Lab5 Packet Sniffing and spoofing

1 Problem Description

Packet sniffing and spoofing are two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. There are many packet sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software. The objective of this lab is for students to master the technologies underlying most of the sniffing and spoofing tools. Students will play with some simple sniffer and spoofing programs, read their source code, modify them, and eventually gain an in-depth understanding of the technical aspects of these programs. At the end of this lab, students should be able to write their own sniffing programs.

NOTE: this experiment needs to be carried out in a linux environment, if you don't have one, you may use virtual machines, you can use virtual box as the VM software and download pre-built virtual machine images from the following link:
http://www.cis.syr.edu/~wedu/seed/lab_env.html

2. Requirements

Task 1:

Writing Packet Sniffing Program 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.

Task 1a:

Understanding sniffex. Please download the sniffex.c program from the tutorial mentioned above, compile and run it. You should provide screendump evidence to show that your program runs successfully and produces expected results.

Please answer the following questions in your lab report:

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

Task 1b:

Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings. In your lab reports, you need to include screendumps to show the results of applying each of these filters.

• Capture the ICMP packets between two specific hosts.

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

Task 1.c:

Sniffing Passwords. Please show how you can use sniffex to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify the sniffex.c a little bit if needed. You also need to start the telnetd server on your VM.

If you are using our pre-built VM, the telnetd server is already installed; just type the following command to start it.

```
% sudo service openbsd-inetd start
```

Task 2 (**Extra credits:** this task is not a must, you get extra 5 points in your final score of this course by finishing this part)

When a normal user sends out a packet, operating systems usually do not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers). OSes will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc. However, if users have the root privilege, they can set any arbitary field in the packet headers. This is called packet spoofing, and it can be done through raw sockets. Raw sockets give programmers the absolute control over the packet construction, allowing programmers to construct any arbitrary packet, including setting the header fields and the payload. Using raw sockets is quite straightforward; it involves four steps: (1) create a raw socket, (2) set socket option, (3) construct the packet, and (4) send out the packet through the raw socket. There are many online tutorials that can teach you how to use raw sockets in C programming. We have linked some tutorials to the lab's web page. Please read them, and learn how to write a packet spoonfing program. We show a simple skeleton of such a program.

```
int sd;
struct sockaddr_in sin;
char buffer[1024]; // You can change the buffer size

/* Create a raw socket with IP protocol. The IPPROTO_RAW parameter
 * tells the sytem that the IP header is already included;
 * this prevents the OS from adding another IP header.  */
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sd < 0) {
    perror("socket() error"); exit(-1);
```

```
}

/* This data structure is needed when sending the packets
 * using sockets. Normally, we need to fill out several
 * fields, but for raw sockets, we only need to fill out
 * this one field */
sin.sin_family = AF_INET;

// Here you can construct the IP packet using buffer[]
//    - construct the IP header ...
//    - construct the TCP/UDP/ICMP header ...
//    - fill in the data part if needed ...
// Note: you should pay attention to the network/host byte order.


/* Send out the IP packet.
 * ip_len is the actual size of the packet. */
if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
              sizeof(sin)) < 0) {
      perror("sendto() error"); exit(-1);
}
```

Task 2.a:

Write a spoofing program. You can write your own packet program or download one. You need to provide evidences (e.g., Wireshark packet trace) to show us that your program successfully sends out spoofed IP packets.

Task 2.b:

Spoof an ICMP Echo Request. Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alieve). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.

Task 2.c:

Spoof an Ethernet Frame. Spoof an Ethernet Frame. Set 01:02:03:04:05:06 as the source address. To tell the system that the packet you constract already includes the Ethernet header, you need to create the raw socket using the following parameters:

```
sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
```

When constructing the packets, the beginning of the buffer[] array should now be the Ethernet header.

Questions. Please answer the following questions.

1. Can you set the IP packet length field to an arbitary value, regardless of how big the actual packet is?
2. Using the raw socket programming, do you have to calculate the checksum for the IP header?

3. Experimental Report

Your submission should include the following:

1. The finished codes
2. A detailed lab report

You need to submit a detailed lab report to describe what you have done and what you have observed; you also 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.