Packet Sniffing and Spoofing Lab

Modified for COMP435, Fall 2017 by Cynthia Sturton, UNC-Chapel Hill.

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`, `Networx`, 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 and spoofing programs.

2 Lab Tasks

2.1 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 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`. 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. You can retrieve and compile the program in terminal using the below commands.

```
wget https://www.tcpdump.org/sniffex.c
gcc -Wall -o sniffex sniffex.c -lpcap
```

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: 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. Take a screenshot of the password and enter it into your report along with a description of what is displayed.

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

2.2 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 arbitary field in the packet headers. This is called packet spoofing, and it can be done through *raw sockets* when using the C programming language. Alternatively, you can create and set the fields of protocol headers using a Python 2 package called Scapy. We will learn more about Scapy in a bit, but for now let us give a description of how this works in C.

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. We have linked some tutorials to the lab's web page. Please read them, and learn how to write a packet spoofing 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. For this task, you are going to spoof a TCP packet using the Scapy Python module. Let your destination address be the localhost and your source address by any valid IP address that is not the same as the localhost. To install Scapy in your VM, use the command: `sudo apt-get install python-scapy`. You can find out detailed information on Scapy's documentation at: `https://scapy.readthedocs.io/en/latest/introduction.html`
The name of your Python program should be: `a5_onyen_tcpspoof.py`

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). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine. The name of your Python program should be: `a5_onyen_icmpspoof.py`

Task 2.c: Spoof an Ethernet Frame. In this task you will spoof an Ethernet Frame. From Wikipedia: "A data packet on an Ethernet link is called an Ethernet packet, which transports an Ethernet frame as its payload. An Ethernet frame is preceded by a preamble and start frame delimiter (SFD), which are both part of the Ethernet packet at the physical layer. Each Ethernet frame starts with an Ethernet header, which contains destination and source MAC addresses as its first two fields." Set 01:02:03:04:05:06 as the source address. When using Scapy, set 1:02:03:04:05:06 as the "src" field in the Ether() object.

Questions. Please answer the following questions.

Question 4: Can you set the IP packet length field to an arbitary value, regardless of how big the actual packet is? What happens when you send it?

Question 5: Using the raw socket programming in C, do you have to calculate the checksum for the IP header? What if you use python's Scapy package?

Question 6: Why do you need the root privilege to run the programs that use Scapy/raw sockets? Where does the program fail if executed without the root privilege?

2.3 EXTRA CREDIT Task 3: Sniff and then Spoof (3 pts final grade)

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two VMs on the same LAN. From VM 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 VM 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 need to write such a program, and include screendumps in your report to show that your program works. Please also attach your code (along with adequate amount of comments) to your report when you submit to Sakai. If you use Scapy, look into using the "sniff" function to receive packets.

3 Guidelines

3.1 VirtualBox Network Configuration for Task 3

In order to complete this task, you will need to configure two VMs in a LAN. The configuration mainly depends on the IP address you ping. Please make sure 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 VM by adding another MAC to the IP address mapping entry. The command is as follows:

  ```
  % sudo arp -s 192.168.56.1 AA:AA:AA:AA:AA:AA
  ```

  IP `192.168.56.1` is the non-existing IP on the same LAN. `AA:AA:AA:AA:AA:AA` is a spoofed MAC address.

- When you ping a non-existing IP outside the LAN, the Network settings will make a difference.

  If you are using the new "NAT Network", no further changes are needed. If your VMs are connected to two virtual networks (one through the "NAT" adapter, and the other through the "Host-only" adapter), there are two things you need to keep in mind to finish the task:

  – If the victim user is trying to ping a non-existing IP outside the local network, the traffic will go to the NAT adapter. VirtualBox permanently disables the promiscuous mode for NAT by default.

  Here we explain how to address these issues:

  Redirect the traffic of the victim VM to the gateway of your "Host-only" network by changing the routing table in the victim VM. You can use the command `route` to configure the routing table. Here is an example:

```
% sudo route add -net 1.1.2.0 netmask 255.255.255.0 gw 192.168.56.1
```

The subnet `1.1.2.0/24` is where the non-existing IP sits. The traffic is redirected to `192.168.56.1`, which is on the same LAN of the victim but does not exist. Before sending out the ICMP request, again the victim VM will send an ARP request to find the gateway. Similarly, you should change the ARP cache as stated in the previous example.

Alternatively, you can setup your two VMs using "Bridged Network" in VirtualBox to do the task. "Bridged Network" in VirtualBox allows you to turn on the promiscuous mode. The following procedure on the VirtualBox achieves it:

```
Settings -> Network -> Adapter 1 tab -> "Attach to" section:
Select "Bridged Adapter"

In "Advanced" section -> "Promiscuous Mode":
Select "Allow All"
```

If you decide to complete the extra credit, create a Python file and name it: `a5_onyen_extracredit.py`

4 Submission

Create a pdf report titled `a5_ONYEN_SniffingLab.pdf` and submit a detailed lab report to describe what you have done and what you have observed for questions 1-6. Also provide explanations to the observations that are interesting or surprising. Submit your files: `a5_onyen_tcpspoof.py`, `a5_onyen_icmpspoof.py`, and `a5_onyen_extracredit.py` if applicable. Please also sufficiently comment your code. Simply attaching code without any explanation will not receive credit.