Writing Programs to Sniff and Spoof Packets using pcap (C programs)

Table of Contents

1.1.1 Overview

Sniffer programs can be easily written using the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. At the end of the sequence, packets will be put in a buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the pcap library. Download Pcap lib Program from given link.

https://www.tcpdump.org/pcap.html.

1.1.2 Task 1: Sniffing - Writing Packet Sniffing Program

The objective of this lab is to understand the sniffing program which uses the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. Understanding sniffex. Please download the sniffex.c program from the tutorial mentioned above, compile and run it. You should provide screen dump evidence to show that your program runs successfully and produces expected results

Attacker machine: 10.0.2.9
Victim machine: 10.0.2.10
Server machine: 10.0.2.29

1.1.2.1 Task 1.1: Understanding how a Sniffer Works

In the below code, we write a sniffer function using the pcap API. It captures all ICMP packets using pcap.

```
int main(){

     pcap_t *handle;
             char errbuf[PCAP_ERRBUF_SIZE];
             struct bpf_program fp;

     char filter_exp[] = "proto ICMP and (host 10.0.2.10 and
10.0.2.2)";//"ip proto icmp";
             bpf_u_int32 net;

     // Step 1: Open live pcap session on NIC with name eth3
     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;
```

Show the captured packet processing. We use the packet and get the details of the packets. Here, we just print the source and destination address of the packet.

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
const u_char *packet)
{
              printf("   Got a Packet number \n", );
              struct ethheader *eth = (struct ethheader *)
packet;
     if(ntohs(eth - >ether_type) == 0x800)
     {
          Struct ipheader *ip = (struct ipheader *) (packet
          +sizeof(struct ethheader));
          printf("  From: %s\n", inet_ntoa(ip->ip_source));
          printf("  To: %s\n", inet_ntoa(ip->ip_destip));
              }
  }
```

Show that when a victim machine (10.0.2.9) sends ICMP packets to the destination address, our sniffer code sniffs all the ICMP packets sent on the network.

Promiscuous Mode On:

Command:

```
gcc -o sniffex sniffex.c -lpcap
sudo ./sniffex
```

Provide a screenshot of your observations.

open another terminal in same VM and ping any ip address

Command:

```
ping 1.2.3.4(any ipaddress)
```

Provide a screenshot of your observations.

Show that when promiscuous mode is switched on the sniffer program can sniff through all the packets in the network.

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 a detailed explanation like the one in the tutorial.

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

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 Off:

Promiscuous mode can be switched off in the attacker machine:

Go to Machine -> Settings -> Network -> Advanced -> Promiscuous mode -> “Deny”

Show that when promiscuous mode is switched off the sniffer program is not able to sniff through all the packets in the network.

Command:

```
gcc -o sniffex sniffex.c -lpcap
sudo ./sniffex
```

open another terminal in the same VM and ping any ip address

```
ping 8.8.8.8
```

Provide a screenshot of your observations.

Show that when promiscuous mode is switched off the sniffer program is not able to sniff through the packets whose destination address being that of the machine running the sniffer program. Write your observations.

When we send packets to a random address 8.8.8.8 from victim machine (10.0.2.10), the attacker using the sniffer program cannot capture the packets as the promiscuous mode is off since the NIC (hardware device) discards the packets that are not being sent to the sniffing machine. But if we send packets to the attacker machine (10.0.2.9) the sniffer program captures this packet since the destination is the 10.0.2.9.

1.1.2.2 Task 1.2: Writing Filters

The objective of this task is to capture certain traffic on the network based on filters. We can provide filters to the sniffer program. In pcap sniffer, when we have a sniffing session opened using “pcap_open_live”, we can create a rule set to filter the traffic which needs to be compiled. The rule set which is in the form of a string is compiled to a form which can be read by pcap. The rule set provided here sniffs the ICMP requests and responses between two given hosts. After compiling, the filter needs to be applied using pcap_setfilter () which preps the sniffer to sniff all the traffic based on the filter. Now, actual packets can be captured using pcap_loop().

i) Capture the ICMP packets between two specific hosts

1.1.2.2.1 In this task we capture all ICMP packets between two hosts. In this task, we need to modify the pcap filter of the sniffer code. The filter will allow us to capture ICMP packets between two hosts .

```
char errbuf[PCAP_ERRBUF_SIZE];

struct bpf_program fp;
char filter_exp [] =”proto ICMP and (host 10.0.2.10 and
10.0.2.2)”://”ip proto icmp”
```

Show that when we send ICMP packets to 10.0.2.2 from 10.0.2.10 using ping command, the sniffer program captures the packets based on the given filter.

Command:

```
gcc -o sniffb1 sniffb1.c -lpcap

sudo ./sniffb1
```

open another terminal in the same VM and ping any ip address

```
ping 10.0.2.2
```

Provide screenshots of your observations.

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

1.1.2.2.2 In this task we capture all TCP packets with a destination port range 10-100. Below screenshot shows the modified sniffer code with the required pcap filter.

```
char errbuf[PCAP_ERRBUF_SIZE];

 struct bpf_program fp;
 char filter_exp [] =”proto TCP and dst portrange 10-100”
```

We send FTP (runs over TCP) packets to the destination machine. As telnet runs over port 21, we should be able to capture all the packets sent with destination port 21.

Show the result of the sniffer program. It captures all the TCP packets with destination port 21.

Command:

```
sudo ./sniffb2
```

```
ftp 10.0.2.3
```

Provide screenshots of your observations.

Note: Observe Source port and Destination port using Wireshark capture.

1.1.2.3 Task 1.3: Sniffing Passwords

The objective of this task is to sniff passwords using the sniffer program. We will connect to a telnet server (running in our VM) and get the password of the user. Our telnet server is running on a machine with IP address 10.0.2.29 (Will be different for students).

Connect to the telnet server and then run the sniffer program, it sniffs all the telnet packets and we can see the password which the user types to login to the server. For telnet make sure you have port set to 23 or use port-range that contains 23.

Command:

telnet 10.0.2.29

Provide screenshots of your observations

1.1.3 Task 2: Spoofing

The objectives of this task is to create raw sockets and send spoof packets to the user/victim machine raw sockets give programmers the absolute control over the packet construction.

1.1.3.1 **Task 2.1 - A Writing a spoofing program**:

In this task we will create a spoofed IP packet and send it to the user machine using raw sockets.

A Filling in Data in Raw Packets When you send out a packet using raw sockets, you basically construct the packet inside a buffer, so when you need to send it out, you simply give the operating system the buffer and the size of the packet. Working directly on the buffer is not easy, so a common way is to typecast the buffer (or part of the buffer) into structures, such as IP header structure, so you can refer to the elements of the buffer using the fields of those structures. You can define the IP, ICMP, TCP, UDP and other header structures in your program. The following example shows how you can construct an packet with UDP data, UDP header and IP header with a random IP address as a source IP address and destination IP address of the machine, this entire packet is sent over the network:

```
void main() {

   char buffer[1500];

   memset(buffer, 0, 1500);
   struct ipheader *ip = (struct ipheader *) buffer;
   struct udpheader *udp = (struct udpheader *) (buffer +
                                                 sizeof(struct
ipheader));

   /*********************************************************
      Step 1: Fill in the UDP data field.
    *********************************************************/
   char *data = buffer + sizeof(struct ipheader) +
                         sizeof(struct udpheader);
   const char *msg = "Hello Server!\n";
   int data_len = strlen(msg);
   strncpy (data, msg, data_len);

   /*********************************************************
      Step 2: Fill in the UDP header.
    *********************************************************/
   udp->udp_sport = htons(12345);
```

```
   udp->udp_dport = htons(9090);
   udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    udp->udp_sum =  0; /* Many OSes ignore this field, so we do
not
                          calculate it. */

   /*********************************************************
      Step 3: Fill in the IP header.
    *********************************************************/
   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("10.0.2.5");
   ip->iph_protocol = IPPROTO_UDP; // The value is 17.
   ip->iph_len = htons(sizeof(struct ipheader) +
                       sizeof(struct udpheader) + data_len);

   /*********************************************************
      Step 4: Finally, send the spoofed packet
    *********************************************************/
   send_raw_ip_packet (ip);

}

/***************************************************************
  Given an IP packet, send it out using a raw socket.
***************************************************************/
void send_raw_ip_packet(struct ipheader* ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    if (sock <0)
    {
       perror("SOCKET CREATION ERROR");
       return;
    }
    else
    {
         printf("SOCKET CREATED\n");
    }
```

```
    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
                     &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    if(sendto(sock, ip, ntohs(ip->iph_len), 0,
            (struct sockaddr *)&dest_info, sizeof(dest_info))<0)
    {
        perror("PACKET NOT SENT\n");
        return;
    }

    close(sock);
}
```

Commands in VM1:

```
sudo ./udpspoof
```

Please provide a screenshot of your observations.

Commands in VM2:

We open a listener at the user end to listen to the connections and we see that the user machine receives the UDP packet sent by the spoof program.

```
nc -luv 8888
```

please provide the screenshot
Before doing nc command please open Wireshark to capture all the data

1.1.3.2 Task 2.2 – Spoof an ICMP Echo Request

In this task we will send a spoofed ICMP request to an existing live machine from a spoofed source IP address.

The code below shows how to create the ICMP packet. The spoofed request is formed by creating our own packet with the header specifications.Here we create an ICMP header with type=8(request) and checksum. Similarly, we will fill the IP header with the source IP address of any machine within the local network and destination Ip address of any remote machine on the internet which is alive.

```
/*************************************************************
******
   Spoof an ICMP echo request using an arbitrary source IP
Address
*************************************************************
******/
void main() {
   char buffer[1500];

   memset(buffer, 0, 1500);

   /*********************************************************
      Step 1: Fill in the ICMP header.
    *********************************************************/
   struct icmpheader *icmp = (struct icmpheader *)
                                        (buffer + sizeof(struct
ipheader));
     icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is
reply.

   // Calculate the checksum for integrity
   icmp->icmp_chksum = 0;
   icmp->icmp_chksum = in_cksum((unsigned short *)icmp,
                                sizeof(struct icmpheader));

   /*********************************************************
      Step 2: Fill in the IP header.
    *********************************************************/
   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("10.0.2.5");
    ip->iph_protocol = IPPROTO_ICMP;
    ip->iph_len = htons(sizeof(struct ipheader) +
                        sizeof(struct icmpheader));
    /*****************************************************
       Step 3: Finally, send the spoofed packet
     *****************************************************/
    send_raw_ip_packet (ip);
}
```

Commands in VM1 :

```
sudo ./icmpspoof
```

please provide the screenshot of your observation

1.1.3.3 Task 2.3 – Sniff and then Spoof

In this task, the victim machine pings a non-existing IP address “1.2.3.4”. As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.
VM A (Victim) : 10.0.2.24
Ping X : 1.2.3.4
VM B (attacker with sniffer-spoofer running): 10.0.2.28

In the below code, we use the PCAP API to listen to the traffic promiscuously and spoof the response if the protocol is ICMP. The code is the same as the one we used for Sniffing.

```
/*********************************************
 * Packet Capturing using raw libpcap
 *********************************************/

#include <pcap.h>
#include <stdio.h>

void 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 eth3
  handle = pcap_open_live("eth3", 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
}


/************************************************
 * Get captured packet
 ************************************************/
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>

/* IP Header */
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));

    /* determine protocol */
    switch(ip->iph_protocol) {
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");
            return;
        case IPPROTO_UDP:
            printf("   Protocol: UDP\n");
            return;
        case IPPROTO_ICMP:
            printf("   Protocol: ICMP\n");
            return;
        default:
            printf("   Protocol: others\n");
            return;
    }
  }
}
```

In the below function we create a buffer of maximum length and fill it with an IP request header. We modify the IP header and ICMP header with our response data. In the new IP header, we interchange the source IP address and destination IP address and send the new IP packet using the raw sockets.

```
/*********************************************
 * Listing 12.8: Spoofing a UDP packet based on a captured UDP
packet
 *********************************************/

void spoof_reply_icmp(struct ipheader* ip)
{
    const char buffer[1500];
    int ip_header_len = ip->iph_ihl * 4;


    // Make a copy from the original packet
    memset((char*)buffer, 0, 1500);
    memcpy((char*)buffer, ip, ntohs(ip->iph_len));

    // Construct icmp header
    struct ipheader  * newip  = (struct ipheader *) buffer;
     struct icmpheader * newicmp = (struct icmpheader *) (buffer
+ ip_header_len);

   newicmp->icmp_type=0; //0 for reply
```

```
   newicmp->icmp_chksum =0;
     newicmp->icmp_chksum=  in_cksum((unsigned  short  *)  newicmp,
sizeof(struct icmpheader));
   newicmp->icmp_seq = count++;


    // Construct the IP header (no change for other fields)
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 50; // Rest the TTL field
    newip->iph_len = ip->iph_len;

    // Send out the spoofed IP packet
    send_raw_ip_packet(newip);
}
```

we should see that the attacker machine sends the spoofed IP packets to the victim machine. The victim machine receives an ICMP reply from the Attacker machine for a non-existing IP address.

Command:

gcc -o sniffspoof sniffspoof.c -lpcap
sudo ./sniffspoof

Open one more terminal on the same VM and ping 1.2.3.4.

Submission:

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanations to the observations that are interesting or surprising. Please also list the important code snippets followed by an explanation. Simply attaching code without any explanation will not receive credits.