Network Security Lab Report 2

--Zhenlei Song
Songzl8_950606@tamu.edu

Task1

1.1 Environment

For this sniffing task, I set up 3 identical VM(VM1 VM2 VM3).

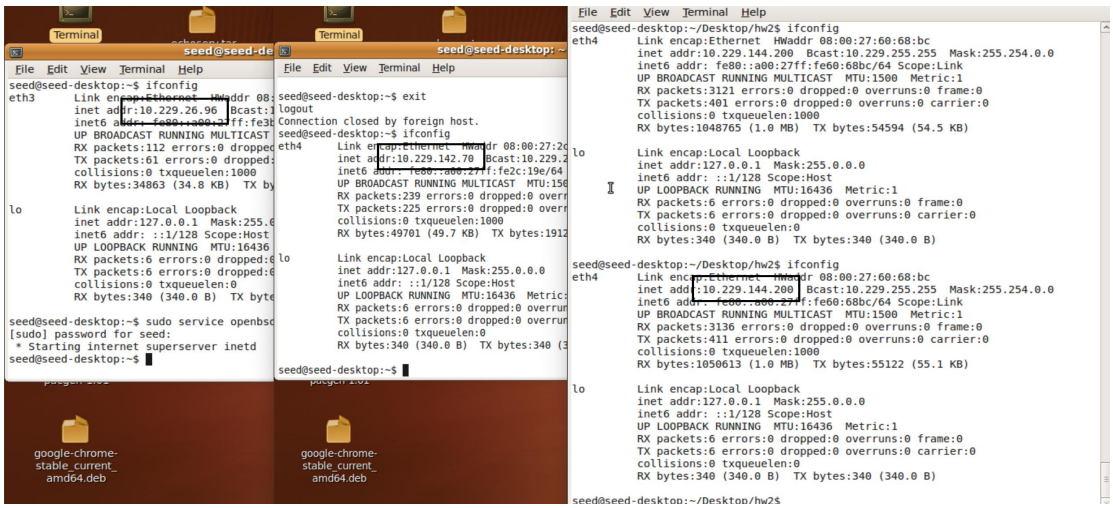

Fig 1.1 VM

And I modify the sniffex.c file for more general use. Let interface and filter expression be input parameters. The usage is like:

```
print_app_usage(void)
{

    printf("Usage: %s (-i [interface]) (-f [filter_expression]) (-p [promiscuous_mode])\n", APP_NAME);
    printf("\n");
    printf("Options:\n");
    printf("    interface    Listen on <interface> for packets.\n");
    printf("\n");

return;
}
```

Fig 1.2 sniffex usage

'-i ' option: determine which net interface to be sniffed, string input

'-f' option: to determine the pcap filter expression, string input.

'-p' option: to determine whether to work under promiscuous or non-promiscuous mode.(only

'true'/'false' received).

1.2 Problem1

The command to compile sniffex.c is

gcc -Wall -o sniffex sniffex.c -l pcap

Some APIs in PCAP lib are called to implement sniffing functionality.

1) pcap_lookupdev

If there is no input network device sniffex will search default device automatically.

2) pcap_lookupnet

This API is used to get net mask and net number from selected network device.

3) pcap_open_live/pcap_close(datalink layer)

Open network device, and returns file handle.

The third parameter defines whether promiscuous mode or not.

4) pcap_datalink

Returns the datalink type of current device. See if we are sniffing a Ethernet device. Below shows the returns value of 'pcap_datalink' which is the enum of all datalink layer protocols.

```
#define DLT_NULL 0 /* BSD loopback encapsulation */
#define DLT_EN10MB 1 /* Ethernet (10Mb) */
#define DLT_EN3MB 2 /* Experimental Ethernet (3Mb) */
#define DLT_AX25 3 /* Amateur Radio AX.25 */
#define DLT_PRONET 4 /* Proteon ProNET Token Ring */
#define DLT_CHAOS 5 /* Chaos */
#define DLT_IEEE802 6 /* 802.5 Token Ring */
#define DLT_ARCNET 7 /* ARCNET, with BSD-style header */
#define DLT_SLIP 8 /* Serial Line IP */
#define DLT_PPP 9 /* Point-to-point Protocol */
#define DLT_FDDI 10 /* FDDI */
```

5) pcap_compile / pcap_setfilter /pcap_freecode(network layer)

Compile filter expression to a defined structure in pcap and set it to current network interface.

Third parameter of pcap_compile receives the filter string.

Pcap_freecode is to remove the network filter object.

6) pcap_loop

Indicate times to loop run, and specify callback function when package is captured.

1.3 Problem2

The network device can only be accessed under root privilege. If not root, the device cannot even be scanned. Even if the network device is specified by input. It still cannot be opened.

```
seed@seed-desktop:~/Desktop/hw2$
seed@seed-desktop:~/Desktop/hw2$
seed@seed-desktop:~/Desktop/hw2$
seed@seed-desktop:~/Desktop/hw2$
seed@seed-desktop:~/Desktop/hw2$
seed@seed-desktop:~/Desktop/hw2$ ./sniffex -f "tcp dst portrange 10-100"
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Couldn't find default device: no suitable device found
seed@seed-desktop:~/Desktop/hw2$ ./sniffex -i eth4 -f "tcp dst portrange 10-100"
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth4
Number of packets: 100
Filter expression: tcp dst portrange 10-100
promicuous: 1
Couldn't open device eth4: socket: Operation not permitted
```

Fig 1.3 run sniffex with no root privilege

1.4 Problem3

The promiscuous mode : all packets pass through current node can be captured.
Non-promicuous mode: only packets whose dst address is current node will be captured.

To switch between promisc mode and non-promisc mode, not only the parameter of sniffex needs to be modified. Also the network setting of VMs needs to modify as well.
I modified sniffex.c use parameter ’-p’ as input to indicate whether use promiscuous or non-promiscuous mode to open network device.

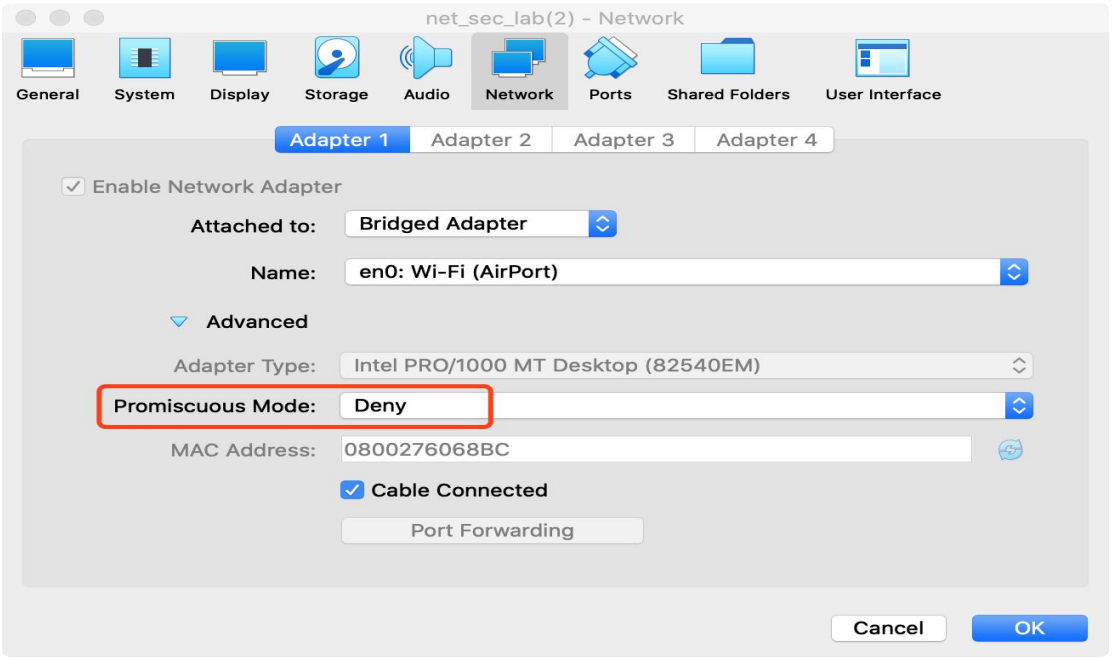

Fig 1.4 change non-promiscuous mode in VM's setting

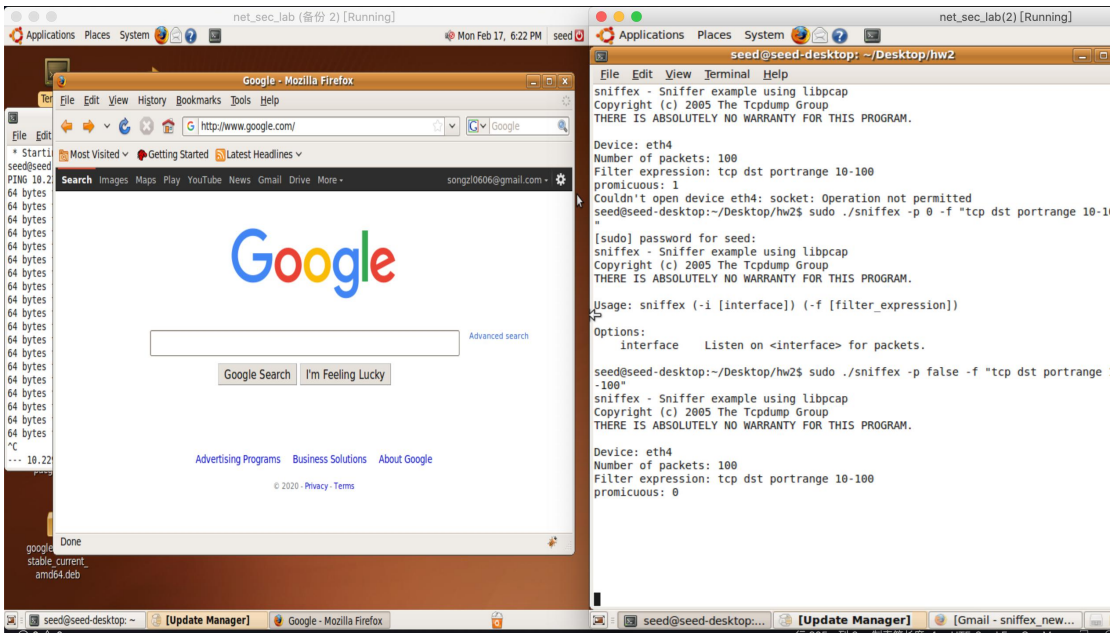

Fig 1.5 non-promiscuous mode

As we can see in Fig 1.5, when the VM network and sniffex are both set to non-promiscuous mode, the HTTP packet from other machines in LAN will not be captured by sniffer. As we can see later, in promiscuous mode which can be captured.

1.5 Problem4

(a)Let VM1 ping VM2, VM3 sniffers packets in promiscuous mode.

The command on VM3 is

sudo ./sniffex -f “ip proto \icmp”

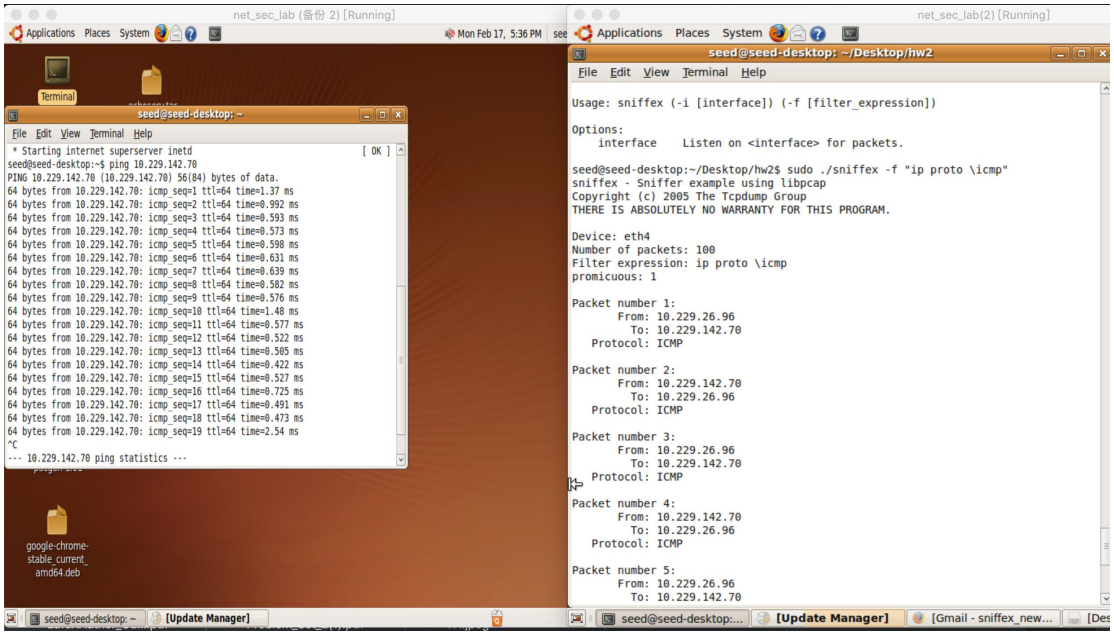

Fig 1.6 filter icmp packets

(b)Let VM1 run browser and seng HTTP requests packets whose port used is 80. Then VM3 sniffers the packets. Here we use the same commands as in last task only the network mode is under promiscuous mode, then packets captured.

The command used on VM3 is:

sudo ./sniffex -f “tcp dst portrange 10-100”

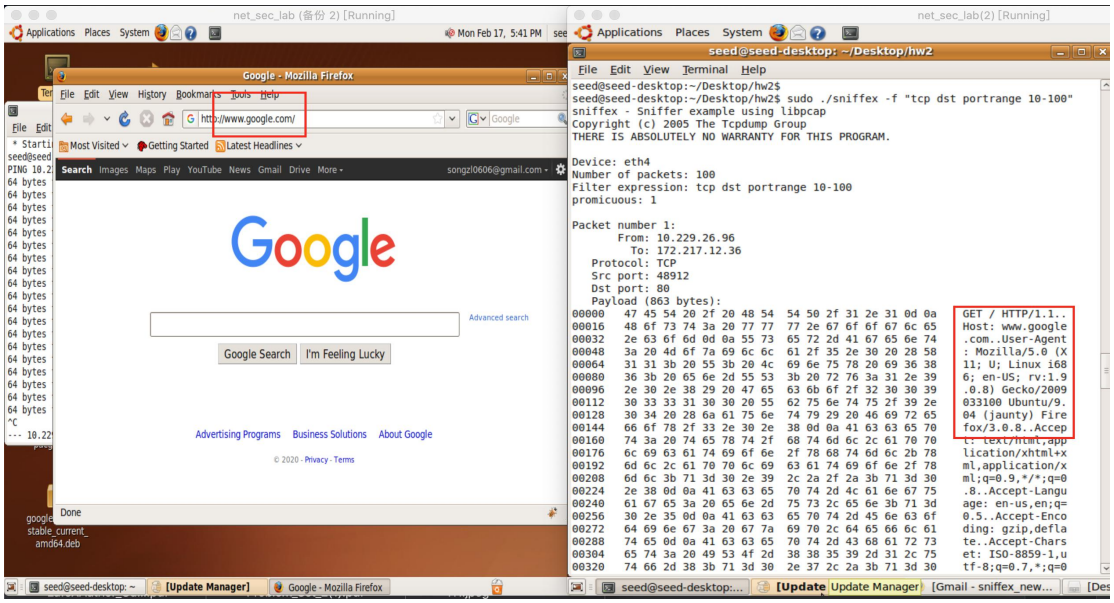

Fig 1.7 filter 10-100 port packets

1.6 Problem 5

Let VM2 open service as telnet server, VM1 try to log into VM2 with telnet. VM3 sniffers the packets during this progress.

Due to telnet use port 23, we will use the command:

sudo ./sniffex -f "tcp port 23"

```
00000   73                                                      s

Packet number 38:
       From: 10.229.26.96
         To: 10.229.142.70
   Protocol: TCP
   Src port: 23
   Dst port: 36936
   Payload (1 bytes):
00000   73                                                      s

Packet number 39:
       From: 10.229.142.70
         To: 10.229.26.96
   Protocol: TCP
   Src port: 36936
   Dst port: 23

Packet number 40:
       From: 10.229.142.70
         To: 10.229.26.96
   Protocol: TCP
   Src port: 36936
   Dst port: 23
   Payload (1 bytes):
00000   65                                                      e

Packet number 41:
       From: 10.229.26.96
         To: 10.229.142.70
   Protocol: TCP
   Src port: 23
   Dst port: 36936
   Payload (1 bytes):
00000   65                                                      e

Packet number 42:
       From: 10.229.142.70
```

Fig 1.8 telnet packet captured

```
   Dst port: 23
   Payload (1 bytes):
00000   65                                                e

Packet number 44:
       From: 10.229.26.96
         To: 10.229.142.70
   Protocol: TCP
   Src port: 23
   Dst port: 36936
   Payload (1 bytes):
00000   65                                                e

Packet number 45:
       From: 10.229.142.70
         To: 10.229.26.96
   Protocol: TCP
   Src port: 36936
   Dst port: 23

Packet number 46:
       From: 10.229.142.70
         To: 10.229.26.96
   Protocol: TCP
   Src port: 36936
   Dst port: 23
   Payload (1 bytes):
00000   64                                                d

Packet number 47:
       From: 10.229.26.96
         To: 10.229.142.70
   Protocol: TCP
   Src port: 23
   Dst port: 36936
   Payload (1 bytes):
00000   64                                                d
```

Fig 1.8 telnet packet captured (continued)

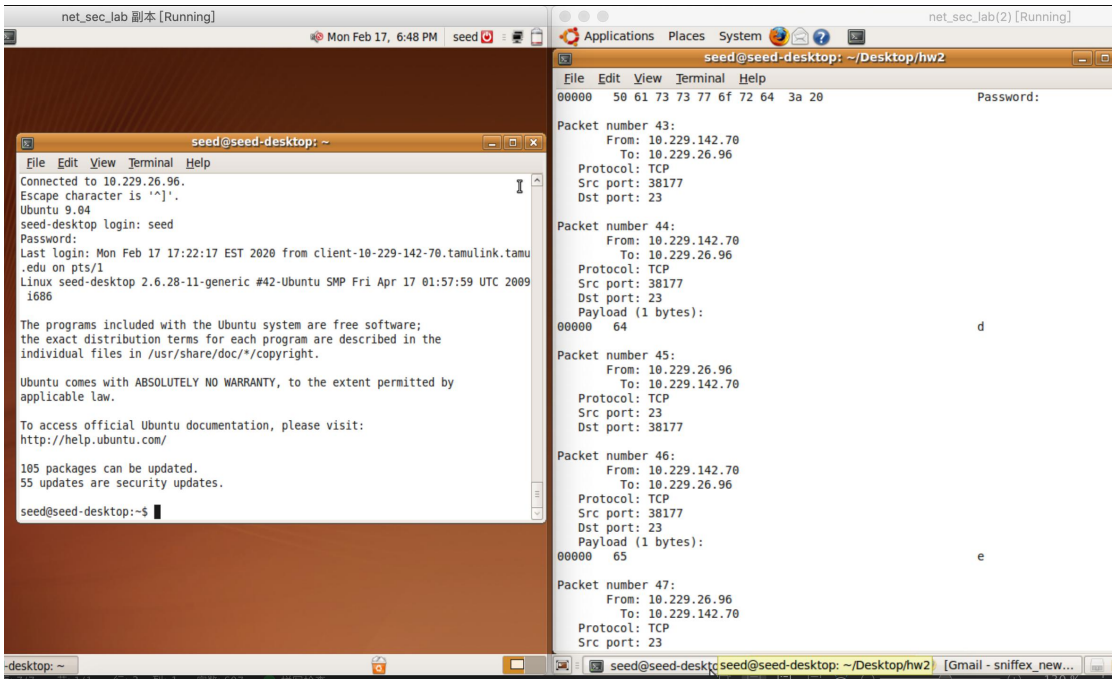

Fig 1.8 telnet packet captured(continued)

As we can see from above screen shots, Due to telnet use plain text, the login name and password is exposed on internet as plain text. Though login name will be echoed to src side.

Task2

A) souce C file rawip.c

I forge a socket to send fixed string “hahaha” in the data section of IP packet.

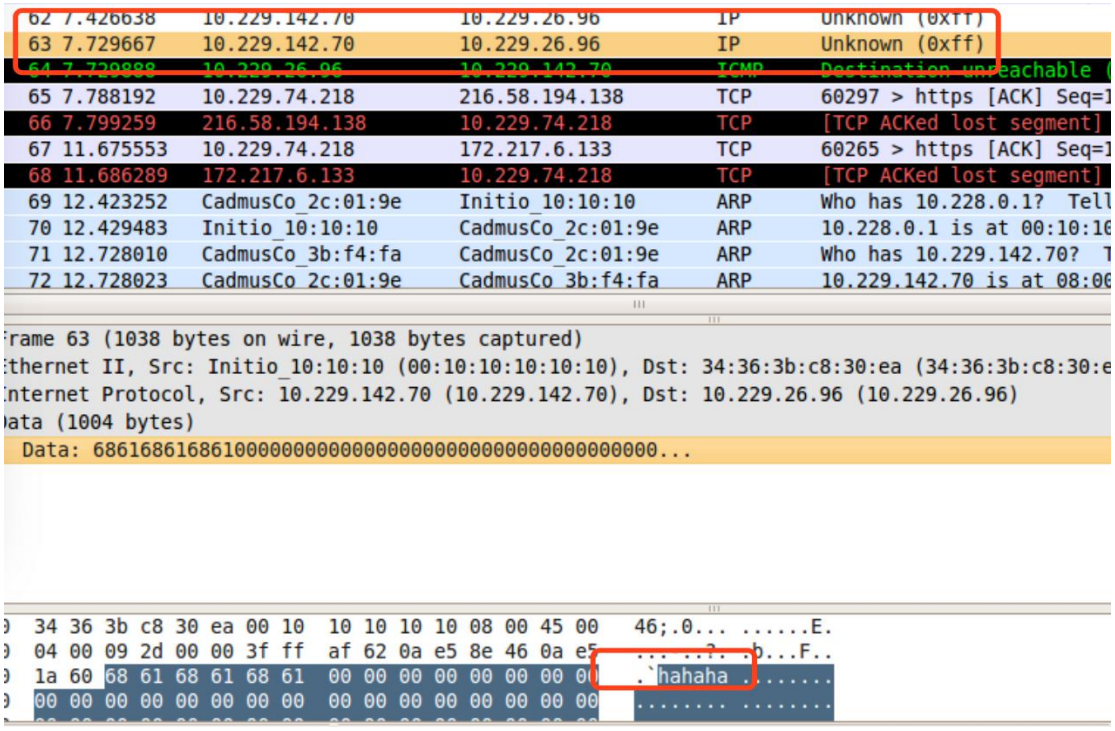

Fig 2.1 fake IP packet

B) Source file rawicmp.c

I made a raw icmp structure to replace the string “hahaha”. By receiving inputs as src IP and dst IP, we can forge any ping packet we want(even though the IP address doesn’t exist). Packets can be shown on Wireshark when filtering ICMP.

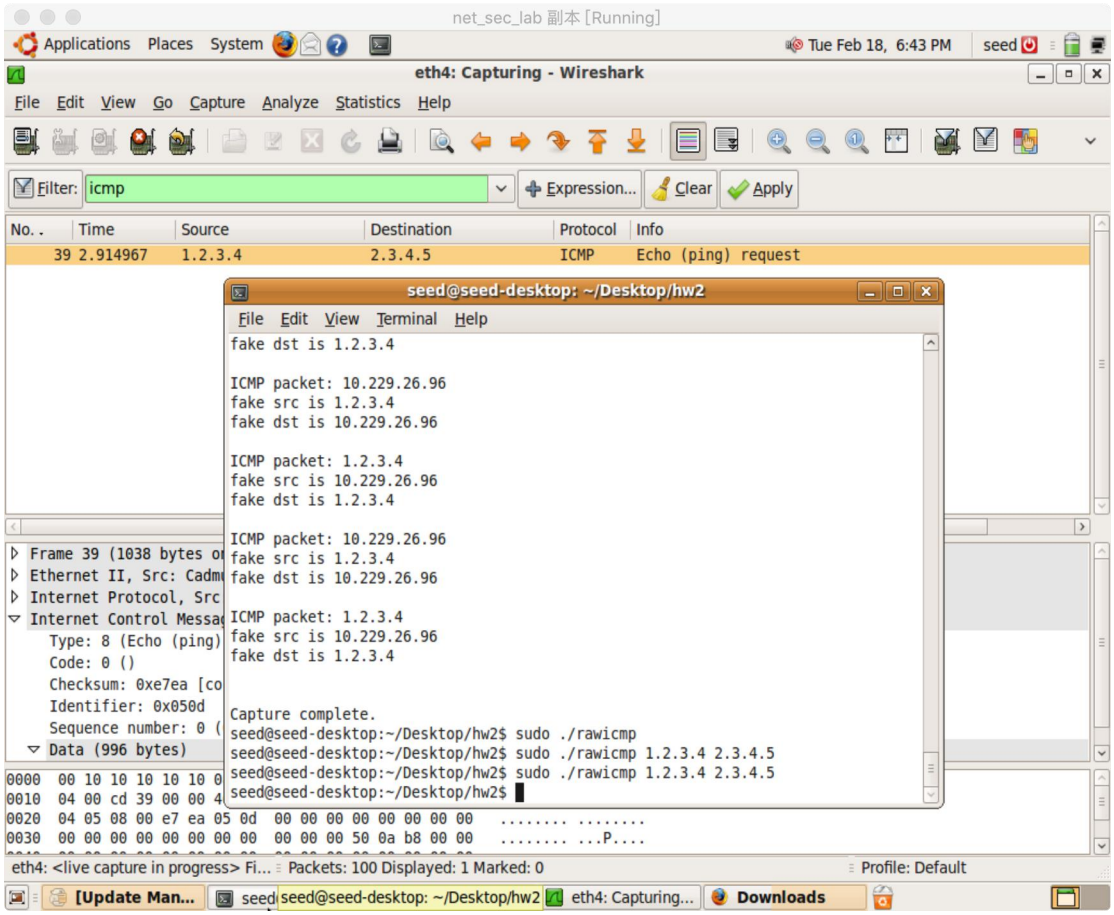

Fig 2.2 Forged ICMP packets

C) 1. Can we set IP length to an arbitrary value?

Yes. Because there are plenty protocols are based on IP protocol, and the packet length varies from each other. So the total length of IP packet vary among plenty of possible values. For a test, I used to set the total IP length in Fig 2.1 as 1024. And I modify it to 512 and re-compile it. And it turns out like in Fig 2.3.

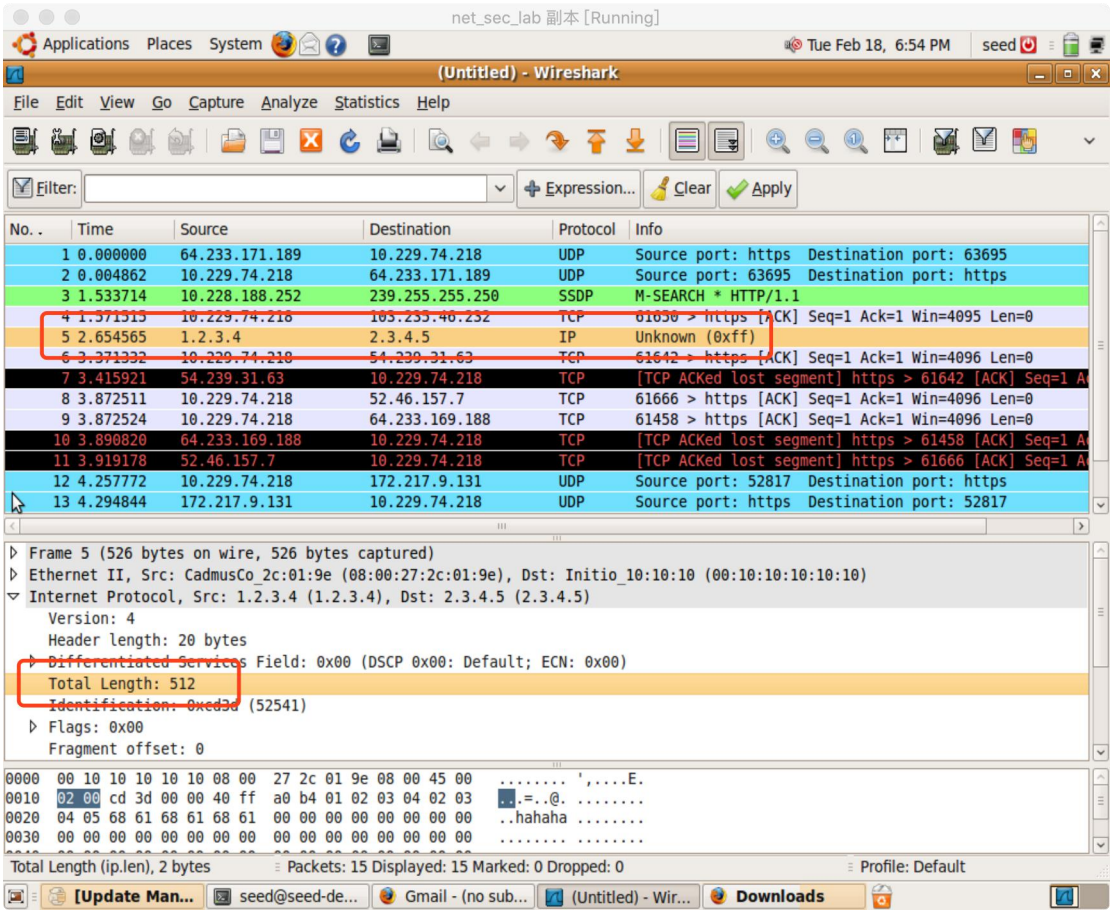

Fig 2.3 Trying to modify IP packet length

2. Do we need to calculate IP header check sum on our own?

No. I didn't call any check sum API or set any value concerned with IP check sum in 'rawip.c'. The IP check sum is automatically calculated by OS.

3. About root privilege.

Yes. Raw socket program requires root privilege. Because 'socket()' API requires root identity. Without root privilege, it runs like below.

```
seed@seed-desktop:~/Desktop/hw2$ ./rawip 1.2.3.4 2.3.4.5
socket() error: Operation not permitted
seed@seed-desktop:~/Desktop/hw2$
```

Fig 2.4 no root privilege

Task3

I use 2 VM in this task. VM1(IP: 10.229.26.96), VM2(IP: 10.229.142.70). On VM1, run "ping 1.2.3.4" while VM2 is sniffing and spoofing response packets. This is obvious that it will not get responded in normal cases. Source file of sniff&spoof program running on VM2 is 'Sniff_n_Spoof.c'. The comparison on VM1 terminal is shown in Fig 3.1. We can see in normal way, all of the ping packets are lost(lower part in Fig 3.1). But when spoofed packets transmitted, some replies are captured(upper part).

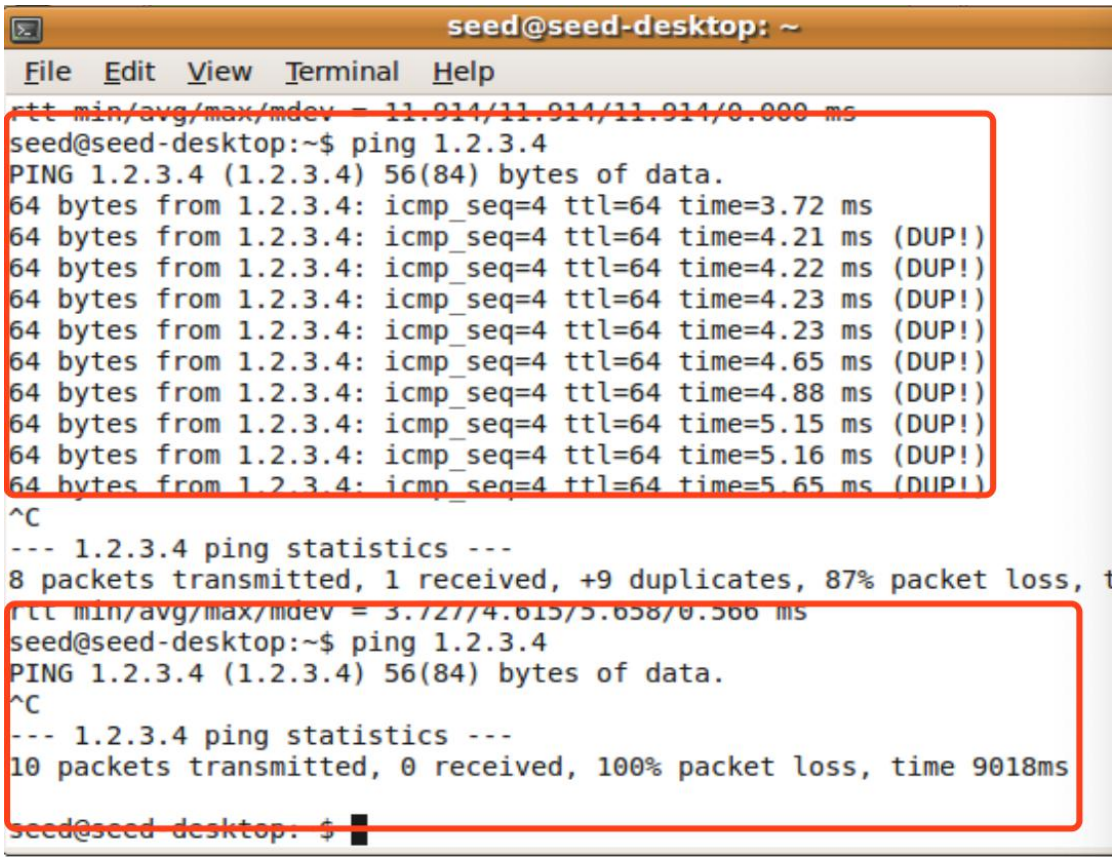

Fig 3.1 Spoofed/not spoofed comparison

The packets captured on wireshark is shown in Fig 3.2.

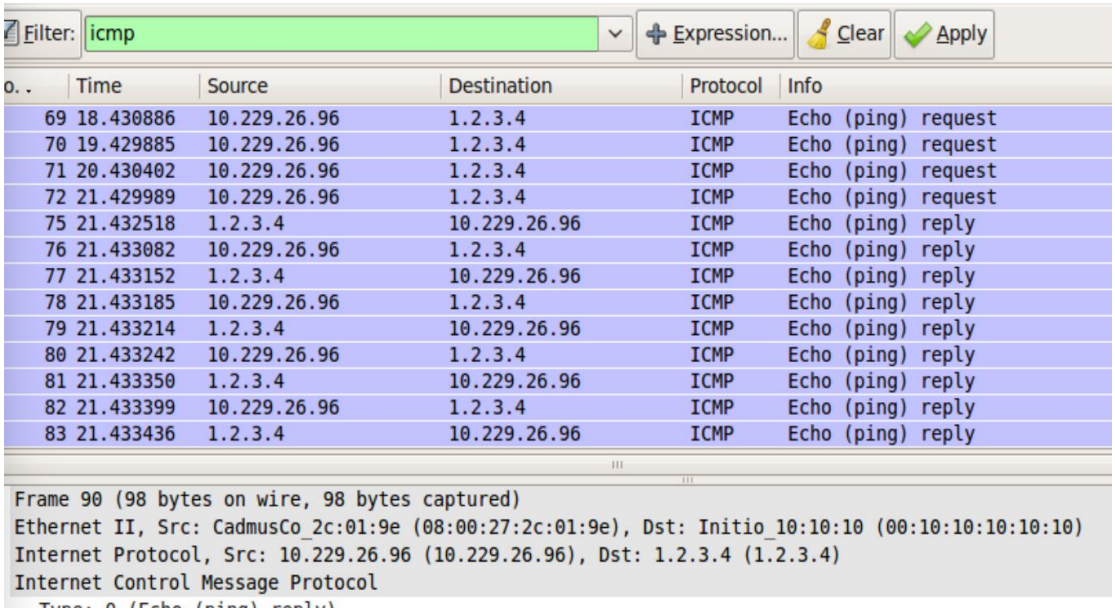

Fig 3.2 packets captured