Packet Sniffing and Spoofing Lab

Due: February 11th by 11:59 PM

1 Overview

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 the source code, modify it, 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 and spoofing programs.

2 Before You Begin

2.1 Remove Old Virtual Machines

Make sure you have a clean VirtualBox configuration before you begin this lab. To do that follow the following steps:

1. Start by making sure you have no running virtual machines on your mcXX machine. To do that run the following command:

   ```
   VBoxManage list runningvms
   ```

   If this command shows any running virtual machines stop them with the following command:

   ```
   VBoxManage controlvm VMNAME poweroff
   ```

   Replace VMNAME with the name of the virtual machine to stop.

2. Now make sure that there are no VirtualBox processes running:

   ```
   ps aux | grep -v grep | grep VBox | grep $USER
   ```

   If this command output any running VBox processes make sure to kill them.

3. Now that you do not have any running virtual machines, to ensure you have a clean configuration run the following command and make it doesn't output anything:

   ```
   VBoxManage list vms
   ```

If the command outputs the existence of any virtual machines make sure you unregister and delete them with the following command:

```
VBoxManage unregistervm VMNAME --delete
```

Replace `VMNAME` with the name of the virtual machine to delete.

NOTE: this command will remove all contents of the virtual machine. Make sure to backup anything you want to save from the virtual machine before unregistering it.

2.2 Remove Old NAT Networks

Make sure you have removed all existing NAT networks before you begin the lab. To ensure you have a clean network configuration run the following command and make sure it doesn't output anything:

```
VBoxManage list natnets
```

If the command outputs the existence of any NAT network make sure you remove it with the following commands:

```
VBoxManage natnetwork stop --netname NETNAME
VBoxManage natnetwork remove --netname NETNAME
VBoxManage dhcpserver remove --netname NETNAME
```

Replace `NETNAME` with the name of the network you want to remove.

2.3 Stop VirtualBox Processes

At this point there shouldn't be any VirtualBox processes running, but to make sure you have a clean configuration run the following command to check for VirtualBox processes:

```
ps aux | grep $USER | grep VBox | grep -v grep
```

If any VirtualBox processes are listed kill them with the following command:

```
kill PID
```

Replace `PID` with the process identifier output from the `ps` command.

3 Lab Setup

For this lab you will need to import two (2) virtual machines. To import a virtual machine, follow the instructions in the `Lab_Setup.pdf`. Once you have two virtual machines imported, registered, and the natnework configured connect to each virtual machine and verify that each can ping an outside IP address (e.g ping www.google.com) and that each virtual machine can ping each other.

4 Lab Tasks

4.1 Task 1: Writing a 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 a 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 1.a: Understanding `sniffex`. In the cs528user home directory is a source file called `sniffex.c`. Instructions on how to compile and run it are within the file. Examine the contents of the sniffex.c file and familiarize yourself with its function. Once you understand how sniffex works answer the following questions.

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.

Task 1.b: Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings. In your lab report, 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 50 - 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 will need to modify the `sniffex.c` for this. The `telnetd` server should already be set up and running on the virtual machine image.

4.2 Task 2: Spoofing

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 arbitrary 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. The following is a skeleton program for sending a raw packet.

```
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. Using the information above as a start, write your own packet spoofing program. Use this program to perform the following tasks. You will need to submit this program along with your report.

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 alive).

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 construct 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.

Question 4: Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

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

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?

Question 7: Please use your own words to describe the sequence of the library calls that are essential for packet spoofing. This is meant to be a summary.

4.3 Task 3: Sniff and then Spoof

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two virtual machines on the same LAN. From virtual machine A, you `ping` an IP X. This will generate an ICMP echo request packet. If X is alive, the `ping` program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on virtual machine B, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the `ping` program will always receive a reply, indicating that X is alive. You will need to submit this program along with your report.

5 Guidelines

5.1 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 show how you can construct an UDP packet:

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

struct udpheader {
   type field;
   ......
}

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

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

// Typecasting the buffer to the UDP header structure
struct udpheader *udp = (struct udpheader *) (buffer
                            + sizeof(struct ipheader));

// Assign value to the IP and UDP header fields.
ip->field = ...;
udp->field = ...;
```

5.2 Network/Host Byte Order and the Conversions

You need to pay attention to the network and host byte orders. If you use an x86 CPU, your host byte order uses *Little Endian*, while the network byte order uses *Big Endian*. Whatever the data you put into the packet buffer has to use the network byte order; if you do not do that, your packet will not be correct. You actually do not need to worry about what kind of Endian your machine is using, and you actually should not worry about it if you want your program to be portable.

What you need to do is to always remember to convert your data to the network byte order when you place the data into the buffer, and convert them to the host byte order when you copy the data from the buffer to a data structure on your computer. If the data is a single byte, you do not need to worry about the order, but if

the data is a `short`, `int`, `long`, or a data type that consists of more than one byte, you need to call one of the following functions to convert the data:

```
htonl(): convert unsigned int from host to network byte order.
ntohl(): reverse of htonl().
htons(): convert unsigned short int from host to network byte order.
ntohs(): reverse of htons().
```

You may also need to use `inet_addr()`, `inet_network()`, `inet_ntoa()`, `inet_aton()` to convert IP addresses from the dotted decimal form (a string) to a 32-bit integer of network/host byte order. Refer to the "man" pages for more information.

5.3 Network Configuration for Task 3

The configuration mainly depends on the IP address you ping. Please make sure to finish the configuration before the task. The details are attached as follows:

- When you ping a non-existing IP in the same LAN, an ARP request will be sent first. If there is no reply, the ICMP requests will not be sent later. So in order to avoid that ARP request which will stop the ICMP packet, you need to change the ARP cache of the victim virtual machine by adding another MAC to the IP address mapping entry. The command is as follows:

  ```
  % sudo arp -s 192.168.15.1 AA:AA:AA:AA:AA:AA
  ```

  IP `192.168.15.1` is the non-existing IP on the same LAN. `AA:AA:AA:AA:AA:AA` is a spoofed MAC address.

6 Submission

6.1 What to Submit

Your submission directory should contain:

- All source files written by you to perform the lab tasks.
- A README text file describing how to compile and execute your source code.
- A detailed report containing an explanation of the observations. Name this report *Analysis-lab1.pdf*. Be sure to put your name in your report.

6.2 How to submit

Submit every time you have a stable version of any part of the functionality. Make sure to submit early to make sure you do not miss the deadline due to any last minute congestion.

1. Login or ssh to an mcXX machine, e.g., mc01.cs.purdue.edu

2. In the parent directory of your submission directory, type the command:

   turnin -c cs528 -p lab1 <submission-dir-name>

   where <submission-dir-name> is the name of the directory containing your files to be submitted. For example, if your program is in a directory `/homes/abc/assignment/src`, make sure you cd to the directory `/homes/abc/assignment` and type:

   turnin -c cs528 -p lab1 src

3. If you wish to, you can verify your submission by typing the following command:

   turnin -v -c cs528 -p lab1

   Do not forget the **-v** above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission).

   Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

   We will check that the submission time-stamp is **before the due date**; we will not accept your submission if its time-stamp is after the due date, even by 3 minutes. Do NOT submit after 11:59 PM Eastern Time.

   NOTE: Do not submit your virtual machine disk image. Copy the files for submission off of your virtual machine and into a directory on the mcXX machine.