PRODIGY INFOTECH: CS_TASK_05

NETWORK PACKET ANALYZER

Check out this project on my github:
https://github.com/sajilsaju/PRODIGY_CS_05.git

PREPARED BY :

Sajil Mohamed V
+91 8714194649
sajilmohamed1234@gmail.com
www.linkedin.com/in/sajilmohamed1718/

Table Of Contents

PRODIGY INFOTECH

1. Introduction

Network traffic analysis is a fundamental aspect of cybersecurity, network management, and troubleshooting. A **Network Packet Analyzer** is a tool used to capture, examine, and interpret network packets that traverse a network. This project involves developing a Python-based packet analyzer that captures packets in real-time, providing insights into the data being transmitted across the network. The tool helps users understand network communication by displaying critical information such as source and destination IP addresses, protocols, ports, and payload content. The primary goal of this project is to create an educational tool for network analysis, ensuring that it is used ethically and within legal boundaries. This project also enhances knowledge of network protocols, packet structures, and traffic patterns, providing a foundation for more advanced network security tools and practices.

PRODIGY INFOTECH

2. Objectives

The Network Packet Analyzer project was undertaken with the following objectives in mind:

1. **Develop a Real-Time Packet Sniffing Tool:** Create a tool that captures and analyzes network packets in real-time to provide insights into network traffic.
2. **Display Detailed Packet Information:** Extract and display relevant details from each packet, such as source and destination IP addresses, protocols, port numbers, and payload content.
3. **Promote Ethical Use and Awareness:** Ensure the tool is designed for educational purposes, with a focus on responsible and ethical use in network analysis.
4. **Enhance Understanding of Network Protocols:** Improve knowledge of various network protocols (such as TCP, UDP, ICMP) and their behavior in a network environment.
5. **Test and Validate Across Multiple Environments:** Verify the tool's effectiveness by testing it across different network interfaces and environments.
6. **Facilitate Learning in Network Security:** Serve as a foundational learning tool for individuals new to network security, enabling them to gain hands-on experience with packet capturing and analysis.

3. Tools & Technologies Used

- **Python:** The primary programming language used to develop the tool due to its simplicity, flexibility, and extensive library support.
- **Scapy**: A Python library for packet manipulation and analysis. It provides powerful functionalities for crafting, sending, receiving, and analyzing packets**.**
- **Linux/Unix Environment:** The tool is designed to run on Linux-based systems, as they provide robust support for network programming and packet manipulation

4. Methodology

The development of the Network Packet Analyzer followed these steps:

Step 1: Requirements Analysis

- Identified the core requirements for the packet analyzer tool, including the ability to capture packets, display relevant information, and run in a Linux environment.
- Chose Python and the Scapy library as the primary technologies for their simplicity and powerful capabilities in handling network packets.

Step 2: Design and Development

- Designed a simple user interface that prompts the user to enter the network interface to monitor.
- Implemented packet capturing using Scapy's sniff function to intercept packets in real-time.

- Developed a callback function to process each captured packet, extracting relevant details such as source and destination IP addresses, protocols, ports, and payload data.
- Ensured that the script runs with root privileges to access the network interface.

Step 3: Testing and Validation

- Tested the tool on multiple network interfaces (e.g., Ethernet, Wi-Fi) to ensure it captures and displays packets correctly.
- Conducted tests in various network environments to verify the tool's ability to handle different types of traffic (e.g., TCP, UDP, ICMP).

Step 4: Documentation and Ethical Considerations

- Prepared detailed documentation, including a README file, to guide users in setting up and using the tool.
- Emphasized the importance of ethical use, ensuring the tool is only used for authorized and educational purposes.

PRODIGY INFOTECH

5. Program Code

Python Code of the program:

```
# Import necessary libraries
from scapy.all import sniff, IP, TCP, UDP, ICMP


def packet_callback(packet):
    # Extract IP layer
    if IP in packet:
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        protocol = packet[IP].proto

        # Determine the protocol type
        if protocol == 6:  # TCP
            proto = "TCP"
            src_port = packet[TCP].sport
            dst_port = packet[TCP].dport
        elif protocol == 17:  # UDP
            proto = "UDP"
            src_port = packet[UDP].sport
```

PRODIGY INFOTECH

```
            dst_port = packet[UDP].dport
        elif protocol == 1:  # ICMP
            proto = "ICMP"
            src_port = None
            dst_port = None
        else:
            proto = "Other"
            src_port = None
            dst_port = None

        # Display packet information
        print(f"Source IP: {ip_src}, Destination IP: {ip_dst}, Protocol: {proto}")
        if src_port and dst_port:
            print(f"Source Port: {src_port}, Destination Port: {dst_port}")

        # Display payload data if available
        if packet.haslayer(TCP) or packet.haslayer(UDP):
            payload = packet[TCP].payload if packet.haslayer(TCP) else
packet[UDP].payload
            print(f"Payload: {str(payload)}")
        print("-" * 50)

def start_sniffing(interface):
```

print(f"Starting packet sniffing on interface {interface}...")

Start sniffing packets on the given interface

sniff(iface=interface, prn=packet_callback, store=0)

if __name__ == "__main__":

Specify the network interface to sniff on (e.g., 'eth0' for Ethernet or 'wlan0' for Wi-Fi)

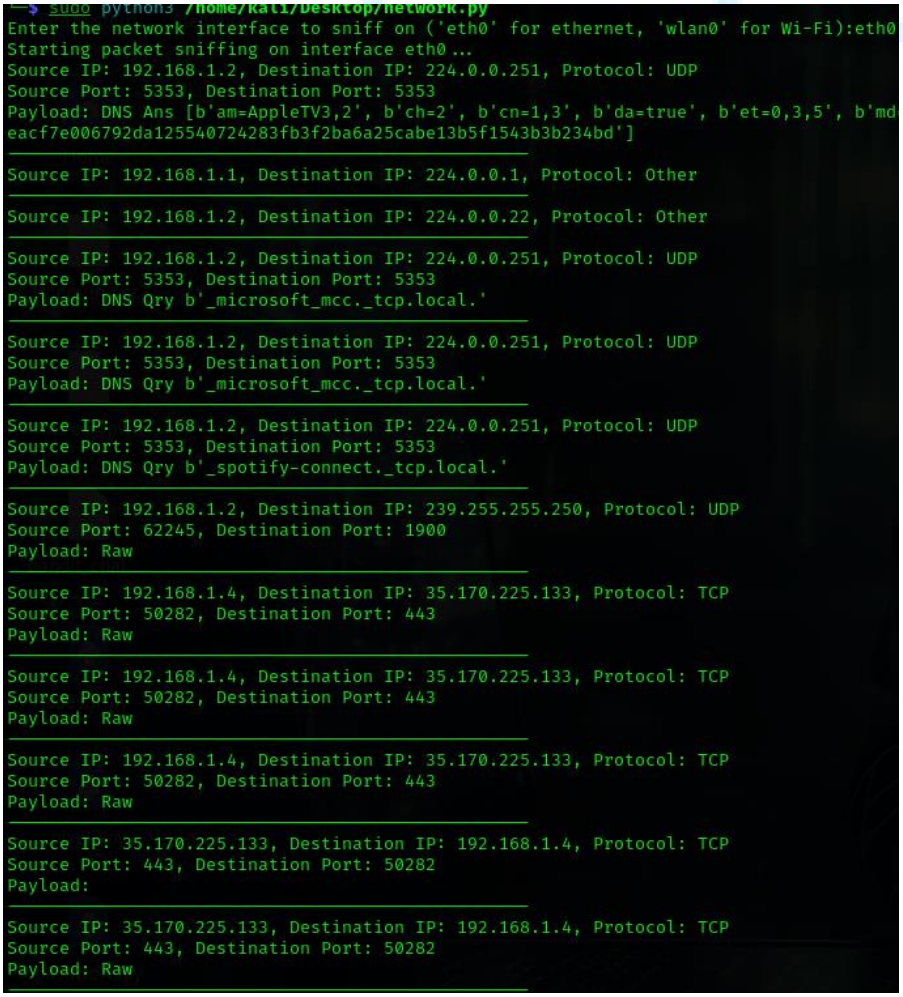
interface = input("Enter the network interface to sniff on ('eth0' for ethernet, 'wlan0' for Wi-Fi):")

start_sniffing(interface)

```python
# Import necessary libraries
from scapy.all import sniff, IP, TCP, UDP, ICMP

def packet_callback(packet):
    # Extract IP layer
    if IP in packet:
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        protocol = packet[IP].proto

        # Determine the protocol type
        if protocol == 6:  # TCP
            proto = "TCP"
            src_port = packet[TCP].sport
            dst_port = packet[TCP].dport
        elif protocol == 17:  # UDP
            proto = "UDP"
            src_port = packet[UDP].sport
            dst_port = packet[UDP].dport
        elif protocol == 1:  # ICMP
            proto = "ICMP"
            src_port = None
            dst_port = None
        else:
            proto = "Other"
            src_port = None
            dst_port = None

        # Display packet information
        print(f"Source IP: {ip_src}, Destination IP: {ip_dst}, Protocol: {proto}")
        if src_port and dst_port:
            print(f"Source Port: {src_port}, Destination Port: {dst_port}")

        # Display payload data if available
        if packet.haslayer(TCP) or packet.haslayer(UDP):
            payload = packet[TCP].payload if packet.haslayer(TCP) else packet[UDP].payload
            print(f"Payload: {str(payload)}")
        print("-" * 50)

def start_sniffing(interface):
    print(f"Starting packet sniffing on interface {interface}...")
    # Start sniffing packets on the given interface
    sniff(iface=interface, prn=packet_callback, store=0)

if __name__ == "__main__":
    # Specify the network interface to sniff on (e.g., 'eth0' for Ethernet or 'wlan0' for Wi-Fi)
    interface = input("Enter the network interface to sniff on (e.g., 'eth0', 'wlan0'): ")
```

6. Results

- **Real-Time Packet Capturing:** The tool was able to capture packets in real-time, displaying crucial information such as source and destination IP addresses, protocols, and ports.
- **Protocol Analysis:** The tool effectively differentiated between various protocols (TCP, UDP, ICMP) and displayed relevant details.
- **Payload Inspection:** The tool displayed payload data for deeper inspection, providing insights into the actual content of the packets.
- **Usability:** The tool was easy to use, requiring only basic command-line knowledge to operate.



7. Knowledge Gained

- **Understanding of Packet Sniffing**: Learned the fundamentals of packet sniffing and how network packets can be captured, analyzed, and monitored in real-time.
- **Network Protocols and Traffic Analysis**: Gained deeper insights into common network protocols such as TCP, UDP, and ICMP, including how they function and the structure of their packets.
- **Practical Use of Scapy Library**: Developed proficiency in using the Scapy library for network programming, particularly in crafting, sending, receiving, and decoding packets.
- **Python for Network Programming**: Enhanced knowledge of Python programming in the context of network applications, particularly in handling sockets and using raw packet data.
- **Ethical Considerations in Cybersecurity**: Developed an understanding of the ethical and legal considerations required when working with network data and analyzing potentially sensitive information.
- **Debugging and Problem Solving**: Improved skills in debugging, particularly in resolving permission-related issues and ensuring that the tool runs correctly on different operating systems and environments.

8. Challenges and Solutions

1. Challenge: Permission Denied Errors

***Solution*:** Encountered “PermissionError” when trying to run the packet-sniffing script due to restricted access to network interfaces. Resolved this by running the script with elevated privileges (using sudo on Linux) to grant the necessary permissions for capturing packets.

2. Challenge: Understanding Packet Structures

Solution: Initially found it challenging to interpret and extract specific information from different packet types. Addressed this by thoroughly studying the Scapy documentation and experimenting with packet inspection functions to learn how to identify and parse various protocol headers.

3. Challenge: Capturing and Handling High Volumes of Traffic

***Solution*:** When analyzing a busy network, the tool needed to handle a high volume of packets efficiently. Implemented packet filtering and limiting logic to capture only relevant packets (e.g., filtering by IP address or protocol type) to reduce overhead and focus on specific traffic.

4. Challenge: Ensuring Ethical Use

***Solution*:** Recognized the importance of ensuring that the tool is used ethically and only on networks where permission has been granted. Documented the ethical considerations clearly in the project README and report, reinforcing that the tool is intended for educational purposes only.

5. Challenge: Cross-Platform Compatibility

***Solution*:** Ensured the tool worked consistently across different Linux distributions by testing it in multiple environments. Addressed compatibility issues by writing platform-independent code and leveraging Python's cross-platform capabilities.

9. Conclusion

The development of the Network Packet Analyzer has provided valuable insights into the intricate workings of network communication and the tools needed to analyze network traffic effectively. Through this project, a practical and educational tool was created that captures and decodes network packets in real time, offering users a deeper understanding of various network protocols, packet structures, and data flows. The project emphasized the importance of ethical considerations in cybersecurity, reinforcing the need to use such tools responsibly and legally.

By leveraging the ***Scapy*** library and Python programming, the tool serves as a robust foundation for learning and experimenting with network traffic analysis. The challenges encountered during development, such as handling permissions and managing large volumes of traffic, provided critical learning experiences that enhanced problem-solving skills and technical knowledge.

Overall, this project not only achieved its technical objectives but also underscored the significance of continuous learning and ethical practice in cybersecurity. The Network Packet Analyzer stands as a useful educational tool for beginners in the field, with potential for further enhancements to expand its capabilities and applications in network security.

10. Future Works

- **Packet Filtering**: Implement filtering options to capture specific types of packets or traffic from certain IP ranges.
- **Logging Functionality**: Add support for logging captured packets to a file for later analysis.
- **Graphical User Interface (GUI)**: Develop a user-friendly GUI to make the tool more accessible to non-technical users.
- **Advanced Protocol Support**: Extend support for additional protocols (e.g., ARP, DNS) to provide more comprehensive analysis capabilities.

11. Reference

- Scapy Documentation: A comprehensive guide to the Scapy library and its capabilities in network packet manipulation and analysis.
- **About Packet Sniffing:** https://www.netscout.com/what-is/sniffer
- Python documentation for understanding socket programming and network interfaces.

PRODIGY INFOTECH