CS5460 – ASSIGNMENT 5

MARCH 28, 2016

VARUN GATTU
A02092613

Packet Sniffing and Spoofing Lab

2. Lab Tasks

2.1. Task 1: Writing Packet Sniffing Program
Task 1a: Understanding Sniffex

Here, I downloaded the program from the provided link and compiled it with –lpcap extension. Then, I executed the program from the root and produced the following output:

```
[04/01/2016 15:45] seed@ubuntu:~/Desktop/Assignment5$ gcc -o sniffex sniffex.c -lpcap
[04/01/2016 15:45] seed@ubuntu:~/Desktop/Assignment5$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth14
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 173.194.192.189
         To: 10.0.2.15
   Protocol: TCP
   Src port: 443
   Dst port: 53527
   Payload (53 bytes):
00000   17 03 01 00 30 6b 3e 43  32 af fc 1d 93 4b 2e 45    ....0k>C2....K.E
00016   5f c5 1f 50 e2 1a 8f 32  1c 25 81 f8 8d 8c 27 04    _..P...2.%....'.
00032   67 cc a7 e7 ea 62 c0 e9  38 7e 2a 15 34 f7 03 40    g....b..8~*.4..@
00048   f8 eb f1 90 00                                      .....

Packet number 2:
       From: 10.0.2.15
         To: 173.194.192.189
   Protocol: TCP
   Src port: 53527
   Dst port: 443

Packet number 3:
       From: 10.0.2.15
         To: 216.58.217.14
   Protocol: TCP
   Src port: 51007
   Dst port: 443
```

```
Packet number 7:
       From: 216.58.217.14
         To: 10.0.2.15
   Protocol: TCP
   Src port: 443
   Dst port: 51007

Packet number 8:
       From: 10.0.2.15
         To: 216.58.217.14
   Protocol: TCP
   Src port: 51007
   Dst port: 443

Packet number 9:
       From: 10.0.2.15
         To: 216.58.217.46
   Protocol: TCP
   Src port: 34238
   Dst port: 443
   Payload (37 bytes):
00000   15 03 01 00 20 b5 90 70  f7 da e8 87 43 c1 66 b2    .... ..p....C.f.
00016   96 e8 34 1b 64 10 80 c5  d8 cb 35 80 77 3a 3f 89    ..4.d.....5.w:?.
00032   bb 30 9d 5e 07                                      .0.^.

Packet number 10:
       From: 216.58.217.46
         To: 10.0.2.15
   Protocol: TCP
   Src port: 443
   Dst port: 34238

Capture complete.
[04/01/2016 15:45] seed@ubuntu:~/Desktop/Assignment5$
```

Problem 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.

Answer:

a. Setting the device: this can be done in two ways:
    a. Device details are provided as a string when executing (argv)
    b. Use pcap_lookupdev() function to set a device on its own and failure results in saving the error message to errbuff
b. Open the device for sniffing: this can be done using pcap_open_live() with the following arguments to it
    a. The device that is specified in step a
    b. Snaplen is the number of bytes to be captured by pcap
    c. Promisc is to set the promiscuous mode
    d. To_ms is the read timeout in milliseconds
    e. Ebuf is to store error messages
c. Filtering the traffic: filter is applied after the expression is compiled.
    a. Pcap_compile() compiles the expression
    b. Pcap_setfilter() applies the filter
d. Actual sniffing: this can be done in two ways – capture single packet at a time or capture a set of packets using loops. The below functions are self-explanatory
    a. Pcap_next()
    b. Pcap_loop()

Problem 2: Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

Answer: Sniffex needs root privileges to execute because it uses pcap_lookupdev() which tries to select a device on its own to sniff. This function will not work if the program is not executed in root.

The output is shown below if tried to execute without root access:

```
[04/01/2016 17:11] seed@ubuntu:~/Desktop/Assignment5$ ./sniffex
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
[04/01/2016 17:11] seed@ubuntu:~/Desktop/Assignment5$
```

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

Answer: Turning on or turning off promiscuous mode can be done in the function call pcap_open_live(). The argument is set to 0 when the mode is off and vice versa. I manually turned it off as shown here:

```
File  Edit  View  Search  Terminal  Help                                  5:23 PM  Seed
        }
        else if (argc > 2) {
                fprintf(stderr, "error: unrecognized command-line options\n\n");
                print_app_usage();
                exit(EXIT_FAILURE);
        }
        else {
                /* find a capture device if not specified on command-line */
                dev = pcap_lookupdev(errbuf);
                if (dev == NULL) {
                        fprintf(stderr, "Couldn't find default device: %s\n",
                            errbuf);
                        exit(EXIT_FAILURE);
                }
        }

        /* get network number and mask associated with capture device */
        if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
                fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
                    dev, errbuf);
                net = 0;
                mask = 0;
        }

        /* print capture info */
        printf("Device: %s\n", dev);
        printf("Number of packets: %d\n", num_packets);
        printf("Filter expression: %s\n", filter_exp);

        /* open capture device */
        /* Promiscous mode on */
        //handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
        /* Promiscous mode off */
        handle = pcap_open_live(dev, SNAP_LEN, 0, 1000, errbuf);
-- INSERT --                                                        552,4-11      93%
```

Then I again compiled and executed the program to get the following output:

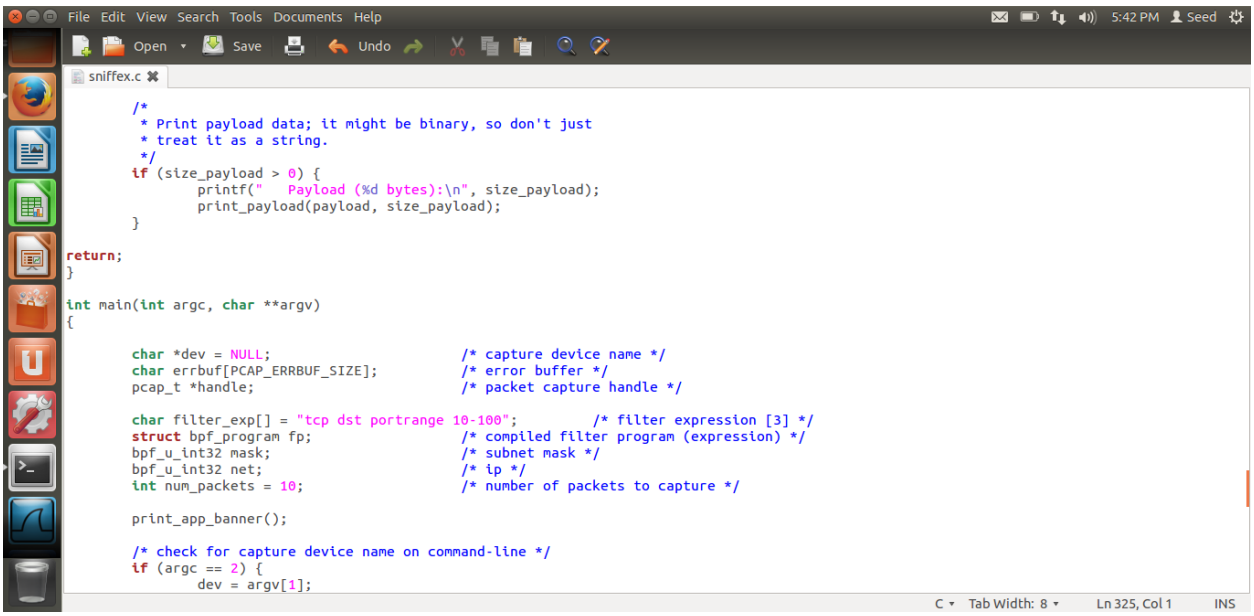
```
File  Edit  View  Search  Terminal  Help                                  5:25 PM  Seed
[04/01/2016 17:24] seed@ubuntu:~/Desktop/Assignment5$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth14
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 10.0.2.15
         To: 216.58.217.46
   Protocol: TCP
   Src port: 48620
   Dst port: 80

Packet number 2:
       From: 216.58.217.46
         To: 10.0.2.15
   Protocol: TCP
   Src port: 80
   Dst port: 48620

Packet number 3:
       From: 10.0.2.15
         To: 216.58.217.46
   Protocol: TCP
   Src port: 48620
   Dst port: 80

Packet number 4:
       From: 10.0.2.15
         To: 216.58.217.46
   Protocol: TCP
   Src port: 48620
```

This shows that when the promiscuous mode is turned off, the sniffer program only captures packets from a single source and single destination IP address unlike when it was turned on, the program captured packets from multiple devices.

Task 1b: Writing filters

- Capture the ICMP packets between two specific hosts
- Capture the TCP packets with destination port range from 10-100

Here, I have changed the filter_exp[] from “ip” to “tcp dst portrange 10-100” to capture all the packets that are sent only to TCP ports with range 10-100. The below screenshots show how and what happened:



It shows that all the captured packets have DST PORT with values 10-100 only.

Task 1c: Sniffing passwords

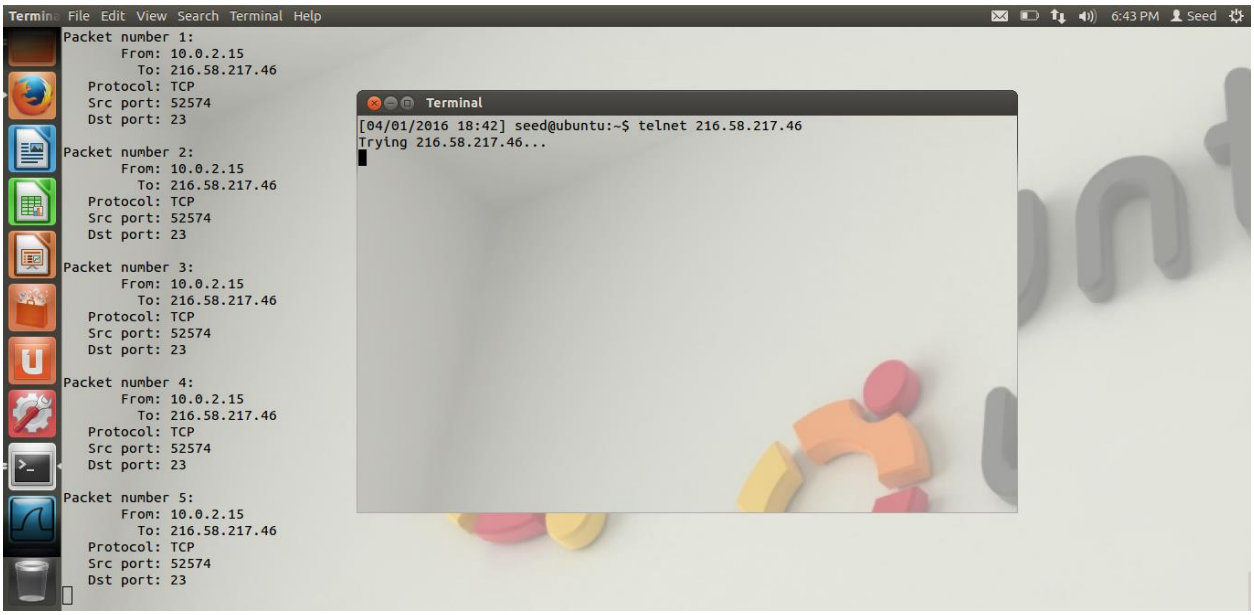
To perform this task, the filter expression needs to be only 23 because we are trying to capture all the telnet packets. Because TCP has a port number of 23. This can be done as follows:

```
sniffex.c
        /* define/compute tcp payload (segment) offset */
        payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);

        /* compute tcp payload (segment) size */
        size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);

        /*
         * Print payload data; it might be binary, so don't just
         * treat it as a string.
         */
        if (size_payload > 0) {
                printf("   Payload (%d bytes):\n", size_payload);
                print_payload(payload, size_payload);
        }

return;
}

int main(int argc, char **argv)
{

        char *dev = NULL;                       /* capture device name */
        char errbuf[PCAP_ERRBUF_SIZE];          /* error buffer */
        pcap_t *handle;                         /* packet capture handle */

        char filter_exp[] = "tcp port 23";              /* filter expression [3] */
        struct bpf_program fp;                  /* compiled filter program (expression) */
        bpf_u_int32 mask;                       /* subnet mask */
        bpf_u_int32 net;                        /* ip */
        int num_packets = 10;                   /* number of packets to capture */
```

We then start the telnetd server like the following:

```
[04/01/2016 17:59] seed@ubuntu:~/Desktop/Assignment5$ sudo service openbsd-inetd start
[sudo] password for seed:
 * Starting internet superserver inetd                                    [ OK ]
[04/01/2016 18:03] seed@ubuntu:~/Desktop/Assignment5$
```

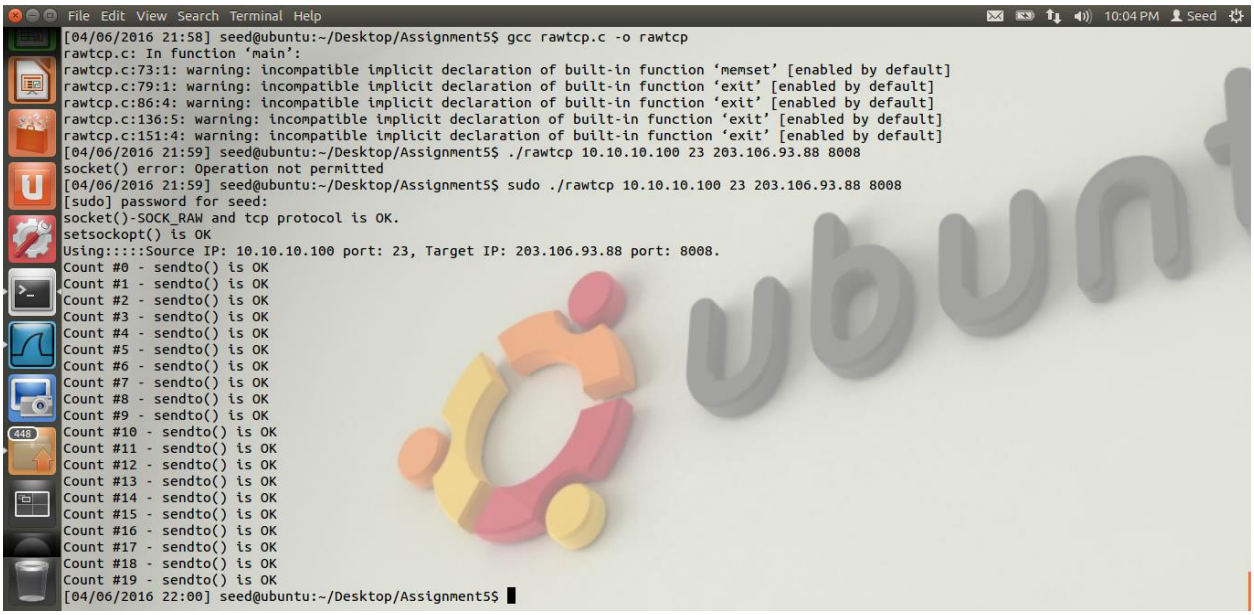
Now, when executing the sniffex program, we invoke an IP address to capture the TCP packets on the flow like the follows:



2.2 Task 2: Spoofing

Task 2.a: Write a Spoofing Program

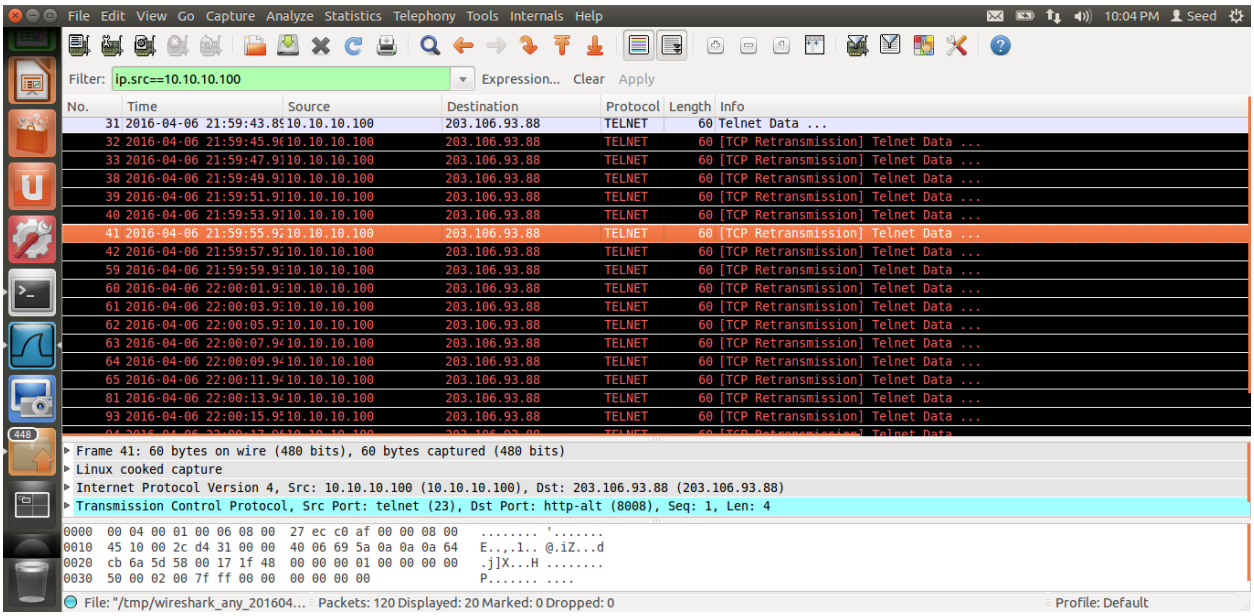
Here, I have used a program for spoofing which sends spoofed out packets to a destination with a different IP address. The output is similar to the below picture:



Here, the program takes 4 parameters as input through command line:

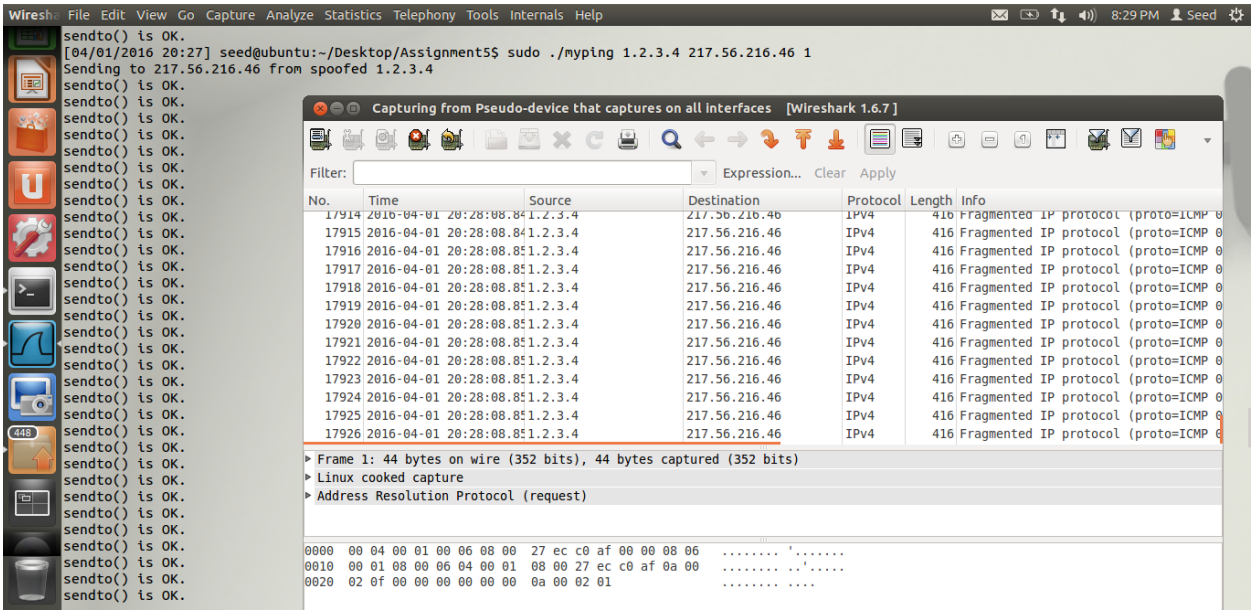
1. Source IP
2. Source port
3. Target IP
4. Target port

It means that the target IP received spoofed packets from the so called source ID which is not the genuine source. The Wireshark trace is shown here:



Task 2.b: Spoof an ICMP Request

Here, we try to spoof with an ICMP request. We try to ping the ICMP server to spoof the echo service it offers. The following output shows how it happens:



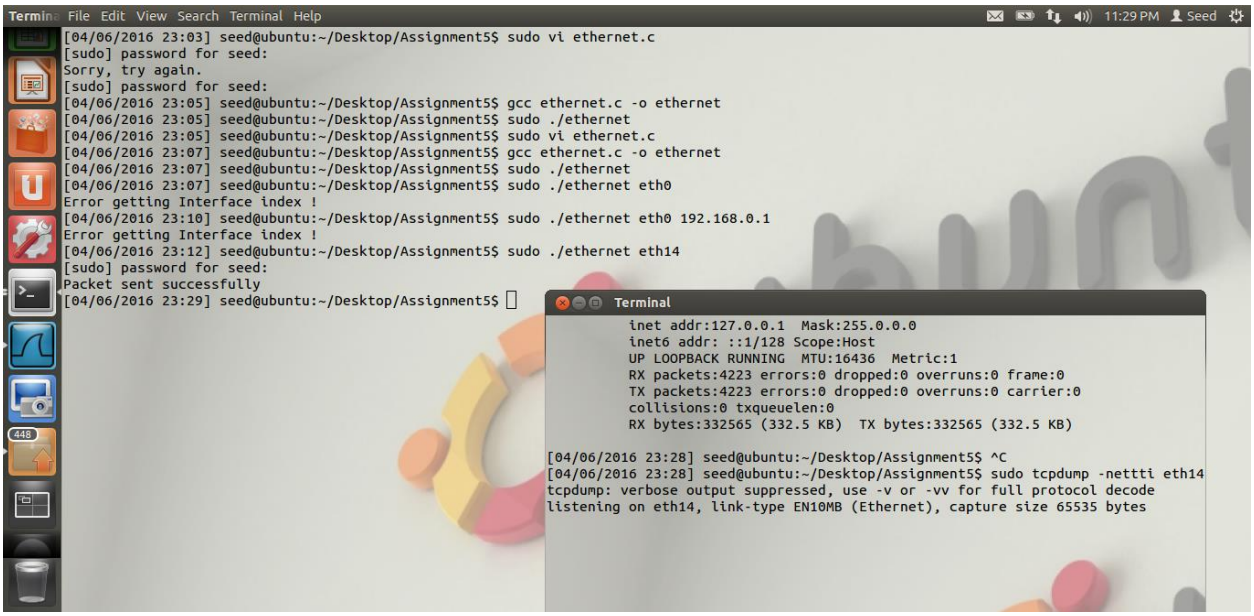
Here, the program takes 3 parameters as input through command line:

1. Source IP
2. Target IP
3. Number of packets to send

The Wireshark trace shows that the attack is working and runs successfully.

Task 2.c: Spoof an Ethernet frame

This was the toughest part to spoof an Ethernet frame. I have used the eth port which is already active on the system to get this work done. The eth port is used as the one that accepts packets and starts listening using tcpdump. Then, in the source program, I have used the eth port as a parameter with the input when I run the program. The output is as displayed:



Questions:

Question 4) Can you set the IP Packet length field to any arbitrary value, regardless of how big the packet is?

Answer: Yes, IP Packet can be initialized to any length no matter what. But, only the length of the raw socket size will be used from it. If the IP packet is 20 bytes and the Raw socket size is 10 bytes, the remaining 10 bytes will not be considered in the IP packet.

```
int SendRawPacket(int rawsock, unsigned char *pkt, int pkt_len)
{
        int sent= 0;

        /* A simple write on the socket ..thats all it takes ! */

        if((sent = write(rawsock, pkt, pkt_len)) != pkt_len)
        {
                /* Error */
                printf("Could only send %d bytes of packet of length %d\n", sent, pkt_len);
                return 0;
        }

        return 1;


}
```

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

Answer: Yes, we compute the checksum when doing socket programming.

```
unsigned short ComputeChecksum(unsigned char *data, int len)
{
        long sum = 0;  /* assume 32 bit long, 16 bit short */
        unsigned short *temp = (unsigned short *)data;

        while(len > 1){
            sum += *temp++;
            if(sum & 0x80000000)   /* if high order bit set, fold */
              sum = (sum & 0xFFFF) + (sum >> 16);
            len -= 2;
        }

        if(len)       /* take care of left over byte */
            sum += (unsigned short) *((unsigned char *)temp);

        while(sum>>16)
            sum = (sum & 0xFFFF) + (sum >> 16);

       return ~sum;
}
```

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?

Answer: We need to run the program only in root privilege.

```
int CreateRawSocket(int protocol_to_sniff)
{
        int rawsock;

        if((rawsock = socket(PF_PACKET, SOCK_RAW, htons(protocol_to_sniff)))== -1)
        {
                perror("Error creating raw socket: ");
                exit(-1);
        }

        return rawsock;
}
```

2.3 Task 3: Sniff and then Spoof

Here, the task is to combine both the above tasks to sniff and then spoof a target. For this, we need to VM's cloned and then the second VM to act as the destination. We will have to compile with –lpcap as we did in the 1st task and then run the program with the port as a parameter. In the other VM, we should trigger the ping command to the so called destination to make sure we receive the packets from the first one.
The below screenshots show how: