Packet Sniffing and Spoofing Lab

This report was written with the help of various GitHub repository reports for the same SEED lab.

1.1A

The program is required to run with root privileges. When a webpage is visited, the packets are captured. If the program is executed with root(sudo) and error is thrown.

1.1B

In scapy, the sniff function has filters where filter="icmp && host 'some ip' && dst port 'port number' " would allow us to capture icmp on a specific ip address for a destination port number. Meanwhile if we wish to filter for a subnet the code would be filter="net subnet address"

1.2

From the code snippet, when we change the line in a, this allows us to spoof with any source IP address. Using wireshark, we can confirm that the source IP is the same as the IP that we typed into the code.

1.3

Using a for loop, we can automate the task where the ttl will increase by 1 each time. Initially, a small ttl value will not be able to reach the destination. So we will get error messages of the packet expired. Then when the destination is reached, we will be able to see how many hops it took. Of course, once the destination sends a reply back any further increase in the ttl will not matter anymore.

1.4

In the sniff spoof program, we capture all of the ping request from machine A to any address and send a reply from machine B where the programs in running. The program will take the ping destination request and spoof the reply back to machine A. Therefore, machine A believes that it has successfully reached the destination. However, it's just machine B sending a response back.

2.1

Executing the provided code will print to the console "Got a packet" every time it sniffs a packet using ICMP.

2.1A

Question 1: There are 2-3 library depending on the version of the code snippet. The code from the PDF has "stdio.h" and "pcap.h" while the online version has an additional library "stdlib.h".

2 of them are calls to the C standard library stdlib and stdio which are mainly used to print to console and to read files. “pcap” is a library hosted by libcap which is under TCPdump. It is used to capture packets.

Question 2: Root privileges are required to read all traffic on the network. Similar to task 1.1A, if the program is run without root privileges an error will be thrown.

Question 3: in the pcap library the promiscuous mode allows the program to spoof IP addresses as setting the value off will only allow the program to sniff packets going to or coming out of the machine.

2.1B

We can change the filter_exp[] to the filter requirements such as “icmp && host ip” or the protocol and port numbers “tcp && dst portrange 1-100”

2.1C

Since the telnet is unencrypted, we can use the sniffing program to capture packets and filter the payload to read the data that is being transferred. The structure of the payload is known so printing the data into human readable ascii would simply be to convert the binary bits to hexadecimals then doing an ascii lookup.

2.2

From the new version of the SEED labs, there are implemented code snippets that can be modified to update the destination IP or port numbers you are looking to attack.

2.2A

Using the provided code in part 2.2, simply opening wireshark allows us to determine that the packets sent out have a source IP that we specified in the program thus ensuring that the spoofing is working accordingly.

2.2B

Question 4: The packet length cannot be set to any arbitrary value since many fields have to setup in accordance with the IP protocol.

Question 5: There is no need to calculate the checksum of an IP packet. The OS/system will fill that part of the IP header when it is sent out.

Question 6: When using raw sockets, it allows the program to read any and all packets both from the user that executes the program and any other packets on the machine. As such, root privilege is required to run those programs. If the permissions are incorrect, then an error will be thrown.

2.3

Using the previous tasks of spoofing, we can modify the function in got_pakcet() to process the ICMP echo coming from the other machine. In this case, we will have to calculate the checksum of the captured packet to properly sent the reply back to the machine that put out the initial ICMP message.

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    printf("Got a packet\n");
    char buffer[1500];
    struct ipheader *captured_ip = (struct ipheader *)(packet + sizeof(struct ethheader));
    struct icmpheader *captured_icmp = (struct icmpheader *)(packet + sizeof(struct ethheader) + sizeof(struct ipheader));

    memset(buffer, 0, 1500);

    struct icmpheader *icmp = (struct icmpheader *)(buffer + sizeof(struct ipheader));

    //ICMP Type: 8 is request, 0 is reply.
    icmp->icmp_type = 0;

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

    //rebuild IP packet
    struct ipheader *ip = (struct ipheader *)buffer;
    ip->iph_ver = 4;
    ip->iph_ihl = captured_ip->iph_ihl;
    ip->iph_ttl = 20;
    ip->iph_sourceip = captured_ip->iph_destip;
    ip->iph_destip = captured_ip->iph_sourceip;
    ip->iph_protocol = IPPROTO_ICMP;
    ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));

    send_raw_ip_packet(ip);

    printf("Send out spoofed reply\n");
}
```