Network Traffic Analysis and Intrusion Detection using Packet Sniffer

Mohammed Abdul Qadeer
Dept. of Computer Engineering,
Aligarh Muslim University,
Aligarh- 202002, India
maqadeer@ieee.org

Mohammad Zahid
Asst. System Engineer,
Tata Consultancy Services,
Trivandrum, India
md.zahid@tcs.com

Arshad Iqbal
Scientist B,
GTRE, DRDO,
Bangalore, India
arshadamu@gmail.com

MisbahurRahman Siddiqui
Univ. Women's Polytechnic,
Aligarh Muslim University,
Aligarh- 202002, India
misbahurrahman@gmail.com

Abstract- **Computer software that can intercept and log traffic passing over a digital network or part of a network is better known as packet sniffer. The sniffer captures these packets by setting the NIC card in the promiscuous mode and eventually decodes them. The decoded information can be used in any way depending upon the intention of the person concerned who decodes the data (i.e. malicious or beneficial purpose). Depending on the network structure one can sniff all or just parts of the traffic from a single machine within the network. However, there are some methods to avoid traffic narrowing by switches to gain access to traffic from other systems on the network. This paper focuses on the basics of packet sniffer and its working, development of the tool on Linux platform and its use for Intrusion Detection. It also discusses ways to detect the presence of such software on the network and to handle them in an efficient way. Focus has also been laid to analyze the bottleneck scenario arising in the network, using this self developed packet sniffer. Before the development of this indigenous software, minute observation has been made on the working behavior of already existing sniffer software such as wireshark (formerly known as ethereal), tcpdump, and snort, which serve as the base for the development of our sniffer software. For the capture of the packets, a library known as libpcap has been used. The development of such software gives a chance to the developer to incorporate the additional features that are not in the existing one.**

Keywords: Packet capture, traffic analysis, libpcap, network monitoring, NIC, promiscuous mode, Berkeley Packet Filter, Network analyzer, packet sniffer, intrusion detection.

I. INTRODUCTION

Packet sniffer is a program running in a network attached device that passively receives all data link layer frames passing through the device's network adapter. It is also known as Network or Protocol Analyzer or Ethernet Sniffer. The packet sniffer captures the data that is addressed to other machines, saving it for later analysis. It can be used legitimately by a network or system administrator to monitor and troubleshoot network traffic. Using the information captured by the packet sniffer an administrator can identify erroneous packets and use the data to pinpoint bottlenecks and help maintain efficient network data transmission. Packet Sniffers were never made to hack or steal information. They had a different goal, to make things secure. But then everything has a dark side. Figure 1 shows the output captured by the Wireshark (packet sniffer software formerly known as Ethereal). In figure 2 we have shown that how the data travels from application layer to the network interface card.

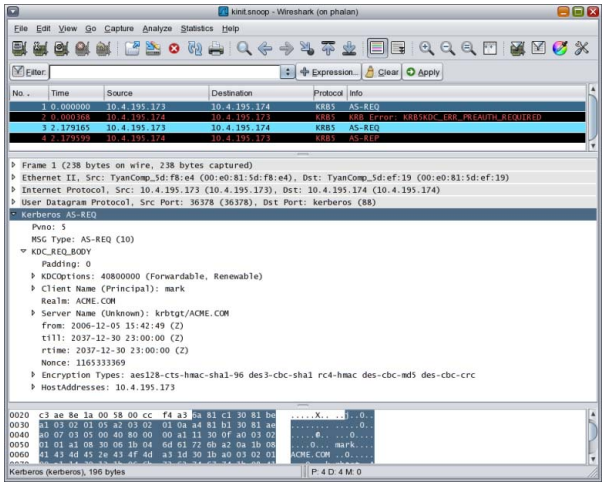

Fig 1: Screen shot of wireshark

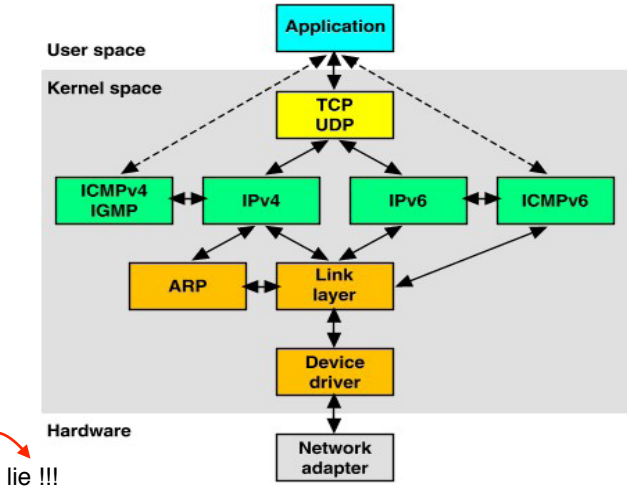

Fig 2: Flow of packets

978-0-7695-3961-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCSN.2010.104

II. LIBRARY : LIBPCAP

Pcap consists of an application programming interface (API) for capturing packets in the network. UNIX like systems implements pcap in the libpcap library; Windows uses a port of libpcap known as WinPcap. LIBPCAP is a widely used standard packet capture library that was developed for use with BPF (Berkely Packet Filter) kernel device [1]. BPF can be considered as an OS kernel extension. It is BPF, which enables communication between operating system and NIC. Libpcap is a C language library that extends the BPF library constructs. Libpcap is used to capture the packets on the network directly from the network adapter. This library is an in built feature of the operating system. It provides packet capturing and filtering capability. It was originally developed by the tcpdump developers in the Network Research Group at Lawrence Berkeley Laboratory [2]. If this library is missing in the operating system, we can install it at a later time, as it is available as an open source.

III. PROMISCUOUS MODE

The network interface card works in two modes

I) Non promiscuous mode (normal mode)

II) Promiscuous mode

When a packet is received by a NIC, it first compares the MAC address of the packet to its own. If the MAC address matches, it accepts the packet otherwise filters it. This is due to the network card discarding all the packets that do not contain its own MAC address, an operation mode called non promiscuous, which basically means that each network card is minding its own business and reading only the frames directed to it. In order to capture the packets, NIC has to be set in the promiscuous mode. Packet sniffers which do sniffing by setting the NIC card of its own system to promiscuous mode, and hence receives all packets even they are not intended for it. So, packet sniffer captures the packets by setting the NIC card into promiscuous mode. To set a network card to promiscuous mode, all we have to do is issue a particular ioctl ( ) call to an open socket on that card and the packets are passed to the kernel. In figure 4 we can see network interface card (NIC). Figure 3 shows how the data sent by device A to device C is also received by device D which is set in promiscuous mode.

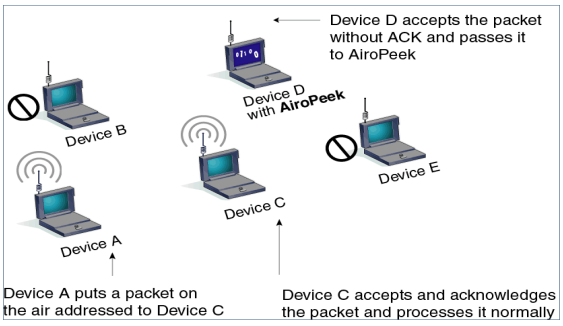

Fig 3: Packet received by device set in promiscuous mode on wireless LAN

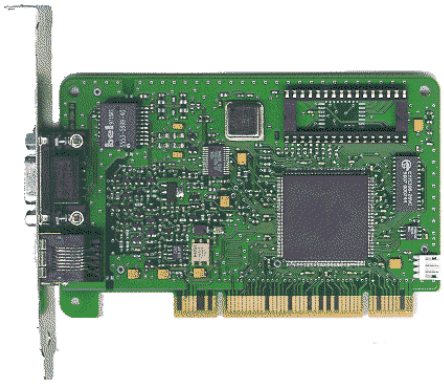

Fig 4: Network Interface card

IV. SNIFFER WORKING MECHANISMS

When the packets are sent from one node to another in the network, a packet has to pass through many intermediate nodes. A node whose NIC is set in the promiscuous mode tends to receives the packet. The packet arriving at the NIC are copied to the device driver memory, which is then passed to the kernel buffer from where it is used by the user application. In Linux kernel, libpcap uses "PF_PACKET" socket which bypasses most packet protocol processing done by the kernel [3]. Each socket has two kernel buffers associated with it for reading and writing. By default in Fedora core 6, the size of each buffer is 109568 bytes. In our packet sniffer, at user level the packets are copied from the kernel buffer into a buffer created by libpcap when a live capture session is created. A single packet is handled by the buffer at a time for the application processing before next packet is copied into it [3]. The new approach taken in the development of our packet sniffer is to improve the performance of packet sniffer, using libpcap to use same buffer space between kernel space and application. Figure 5 shows the interface of our packet sniffer while capturing packets.

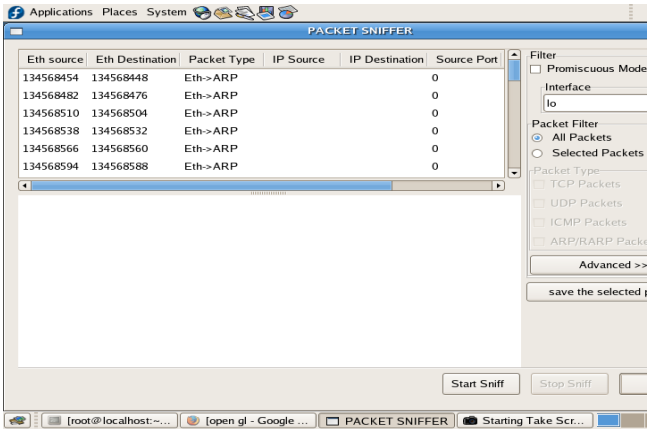

Fig 5: Packet sniffer while capturing session

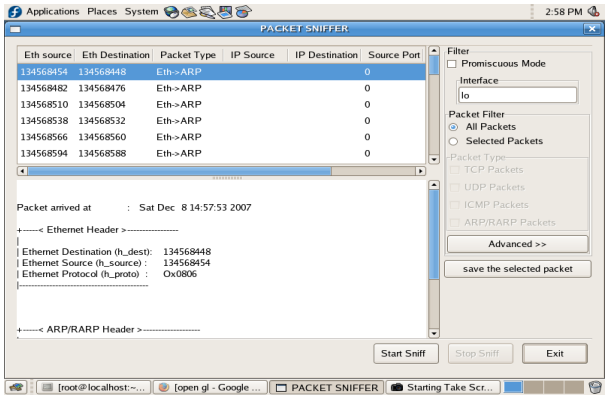

Fig 6: Shows the details of selected packet

V. Basic Steps for the Development of Packet Sniffer on Linux Platform

We are going to discuss the basic steps that we have taken during the development of our packet sniffer.

A. *Socket Creation*

Socket is a bi-directional communication abstraction via which an application can send and receive data

There are many types of socket:

SOCK_STREAM: TCP (connection oriented, guaranteed delivery)

SOCK_DGRAM: UDP (datagram based communication)

SOCK_RAW: allow access to the network layer. This can be build ICMP message or Custom IP packet.

SOCK_PACKET: allows access to the link layer (e.g. Ethernet). When a socket is created, a socket stream, similar to the file stream, is created, through which data is read [4].

B. *To Set NIC in Promiscuous Mode*

To enable the packet sniffer to capture the packets, the NIC of the node on which sniffer software is running has to be set on promiscuous mode. In our packet sniffer it was implemented by issuing an ioctl ( ) call to an open socket on that card. The *ioctl* system call takes three arguments;

- The socket stream descriptor.
- The function that the *ioctl* function is supposed to perform.
- Reference to the *ifreq* member [4]

Since this is a potentially security-threatening operation, the call is only allowed for the root user. Supposing that “sock" contains an already open socket, the following instructions will do the trick:

```
ioctl (sock, SIOCGIFFLAGS, & ether);
ethreq.ifr_flags |= IFF_PROMISC;
ioctl (sock, SIOCGIFFLAGS, & ether);
```

The first ioctl reads the current value of the Ethernet card flags; the flags are then ORed with IFF_PROMISC, which enables promiscuous mode and are written back to the card with the second ioctl.

C. *Protocol Interpretation*

In order to interpret the protocol, the developer should have some basic knowledge of protocol that he wishes to sniff. In our sniffer which we developed on Linux platform we interpreted the protocols such as IP, TCP, UDP, ICMP protocols by including the headers as;

<linux/tcp.h>, <linux/udp.h>, <linux/ip.h> and <linux/icmp.h>.

In the figures below we are showing some packet header formats;

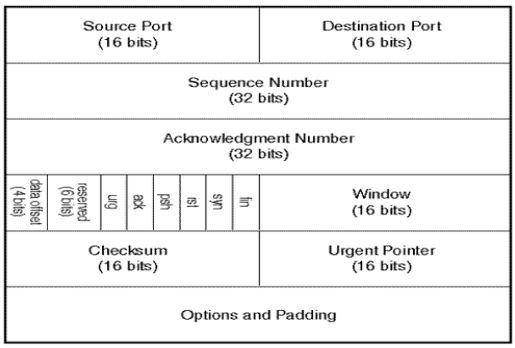

Fig 7: TCP protocol header fields

Fig 8: UDP protocol header fields

VI. Linux Filter

As network traffic increases, the sniffer will start losing packets since the PC will not be able to process them quickly enough. The solution to this problem is to filter the packets you receive, and print out information only on those you are interested in. One idea would be to insert an “if statement" in the sniffer's source; this would help polish the output of the sniffer, but it would not be very efficient in terms of performance. The kernel would still pull up all the packets flowing on the network, thus wasting processing time, and the sniffer would still examine each packet header to decide whether to print out the related data or not. The optimal solution to this problem is to put the filter as early as possible in the packet-processing chain (it starts at the

network driver level and ends at the application level, see Figure 9). The Linux kernel allows us to put a filter, called an LPF, directly inside the PF_PACKET protocol-processing routines, which are run shortly after the network card reception interrupt has been served. The filter decides which packets shall be relayed to the application and which ones should be discarded.

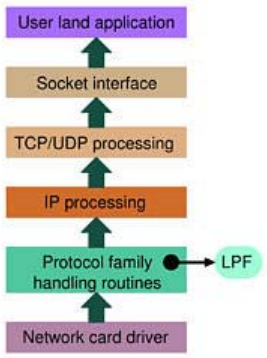

Fig 9: Filter processing chain

VII. Methods To Sniff On Switch

Now we are going to discuss the methods that can be used to sniff the packets on the switch, being an intelligent device.

A. *ARP Spoofing*

As we know that ARP is used to obtain the MAC address of the destination machine with which we wish to communicate. The ARP is stateless, we can send an ARP reply, even if one has not been asked for and such a reply will be accepted. Ideally, when you want to sniff the traffic originating from a machine, you need to ARP spoof the gateway of the network. The ARP cache of that machine will now have a wrong entry for the gateway and is said to be "poisoned". This way all the traffic from that machine destined for the gateway will pass through your machine. Another trick that can be used is to poison a hosts ARP cache by setting the gateway's MAC address to FF:FF:FF:FF:FF:FF (also known as the broadcast MAC). There are various utilities available for ARP spoofing. An excellent tool for this is the arpspoof utility that comes with the dsniff suite.

B. *MAC Flooding*

Switches keep a translation table that maps various MAC addresses to the physical ports on the switch. As a result of this, a switch can intelligently route packets from one host to another, but it has a limited memory for this work. MAC flooding makes use of this limitation to bombard the switch with fake MAC addresses until the switch can't keep up. The switch then enters into what is known as a `failopen mode', wherein it starts acting as a hub by broadcasting packets to all the machines on the network. Once that happens sniffing can be performed easily. MAC flooding can be performed by using macof, a utility which comes with dsniff suite.

VIII. Bottleneck Analysis

With the increase of traffic in the network, the rate of the packets being received by the node also increases. On the arrival of the packet at NIC, they have to be transferred to the main memory for processing. A single packet is transferred over the bus. As we know that the PCI bus has actual transfer of not more than 40 to 50 Mbps because a device can have control over the bus for certain amount of time or cycles, after that it has to transfer the control of the bus [2]. And we know that the slowest component of a PC is disk drive so, bottleneck is created in writing the packets to disk in traffic sensitive network. To handle the bottle neck we can make an effort to use buffering in the user level application. According to this solution, some amount of RAM can be used as buffer to overcome bottleneck [1].

IX. Detection of Packet Sniffer

Since the packet sniffer has been designed as a solution to many network problems. But one can not ignore its malicious use. Sniffers are very hard to detect due to its passiveness but there is always a way, and some of them are given below;

A. *ARP Detection Technique*

As we know that sniffing host receives all the packets, including those that are not destined for it. Sniffing host makes mistakes by responding to such packets that are supposed to be filtered by it. So, if an ARP packet is sent to every host and ARP packet is configured such that it does not have broadcast address as destination address and if some host respond to such packets, then those host have there NIC set into promiscuous mode [5]. As we know that Windows is not an open source OS, so we can't analyze its software filter behavior as we do in Linux. In Linux we can analyze the behavior of filter by examining the source code of this OS. So, here we are presenting some addresses to do it on Windows. They are as follows;

- FF-FF-FF-FF-FF-FF Broadcast address: The packet having this address is received by all nodes and responded by them.
- FF-FF-FF-FF-FF-FE fake broadcast address: This address is fake broadcast address in which last 1 bit is missing. By this address we check whether the filter examines all the bits of address and respond to it.
- FF-FF-00-00-00-00 fake broadcast 16 bit address: In this address we can see those first 16 bits are same as broadcast address.

- FF: 00:00:00:00:00 fake broadcast 8 bits: This address is fake broadcast address whose first 8 bits are same as the broadcast address [6].

B. *RTT Detection*

RTT stands for Round Trip Time. It is the time that the packet takes to reach the destination along with the time which is taken by response to reach the source. In this technique first the packets are sent to the host with normal mode and RTT is recorded. Now the same host is set to promiscuous mode and same set of packets are sent and again RTT is recorded. The idea behind this technique is that RTT measurement increases when the host is in promiscuous mode, as all packets are captured in comparison to host that is in normal mode [7].

C. *SNMP Monitoring*

SNMP is widely employed to monitor, control, and configure network elements. By the help of this protocol network managers locate and correct the network problems. SNMP client is invoked by the managers on the local node, and by the help of this client node they contact one or more SNMP servers. SNMP uses a fetch and store model in which each server maintains a variable that include statistics, as count of packet received [4]. By the help of SNMP one can detect the presence of sniffer in the network by connecting and disconnecting to the ports.

X. Intrusion Detection using Packet Sniffer

The term "Intrusion Detection" implies discovering attacks and threats throughout an enterprise or organization, and responding to those discoveries. Some of the automated responses typically include notifying a security administrator via a console, e-mail, stopping the offending session, shutting the system down, turning off down Internet links, or executing a predefined command procedure. In context to our paper, as we know that packet sniffer can be used for malicious purpose the same can be used for intrusion detection also. Using this methodology, the Intrusion Detection software is placed on the system, which puts the Ethernet card in "promiscuous mode" so that the software can read and analyze all traffic. It does this by examining both the packet header fields and packet contents. The Intrusion Detection software like packet sniffers includes an engine, which looks for specific types of network attacks, such as IP spoofing and packet floods. When the packet sniffer detects a potential problem it responds immediately by notifying to the administrator by various mode such as console, beeping a pager, sending an e-mail, or even shutting down the network session. The diagram below shows a typical deployment of sniffers for doing packet analysis. A sniffer is placed outside the firewall to detect attack attempts coming from the Internet. A sniffer is also placed inside the network to detect Internet attacks, which penetrate the firewall and to assist in detecting internal attacks and threats.

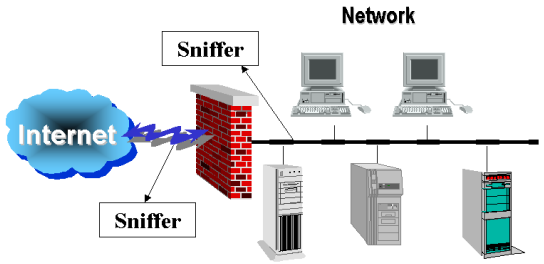

Fig 10: Deployment of packet sniffer for intrusion detection

XI. Conclusion & Future Work

This packet sniffer can be enhanced in future by incorporating features like making the packet sniffer program platform independent, filtering the packets using filter table, filtering the suspect content from the network traffic and gather and report network statistics. A packet sniffer is not just a hacker's tool. It can be used for network traffic monitoring, traffic analysis, troubleshooting and other useful purposes. However, a user can employ a number of techniques to detect sniffers on the network as discussed in this paper and protect the data from being sniffed.

References

[1] G. Varghese, "Network Algorithmic: An Interdisciplinary Approach to Designing Fast Networked Devices", San Francisco, CA: Morgan Kaufmann, 2005.

[2] J. Cleary, S. Donnelly, I. Graham, "Design Principles for Accurate Passive Measurement," in Proc. PAM 2000 Passive and Active Measurement Workshop (Apr. 2000).

[3] A. Dabir, A. Matrawy, "Bottleneck Analysis of Traffic Monitoring Using Wireshark", 4th International Conference on Innovations in Information Technology, 2007, IEEE Innovations '07, 18-20 Nov. 2007, Page(s):158 - 162

[4] S. Ansari, Rajeev S.G. and Chandrasekhar H.S, "Packet Sniffing: A brief Introduction", IEEE Potentials, Dec 2002- Jan 2003, Volume:21, Issue:5, pp:17 – 19

[5] Daiji Sanai, "Detection of Promiscuous Nodes Using ARP Packet", http://www.securityfriday.com/

[6] Ryan Spangler , Packet Sniffer Detection with AntiSniff, University of Wisconsin – Whitewater, Department of Computer and Network Administration, May 2003

[7] Zouheir Trabelsi, Hamza Rahmani, Kamel Kaouech, Mounir Frikha, "Malicious Sniffing System Detection Platform", Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04), IEEE Computer Society

[8] Hornig, C., "A Standard for the Transmission of IP Data grams over Ethernet Networks", RFC-894, Symbolic Cambridge Research Center, April 1984.