**Project Report: Real-Time Network Packet Sniffer with Alert System**

# **Abstract**

This report details the design, development, and implementation of a real-time network traffic sniffer with an integrated anomaly detection system. The primary objective of this project was to create a robust tool capable of capturing, analyzing, and logging network packets from a live interface. The system successfully identifies common network threats, such as port scanning and traffic flooding, by monitoring packet flow against predefined thresholds. Key functionalities include logging all captured packet metadata to an SQLite database, generating real-time alerts for suspicious activities, and providing a live graphical visualization of network traffic. The final deliverable is a functional command-line and GUI-based application built using Python and several specialized libraries, serving as an effective solution for basic network security monitoring.

# **Introduction**

In an era of increasing cyber threats, effective network monitoring is a fundamental component of any robust security posture. The ability to observe and analyze network traffic in real-time allows for the early detection of malicious activities, potential security breaches, and network misconfigurations. Manual traffic analysis is often impractical due to the sheer volume of data traversing a network. The objective of this project was to design and implement an automated tool to address this challenge. The goal was to build a Network Packet Sniffer that not only captures and displays packet data but also intelligently analyzes it to detect anomalies. By setting thresholds for common attack patterns, the system can provide immediate alerts to a network administrator, enabling a faster response to potential threats. This report outlines the architecture, tools, and methodology used to build this comprehensive monitoring solution.

# **Tools and Technologies Used**

The project was built with **Python 3** as the core language, leveraging several powerful open-source tools:

* **Scapy** – for low-level packet capture, analysis, and manipulation.
* **SQLite3** – for lightweight, file-based data storage of packet logs and alerts.
* **Matplotlib** – to visualize live network traffic in an optional GUI.
* **Threading** – to enable concurrent packet sniffing, analysis, and visualization.
* **VS Code** – for efficient coding, debugging, and project management.

# **Steps Involved in Building the Project**

The development process was structured into a series of logical, iterative steps to ensure each component was functional before integrating it into the larger system. Step 1: Environment and Database Setup The initial phase involved setting up a dedicated project environment using a Python virtual environment (venv) to manage dependencies. A setup script (database\_setup.py) was created to initialize the SQLite database, defining two primary tables: one (packets) for storing metadata of every captured packet and another (alerts) for logging security events. Step 2: Core Packet Sniffing Logic Using the Scapy library, the main sniffing engine was developed. The sniff() function from Scapy was configured to capture packets from the network interface in real-time. A callback function, process\_packet, was implemented to be executed for each packet captured. This function was responsible for extracting relevant header information, such as source/destination IP addresses, ports, protocol, and flags. Step 3: Anomaly Detection Implementation This step focused on building the intelligence of the system. Two primary anomaly detection rules were implemented: Flood Detection: A dictionary was used to count the number of packets received per second. If this count exceeded a predefined FLOOD\_THRESHOLD, an alert was triggered. Port Scan Detection: A dictionary tracked the unique destination ports being contacted by each source IP. If a single IP address attempted to connect to a number of ports exceeding the PORT\_SCAN\_THRESHOLD within a time window, a port scan alert was generated. Step 4: Alerting and Logging System An integrated log\_alert function was created to serve as a central hub for all alerts. When an anomaly was detected, this function would: - Print a color-coded alert message to the console. - Write the alert details to a persistent log file (alerts.log). - Insert a record of the alert into the alerts table in the SQLite database. Step 5: GUI for Live Traffic Visualization To provide a more intuitive user experience, a graphical interface was added using Matplotlib. The GUI runs in a separate thread and uses FuncAnimation to update a line graph every second. This graph plots the total number of packets per second, giving a live visual representation of network load and helping to spot traffic spikes instantly. Threading was essential to ensure the GUI remained responsive while the sniffing and analysis continued in the background.

# **Conclusion**

The project successfully culminated in the creation of a functional and effective network monitoring tool that meets all the initial objectives. The application is capable of real-time packet capture, analysis, and persistent logging. Its anomaly detection engine provides a valuable first line of defense by identifying suspicious traffic patterns, and the live GUI offers an accessible way to visualize network activity.