CodeAlpha

Cyber Security

(TASK-1)

BASIC NETWORK SNIFFER

Build a network sniffer in Python that captures and analyzes network traffic. This project will help you understand how data flows on a network and how network packets are structured.

To build a network sniffer in Python, you can use the scapy library, which provides powerful tools for capturing, dissecting, and analyzing network packets.

Below is a step-by-step guide to building a simple network sniffer:

Requirements:

1. **Install Scapy**: You need the Scapy library to interact with network packets. By giving command "pip install scapy"
2. Permissions: Capturing network traffic typically requires administrative (root) permissions.

Steps to Build the Network Sniffer:

1. **Import Necessary Libraries**:
    a. We'll use scapy to capture and analyze packets.
    b. We'll also use the time module for timestamping and other analysis.
2. **Packet Sniffing**:
    a. We'll use the sniff function from Scapy to capture network packets.
    b. The filter parameter allows us to capture specific types of packets (e.g., TCP, UDP, ICMP).

c. The prn parameter allows us to define a custom function to process each captured packet.

3. **Packet Analysis**:
    a. For each packet captured, we'll display the basic information such as source/destination IP addresses, protocol type, and other relevant details.
    b. Optionally, we can save the packets or filter them further based on certain criteria.

Here's the code to build a simple network sniffer:

Code Example:

import **scapy.all** as scapy import time

#Callback function to process each packet

def packet_callback(packet): # Check if packet has an IP layer if packet.haslayer(scapy.IP): ip_src = packet[scapy.IP].src ip_dst = packet[scapy.IP].dst protocol = packet[scapy.IP].proto

```
    # Check if the packet is a TCP packet
    if packet.haslayer(scapy.TCP):
        tcp_src_port = packet[scapy.TCP].sport
        tcp_dst_port = packet[scapy.TCP].dport

print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}]
TCP Packet: {ip_src} -> {ip_dst} (Src Port:
{tcp_src_port}, Dst Port: {tcp_dst_port})")

    # Check if the packet is a UDP packet
    elif packet.haslayer(scapy.UDP):
        udp_src_port = packet[scapy.UDP].sport
        udp_dst_port = packet[scapy.UDP].dport

print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}]
UDP Packet: {ip_src} -> {ip_dst} (Src Port:
{udp_src_port}, Dst Port: {udp_dst_port})")

    # Check if the packet is an ICMP packet
    elif packet.haslayer(scapy.ICMP):

print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}]
ICMP Packet: {ip_src} -> {ip_dst} (ICMP Type:
```

{packet[scapy.ICMP].type})")

# General IP packet (non-TCP, UDP, or ICMP)
else:

print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}] IP Packet: {ip_src} -> {ip_dst} (Protocol: {protocol})")

#Main function to start sniffing

def start_sniffing(interface="eth0"): print(f"Starting packet sniffer on interface {interface}...")

Sniff packets indefinitely with a callback function scapy.sniff(iface=interface, prn=packet_callback, store=0)

if **name** == "**main**": # Change the interface to your network interface, e.g., 'eth0', 'wlan0', etc. start_sniffing("eth0") # Replace 'eth0' with your network interface name

Explanation:

packet_callback(packet):

This function is called for each captured packet.

We check if the packet has an IP layer (packet.haslayer(scapy.IP)).

Depending on the protocol (TCP, UDP, ICMP), we extract relevant information like source/destination IP and port numbers.

We also print a timestamp for each packet capture.

start_sniffing(interface="eth0"):

The scapy.sniff() function listens on the specified network interface (eth0 is common for wired Ethernet interfaces).

The prn=packet_callback argument tells Scapy to call the packet_callback() function for each captured packet.

The store=0 argument tells Scapy not to keep a copy of the packets in memory (just process them).

Running the Sniffer:

1.Run the Script:

You'll need to run this script with administrative privileges (for example, using sudo on Linux).

On Linux or macOS, use the command: **sudo python3 network_sniffer.py**

2.Network Interface:

a. Replace "eth0" with the correct network interface name for your system (e.g., wlan0 for Wi-Fi on Linux or macOS).
b. You can list network interfaces using ifconfig (Linux/macOS) or ipconfig (Windows).

Enhancements:

1. **Filter Specific Traffic**: Use the filter parameter to capture specific types of packets (e.g., filter="tcp" to capture only TCP traffic).
2. **Packet Analysis**: Analyze packet contents in more depth, including HTTP payloads or other protocols.
3. **Packet Logging**: Save captured packets to a file for further analysis using Scapy's wrpcap() function.
4. **GUI**: Build a graphical interface to display packet statistics in real time.

Example output:

[2025-01-11 13:35:45] TCP Packet: 192.168.1.2 -> 192.168.1.3 (Src Port: 12345, Dst Port: 80)
[2025-01-11 13:35:45] ICMP Packet: 192.168.1.2 -> 192.168.1.1 (ICMP Type: 8)
[2025-01-11 13:35:45] UDP Packet: 192.168.1.2 -> 192.168.1.1 (Src Port: 53, Dst Port: 12345)

Notes:

1.This script works for basic sniffing and analysis. If you want to analyze specific protocols (e.g., HTTP, DNS), you can further dissect the packet payload.

2.Be sure to use this script responsibly and only on networks where you have permission to capture traffic.