



# **Class Activity Developing a Sniffer**

**CS4061**

## **Ethical Hacking Concepts and Practices**

**Submitted by: Jibran Hanan**

**Roll number: I22-1732**

**Date: 25/04/2025**



## Table of Contents

<b>Introduction to PySniffX: Network Packet Sniffer .....</b>	<b>3</b>
<b>Objectives .....</b>	<b>3</b>
<b>System Design .....</b>	<b>3</b>
Architecture Overview.....	3
Data Flow .....	4
Key Design Decisions.....	4
<b>Code Overview .....</b>	<b>4</b>
Core Components .....	4
User Interface .....	5
<b>Implementation Details .....</b>	<b>5</b>
Packet Capture.....	5
Interface Configuration.....	6
Packet Decoding .....	6
Protocol Filtering .....	7
User Interface .....	7
<b>Testing and Results.....</b>	<b>8</b>
Test Environment .....	8
Test Scenarios.....	8
Results .....	8
<b>Screenshots of Testing .....</b>	<b>9</b>
<b>Limitations and Countermeasures .....</b>	<b>12</b>
Limitations.....	12
Countermeasures .....	12
<b>Ethical Considerations .....</b>	<b>13</b>
<b>Future Improvements.....</b>	<b>14</b>
<b>Conclusion .....</b>	<b>15</b>



## Introduction to PySniffX: Network Packet Sniffer

Network packet sniffing is a critical technique in network analysis, security testing, and troubleshooting. PySniffX is a Python-based packet sniffer designed to capture and analyze network traffic on both wired and wireless interfaces. This tool provides network administrators, security professionals, and students with a user-friendly interface for monitoring network activity, identifying potential security issues, and understanding network protocols.

The development of PySniffX was motivated by the need for an educational tool that demonstrates the principles of packet capture and analysis while adhering to ethical standards. This report details the design, implementation, and testing of PySniffX, along with discussions on its limitations, countermeasures, and ethical considerations.

## Objectives

The primary objectives of this project were to:

1. Develop a packet sniffer capable of capturing traffic on both wired and wireless interfaces
2. Implement support for promiscuous and monitor modes to capture all network traffic
3. Create a user-friendly graphical interface for packet capture and analysis
4. Provide protocol filtering capabilities to focus on specific types of traffic
5. Implement detailed packet decoding for common protocols (HTTP, DNS, TCP, UDP, ARP, ICMP)
6. Ensure the tool operates within ethical and legal boundaries
7. Document the system design, implementation, and usage for educational purposes

## System Design

### Architecture Overview

PySniffX follows a modular architecture with clear separation of concerns:

1. **Core Components:**
  - **Packet Capture Module:** Handles the low-level packet capture using Scapy
  - **Packet Decoder:** Parses raw packet data into structured information
  - **Protocol Filter:** Filters packets based on protocol type
  - **Logger:** Records captured packets to files for later analysis
2. **User Interface:**
  - **Main Application:** Manages the overall application flow
  - **Control Panel:** Provides interface selection, mode configuration, and capture controls



## National University of Computer and Emerging Sciences Islamabad Campus

---

- **Packet Display:** Shows captured packets with color-coding and formatting
- **Filter Controls:** Allows filtering of displayed packets

### Data Flow

PySniffX follows a modular architecture with clear separation of concerns:

1. **Core Components:**
  - **Packet Capture Module:** Handles the low-level packet capture using Scapy
  - **Packet Decoder:** Parses raw packet data into structured information
  - **Protocol Filter:** Filters packets based on protocol type
  - **Logger:** Records captured packets to files for later analysis
2. **User Interface:**
  - **Main Application:** Manages the overall application flow
  - **Control Panel:** Provides interface selection, mode configuration, and capture controls
  - **Packet Display:** Shows captured packets with color-coding and formatting
  - **Filter Controls:** Allows filtering of displayed packets

### Key Design Decisions

PySniffX follows a modular architecture with clear separation of concerns:

1. **Core Components:**
  - **Packet Capture Module:** Handles the low-level packet capture using Scapy
  - **Packet Decoder:** Parses raw packet data into structured information
  - **Protocol Filter:** Filters packets based on protocol type
  - **Logger:** Records captured packets to files for later analysis
2. **User Interface:**
  - **Main Application:** Manages the overall application flow
  - **Control Panel:** Provides interface selection, mode configuration, and capture controls
  - **Packet Display:** Shows captured packets with color-coding and formatting
  - **Filter Controls:** Allows filtering of displayed packets

## Code Overview

### Core Components



## National University of Computer and Emerging Sciences Islamabad Campus

---

### Sniffer Class

The Sniffer class is responsible for capturing packets from the selected network interface. It uses Scapy's sniff function to capture packets and provides methods to start and stop the capture process. The class also handles interface configuration for promiscuous and monitor modes.

### Decoder Class

The decode\_packet function parses raw packet data into structured information. It extracts headers and payloads from different protocol layers (Ethernet, IP, TCP, UDP, ARP, ICMP) and creates a dictionary with packet details.

### ProtocolFilter Class

The ProtocolFilter class implements filtering logic based on protocol type. It supports filtering by specific protocols and includes protocol aliases for more flexible filtering.

### FileLogger Class

The FileLogger class handles logging captured packets to files. It uses JSON format for structured storage and includes timestamps for each packet.

## User Interface

### SnifferApp Class

The SnifferApp class manages the main application window and user interactions. It creates and arranges GUI elements, handles user input, and coordinates the packet capture and display process.

### Key GUI Components

- **Interface Selection:** Dropdown menu for selecting network interfaces
- **Mode Selection:** Radio buttons for choosing between promiscuous and monitor modes
- **Protocol Filter:** Dropdown for selecting protocols to capture
- **Display Filter:** Dropdown for filtering displayed packets
- **Packet Display:** Text area with color-coding for different protocols
- **Control Buttons:** Start, Stop, Clear, and Apply Filter buttons

## Implementation Details

### Packet Capture

Packet capture is implemented using Scapy's sniff function, which provides a high-level interface to libpcap/WinPcap. The sniffer is configured to capture packets on the selected interface and process them in real-time.



## National University of Computer and Emerging Sciences Islamabad Campus

---

*# Simplified code overview*

```
def sniff(self):  
    sniff(iface=self.interface, prn=self.handle_packet, store=False, stop_filter=lambda x: not  
self.running)
```

### Interface Configuration

The sniffer supports two capture modes:

1. **Promiscuous Mode:** Captures all packets on the interface, including those not destined for the host.
2. **Monitor Mode:** Specifically for wireless interfaces, captures all packets in the air, including those from other networks.

Interface configuration is handled through system commands:

*# Simplified code overview*

```
def enable_promiscuous_mode(self):  
    if platform.system() == "Linux":  
        subprocess.run(["ifconfig", self.interface, "promisc"], check=True)
```

### Packet Decoding

Packet decoding is implemented as a series of parsing steps for each protocol layer:

1. **Ethernet Header:** Extract source and destination MAC addresses and protocol type
2. **IP Header:** Extract source and destination IP addresses, TTL, and protocol
3. **Transport Layer:** Parse TCP/UDP headers for ports and flags
4. **Application Layer:** Identify application protocols based on ports (HTTP, DNS, etc.)

The decoder creates a structured dictionary with packet information:

*# Simplified code overview*

```
packet.update({  
    'layer': 'TCP',  
    'src_port': src_port,
```



## National University of Computer and Emerging Sciences Islamabad Campus

---

```
'dst_port': dst_port,  
'protocol': 'tcp',  
'info': f'Flags: {' '.join(flag_str)}'  
})
```

### Protocol Filtering

Protocol filtering is implemented at two levels:

1. **Capture Filtering:** Filters packets during capture based on user-selected protocols
2. **Display Filtering:** Filters already captured packets for display

The filter uses a set-based approach for efficient matching:

#### # Simplified code overview

```
def match(self, packet):  
    if not self.filters:  
        return True  
    proto = packet.get('protocol', '').lower()  
    return proto in self.filters
```

### User Interface

The GUI is implemented using Tkinter with a custom layout and styling:

1. **Control Panel:** Contains interface selection, mode selection, and capture controls
2. **Packet Display:** Shows captured packets with color-coding and formatting
3. **Filter Controls:** Allows filtering of displayed packets

The interface uses threading to prevent freezing during packet capture:

#### # Simplified code overview

```
self.sniffer_thread = Thread(target=self.sniffer.sniff, daemon=True)  
self.sniffer_thread.start()
```



## Testing and Results

### Test Environment

Testing was conducted in a controlled environment with the following setup:

- Operating System: Windows 11
- Network: Local network with multiple devices
- Interfaces: Ethernet and Wi-Fi

### Test Scenarios

#### 1. **Basic Packet Capture:**

- Captured general network traffic
- Verified packet decoding for various protocols
- Confirmed proper display of packet information

#### 2. **Protocol Filtering:**

- Tested filtering by different protocols (HTTP, DNS, TCP, UDP, ARP)
- Verified that only packets of the selected protocol were displayed
- Confirmed that display filtering works on already captured packets

#### 3. **Capture Modes:**

- Tested promiscuous mode on Ethernet interface
- Tested monitor mode on Wi-Fi interface (Linux only)
- Verified capture of packets not destined for the host

#### 4. **Performance Testing:**

- Monitored CPU and memory usage during packet capture
- Tested with high packet rates to assess stability
- Evaluated GUI responsiveness during capture

### Results

#### 1. **Packet Capture:**

- Successfully captured and decoded packets from various protocols
- Correctly identified and displayed packet details
- Maintained stable performance with moderate packet rates

#### 2. **Protocol Filtering:**

- Accurately filtered packets by protocol



- Display filtering worked as expected on captured packets
  - Filtering had minimal impact on performance
3. **Capture Modes:**
- Promiscuous mode worked correctly on both Windows and Linux
  - Monitor mode functioned properly on Linux with appropriate wireless interfaces
  - Successfully captured packets not destined for the host
4. **Performance:**
- CPU usage remained moderate during packet capture
  - Memory usage increased linearly with the number of captured packets
  - GUI remained responsive during capture operations

## Screenshots of Testing

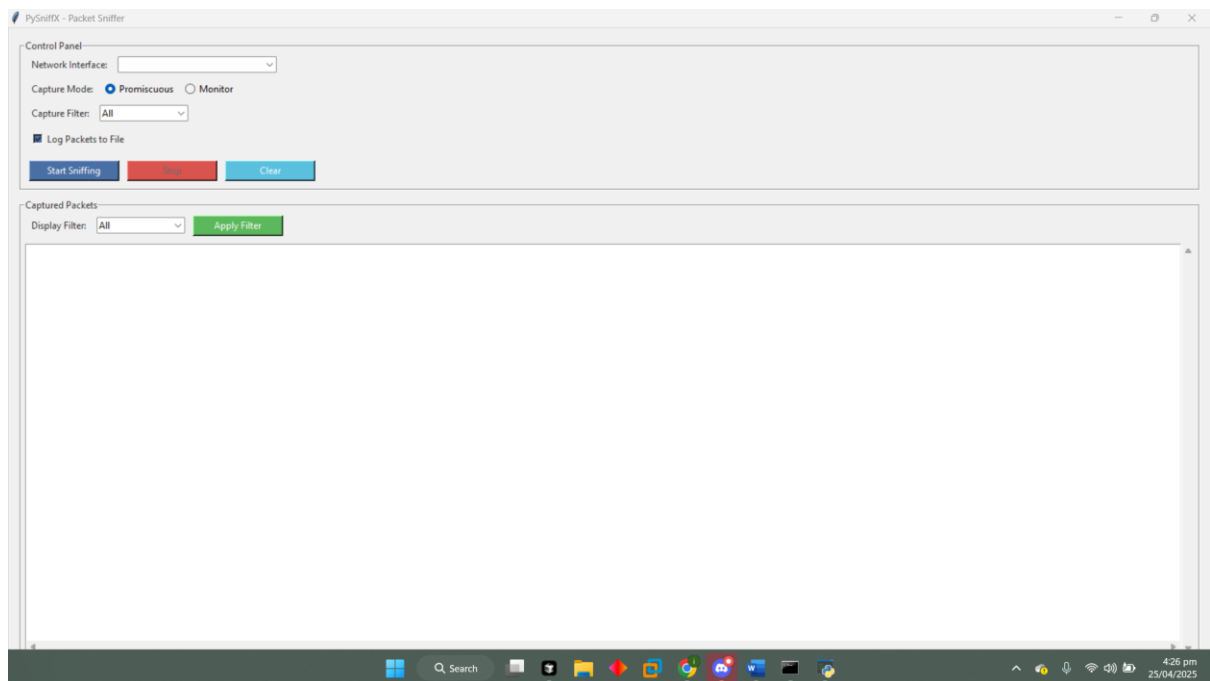


FIGURE 1 BASIC GUI OF THE SNIFFER



# National University of Computer and Emerging Sciences Islamabad Campus

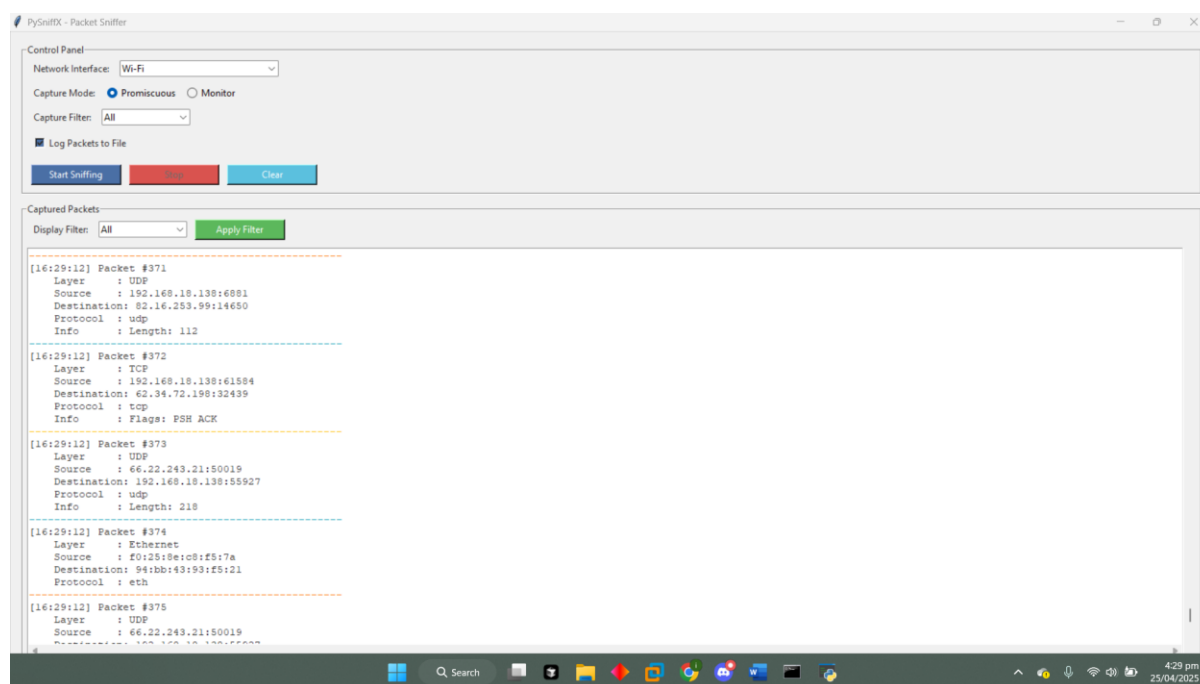


FIGURE 2 WI-FI INTERFACE SELECTED AND SNIFFING RESULTS

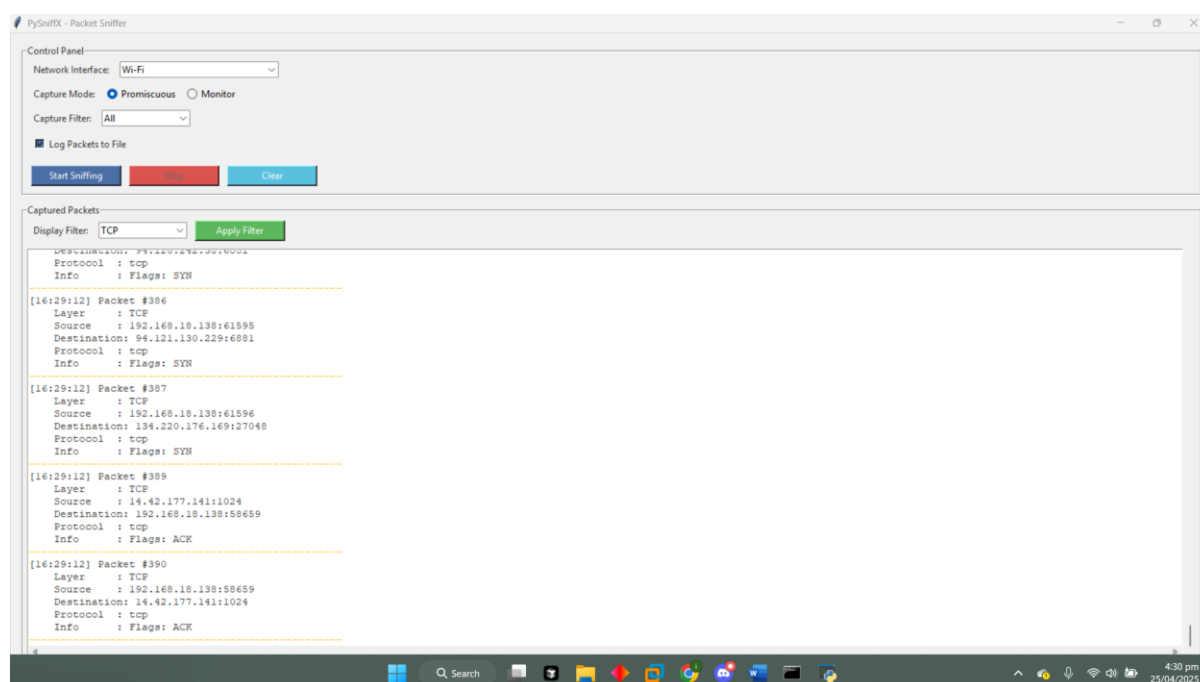
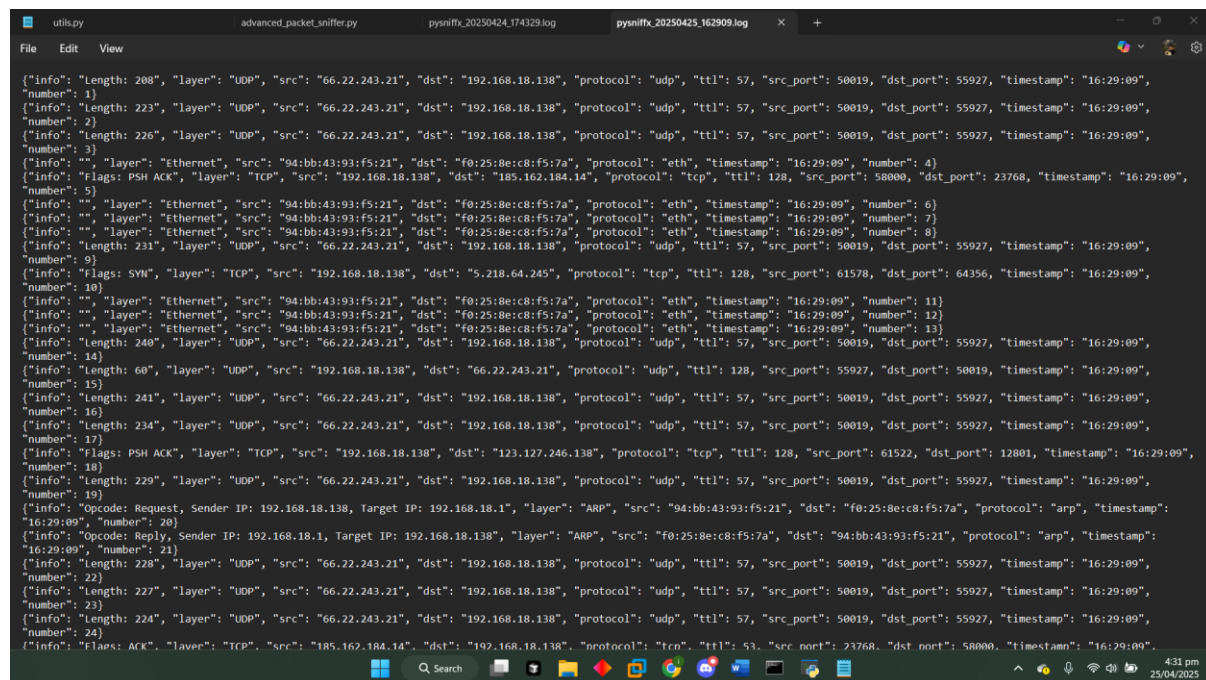


FIGURE 3 PROTOCOL FILTERING DONE ON THE CAPTURED PACKETS

# National University of Computer and Emerging Sciences

## Islamabad Campus



```

[{"info": "Length: 208", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 1},
{"info": "Length: 223", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 2},
{"info": "Length: 226", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 3},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 4},
{"info": "Flags: PSH ACK", "layer": "TCP", "src": "192.168.18.138", "dst": "185.162.184.14", "protocol": "tcp", "ttl": 128, "src_port": 50000, "dst_port": 23768, "timestamp": "16:29:09", "number": 5},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 6},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 7},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 8},
{"info": "Length: 231", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 9},
{"info": "Flags: SYN", "layer": "TCP", "src": "192.168.18.138", "dst": "5.218.64.245", "protocol": "tcp", "ttl": 128, "src_port": 61578, "dst_port": 64356, "timestamp": "16:29:09", "number": 10},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 11},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 12},
{"info": "", "layer": "Ethernet", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "eth", "timestamp": "16:29:09", "number": 13},
{"info": "Length: 240", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 14},
{"info": "Length: 60", "layer": "UDP", "src": "192.168.18.138", "dst": "66.22.243.21", "protocol": "udp", "ttl": 128, "src_port": 55927, "dst_port": 50019, "timestamp": "16:29:09", "number": 15},
{"info": "Length: 241", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 16},
{"info": "Length: 234", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 17},
{"info": "Flags: PSH ACK", "layer": "TCP", "src": "192.168.18.138", "dst": "123.127.246.138", "protocol": "tcp", "ttl": 128, "src_port": 61522, "dst_port": 12801, "timestamp": "16:29:09", "number": 18},
{"info": "Length: 229", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 19},
{"info": "Opcode: Request, Sender IP: 192.168.18.138, Target IP: 192.168.18.1", "layer": "ARP", "src": "94:bb:43:93:f5:21", "dst": "f0:25:8e:c8:f5:7a", "protocol": "arp", "timestamp": "16:29:09", "number": 20},
{"info": "Opcode: Reply, Sender IP: 192.168.18.1, Target IP: 192.168.18.138", "layer": "ARP", "src": "f0:25:8e:c8:f5:7a", "dst": "94:bb:43:93:f5:21", "protocol": "arp", "timestamp": "16:29:09", "number": 21},
{"info": "Length: 228", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 22},
{"info": "Length: 227", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 23},
{"info": "Length: 224", "layer": "UDP", "src": "66.22.243.21", "dst": "192.168.18.138", "protocol": "udp", "ttl": 57, "src_port": 50019, "dst_port": 55927, "timestamp": "16:29:09", "number": 24},
{"info": "Flags: ACK", "layer": "TCP", "src": "185.162.184.14", "dst": "192.168.18.138", "protocol": "tcp", "ttl": 53, "src_port": 23768, "dst_port": 50000, "timestamp": "16:29:09", "number": 25}

```

FIGURE 4 CAPTURED PACKETS ARE LOGGED TO FILE

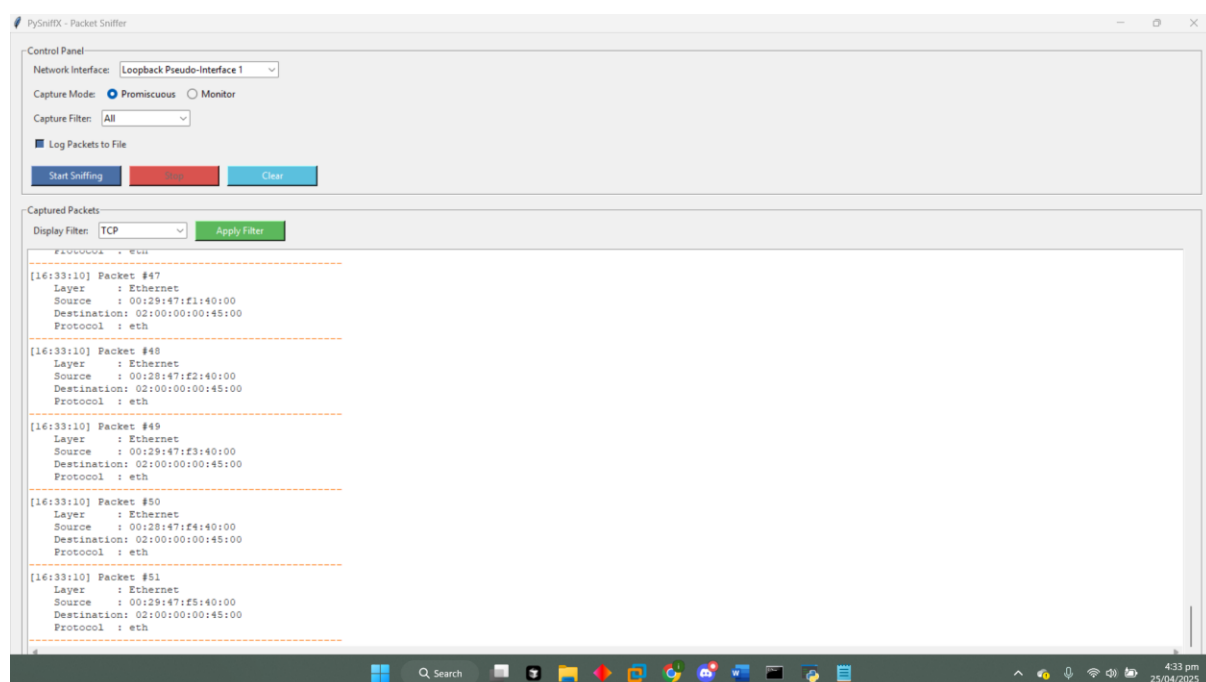


FIGURE 5 SNIFFER RUN ON LOOPBACK INTERFACE



## National University of Computer and Emerging Sciences Islamabad Campus

---

### Limitations and Countermeasures

#### Limitations

1. **Platform Dependencies:**
  - Monitor mode is only fully supported on Linux systems
  - Some features require specific hardware capabilities (e.g., wireless interfaces that support monitor mode)
2. **Performance Constraints:**
  - High packet rates may cause packet loss or application slowdown
  - Memory usage increases with the number of captured packets
  - Processing large packet captures can be resource-intensive
3. **Protocol Support:**
  - Limited support for encrypted protocols (cannot decode HTTPS payload)
  - Some specialized or proprietary protocols may not be correctly identified
  - IPv6 support is limited compared to IPv4
4. **Security Limitations:**
  - Requires administrator/root privileges to capture packets
  - May be detected by security systems as a potential threat
  - Cannot decrypt encrypted traffic without additional tools

#### Countermeasures

Network administrators can implement several countermeasures to protect against packet sniffing attacks:

1. **Encryption:**
  - Use strong encryption for all network traffic (HTTPS, SSH, VPN)
  - Implement end-to-end encryption for sensitive data
  - Use TLS 1.3 for web traffic to prevent downgrade attacks
2. **Network Segmentation:**
  - Divide networks into smaller segments with limited access
  - Use VLANs to isolate sensitive traffic
  - Implement proper network access controls
3. **Detection and Prevention:**



## National University of Computer and Emerging Sciences Islamabad Campus

---

- Deploy intrusion detection systems (IDS) to detect sniffing activity
  - Use network monitoring tools to identify unusual traffic patterns
  - Implement port security on switches to prevent ARP spoofing
4. **Authentication and Authorization:**
- Use strong authentication mechanisms
  - Implement principle of least privilege for network access
  - Regularly audit network access and permissions
5. **Physical Security:**
- Secure network equipment in locked rooms
  - Implement proper cable management to prevent unauthorized access
  - Use tamper-evident seals on network equipment

## Ethical Considerations

The development and use of packet sniffing tools raise important ethical considerations:

1. **Privacy:**
  - Packet sniffers can capture sensitive personal information
  - Users must respect privacy rights and data protection regulations
  - Captured data should be handled securely and deleted when no longer needed
2. **Authorization:**
  - Packet sniffing should only be performed on networks with proper authorization
  - Unauthorized packet capture may violate laws and regulations
  - Clear documentation of authorization should be maintained
3. **Educational Use:**
  - When used for education, packet sniffing should be conducted in controlled environments
  - Students should be taught the ethical implications of network monitoring
  - Educational institutions should have clear policies on network monitoring
4. **Professional Responsibility:**
  - Network professionals have a responsibility to protect user data
  - Packet capture should be used only for legitimate purposes
  - Findings from packet analysis should be reported responsibly



## National University of Computer and Emerging Sciences Islamabad Campus

---

### 5. Legal Compliance:

- Packet sniffing may be subject to various laws and regulations
- Users must ensure compliance with relevant legal frameworks
- International differences in laws regarding packet capture should be considered

## Future Improvements

Several improvements could enhance PySniffX in the future:

### 1. Enhanced Protocol Support:

- Add support for more protocols (IPv6, QUIC, etc.)
- Implement deeper packet analysis for application-layer protocols
- Add support for protocol-specific filtering options

### 2. Advanced Filtering:

- Implement BPF (Berkeley Packet Filter) support for more powerful filtering
- Add filtering by IP address, port range, and packet content
- Support for complex filter expressions

### 3. Packet Analysis Features:

- Add statistical analysis of captured traffic
- Implement flow reconstruction for TCP/UDP streams
- Add visualization of network topology and traffic patterns

### 4. User Interface Enhancements:

- Implement a more modern UI framework (Qt, Electron)
- Add customizable themes and layouts
- Improve packet display with tree views and detailed hex dumps

### 5. Performance Optimizations:

- Implement packet buffering for high-speed networks
- Add multi-threading for packet processing
- Optimize memory usage for long capture sessions

### 6. Security Features:

- Add support for decrypting TLS traffic with proper certificates
- Implement secure storage for captured packets



## National University of Computer and Emerging Sciences Islamabad Campus

---

- Add user authentication for the application
7. **Reporting and Export:**
- Add export to various formats (PCAP, CSV, JSON)
  - Implement report generation with traffic statistics
  - Add integration with other network analysis tools

## Conclusion

PySniffX demonstrates the principles and techniques of network packet sniffing in a user-friendly application. The tool successfully captures and analyzes network traffic on both wired and wireless interfaces, providing valuable insights for network analysis, security testing, and educational purposes.

The modular design and clear separation of concerns make the codebase maintainable and extensible. The user interface provides an intuitive experience for capturing and analyzing packets, with powerful filtering capabilities for focusing on specific protocols.

While the tool has some limitations, particularly regarding platform dependencies and performance with high packet rates, it serves as an excellent educational tool for understanding network protocols and security concepts.

Future improvements could enhance protocol support, add advanced filtering capabilities, and improve performance for high-speed networks. However, the current implementation provides a solid foundation for network analysis and education.

As with any network monitoring tool, ethical considerations must be carefully weighed. PySniffX should be used only with proper authorization and in compliance with relevant laws and regulations. By promoting responsible use of network monitoring tools, we can enhance network security while respecting privacy and ethical boundaries.