CREATING A PACKET TRACER USING RAW-SOCKET IN LINUX

Sulabh Kumar Jain

School of Computer Science and Engineering

Vellore Institute Of Technology.

Chennai,India

sulabhkumar.jain2019@vitstudent.ac.in

Abstract— **In this paper, we will discuss the implementation of Packet Sniffer tool. For this purpose, I will use the Linux Operating System, and will create a sniffer using C Language. The sniffer is a user application that is used to capture and display the packet information, such as the TCP header, UDP header, IP header, Ethernet header, and of-course the payload, that is sent when electronic devices communicate.**

Linux provides two methods of creating a packet sniffer. First, the libpcap library, and second, by using a raw socket. There are various tools available in the market such as Wireshark, TCPdump that are built using the libcap library. In this paper, we will discuss the implementation of Packet Sniffer using the Raw Socket Method, which is said to be a faster method, performance wise.

As you read through this paper, you will learn to convert your Network Interface Card to the promiscuous mode, which makes the NIC to accept all the packets even if the packet does not contain its MAC address as the destination.

Keywords—Packet-Sniffer;Raw-Socket;Linux;Sniffing;Promiscuous;NIC;

Introduction

With the advancement of the internet, a person in one part of the world can easily communicate with another person at another part of the world in a matter of milliseconds. A person can log into his facebook account, and will be connected with all his friends no matter where there are. He can access google and search for a page, and hundreds of pages will be at it his disposal in a second. Have you wondered the process behind how this communication takes place? Firstly, any message that is being communicated travels inform of packets, whether it be through a wire, or using a wireless communication. If a PC A wants to communicate with PC B, it will do so by sending messages inform of packets which will be received and read by PC B. With the increase in the internet user, there has been a significant increase in the transmission of such packets. Now comes the role of a Packet Sniffer. Sniffers have two components, packet capture and packet analyzer. Packet capture, as the name suggests is used to capture the packets, and the analyzer is used to store and analyze the information contained in those captured packets. Packet sniffers are used by network administrators to monitor the activity of the users in the network. It helps the administrator to examine if a user tries to search for an illegal website. It also enables the administrator to find the cause of network congestion, if any. The tool can also be used by a hacker to sniff into activity of the users in the network.

I. SIMILAR WORKS:

The paper Design and Implementation of V6SNIFF: an Efficient IPv6 Packet Sniffer have analysed the performance of the IPV6 packet sniffer using the libpcap library and the raw socket in the Linux Environment.

Packet Sniffing for Automated Chat Room Monitoring and Evidence Preservation has discussed how a packet sniffer can be used to automatically monitor and filter the chat rooms, and store any malicious activity in the database for further investigation or as a proof for such unsolicited activity.

A New Hybrid Network Sniffer Model Based on Pcap Language and Sockets (Pcapsocks) discusses libpcap whose main task is to capture packets in lower level. Libcap provides filtering options such as BPF filter. Libnet manipulates mostly the high level packets, and also the low level modules. They do not provide filtering facilities for filtering the packets.The paper has come up with a solution to sniff packets in the lower level, that is the physical layer and the data-link layer of the OSI model. They have also included the Intrusion Detection and Prevention System, and also storing of data received.

Packet Sniffing: A Brief Introduction This paper has come up with the solution of how to convert an NIC into promiscuous mode after which it will be able to sniff any packets in the network, even if they are not destined for the host. This paper have used Linux Environment and C-language to do so.

Packet Sniffer This paper gives us the basic idea of how a packet sniffer works. Firstly, frame is captured in the data-link layer. Secondly, the frames are unpacked and all the necessary information is extracted from the frame. The payload in the data-link layer is the data received from the network layer(including the network layer data). Now, the network layer is unpacked to get the header information of the network layer. This goes on and the headers of the respective layers is extracted by the that layer.

There have been other papers as well that have all discussed the basic principle of packet sniffer and how they can be used with host connected in switched, hubs, etc. They have all come up with how you can check if your system is being sniffed, and how to avoid it. For my paper, I have created a sniffer from scratch using the raw socket in the linux environment.

II. ARCHITECTURE:

This paper deals with the implementation of Packet Sniffer using the RAW socket. There are other methods to do the same, such as using the "libpcap library". There are already available tools in the market, such as the wireshark, tcpdump, which used the library. The difference between the two methodology is given in the methodology section in the form of a diagram. We will use the raw socket which will use AF_PACKET as the domain. This will help the user to extract the entire packet to the user-space including the data-link header and trailer. We can also manually create the entire packet without the kernel doing it for us, but that is not the scope of this paper. Once we have extracted the packet, we will create the pre-defined structure to store the information included in the header and the payload. We will extract information such as the source and destination ip and mac address, IPV4 header as a whole, tcp/udp header, ethernet header, and the payload.

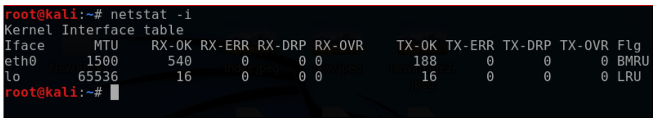
The sniffer can be created for both wireless and wired connection. If the host is connected using ethernet to the network, you can use the ethernet interface which is usually "eth0" in the linux environment. For host using wireless connection, you can use the interface which is dedicated to the wireless connection, which is usually "wifi0" in a linux system. You can check the interfaces and their respective information using the "ifconfig" command in the terminal.

III. METHODOLOGY:

Promiscuous Mode:

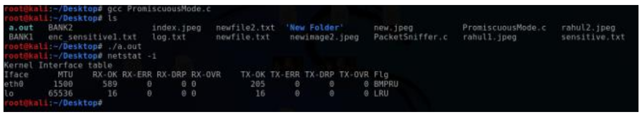
Every communicating device has a Network Interface Card which acts as an entry and exit point of a packet into or from the host device. The NIC checks for the destination IP address in the incoming packet. If the IP does not match with the IP of the host, the packet is dropped. A packet sniffer converts the NIC into PROMISCUOUS MODE. Now, the NIC will accept all the packets, even if the packet is not meant for the host.

OUTPUT BEFORE RUNNING THIS PROGRAM



```
root@kali:~# netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500      540      0      0 0           188      0      0      0 BMRU
lo       65536       16      0      0 0            16      0      0      0 LRU
root@kali:~# 
```

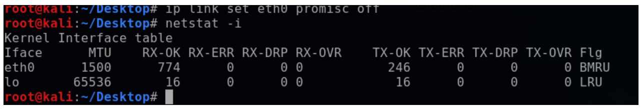
OUTPUT AFTER RUNNING THE PROGRAM



```
root@kali:~/Desktop# gcc PromiscuousMode.c
root@kali:~/Desktop# ls
 a.out   BANK2                index.jpeg   newfile2.txt  'New Folder'     new.jpeg         PromiscuousMode.c   rahul2.jpeg
 BANK1   enc_sensitive1.txt   log.txt      newfile.txt    newimage2.jpeg  PacketSniffer.c  rahul1.jpeg         sensitive.txt
root@kali:~/Desktop# ./a.out
root@kali:~/Desktop# netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500      589      0      0 0           205      0      0      0 BMPRU
lo       65536       16      0      0 0            16      0      0      0 LRU
root@kali:~/Desktop#
```

Here, you will notice that the "P" flag has been added to the "eth0" interface which means the ethernet interface has been converted to promiscuous mode.

Converting back to default mode after doing the task



```
root@kali:~/Desktop# ip link set eth0 promisc off
root@kali:~/Desktop# netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500      774      0      0 0           246      0      0      0 BMRU
lo       65536       16      0      0 0            16      0      0      0 LRU
root@kali:~/Desktop# 
```

CONVERTING THE NIC INTO PROMISCUOUS MODE:

The "ifreq" structure deals with the network devices. The user must fill the "ifr_name" of the ifreq structure(like eth0 for ethernet or wifi0 for wireless connection). The user will use the "ioctl()" system call. The ioctl() in its second parameter takes certains flag which inturn tells the ioctl() to perform certain task. In our case, we will provide the SIOCGIFFLAGS flag which will return all the flags of each and every interface.Now, you must set the "promisc flag" of the "ifreq" structure. This will make the NIC into promiscuous mode.

Ifr.ifr_flags |= IFF _PROMISC;

Finally, set this flags to the NIC using the "ioctl()" system call. This time, the second parameter will be SIOCSIFFLAGS

Socket

This is the endpoint of communication. If any two devices wants to communicate with each other, they must open their sockets. Any communication will happen between the two sockets.

int sockfd = socket(domain,type,protocol)

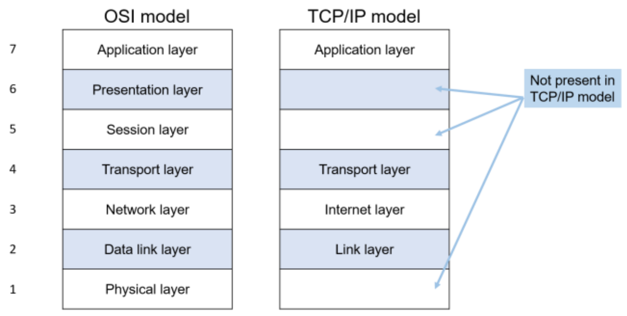
When a user creates a normal socket, all he has to do is pass the data to the socket buffer. The operating system will append the headers as follows:

As the date passes down the layer for transmission, the Transport Layer will add it header, followed by the Network Layer header. The Data-link layer will add it header and trailer, and pass the packets for transmission through the wire, or wireless medium. When the packet is received by the destined device, its Data-link layer will first remove the header and trailer appended by the data-link layer of the source device. The packet will then move up the layer to the Network layer which will remove its respective header, and send the packet to the Transport layer which in turn will remove it header, and send the data to the intended application in the Application Layer.

From this type of socket creation, we can come into conclusion that the user receives only the data and not the headers.

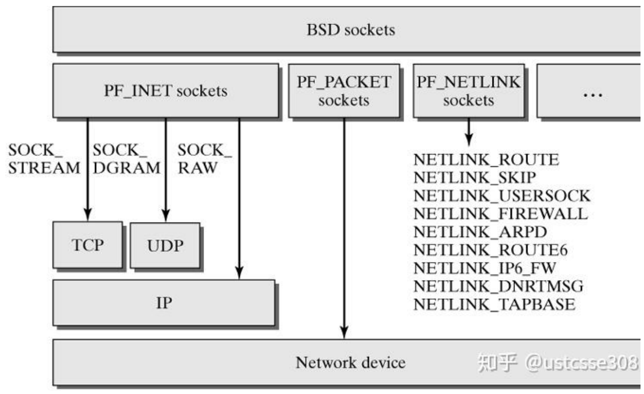
TCP Model:

This is a four layer model, built over the OSI model, on the basis of encapsulation and decapsulation of packets. This model is used by all the devices as this is a common agreement by which the devices can communicate with each other.

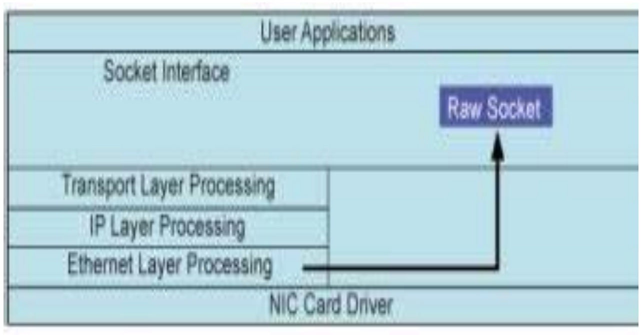


Raw Socket:

In the normal socket creation, we had learned that the user cannot write or read any header, as this was only manipulated by the kernel. Raw socket, however, allows the user to read and write data/headers. This will allow the user to create and read the entire packet from the application layer.



Normal BSD socket



RAW socket

From the above images, we can conclude that the packets that are created using raw sockets does not pass through the networking stack of the kernel, and as a result cannot be manipulated by the layers.

Creating a RAW socket

int sockfd = socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL));

SOCK_RAW: This socket is a raw socket.

AF_PACKET and ETH_P_ALL: These allow the entire packet, no matter of which protocol they belong to, to be accessed by the user application (inclusive of the ethernet header).

Accessibility of Raw Socket:

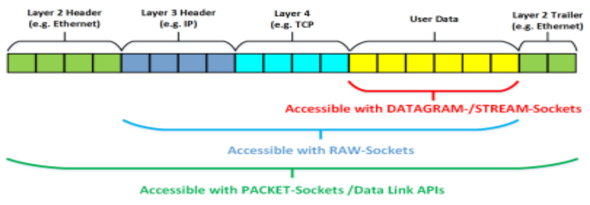

Layer 2 Header (e.g. Ethernet)
Layer 3 Header (e.g. IP)
Layer 4 (e.g. TCP
User Data
Layer 2 Trailer (e.g. Ethernet)
Accessible with DATAGRAM-/STREAM-Sockets
Accessible with RAW-Sockets
Accessible with PACKET-Sockets /Data Link APIs

Figure 2: Overview over the layers and access possibilities

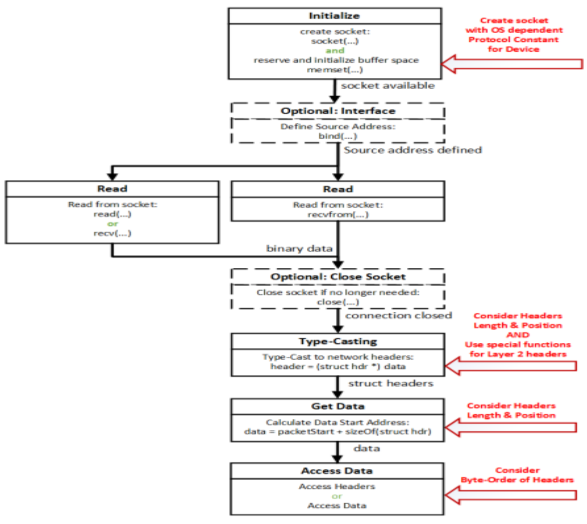

Initialize
create socket:
socket(...)
and
reserve and initialize buffer space
memset(...)
Create socket with OS dependent Protocol Constant for Device
socket available
Optional: Interface
Define Source Address:
bind(...)
Source address defined
Read
Read from socket:
read(...)
or
recv(...)
Read
Read from socket:
recvfrom(...)
binary data
Optional: Close Socket
Close socket if no longer needed:
close(...)
connection closed
Consider Headers Length & Position AND Use special functions for Layer 2 headers
Type-Casting
Type-Cast to network headers:
header = (struct hdr *) data
struct headers
Get Data
Calculate Data Start Address:
data = packetStart + sizeOf(struct hdr)
Consider Headers Length & Position
data
Access Data
Access Headers
or
Access Data
Consider Byte-Order of Headers

Figure 7: Structure of a PACKET-socket layer 2 Read Operation

After doing the above steps, all the user has to do is to get the pointer to the start of the header, and typecast it to struct datatype of the respective header.

This structure contains the ethernet header of the packet.

struct ethhdr *eth = (struct ethhdr *)(buffer);

This structure contains the ip header of the packet.

struct iphdr *ip = (struct iphdr*)(buffer + sizeof(struct ethhdr));

This structure contains the tcp protocol header of the packet.

struct tcphdr *tcp = (struct tcphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));

This structure contains the udp header of the packet.

struct udphdr *udp = (struct udphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));

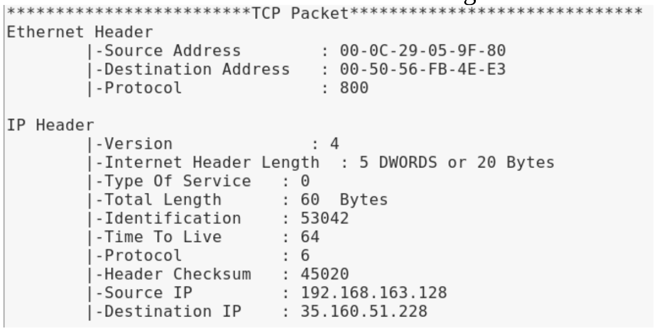
This variable contains the payload.

unsigned char * data = (buffer + iphdrlen + sizeof(struct ethhdr) + sizeof(struct udphdr));…………After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

RESULT

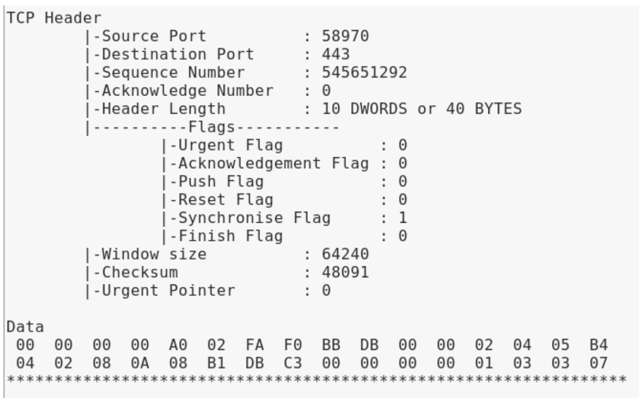
First run the promiscuous program to convert the NIC into promiscuous mode. After the NIC has been set to the promiscuous mode, run the sniffer program to sniff all the traffic on the network. I have created a log file which will store the information of the sniffed packets, and will yield the result after the sniffer program had been stopped. Also, do not forget to change back the NIC from the promiscuous mode to the default mode.
You can check the status of the NIC using "netstat -i" in the command prompt. If the promiscuous mode is turned on, the "P" flag will be visible.

Here is the screenshot of the log file.



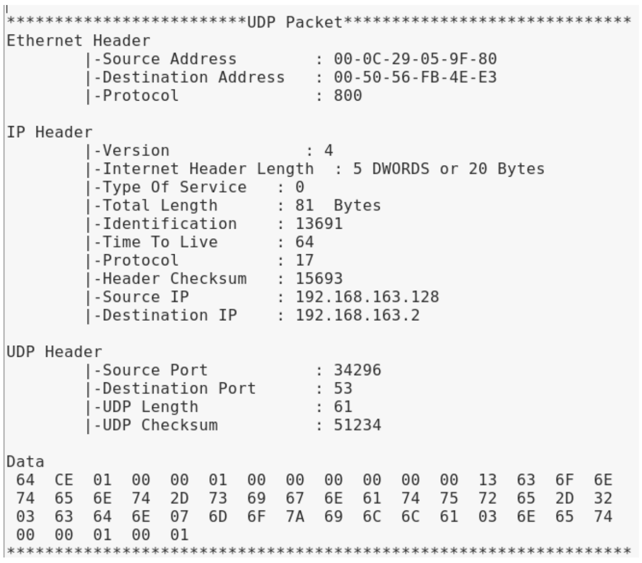
```
*************************TCP Packet*****************************
Ethernet Header
        |-Source Address         : 00-0C-29-05-9F-80
        |-Destination Address    : 00-50-56-FB-4E-E3
        |-Protocol               : 800

IP Header
        |-Version                  : 4
        |-Internet Header Length   : 5 DWORDS or 20 Bytes
        |-Type Of Service    : 0
        |-Total Length       : 60  Bytes
        |-Identification     : 53042
        |-Time To Live       : 64
        |-Protocol           : 6
        |-Header Checksum    : 45020
        |-Source IP          : 192.168.163.128
        |-Destination IP     : 35.160.51.228
```



```
TCP Header
        |-Source Port            : 58970
        |-Destination Port       : 443
        |-Sequence Number        : 545651292
        |-Acknowledge Number     : 0
        |-Header Length          : 10 DWORDS or 40 BYTES
        |----------Flags-----------
                |-Urgent Flag             : 0
                |-Acknowledgement Flag : 0
                |-Push Flag               : 0
                |-Reset Flag              : 0
                |-Synchronise Flag        : 1
                |-Finish Flag             : 0
        |-Window size            : 64240
        |-Checksum               : 48091
        |-Urgent Pointer         : 0

Data
 00  00  00  00  A0  02  FA  F0  BB  DB  00  00  02  04  05  B4
 04  02  08  0A  08  B1  DB  C3  00  00  00  00  01  03  03  07
*****************************************************************
```



```
*************************UDP Packet*****************************
Ethernet Header
        |-Source Address         : 00-0C-29-05-9F-80
        |-Destination Address    : 00-50-56-FB-4E-E3
        |-Protocol               : 800

IP Header
        |-Version                  : 4
        |-Internet Header Length   : 5 DWORDS or 20 Bytes
        |-Type Of Service    : 0
        |-Total Length       : 81  Bytes
        |-Identification     : 13691
        |-Time To Live       : 64
        |-Protocol           : 17
        |-Header Checksum    : 15693
        |-Source IP          : 192.168.163.128
        |-Destination IP     : 192.168.163.2

UDP Header
        |-Source Port            : 34296
        |-Destination Port       : 53
        |-UDP Length             : 61
        |-UDP Checksum           : 51234

Data
 64  CE  01  00  00  01  00  00  00  00  00  00  13  63  6F  6E
 74  65  6E  74  2D  73  69  67  6E  61  74  75  72  65  2D  32
 03  63  64  6E  07  6D  6F  7A  69  6C  6C  61  03  6E  65  74
 00  00  01  00  01
*****************************************************************
```

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.

CONCLUSION

We have learned that a packet sniffer can be created using both RAW socket and built-in libpcap library. You can capture the packets coming in through both wired and wireless interfaces. We now understand how to create a sniffer using the RAW socket. We have also learnt how to convert the default mode of a NIC to a promiscuous mode for it to capture all the traffic in the network. Since the NIC is a hardware, it will be difficult for it to capture all the packets, therefore, it is advisable to use an analyser to only accept a certain type of protocols. This will improve the performance of the sniffer. I would like to conclude by saying that a sniffer should not be utilized for malicious active. It should only be used to monitor the traffic in a home network or the office by the network administrator, to avoid network congestion.

Acknowledgment

References

[1] Design and Implementation of V6SNIFF: an Efficient IPv6 Packet Sniffer.

[2] Packet Sniffing for Automated Chat Room Monitoring and Evidence Preservation.

[3] A New Hybrid Network Sniffer Model Based on Pcap Language and Sockets (Pcapsocks)

[4] Packet Sniffing: A Brief Introduction

[5] A Protocol Based Packet Sniffer

[6] PACKET SNIFFER

[7] Packet Sniffer – A Comparative Study

[8] Network Traffic Monitoring and Analysis using Packet Sniffer

[9] An Approach to Detect Packets Using Packet Sniffing.

[10] Development of a network packet sniffing tool for internet protocol generations

[11] Kernel.org/doc/html

[12] The Linux Foundation-wiki

[13] embeddedlinux.org.cn