Network Packet Sniffing

Sniffing or network packet sniffing is the process of monitoring and capturing all the packets passing through a given network using sniffing tools. It is a form wherein, we can “tap phone wires” and get to know the conversation. It is also called **wiretapping** and can be applied to the computer networks.

There is so much possibility that if a set of enterprise switch ports is open, then one of their employees can sniff the whole traffic of the network. Anyone in the same physical location can plug into the network using Ethernet cable or connect wirelessly to that network and sniff the total traffic.

In other words, Sniffing allows you to see all sorts of traffic, both protected and unprotected. In the right conditions and with the right protocols in place, an attacking party may be able to gather information that can be used for further attacks or to cause other issues for the network or system owner.

What can be sniffed?

One can sniff the following sensitive information from a network −

- Email traffic
- FTP passwords
- Web traffics
- Telnet passwords
- Router configuration
- Chat sessions
- DNS traffic

How does sniffing work?

A sniffer normally turns the NIC of the system to the promiscuous mode so that it listens to all the data transmitted on its segment.

The promiscuous mode refers to the unique way of Ethernet hardware, in particular, network interface cards (NICs), that allows an NIC to receive all traffic on the network, even if it is not addressed to this NIC. By default, an NIC ignores all traffic that is not addressed to it, which is done by comparing the destination address of the Ethernet packet with the hardware address (MAC) of the device. While this makes perfect sense for networking, non-promiscuous mode makes it difficult to use network monitoring and analysis software for diagnosing connectivity issues or traffic accounting.

A sniffer can continuously monitor all the traffic to a computer through the NIC by decoding the information encapsulated in the data packets.

Types of Sniffing

Sniffing can be either Active or Passive in nature. We will now learn about the different types of sniffing.

Passive Sniffing

In passive sniffing, the traffic is locked but it is not altered in any way. Passive sniffing allows listening only. It works with the Hub devices. On a hub device, the traffic is sent to all the ports. In a network that uses hubs to connect systems, all hosts on the network can see the traffic. Therefore, an attacker can easily capture traffic going through.

The good news is that hubs have almost become obsolete in recent times. Most modern networks use switches. Hence, passive sniffing is no more effective.

Active Sniffing

In active sniffing, the traffic is not only locked and monitored, but it may also be altered in some way as determined by the attack. Active sniffing is used to sniff a switch-based network. It involves injecting address resolution packets (ARP) into a target network to flood on the switch content addressable memory (CAM) table. CAM keeps track of which host is connected to which port.

Following are the Active Sniffing Techniques −

- MAC Flooding
- DHCP Attacks
- DNS Poisoning
- Spoofing Attacks
- ARP Poisoning

The Sniffing Effects on Protocols

Protocols such as the **tried and true TCP/IP** were never designed with security in mind. Such protocols do not offer much resistance to potential intruders. Following are the different protocols that lend themselves to easy sniffing −

HTTP

It is used to send information in clear text without any encryption and thus a real target.

SMTP (Simple Mail Transfer Protocol)

SMTP is utilized in the transfer of emails. This protocol is efficient, but it does not include any protection against sniffing.

NNTP (Network News Transfer Protocol)

It is used for all types of communication. A major drawback with this is that data and even passwords are sent over the network as clear text.

POP (Post Office Protocol)

POP is strictly used to receive emails from the servers. This protocol does not include protection against sniffing because it can be trapped.

FTP (File Transfer Protocol)

FTP is used to send and receive files, but it does not offer any security features. All the data is sent as clear text that can be easily sniffed.

IMAP (Internet Message Access Protocol)

IMAP is same as SMTP in its functions, but it is highly vulnerable to sniffing.

Telnet

Telnet sends everything (usernames, passwords, keystrokes) over the network as clear text and hence, it can be easily sniffed.

Sniffers are not the dumb utilities that allow you to view only live traffic. If you really want to analyze each packet, save the capture and review it whenever time allows.

Implementation using Python

Before implementing the raw socket sniffer, let us understand the **struct** method as described below −

struct.pack(fmt, a1,a2,…)

As the name suggests, this method is used to return the string, which is packed according to the given format. The string contains the values a1, a2 and so on.

struct.unpack(fmt, string)

As the name suggests, this method unpacks the string according to a given format.

In the following example of raw socket sniffer IP header, which is the next 20 bytes in the packet and among these 20 bytes we are interested in the last 8 bytes. The latter bytes show if the source and destination IP address are parsing −

Now, we need to import some basic modules as follows −

```
import socket
import struct
import binascii
```

Now, we will create a socket, which will have three parameters. The first parameter tells us about the packet interface — PF_PACKET for Linux specific and AF_INET for windows; the second parameter tells us that it is a raw socket and the third parameter tells us about the protocol we are interested in —0x0800 used for IP protocol.

```
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket. htons(0x0800))
```

Now, we need to call the **recvfrom()** method to receive the packet.

```
while True:
   packet = s.recvfrom(2048)
```

In the following line of code, we are ripping the Ethernet header −

```
ethernet_header = packet[0][0:14]
```

With the following line of code, we are parsing and unpacking the header with the **struct** method −

```
eth_header = struct.unpack("!6s6s2s", ethernet_header)
```

The following line of code will return a tuple with three hex values, converted by **hexify** in the **binascii** module −

```
print "Destination MAC:" + binascii.hexlify(eth_header[0]) + " Source MAC:" + binas
```

We can now get the IP header by executing the following line of code −

```
ipheader = pkt[0][14:34]
ip_header = struct.unpack("!12s4s4s", ipheader)
print "Source IP:" + socket.inet_ntoa(ip_header[1]) + " Destination IP:" + socket.i
```

Similarly, we can also parse the TCP header.