AIM : Write a program to sniff packet sent over the local network and analyze it.

Performance (3)	Understanding (1)	Regularity (1)	Total (5)	Sign of Staff

Introduction

Have you ever thought about how your computer talks with others on a network? Would you like to listen to, or "sniff", the conversation? Network engineers, system administrators, security professionals and, unfortunately, crackers have long used a tool that allows them to do exactly that. This nifty utility, known as a *sniffer*, can be found in the arsenal of every network guru, where it's likely used every day for a variety of tasks. This article will offer a brief overview of sniffers, including what they do, how they work, why users need to be aware of them, and what users can do to protect themselves against the illegitimate use of sniffers.

What is a Sniffer?

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 a Sniffer Work?

- Before we can explore how a sniffer operates, it may be helpful to examine what enables the tool to work.
- During normal tasks such as Web surfing and messaging, computers are constantly communicating with other machines.
- Obviously, a user should be able to see all the traffic traveling to or from their machine.

Most PCs, however, are on a Local Area Network (LAN), meaning they share a connection with several other computers.

- If the network is not switched (a switch is a device that filters and forwards packets between segments of the LAN), the traffic destined for any machine on a segment is broadcast to every machine on that segment.
- This means that a computer actually sees the data traveling to and from each of its neighbors, but ignores it, unless otherwise instructed.
- We can now begin to understand the magic behind a sniffer. The sniffer program tells a computer, specifically its Network Interface Card (NIC), to stop ignoring all the traffic headed to other computers and pay attention to them.
- It does this by placing the NIC in a state known as *promiscuous* mode. Once a NIC is promiscuous, a status that requires administrative or root privileges, a machine can see all the data transmitted on its segment.
- The program then begins a constant read of all information entering the PC via the network card.

Implementation using Python

1. **Importing Required Libraries**:

   from scapy.all import *

   This line imports all the necessary functions and classes from the **scapy** library. Scapy is a powerful packet manipulation tool and library that allows you to capture, decode, and analyze network packets.

2. **Defining the Packet Handler Function**:

   def packet_handler(packet):

   This line defines a function named **packet_handler** that will be called for each packet sniffed by Scapy. The function takes one argument, **packet**, which represents the packet being sniffed.

3. **Checking for IP Packet**:

   if IP in packet:

   This line checks if the packet is an IP packet. If the packet contains an IP layer, the code inside the **if** block will be executed.

4. **Extracting IP Information**:

   src_ip = packet[IP].src dst_ip = packet[IP].dst protocol = packet[IP].proto length = len(packet)

   Here, we extract information from the IP layer of the packet. We retrieve the source IP address (**src_ip**), destination IP address (**dst_ip**), protocol (**protocol**), and length of the packet (**length**).

5. **Printing IP Packet Information**:

   print(f"IP Packet: Source IP: {src_ip}, Destination IP: {dst_ip}, Protocol: {protocol}, Length: {length}")

   This line prints out the information extracted from the IP layer of the packet.

6. **Checking for TCP and UDP Packets**:

   if TCP in packet: src_port = packet[TCP].sport dst_port = packet[TCP].dport print(f" TCP Packet: Source Port: {src_port}, Destination Port: {dst_port}") if UDP in packet: src_port = packet[UDP].sport dst_port = packet[UDP].dport print(f" UDP Packet: Source Port: {src_port}, Destination Port: {dst_port}")

   These blocks of code check if the packet contains TCP or UDP layers. If so, they extract source and destination port information and print it out.

7. **Starting Packet Sniffing**:

   print("Sniffing packets... Press Ctrl+C to stop.") sniff(prn=packet_handler, store=0)

   These lines start packet sniffing using Scapy's `sniff` function. The `prn` parameter specifies the callback function (`packet_handler`) to be called for each packet sniffed. The `store` parameter is set to `0` to prevent Scapy from storing packets in memory.

Conclusion:

Hence, we Conclude when we run these program different packets are analyzed by extracting Headers Fields different fields of captured packet.