Lab Report

Task1

Task 1a:

(1)The sequence of the library calls that are essential for sniffer programs are as below:

Setting the device

Opening the device for sniffing

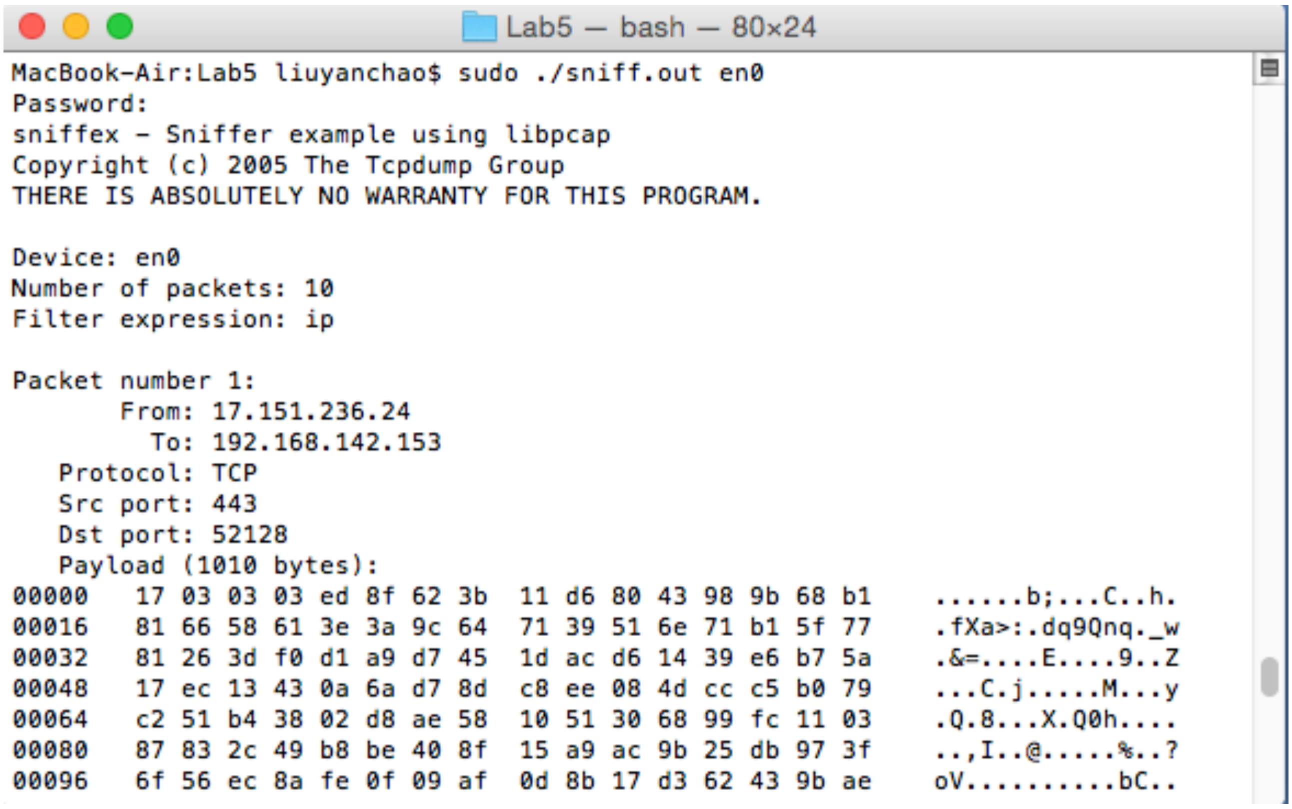
The actual sniffing

Filtering traffic

(2)If executed without the root privilege, the program will alert “Couldn't open device en0: en0: You don't have permission to capture on that device ((cannot open BPF device) /dev/bpf0: Permission denied)”. Because of the hierarchy of network, the OS hides the details below the application layer. For users, they do not need to know what the packet’s content is, but for some crackers they could get useful things for their evil. Another reason is that the program calls the system API. So the root privilege is requested.

(3)There are many broadcast packets in ethernet. When the promiscuous mode is turned on, we can receive some broadcast or multicast packets that may be not sent to us. The evidence is that I can receive the packet with broadcast or multicast address.

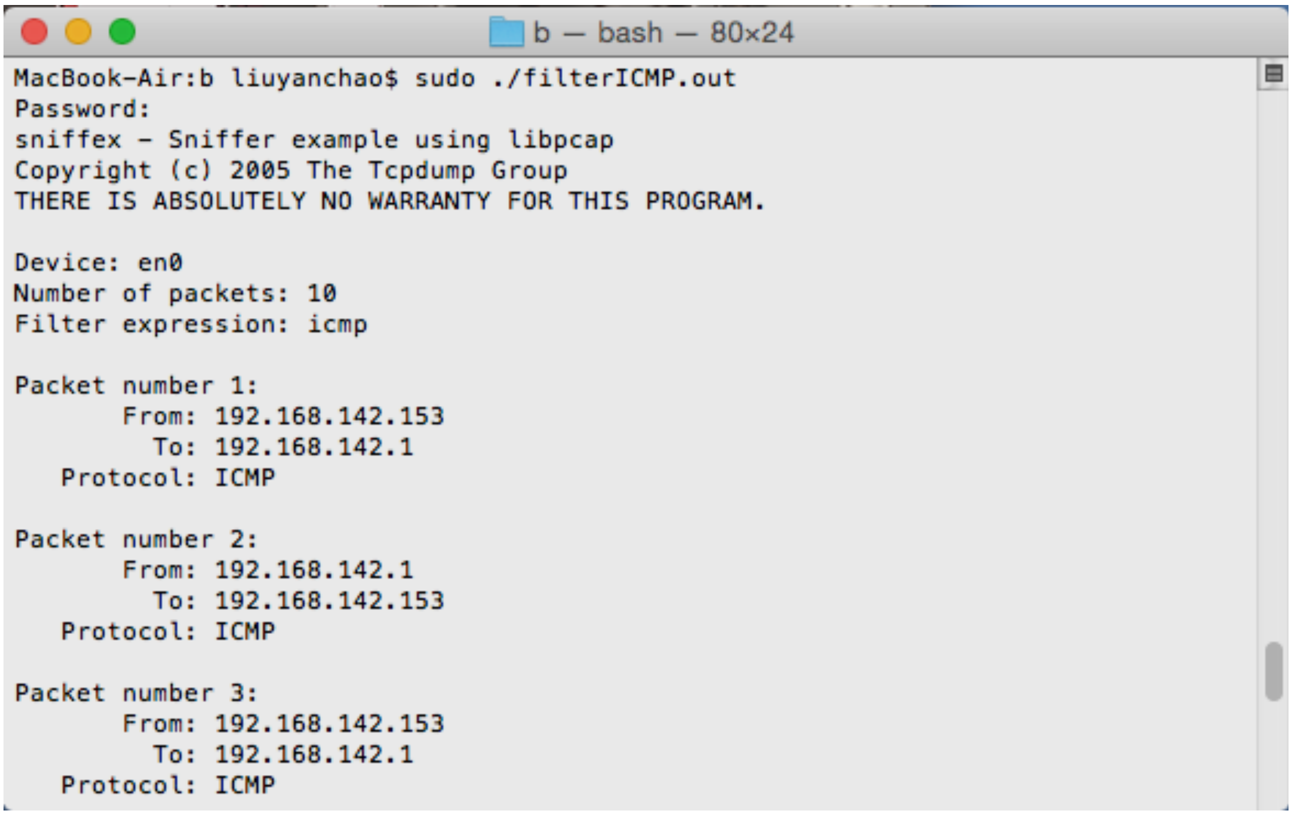
screendump evidence to show that your program runs successfully and produces expected results:



```
MacBook-Air:Lab5 liuyanchao$ sudo ./sniff.out en0
Password:
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: en0
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 17.151.236.24
         To: 192.168.142.153
   Protocol: TCP
   Src port: 443
   Dst port: 52128
   Payload (1010 bytes):
00000   17 03 03 03 ed 8f 62 3b  11 d6 80 43 98 9b 68 b1    ......b;...C..h.
00016   81 66 58 61 3e 3a 9c 64  71 39 51 6e 71 b1 5f 77    .fXa>:.dq9Qnq._w
00032   81 26 3d f0 d1 a9 d7 45  1d ac d6 14 39 e6 b7 5a    .&=....E....9..Z
00048   17 ec 13 43 0a 6a d7 8d  c8 ee 08 4d cc c5 b0 79    ...C.j.....M...y
00064   c2 51 b4 38 02 d8 ae 58  10 51 30 68 99 fc 11 03    .Q.8...X.Q0h....
00080   87 83 2c 49 b8 be 40 8f  15 a9 ac 9b 25 db 97 3f    ..,I..@.....%..?
00096   6f 56 ec 8a fe 0f 09 af  0d 8b 17 d3 62 43 9b ae    oV..........bC..
```

Task 1b:

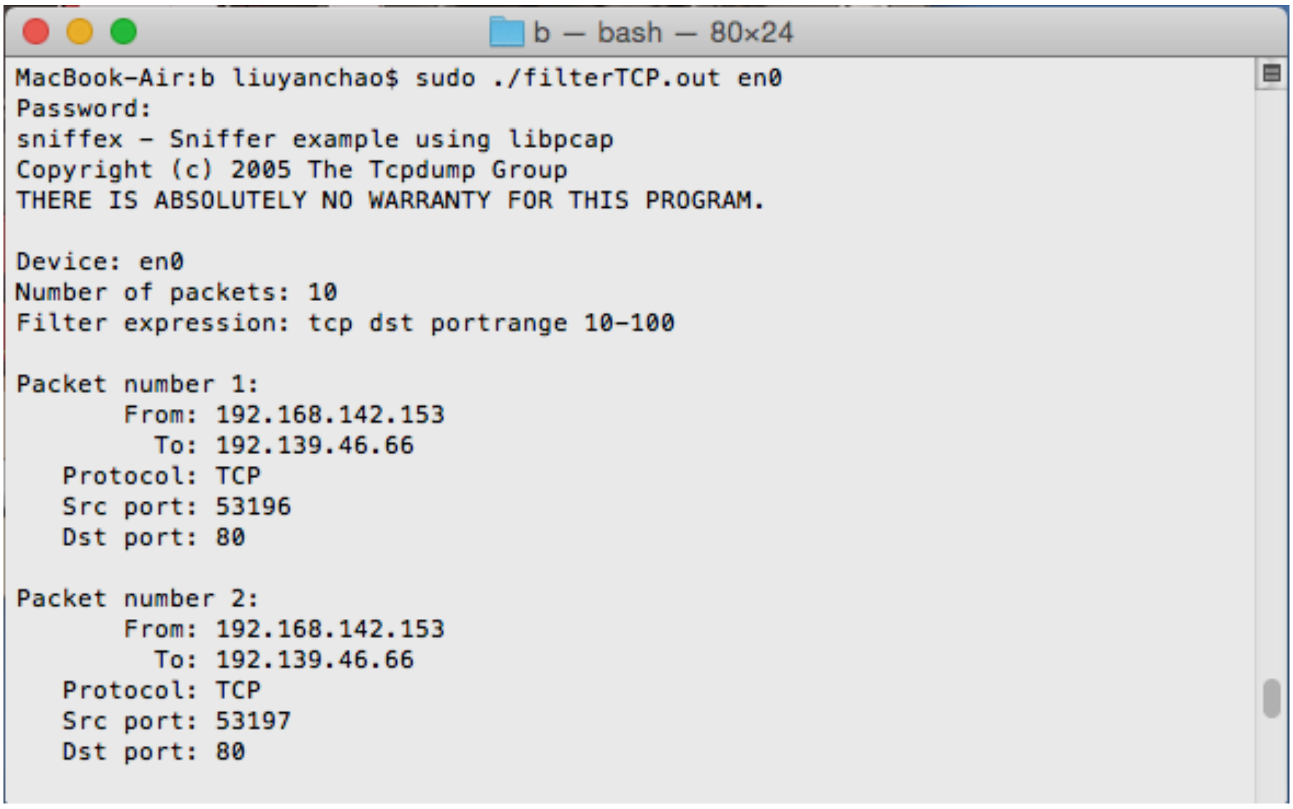
- Capture the ICMP packets between two specific hosts



b — bash — 80×24

```
MacBook-Air:b liuyanchao$ sudo ./filterICMP.out
Password:
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: en0
Number of packets: 10
Filter expression: icmp

Packet number 1:
       From: 192.168.142.153
         To: 192.168.142.1
   Protocol: ICMP

Packet number 2:
       From: 192.168.142.1
         To: 192.168.142.153
   Protocol: ICMP

Packet number 3:
       From: 192.168.142.153
         To: 192.168.142.1
   Protocol: ICMP
```

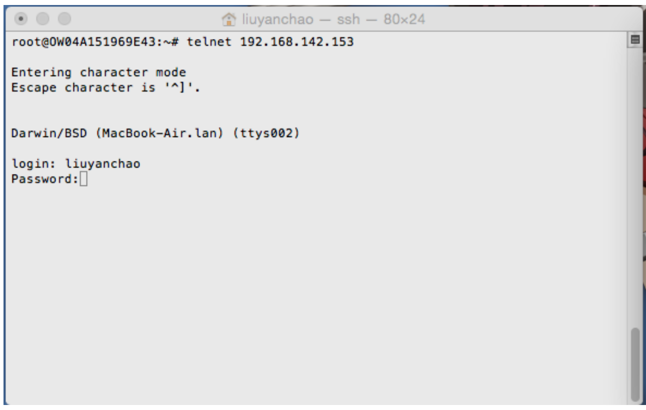
- Capture the TCP packets that have a destination port range from to port 10 - 100



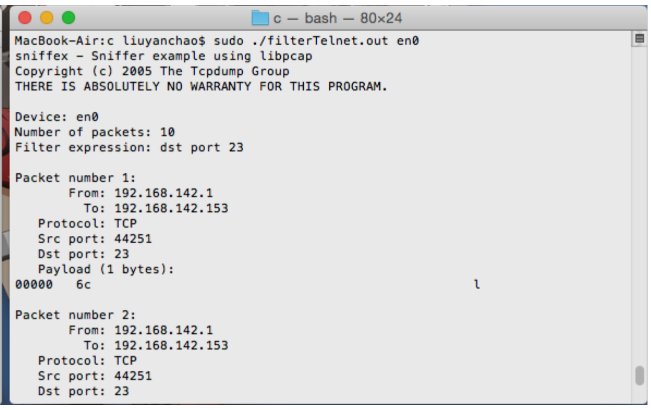
b — bash — 80×24

```
MacBook-Air:b liuyanchao$ sudo ./filterTCP.out en0
Password:
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: en0
Number of packets: 10
Filter expression: tcp dst portrange 10-100

Packet number 1:
       From: 192.168.142.153
         To: 192.139.46.66
   Protocol: TCP
   Src port: 53196
   Dst port: 80

Packet number 2:
       From: 192.168.142.153
         To: 192.139.46.66
   Protocol: TCP
   Src port: 53197
   Dst port: 80
```

Task 1.c:



```
root@OW04A151969E43:~# telnet 192.168.142.153

Entering character mode
Escape character is '^]'.


Darwin/BSD (MacBook-Air.lan) (ttys002)

login: liuyanchao
Password:
```



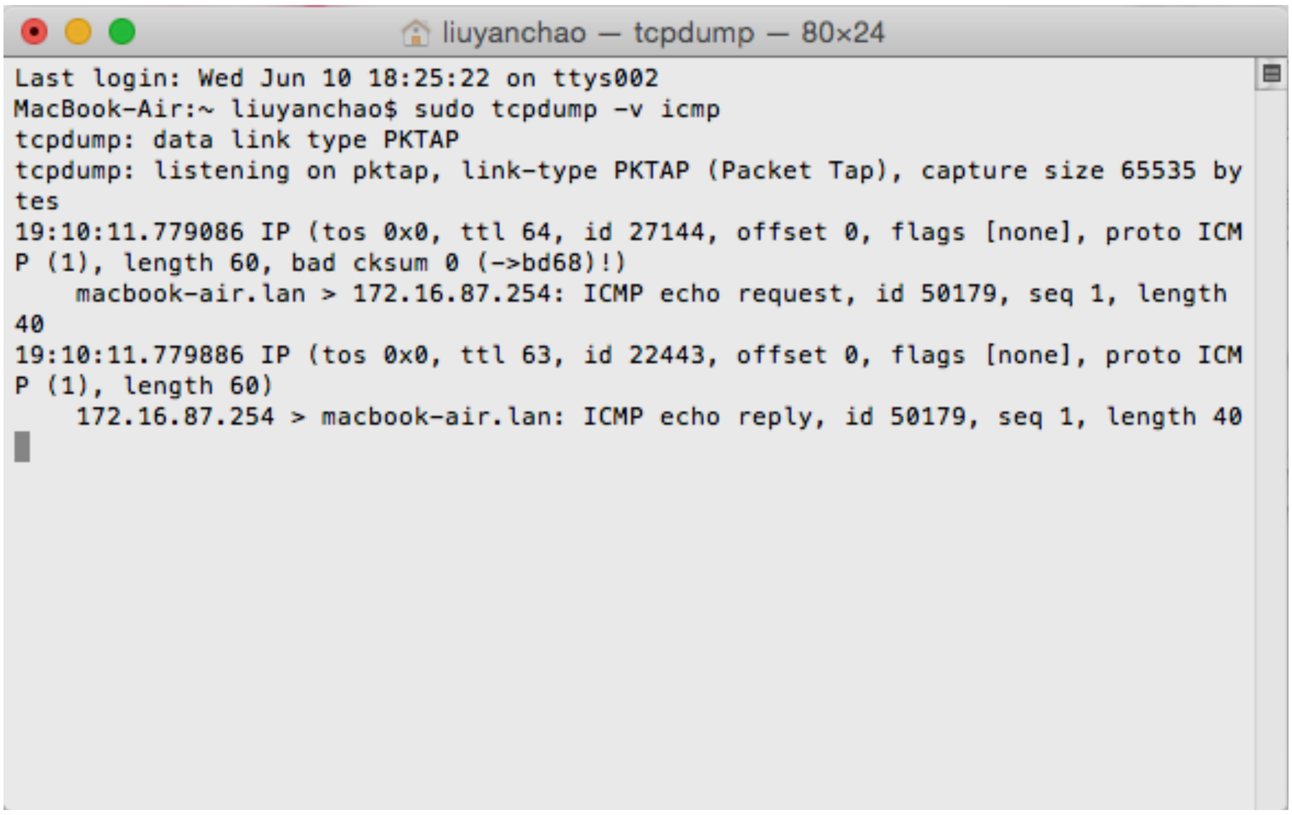
```
MacBook-Air:c liuyanchao$ sudo ./filterTelnet.out en0
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: en0
Number of packets: 10
Filter expression: dst port 23

Packet number 1:
       From: 192.168.142.1
         To: 192.168.142.153
   Protocol: TCP
   Src port: 44251
   Dst port: 23
   Payload (1 bytes):
00000   6c                                                  l

Packet number 2:
       From: 192.168.142.1
         To: 192.168.142.153
   Protocol: TCP
   Src port: 44251
   Dst port: 23
```

Use the filter to catch packets with dst port 23. As we can see the sniffer get the first alphabet of the username liuyanchao — l.

Task2

Task 2.a:

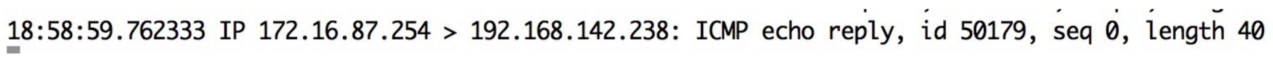
I find a ICMP spoofing program and change the source ip to mine. Here is the packet caught by tcpdump below:



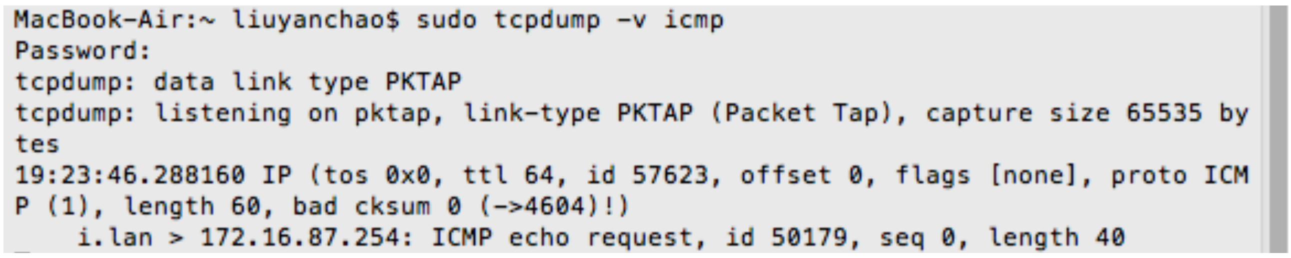
```
Last login: Wed Jun 10 18:25:22 on ttys002
MacBook-Air:~ liuyanchao$ sudo tcpdump -v icmp
tcpdump: data link type PKTAP
tcpdump: listening on pktap, link-type PKTAP (Packet Tap), capture size 65535 by
tes
19:10:11.779086 IP (tos 0x0, ttl 64, id 27144, offset 0, flags [none], proto ICM
P (1), length 60, bad cksum 0 (->bd68)!)
    macbook-air.lan > 172.16.87.254: ICMP echo request, id 50179, seq 1, length
40
19:10:11.779886 IP (tos 0x0, ttl 63, id 22443, offset 0, flags [none], proto ICM
P (1), length 60)
    172.16.87.254 > macbook-air.lan: ICMP echo reply, id 50179, seq 1, length 40
```

Task 2.b:

I spoofed an ICMP Echo Request packet on behalf of my friend's machine, and the packet was sent to the gateway of school network:172.16.87.254. The tcpdump caught the request packet and the reply packet.



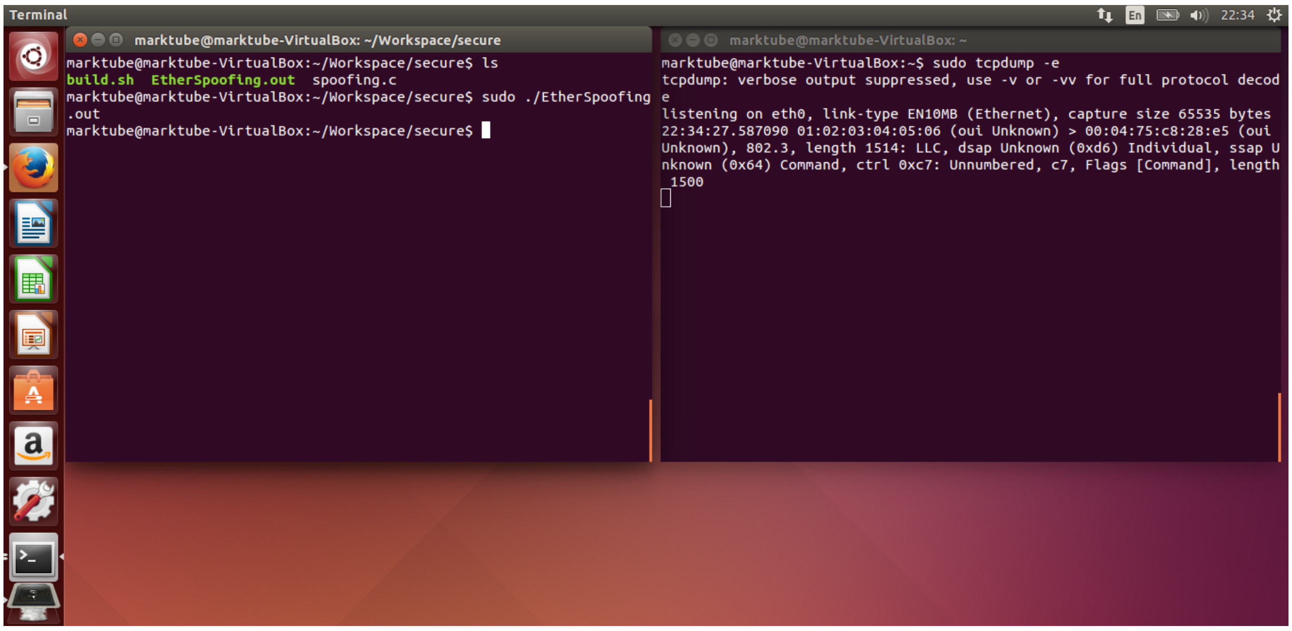
```
18:58:59.762333 IP 172.16.87.254 > 192.168.142.238: ICMP echo reply, id 50179, seq 0, length 40
```



```
MacBook-Air:~ liuyanchao$ sudo tcpdump -v icmp
Password:
tcpdump: data link type PKTAP
tcpdump: listening on pktap, link-type PKTAP (Packet Tap), capture size 65535 by
tes
19:23:46.288160 IP (tos 0x0, ttl 64, id 57623, offset 0, flags [none], proto ICM
P (1), length 60, bad cksum 0 (->4604)!)
    i.lan > 172.16.87.254: ICMP echo request, id 50179, seq 0, length 40
```

Task 2.c:

The source code need to be compile on linux. I tested it on Ubuntu 13.05, and successfully get the frame by tcpdump. Here is the screenshot below:



Terminal
22:34
marktube@marktube-VirtualBox: ~/Workspace/secure
marktube@marktube-VirtualBox: ~

```
marktube@marktube-VirtualBox:~/Workspace/secure$ ls
build.sh  EtherSpoofing.out  spoofing.c
marktube@marktube-VirtualBox:~/Workspace/secure$ sudo ./EtherSpoofing
.out
marktube@marktube-VirtualBox:~/Workspace/secure$
```

```
marktube@marktube-VirtualBox:~$ sudo tcpdump -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decod
e
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:34:27.587090 01:02:03:04:05:06 (oui Unknown) > 00:04:75:c8:28:e5 (oui
Unknown), 802.3, length 1514: LLC, dsap Unknown (0xd6) Individual, ssap U
nknown (0x64) Command, ctrl 0xc7: Unnumbered, c7, Flags [Command], length
 1500
```

Questions. Please answer the following questions.

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

Answer: The length field should be the length of IP packet. Otherwise the sendto function will alert the error: Invalid Argument!

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

Answer: We don't have to calculate the checksum for the IP header. It will be filled by the system.