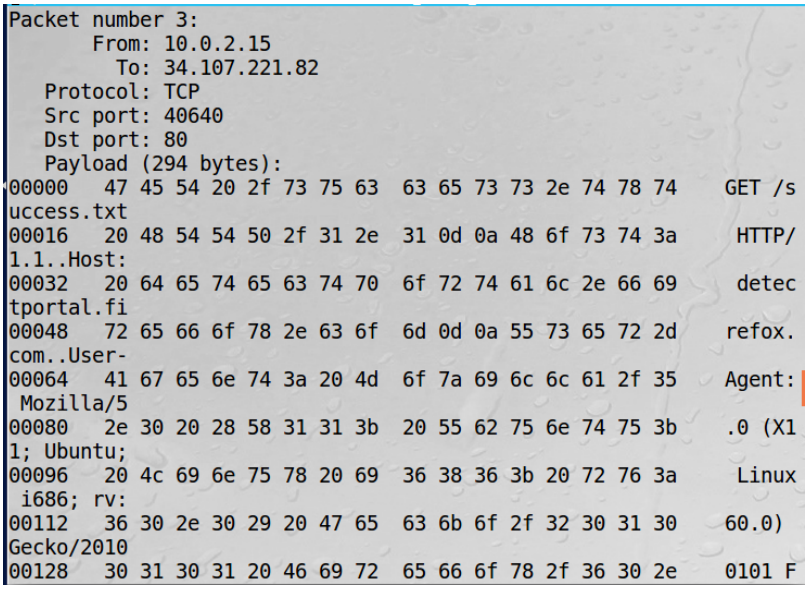
- TCP packets that have a destination port range from port 10 - 100. As you can see below all the dst ports are 80 which are within the range and the Protocol for each packet is tcp as specified in the filter expression
    - 

```
root@VM:/media/sf_shared_folder# ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

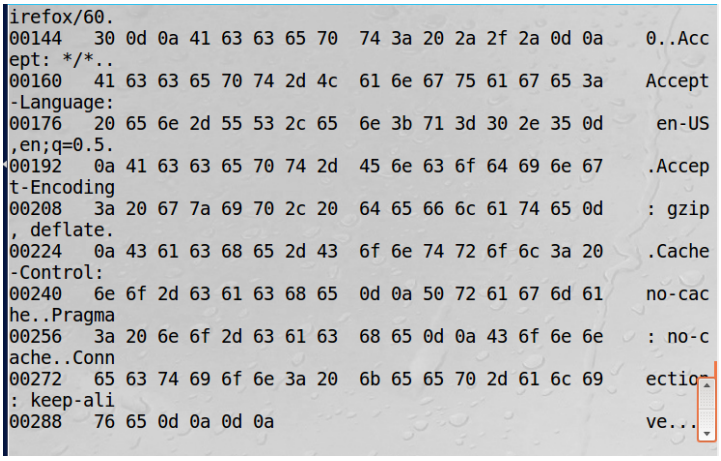
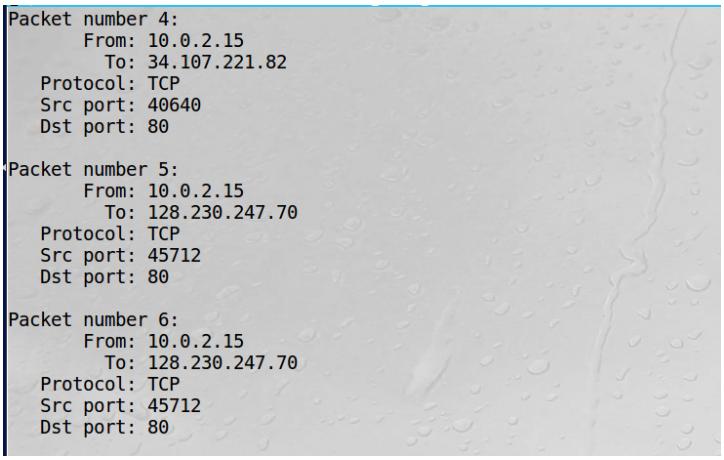
Device: enp0s3
Number of packets: 10
Filter expression: tcp dst portrange 10-100

Packet number 1:
       From: 10.0.2.15
         To: 34.107.221.82
   Protocol: TCP
   Src port: 40640
   Dst port: 80

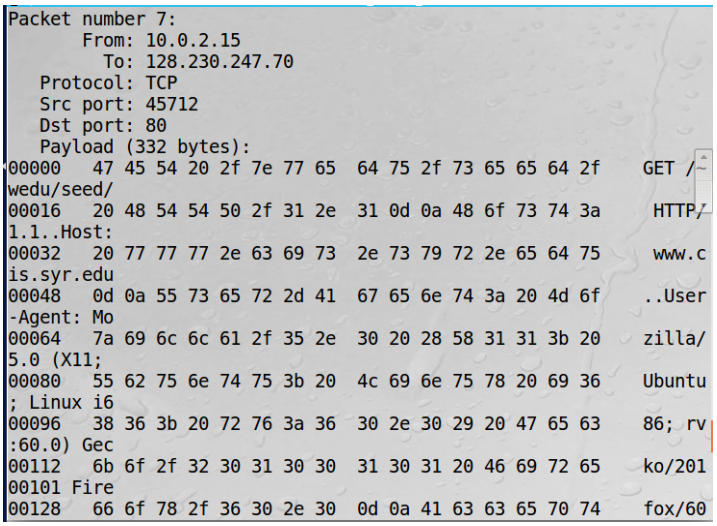
Packet number 2:
       From: 10.0.2.15
         To: 34.107.221.82
   Protocol: TCP
   Src port: 40640
   Dst port: 80
```

    - 

```
Packet number 3:
       From: 10.0.2.15
         To: 34.107.221.82
   Protocol: TCP
   Src port: 40640
   Dst port: 80
   Payload (294 bytes):
00000   47 45 54 20 2f 73 75 63  63 65 73 73 2e 74 78 74    GET /s
uccess.txt
00016   20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a     HTTP/
1.1..Host:
00032   20 64 65 74 65 63 74 70  6f 72 74 61 6c 2e 66 69     detec
tportal.fi
00048   72 65 66 6f 78 2e 63 6f  6d 0d 0a 55 73 65 72 2d    refox.
com..User-
00064   41 67 65 6e 74 3a 20 4d  6f 7a 69 6c 6c 61 2f 35    Agent:
 Mozilla/5
00080   2e 30 20 28 58 31 31 3b  20 55 62 75 6e 74 75 3b    .0 (X1
1; Ubuntu;
00096   20 4c 69 6e 75 78 20 69  36 38 36 3b 20 72 76 3a     Linux
 i686; rv:
00112   36 30 2e 30 29 20 47 65  63 6b 6f 2f 32 30 31 30    60.0)
Gecko/2010
00128   30 31 30 31 20 46 69 72  65 66 6f 78 2f 36 30 2e    0101 F
```

- 
```
irefox/60.
00144   30 0d 0a 41 63 63 65 70  74 3a 20 2a 2f 2a 0d 0a    0..Acc
ept: */*..
00160   41 63 63 65 70 74 2d 4c  61 6e 67 75 61 67 65 3a    Accept
-Language:
00176   20 65 6e 2d 55 53 2c 65  6e 3b 71 3d 30 2e 35 0d     en-US
,en;q=0.5.
00192   0a 41 63 63 65 70 74 2d  45 6e 63 6f 64 69 6e 67    .Accep
t-Encoding
00208   3a 20 67 7a 69 70 2c 20  64 65 66 6c 61 74 65 0d    : gzip
, deflate.
00224   0a 43 61 63 68 65 2d 43  6f 6e 74 72 6f 6c 3a 20    .Cache
-Control:
00240   6e 6f 2d 63 61 63 68 65  0d 0a 50 72 61 67 6d 61    no-cac
he..Pragma
00256   3a 20 6e 6f 2d 63 61 63  68 65 0d 0a 43 6f 6e 6e    : no-c
ache..Conn
00272   65 63 74 69 6f 6e 3a 20  6b 65 65 70 2d 61 6c 69    ectio
: keep-ali
00288   76 65 0d 0a 0d 0a                                  ve...
```
- 
```
Packet number 4:
       From: 10.0.2.15
         To: 34.107.221.82
   Protocol: TCP
   Src port: 40640
   Dst port: 80

Packet number 5:
       From: 10.0.2.15
         To: 128.230.247.70
   Protocol: TCP
   Src port: 45712
   Dst port: 80

Packet number 6:
       From: 10.0.2.15
         To: 128.230.247.70
   Protocol: TCP
   Src port: 45712
   Dst port: 80
```
- 
```
Packet number 7:
       From: 10.0.2.15
         To: 128.230.247.70
   Protocol: TCP
   Src port: 45712
   Dst port: 80
   Payload (332 bytes):
00000   47 45 54 20 2f 7e 77 65  64 75 2f 73 65 65 64 2f    GET /~
wedu/seed/
00016   20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a     HTTP/
1.1..Host:
00032   20 77 77 77 2e 63 69 73  2e 73 79 72 2e 65 64 75     www.c
is.syr.edu
00048   0d 0a 55 73 65 72 2d 41  67 65 6e 74 3a 20 4d 6f    ..User
-Agent: Mo
00064   7a 69 6c 6c 61 2f 35 2e  30 20 28 58 31 31 3b 20    zilla/
5.0 (X11;
00080   55 62 75 6e 74 75 3b 20  4c 69 6e 75 78 20 69 36    Ubuntu
; Linux i6
00096   38 36 3b 20 72 76 3a 36  30 2e 30 29 20 47 65 63    86; rv
:60.0) Gec
00112   6b 6f 2f 32 30 31 30 30  31 30 31 20 46 69 72 65    ko/201
00101 Fire
00128   66 6f 78 2f 36 30 2e 30  0d 0a 41 63 63 65 70 74    fox/60
```

```
.0..Accept
00144   3a 20 74 65 78 74 2f 68  74 6d 6c 2c 61 70 70 6c    : text
/html,appl
00160   69 63 61 74 69 6f 6e 2f  78 68 74 6d 6c 2b 78 6d    icatio
n/xhtml+xm
00176   6c 2c 61 70 70 6c 69 63  61 74 69 6f 6e 2f 78 6d    l,appl
ication/xm
00192   6c 3b 71 3d 30 2e 39 2c  2a 2f 2a 3b 71 3d 30 2e    l;q=0.
9,*/*;q=0.
00208   38 0d 0a 41 63 63 65 70  74 2d 4c 61 6e 67 75 61    8..Acc
ept-Langua
00224   67 65 3a 20 65 6e 2d 55  53 2c 65 6e 3b 71 3d 30    ge: en
-US,en;q=0
00240   2e 35 0d 0a 41 63 63 65  70 74 2d 45 6e 63 6f 64    .5..Ac
cept-Encod
00256   69 6e 67 3a 20 67 7a 69  70 2c 20 64 65 66 6c 61    ing: g
zip, defla
00272   74 65 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20    te..Co
nnection:
00288   6b 65 65 70 2d 61 6c 69  76 65 0d 0a 55 70 67 72    keep-a
live..Upgr
00304   61 64 65 2d 49 6e 73 65  63 75 72 65 2d 52 65 71    ade-In
secure-Req
00320   75 65 73 74 73 3a 20 31  0d 0a 0d 0a                uests:
```

```
 1....

Packet number 8:
       From: 10.0.2.15
         To: 128.230.247.70
   Protocol: TCP
   Src port: 45712
   Dst port: 80

Packet number 9:
       From: 10.0.2.15
         To: 23.47.52.114
   Protocol: TCP
   Src port: 48756
   Dst port: 80

Packet number 10:
       From: 10.0.2.15
         To: 23.47.52.114
   Protocol: TCP
   Src port: 48758
   Dst port: 80

Capture complete.
```

Problem 5:
Live

```
root@VM:/media/sf_shared_folder# ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 10
Filter expression: tcp port 23

Packet number 1:
       From: 10.0.2.5
         To: 10.0.2.4
   Protocol: TCP
   Src port: 60708
   Dst port: 23
   Payload (1 bytes):
00000   64                                                d

Packet number 2:
       From: 10.0.2.4
         To: 10.0.2.5
   Protocol: TCP
   Src port: 23
   Dst port: 60708
Packet number 3:
       From: 10.0.2.5
         To: 10.0.2.4
   Protocol: TCP
   Src port: 60708
   Dst port: 23
   Payload (1 bytes):
00000   65                                                e

Packet number 4:
       From: 10.0.2.4
         To: 10.0.2.5
   Protocol: TCP
   Src port: 23
   Dst port: 60708

Packet number 5:
       From: 10.0.2.5
         To: 10.0.2.4
   Protocol: TCP
   Src port: 60708
   Dst port: 23
   Payload (1 bytes):
00000   65                                                e

Packet number 6:
       From: 10.0.2.4
         To: 10.0.2.5
   Protocol: TCP
   Src port: 23
   Dst port: 60708

Packet number 7:
       From: 10.0.2.5
         To: 10.0.2.4
   Protocol: TCP
   Src port: 60708
   Dst port: 23
   Payload (1 bytes):
00000   73                                                s

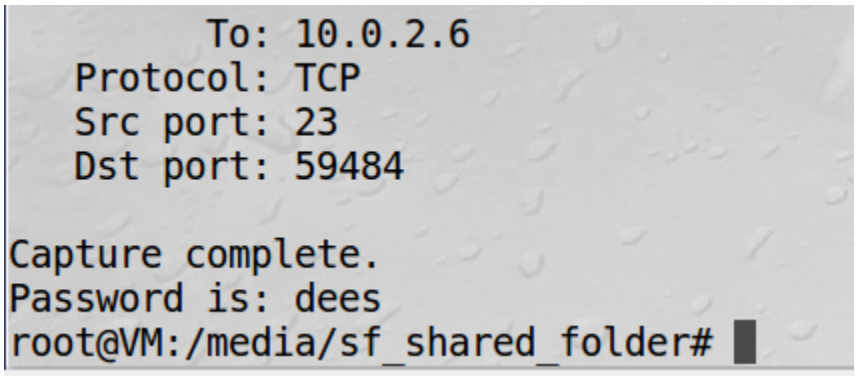
Packet number 8:
       From: 10.0.2.4
         To: 10.0.2.5
   Protocol: TCP
   Src port: 23
   Dst port: 60708

Packet number 9:
       From: 10.0.2.5
         To: 10.0.2.4
   Protocol: TCP
   Src port: 60708
   Dst port: 23
   Payload (2 bytes):
00000   0d 00                                             ..

Packet number 10:
       From: 10.0.2.4
         To: 10.0.2.5
   Protocol: TCP
   Src port: 23
   Dst port: 60708

Capture complete.
root@VM:/media/sf_shared_folder#
```

I took these screenshots before I added code to capture the password on the terminal. But below you can see I have shown the password from more than just printing the packets

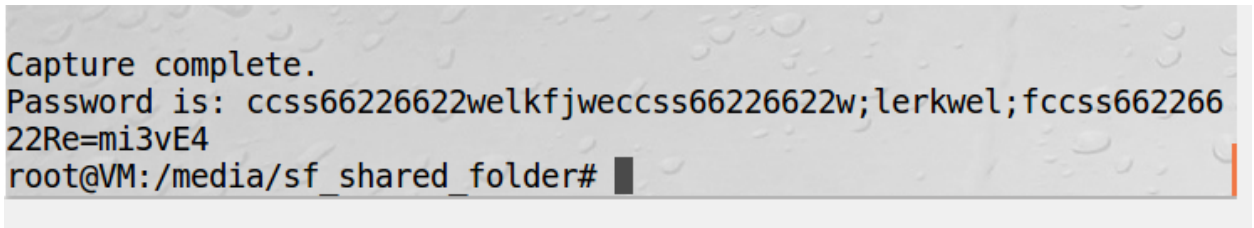


Offline:

The code in sniffex has been changed to account for an offline trace file with a simple if statment.

```
    if(argc == 2){
        offline = 1;
        handle = pcap_open_offline(fname, errbuf);
    } else{
        offline = 0;
        handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
    }
```

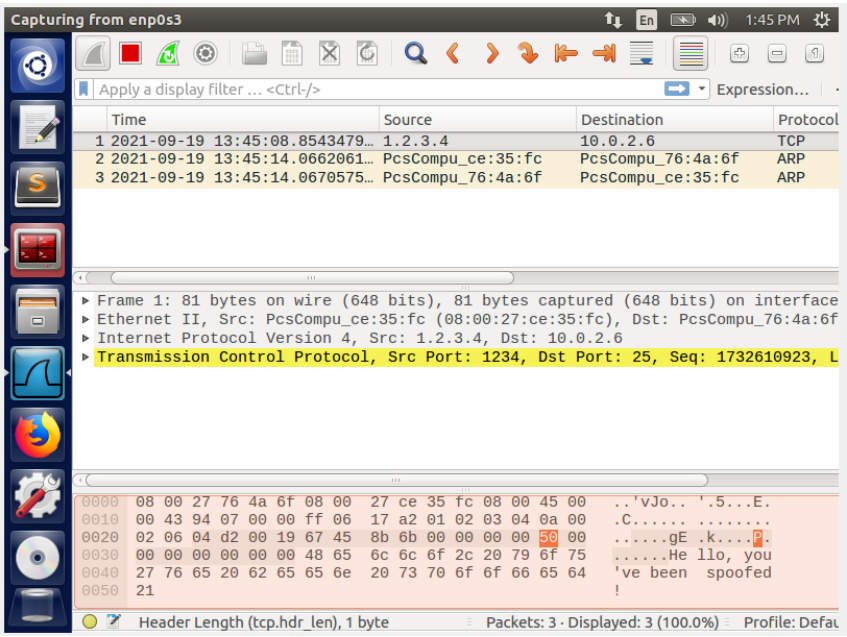
When running the code you have to run it as ./sniffex tfsession.pcap. I am able to print out the packets, so I know the password is Re=mi3vE4. My code below is also printing the failed login attempts, but you can still see the password.



Task 2: Spoofing

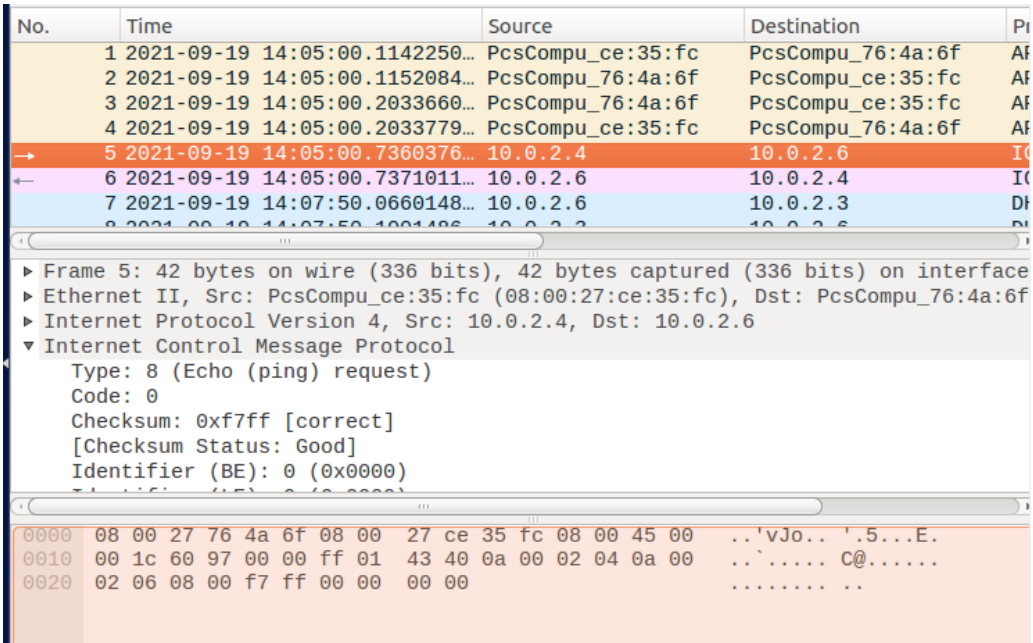
For this task, I create the spoofing program by using the template from reference 4 of the HW1 pdf, also following the instructions. So first I created structs for all the different protocol types and then I used a checksum function from the cpluplus document I referenced. For the main function, I created the raw socket and set the fields necessary. Then I created the ip headers and the tcp/icmp headers for the respective tasks. After that, I used setsockopt as said in the reference and then I sent the spoofed packet out using the send to function. This is seen in my code which is located in the zip.

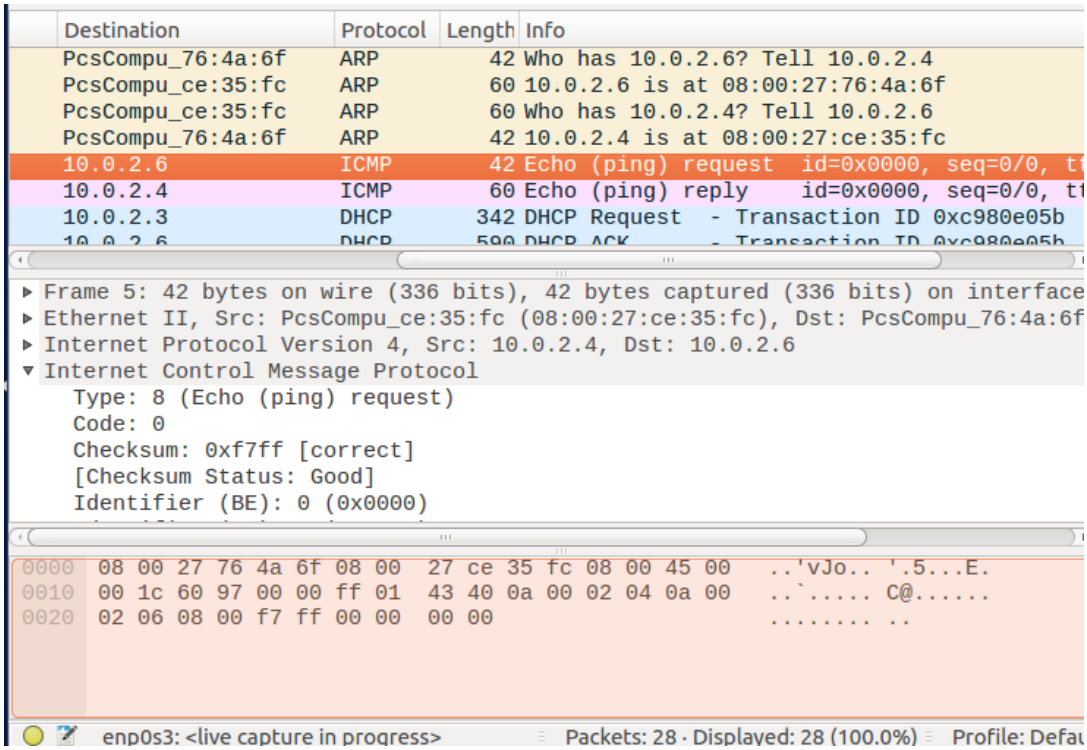
Task 2a



From the picture, you can see that I spoofed a TCP packet to the victim's vm with ip address 10.0.2.6.

<u>Task 2b:</u> *I couldn't make my terminal bigger so I scrolled so you could see all the information in both pictures



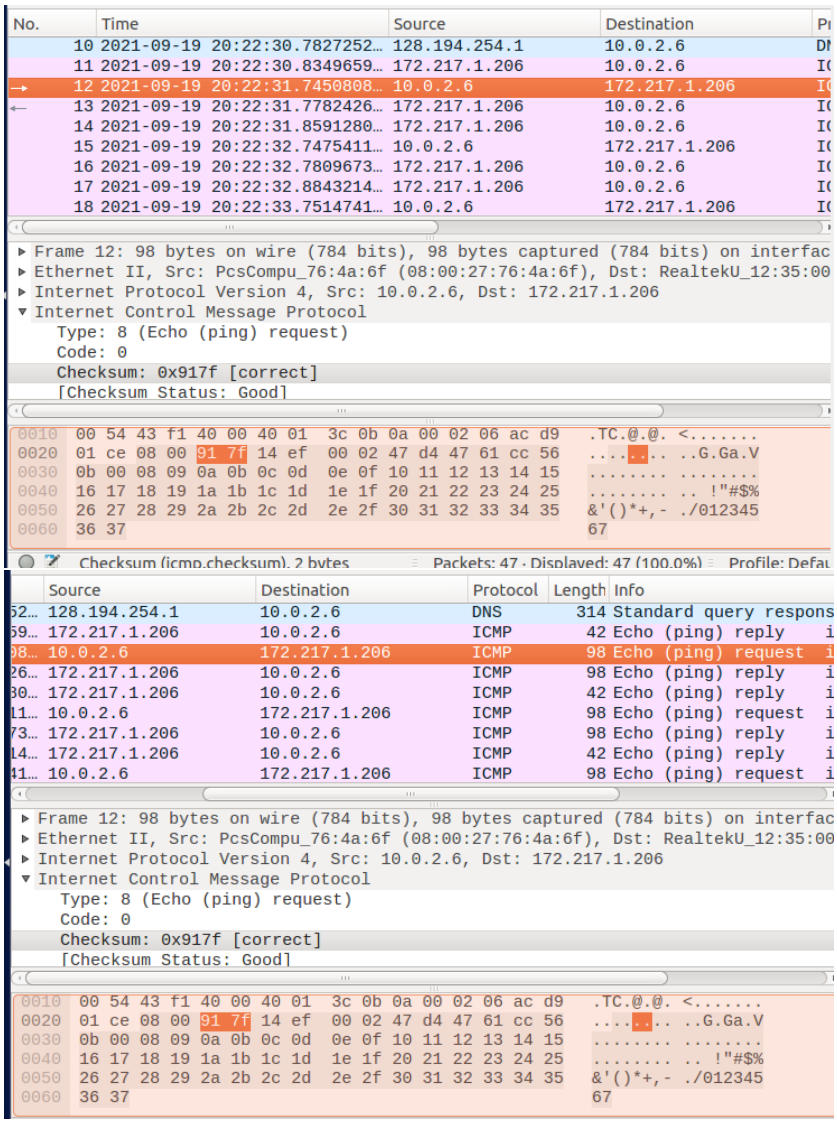


Here you can see that I spoofed an ICMP echo request packet on behalf of another machine. So the request was sent out, and the reply was sent back. For this task used reference 4 from the HW1 pdf.

Task 2c

1. I think you can because the system would recalculate and set the value to the correct length. So you can set bigger up to a maximum value of 1500 bytes. Once you pass the cutoff it won't work. Also if you set it smaller, that won't work as well. So yes to a certain point, but no if you pass those thresholds.
2. If you don't calculate the correct checksum, then the packet will be dropped when it reaches its destination, so it won't reply. This happened to me many times when trying to get the echo request to work because I had the wrong checksum value.
3. The seed users(regular users) don't have access to the commands and permissions needed to create a raw socket. So it would fail when trying to create the socket at this command -> socket(AF_INET, SOCK_RAW, IPPROTO_RAW). I ran in as a regular user and that is where it failed.

Task 3



So for the sniff and spoof code, I figured the best way to approach this was to sniff until I received a packet. Once that occurs, I would then use the spoofing code to create my own spoofed packet and then send it back out to the victim. So to combine my sniffex.c and spoofing.c code I copied the sniffex file and then added the spoofing.c code to the got_packet function since that is where the packet is dealt with. Of course, I also added all of the #includes, #defines, and structs. For the headers, we want our spoofed headers to be identical to what the normal icmp headers of the packet received was, so the code below demonstrates that.

```
        iphr->ip_id = ip->ip_id;
        iphr->ip_off = ip->ip_off;
        iphr->ip_ttl = ip->ip_ttl;
        iphr->ip_p = IPPROTO_ICMP;
        iphr->ip_sum= 0; //Set to 0 before calculating checksum
```

```
        iphr->ip_src = ip->ip_dst;
        iphr->ip_dst = sin.sin_addr.s_addr;
```

I took the spoofed packet's src ip and made it equal to the dst ip of the packet it sniffed because we want it to reply back where it came from and the destination ip of the spoofed packet will go back to the victim ip. The code above is mostly what changed from the original spoofing.c file. What I found interesting was that there were two replies back, but that makes sense because there was the original one that was sent back, and then of course the spoofed one.