JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY , SECTOR - 62 , NOIDA

Network Packet Sniffer with Socket Programming and Wireshark Integration

Computer Networks & Security Lab (18B15CS212)

Submitted To: Dr. Kavita Pandey & Dr. Somya Jain

Submitted By:

Serial No.	Name	Enrollment No.
1.	Aniket Gupta	22104011
2.	Tejas Goel	22104013
3.	Harshit Raj	22104022
4.	Priyansh Arora	22104025

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who contributed to the successful completion of our project, "Network Packet Sniffer with Socket Programming and Wireshark Integration."

First and foremost, we are deeply thankful to our lab teachers, Dr. Kavita Pandey & Dr. Somya Jain for their valuable guidance, constructive feedback, and constant encouragement throughout the development of this project. Their expertise and insights have been instrumental in shaping this work.

We extend our heartfelt thanks to JIIT, for providing us with the resources and platform to explore this topic and enhance our technical skills. Special thanks to our faculty members and peers for their support and suggestions during various stages of the project.

We are also grateful to the developers and maintainers of Python, Scapy, and Wireshark, whose powerful tools and libraries made this project possible. Their open-source contributions have been a cornerstone of innovation and learning.

This project has been an enriching experience, providing us with hands-on knowledge of network traffic analysis, packet capturing, and the integration of real-time tools like Wireshark. We hope that this work serves as a stepping stone for further exploration in the field of networking and cybersecurity.

INTRODUCTION

In today's interconnected digital world, analysing and monitoring network traffic is essential for ensuring robust network security, optimising performance, and troubleshooting issues. This project, **"Network Packet Sniffer with Socket Programming and Wireshark Integration,"** aims to develop a lightweight, Python-based tool to capture and analyse network traffic. The project combines the power of Scapy, a versatile Python library for packet crafting and sniffing, with the renowned packet analysis tool, Wireshark.

How the Project Works

1. **Packet Sniffing**: Using Python's socket programming and Scapy library, the sniffer listens to network traffic on the specified interface. It captures all types of packets, including Ethernet, IP, TCP, UDP, and ICMP.
2. **Packet Processing**: The captured packets are processed to extract critical information such as protocol type, source/destination

WHAT IS A PACKET SNIFFER?

A packet sniffer is a utility that captures data packets traversing a network. These packets carry valuable information, such as protocol type, source and destination IP addresses, ports, and even payload data. Packet sniffers are extensively used in fields such as:

- **Network Administration**: To troubleshoot network performance issues.
- **Cybersecurity**: To detect and prevent unauthorised access or malicious activities.
- **Education and Research**: To understand network protocols and traffic patterns.

KEY OBJECTIVES

The primary objective of this project is to create a network packet sniffer capable of:

1. Capturing network traffic in real-time.
2. Saving captured packets into a `.pcap` file for further analysis.
3. Launching Wireshark for advanced packet dissection and visualisation.
4. Monitor live network traffic across various layers of the OSI model.
5. Leverage Wireshark, a leading network protocol analyzer, for advanced packet dissection.

This project provides users with a comprehensive solution for real-time and offline network monitoring needs, catering to a variety of technical and educational applications.

KEY FEATURES

1. **Real-Time Packet Capture:**

   The tool captures packets traversing the network interface, including Ethernet, IP, TCP, UDP, and ICMP packets.

   Provides live logging of captured packet details such as source/destination addresses, protocols, and ports.

2. **PCAP File Handling:**

   Captured packets are saved into an industry-standard `.pcap` file, ensuring compatibility with third-party tools like Wireshark.

3. **Wireshark Integration:**

   After completing the packet capture, the tool automatically opens Wireshark to analyze the `.pcap` file, providing deep insights into the captured traffic.

4. **Configurable Packet Limit:**

   Users can limit the number of packets to capture, ensuring the tool remains efficient and tailored to specific use cases.

TECHNOLOGIES AND TOOLS USED

- **Python:** The core programming language used for developing the sniffer and handling network traffic with Scapy.
- **Scapy:** A powerful Python library used for packet sniffing, crafting, and saving packets in `.pcap` format.
- **Wireshark:** A widely-used network protocol analyzer for visualizing and dissecting network traffic.
- **Socket Programming:** The backbone of the project, enabling low-level interaction with the network.

RELEVANCE AND USE CASE OF THIS PROJECT

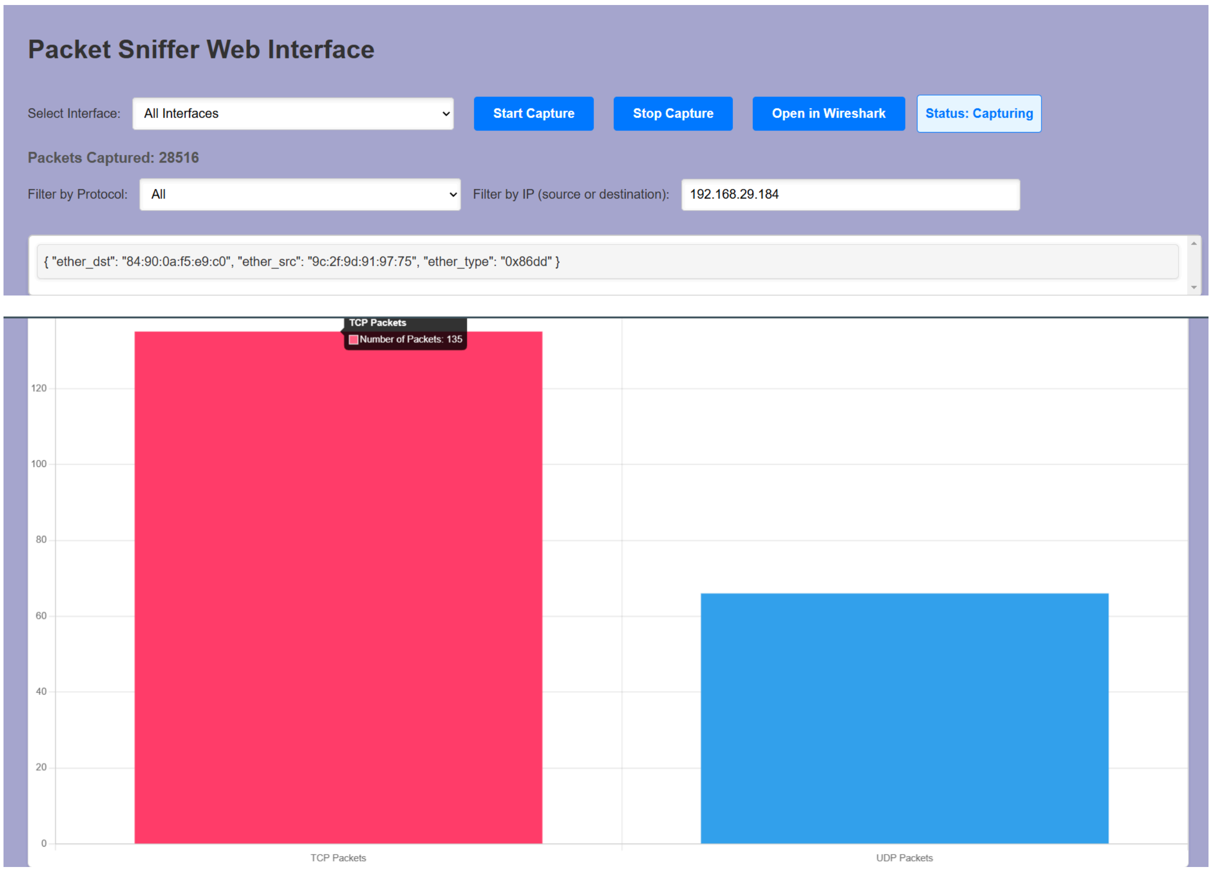
This project demonstrates a practical application of network programming concepts and integrates industry-standard tools for packet analysis. It serves multiple purposes:

- **For Cybersecurity:** Enables the identification of potential threats, such as malicious traffic or unauthorized access attempts.
- **For Network Engineers:** Assists in debugging network performance issues, such as latency or packet loss.
- **For Educational Purposes:** Offers a hands-on understanding of network protocols and traffic patterns.

CODE FOR REFERENCE

```
from flask import Flask, render_template, jsonify, request
from scapy.all import sniff, PcapWriter, Ether, IP, ICMP, TCP, UDP, get_if_list
import threading
import subprocess
import logging

# Setup logging
logging.basicConfig(level=logging.INFO)

# Global variables
pcap_writer = None
packets = []  # To store packet information for display
sniffing_active = False
tcp_count = 0
udp_count = 0

# Function to process packets
def process_packet(packet):
    global packets, pcap_writer, tcp_count, udp_count

    logging.info("Packet captured")
    packet_info = {}

    # Ethernet Frame
    if packet.haslayer(Ether):
        ether = packet[Ether]
        packet_info['ether_dst'] = ether.dst
        packet_info['ether_src'] = ether.src
        packet_info['ether_type'] = hex(ether.type)

    # IPv4 Packet
    if packet.haslayer(IP):
        ip = packet[IP]
        packet_info['ip_src'] = ip.src
        packet_info['ip_dst'] = ip.dst
        packet_info['ip_proto'] = ip.proto

        # ICMP
        if ip.proto == 1 and packet.haslayer(ICMP):
            icmp = packet[ICMP]
            packet_info['icmp_type'] = icmp.type
            packet_info['icmp_code'] = icmp.code

        # TCP
        elif ip.proto == 6 and packet.haslayer(TCP):
            tcp = packet[TCP]
            packet_info['tcp_sport'] = tcp.sport
            packet_info['tcp_dport'] = tcp.dport
            tcp_count += 1  # Increment TCP packet count

        # UDP
        elif ip.proto == 17 and packet.haslayer(UDP):
            udp = packet[UDP]
            packet_info['udp_sport'] = udp.sport
            packet_info['udp_dport'] = udp.dport
            udp_count += 1  # Increment UDP packet count

    packets.append(packet_info)

    # Write the packet to the PCAP file
    if pcap_writer:
        pcap_writer.write(packet)
        logging.info("Packet written to capture.pcap")

# Function to start packet sniffing
def start_sniffing(interface=None):
    global sniffing_active, pcap_writer, tcp_count, udp_count
    sniffing_active = True
    packets.clear()  # Clear old packet data
    tcp_count = 0
    udp_count = 0

    # Create a pcap writer
    pcap_writer = PcapWriter("capture.pcap", append=True, sync=True)

    # Start sniffing in a separate thread
    logging.info(f"Starting sniffing on interface: {interface or 'all interfaces'}...")
    sniff_thread = threading.Thread(
        target=lambda: sniff(
            iface=interface,  # Specify the interface (or None for all)
            prn=process_packet,
            store=False,
            stop_filter=lambda p: not sniffing_active
        )
    )
    sniff_thread.start()
    logging.info("Sniffing thread started.")

# Function to stop packet sniffing
def stop_sniffing():
    global sniffing_active
    sniffing_active = False
    logging.info("Sniffing stopped.")

# Function to open Wireshark
def open_with_wireshark(pcap_file):
    try:
        subprocess.run(["C:\\Program Files\\Wireshark\\Wireshark.exe", pcap_file])
        logging.info("Wireshark opened successfully.")
    except FileNotFoundError:
        logging.error("Wireshark not found. Ensure it is installed and in your PATH.")

# Flask app setup
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/start_capture', methods=['POST'])
def start_capture():
    interface = request.json.get("interface", None)  # Optionally get the interface from the request
    start_sniffing(interface=interface)
    return jsonify({'status': 'Capture started', 'interface': interface or 'all interfaces'})

@app.route('/stop_capture', methods=['POST'])
def stop_capture():
    stop_sniffing()
    return jsonify({'status': 'Capture stopped'})

@app.route('/get_packets', methods=['GET'])
def get_packets():
    return jsonify(packets)

@app.route('/status', methods=['GET'])
def status():
    return jsonify({
        'status': 'Capturing' if sniffing_active else 'Idle',
        'packet_count': len(packets),
        'tcp_count': tcp_count,
        'udp_count': udp_count,
        'interfaces': get_if_list()  # Return available network interfaces
    })

@app.route('/open_wireshark', methods=['GET'])
def open_wireshark():
    open_with_wireshark("capture.pcap")
    return jsonify({'status': 'Wireshark opened with capture.pcap file'})

if __name__ == '__main__':
    # Print available interfaces for convenience
    logging.info(f"Available network interfaces: {get_if_list()}")
    app.run(debug=True)
```

SNAPSHOTS OF PROJECT


Packet Sniffer Web Interface

Select Interface: All Interfaces

Packets Captured: 28516

Filter by Protocol: All

Filter by IP (source or destination): 192.168.29.184

```
{ "ether_dst": "84:90:0a:f5:e9:c0", "ether_src": "9c:2f:9d:91:97:75", "ether_type": "0x86dd" }
```

CONCLUSION

This project demonstrates the development of a real-time network packet capture and analysis tool using Flask and Scapy, integrated with a user-friendly web interface. The key features include:

1. Packet Sniffing and Analysis: Captures network packets in real-time, processes them to extract essential information (Ethernet, IP, TCP, UDP, ICMP), and categorises them efficiently.
2. PCAP File Generation: Saves captured packets to a .pcap file, allowing further analysis in tools like Wireshark.
3. Dynamic Web Interface: Provides an intuitive and interactive interface for starting/stopping packet capture, viewing real-time packet statistics, and opening the .pcap file in Wireshark.
4. Multi-Interface Support: Enables packet sniffing across multiple network interfaces, improving adaptability for diverse environments.
5. Modular and Scalable Design: Implements threading for concurrent sniffing and Flask for RESTful interactions, ensuring scalability and performance.

This project successfully bridges the gap between low-level network monitoring and user-accessible analytics, making it a valuable tool for network troubleshooting, security monitoring, and educational purposes.