Assignment 2

Table of Contents

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

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 buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the pcap library. Tim Carstens has written a tutorial on how to use pcap library to write a sniffer program. The tutorial is available at http://www.tcpdump.org/pcap.htm.

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.

Task 1.1: Writing Packet Sniffing Program

The objective of this lab is to understand sniffing program which uses 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

Task 1.1A: Understanding how a Sniffer Works

In the below code, we write a sniffer function using 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));

        }

}
```

Command:

```
ping 10.0.2.2
```

Provide a screenshot of your observations.

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

Command:

```
sudo ./sniffex
```

Provide a screenshot of your observations.

Promiscuous Mode On:

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

Commands:

```
sudo gcc -o sniff sniff.c -lpcap

sudo ./sniff

ping 8.8.8.8
```

Provide screenshots of your observations.

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.

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
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 which destination address being that of the machine running the sniffer program.

Command:

```
ping 1.2.3.4
```

Provide screenshots of your observations.

```
gcc -o sniffex sniffex.c -lpcap
sudo ./sniffex
ping 10.0.2.4
```

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.

Task 1.2B: Writing Filters

The objective of this task 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().

Capture the ICMP packets between two specific hosts

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 10.0.2.10 and 10.0.2.9

```
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
```

Show that when we send ICMP packets to 10.0.2.2 from 10.0.2.24 using ping command, the sniffer program captures the packets based on the given filter.

Command:

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

sudo ./sniffb1

ping 10.0.2.2
```

Provide screenshots of your observations.

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

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";
bpf_u_int32 net;
// Step 1: Open live pcap session on NIC with name eth3
```

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

Show that 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.

Task 1.3C: 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 password of the user. Our telnet server is running in machine with IP address 10.0.2.

Task 2.2: 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

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

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 structres in your program. The following example show how you can construct an UDP packet:

```
struct ipheader {
    typefield;
    .....
}
struct udpheader { type field;
    ......
}
```

// This buffer will be used to construct rawpacket. char buffer[1024];

// Typecasting the buffer to the IP headerstructure struct ipheader *ip
= (struct ipheader *) buffer;

// Typecasting the buffer to the UDP headerstructure

```
 struct udpheader *udp = (struct udpheader *) (buffer+ sizeof(struct ipheader));
// Assign value to the IP and UDP headerfields. ip->field = ...;
 udp->field = ...;
```

commands in vm1:

sudo ./udpspoof

please provide the screen shot

commands in vm2:

nc -luv 8888

please provide the screen shot

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

Submission

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