Computer Network Security
Lab 2
Sniffing and Spoofing using PCAP library

-PES1201801948
-Bharath S Bhambore
-Section H

Lab Setup :
Attacker Machine :
Machine Name : Ubuntu 16.04 - 1 [Black Terminal]
IP : 10.0.2.9

Victim Machine :
Machine Name : Ubuntu 16.04 - 2 [White Terminal]
IP : 10.0.2.10

Machine 3 :
Machine Name : Ubuntu 16.04
IP : 10.0.2.8

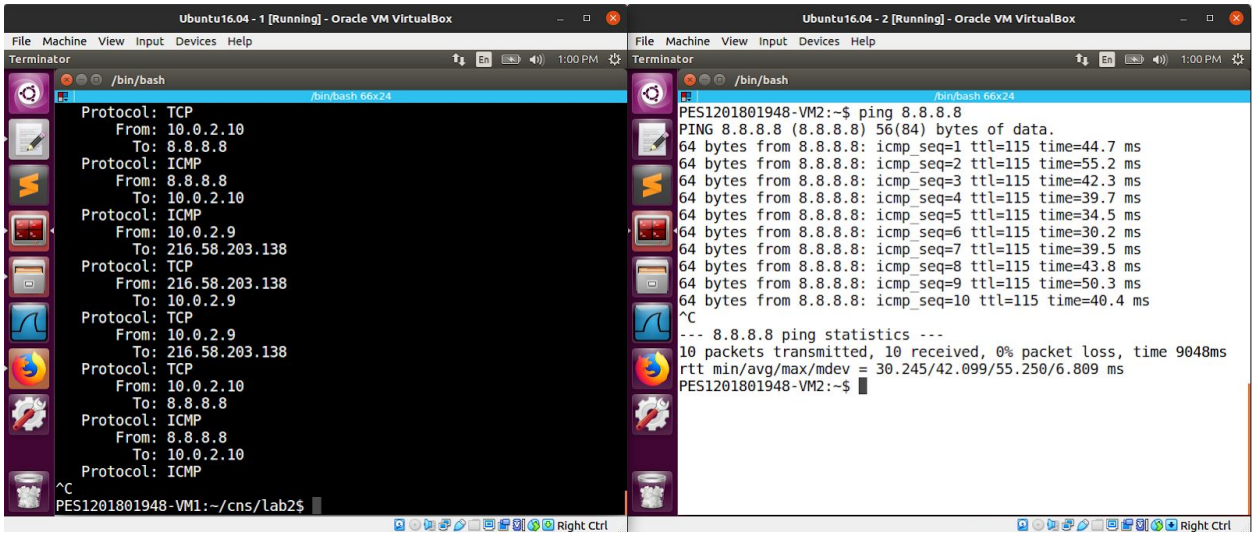
Task 1 : Writing Programs to Sniff and Spoof Packets using pcap library

Task 1.1: Writing Packet Sniffing Program :

Running the code :

Code given below : with promiscuous mode on

```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
/* Ethernet header */
struct ethheader {
  u_char  ether_dhost[6]; /* destination host address */
  u_char  ether_shost[6]; /* source host address */
  u_short ether_type;     /* protocol type (IP, ARP, RARP, etc) */
};
struct ipheader {
  unsigned char      iph_ihl:4, //IP header length
                     iph_ver:4; //IP version
  unsigned char      iph_tos; //Type of service
  unsigned short int iph_len; //IP Packet length (data + header)
  unsigned short int iph_ident; //Identification
  unsigned short int iph_flag:3, //Fragmentation flags
                     iph_offset:13; //Flags offset
  unsigned char      iph_ttl; //Time to Live
  unsigned char      iph_protocol; //Protocol type
  unsigned short int iph_chksum; //IP datagram checksum
  struct  in_addr    iph_sourceip; //Source IP address
  struct  in_addr    iph_destip;   //Destination IP address
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;

  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *)
                           (packet + sizeof(struct ethheader));

    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));
    printf("         To: %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 icmp";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

  // Step 2: Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);
  pcap_setfilter(handle, &fp);

  // Step 3: Capture packets
  pcap_loop(handle, -1, got_packet, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}
```



The attacker [left terminal] is successfully reading all the packets on the network sent by the victim client ping.

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.

-We firstly open a pcap live session which initializes and binds a socket along with the promiscuous mode setting [0 or 1], then write the filters that are to be used by the socket [ip protocols, src and dest addresses can be specified] which involve the pcap_compile() and pcap_setfilter() functions. Session is started using the pcap_loop(), the session is closed using the pcap_close() function call.

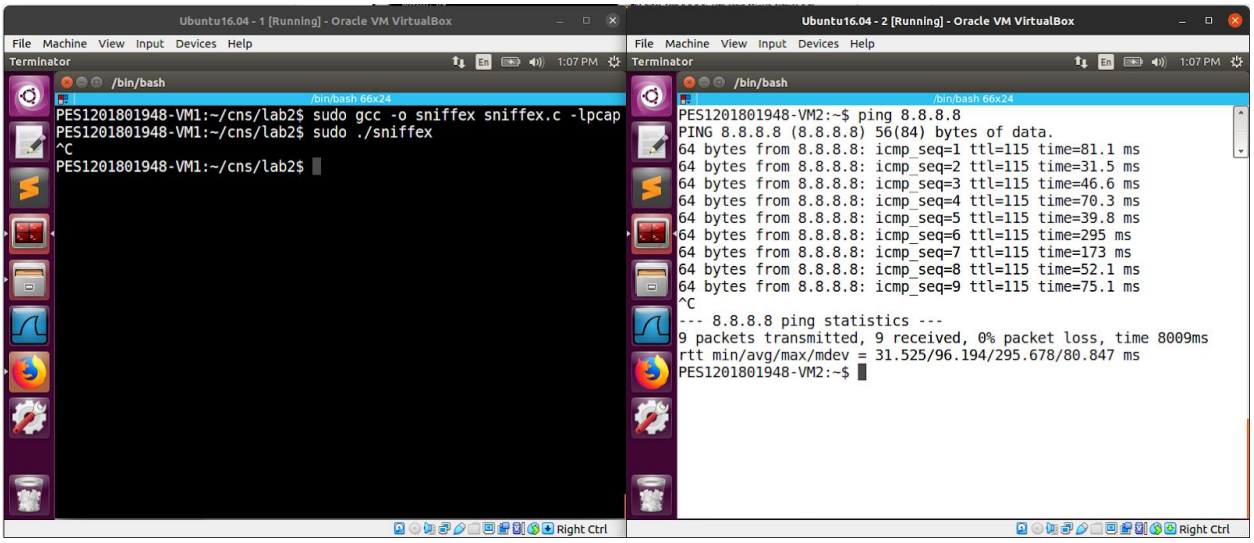
Problem 2: Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

-We need root privileges to initialize a packet, since the program needs to access the network interface card. Running without Sudo permission, we get a segmentation fault which fails at the pcap_open_live() function.

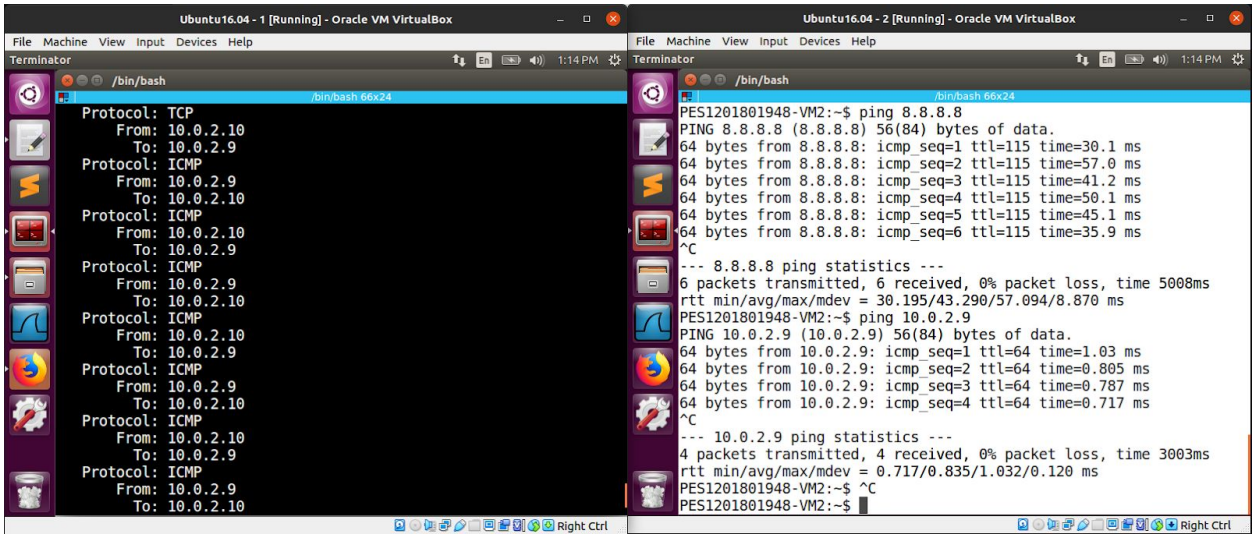
Problem 3: Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this

Promiscuous mode : enables the interface to read and intercept any packet on the interface.
Turning it off : setting the 3rd argument to 0

```
// Step 1: Open live pcap session on NIC with name enp0s3
handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);
```



As we can see, the attacker can’t sniff the packets passing on the network by the ping on the victim machine



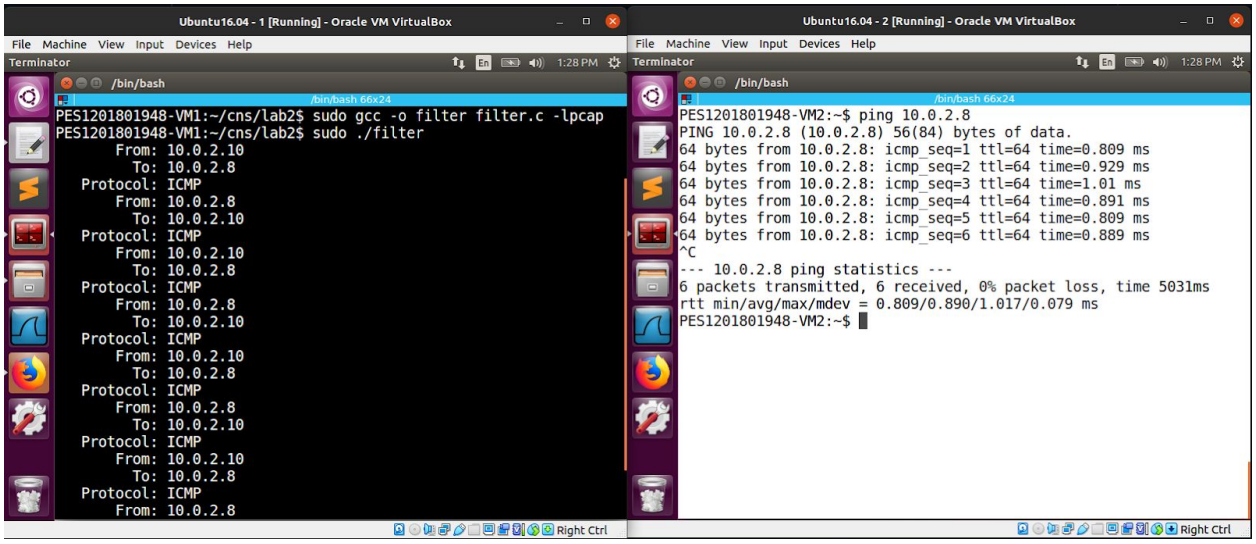
As we can see, the attacker captures icmp packets even when promiscuous mode is off only if the destination address in the ping is itself ie 10.0.2.9

Task 1.2B: Writing Filters

Capture the ICMP packets between two specific hosts

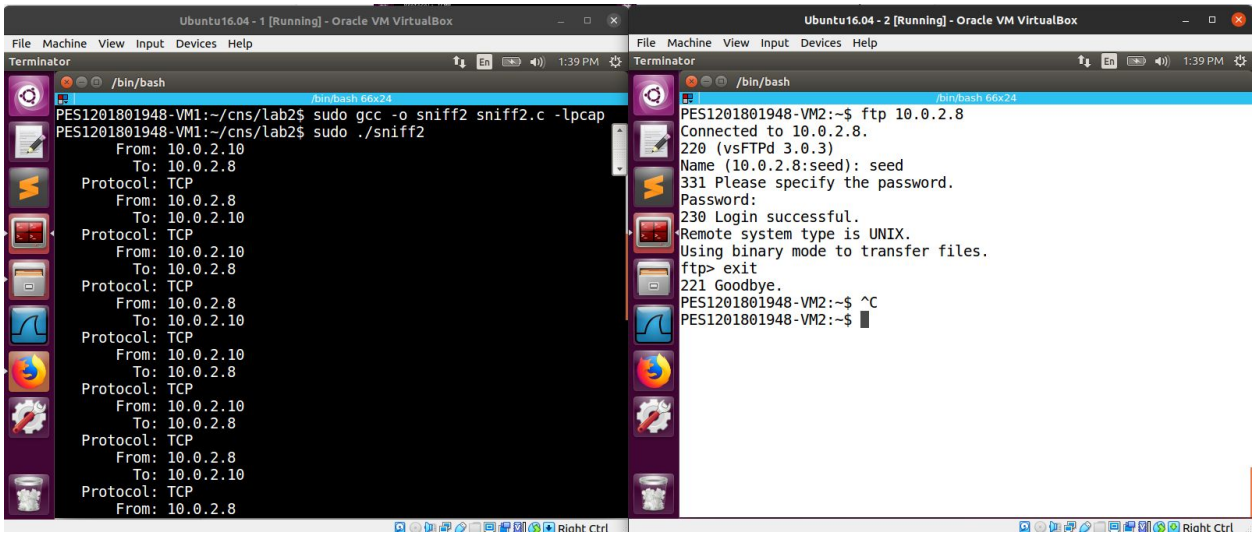
Filter expression used :
Src ip is victim machine,
dst ip is the machine 3

```
struct bpf_program fp;
char filter_exp[] = "ip proto icmp src host 10.0.2.10 and dst host 10.0.2.8";
bpf_u_int32 net;
```



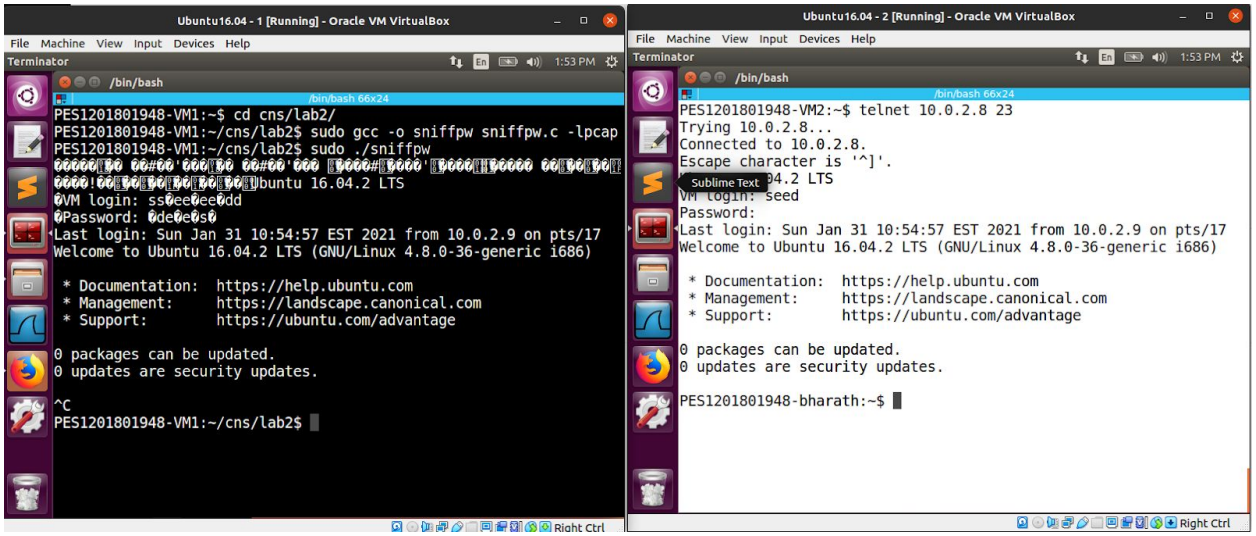
The attacker is able to view the packets sent from the victim machine [10.0.2.10]to machine 3 [10.0.2.8]

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



As we can see, the tcp packets are captured when an ftp request to the machine [10.0.2.8] is spun up from the victim machine [10.0.2.10].

Task 1.3C: Sniffing Passwords



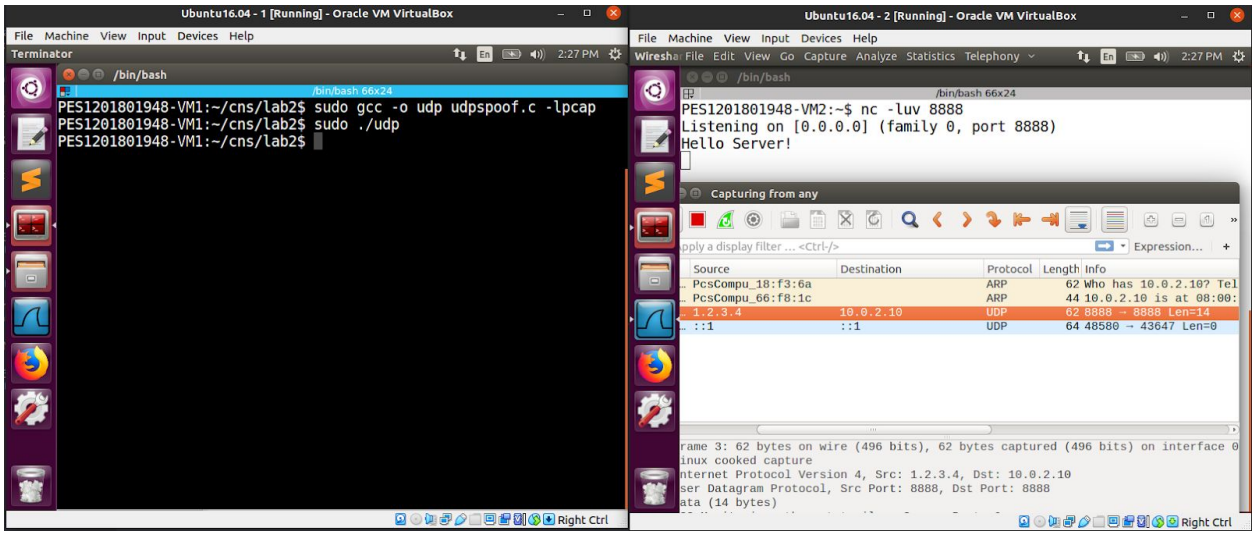
As we can see, we captured the tcp header of the telnet tcp packet thereby getting the login credentials using the below code.

Code :

```c
struct ethheader {
  u_char ether_dhost[6];
  u_char ether_shost[6];
  u_short 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;
};

typedef u_int tcp_seq;
struct tcpheader {
  u_short th_sport; /* source port */
  u_short th_dport; /* destination port */
  tcp_seq th_seq;   /* sequence number */
  tcp_seq th_ack;   /* acknowledgement number */
  u_char th_offx2;  /* data offset, rsvd */
  #define TH_OFF(th)  (((th)->th_offx2 & 0xf0) >> 4)
  u_char th_flags;
  #define TH_FIN 0x01
  #define TH_SYN 0x02
  #define TH_RST 0x04
  #define TH_PUSH 0x08
  #define TH_ACK 0x10
  #define TH_URG 0x20
  #define TH_ECE 0x40
  #define TH_CWR 0x80
  #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
  u_short th_win;   /* window */
  u_short th_sum;   /* checksum */
  u_short th_urp;   /* urgent pointer */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{ char *data;
  int i, size_tcp;
  struct ethheader *eth = (struct ethheader *)packet;
  if (ntohs(eth->ether_type) == 0x0800){
    struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
    int ip_header_len = ip->iph_ihl * 4;
    struct tcpheader *tcp = (struct tcpheader *)((u_char *)ip + ip_header_len);
    size_tcp = TH_OFF(tcp)*4;
    data = (u_char *)(packet + 14 + ip_header_len + size_tcp);
    printf("%s",data);
  }
}
int main(){
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "port 23";
bpf_u_int32 net;
// Step 1: Open live pcap session on NIC with name enp0s3
handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
```

The tcpheader structure defines all the fields in a TCP packet. Thereby helping us capture the data present in the TCP header which contains the login credentials.

Task 2 : Spoofing



We can observe that we successfully spoofed the UDP packet.
Looking at the wireshark capture, we can see we successfully sent out spoofed udp packets