**Real-Time Intrusion Detection System**

Project submitted for the partial fulfillment of the requirements for the course

**CSE 337L: Cryptography Lab**

Offered by the

**Department Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

1) Darbha Sudha Vaishnavi, AP22110010201

2) Indu kukatikonda , AP22110011475

3) Roda Chinthapalli, AP22110011496

4) Raya Srujana Reddy, AP22110011502

5) Tejaswi Darsi, AP22110011503

**A picture containing text

Description automatically generated**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[April, 2025]**

**Contents:**

1. **Introduction**

**Purpose of the Project**

The purpose of this project is to develop a real-time Intrusion Detection System (IDS) that monitors network traffic, identifies potential security threats, and provides immediate feedback to network administrators. In an era where cyber threats such as Denial-of-Service (DoS) attacks, port scanning, and malware are prevalent, the need for proactive security measures is critical. This IDS leverages machine learning to classify network packets as "Normal" or "Attack (Alert)" based on features extracted from live traffic, offering a scalable solution for enhancing network security. The system is implemented using Python, with Scapy for packet capture, Flask for a web-based dashboard, and a Logistic Regression model trained on the KDD dataset.

**Background**

**Why It Is Relevant**

Intrusion detection is increasingly vital due to the rising sophistication of cyber-attacks, which exploit network vulnerabilities and threaten organizations with downtime, data breaches, and financial losses. For example, a Denial-of-Service (DoS) attack can flood a server with SYN packets, as seen in the project logs where serror\_rate reached 0.75, indicating a potential SYN flood that exhausts resources. This highlights the need for real-time detection to mitigate such threats effectively in modern networks.

**Possible Ways to Solve It**

Several approaches have been developed to address the challenge of intrusion detection, each with distinct methodologies and trade-offs:

* **Signature-Based Detection**: This method relies on a database of known attack signatures or patterns, such as specific packet sequences associated with malware or exploits. It is highly effective for detecting previously identified threats, such as a known virus signature, and is commonly used in antivirus software. However, its limitation lies in its inability to detect novel or zero-day attacks that lack pre-defined signatures, rendering it ineffective against evolving threats.
* **Anomaly-Based Detection**: This approach uses statistical models, machine learning, or behavioral analysis to establish a baseline of normal network activity and flag deviations as potential intrusions. It is particularly suitable for identifying unknown threats, such as unusual traffic spikes or atypical packet behaviors (e.g., the high serror\_rate in the project logs). The advantage is its adaptability to new attack vectors, but it can suffer from false positives, misclassifying legitimate anomalies (e.g., a sudden legitimate traffic surge) as attacks, necessitating manual verification.
* **Hybrid Systems**: These combine signature-based and anomaly-based techniques to leverage the strengths of both. For example, a hybrid system might use signature detection to catch known malware while employing anomaly detection to flag unusual patterns, such as a sudden increase in outbound traffic indicative of a data exfiltration attempt. This approach offers comprehensive coverage but increases complexity and resource demands.

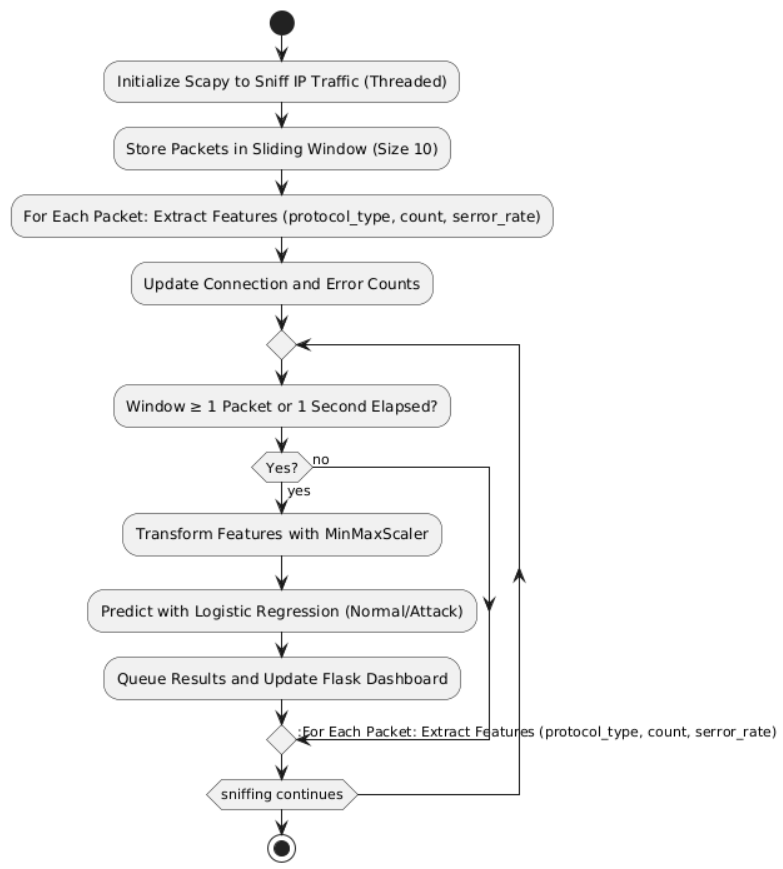
This project adopts an anomaly-based approach with machine learning, specifically using a Logistic Regression model trained on the KDD dataset. This choice offers adaptability to evolving threats by learning from historical data, enabling the system to detect anomalies like SYN floods or port scans in real-time. The use of live packet analysis further enhances its relevance, providing immediate insights into network security.

1. **Proposed Approach**

**Explain the Approach with Algorithm and Flowchart**

The proposed approach for the Real-Time Intrusion Detection System (IDS) involves a systematic process of capturing network packets, extracting relevant features, and classifying them using a pre-trained machine learning model. The algorithm is outlined as follows:

1. Initialize Scapy to sniff packets with a filter for IP traffic, executed in a separate thread to ensure continuous capture without blocking the main application.
2. Store captured packets in a sliding window of size 10, maintaining a recent history of network activity and discarding the oldest packet as new ones arrive.
3. For each packet:
   * Extract features such as protocol\_type, count (number of connections to a destination IP), and serror\_rate (ratio of SYN errors) from packet headers and aggregated counts.
   * Update connection counts (connection\_counts) and error counts (serror\_counts, srv\_serror\_counts) for real-time tracking.
   * Proceed to the next step if the sliding window contains at least 1 packet or 1 second has elapsed since the last prediction.
4. Transform the extracted features using a pre-trained MinMaxScaler to normalize values, ensuring compatibility with the training data.
5. Predict the packet’s classification using a pre-trained Logistic Regression model, generating probabilities for "Normal" and "Attack" classes, with a 0.5 threshold determining the final label ("Normal" or "Attack (Alert)").
6. Queue the prediction results, including raw data, scaled features, probabilities, and labels, and update the Flask dashboard for real-time display.



**Explain the System Design**

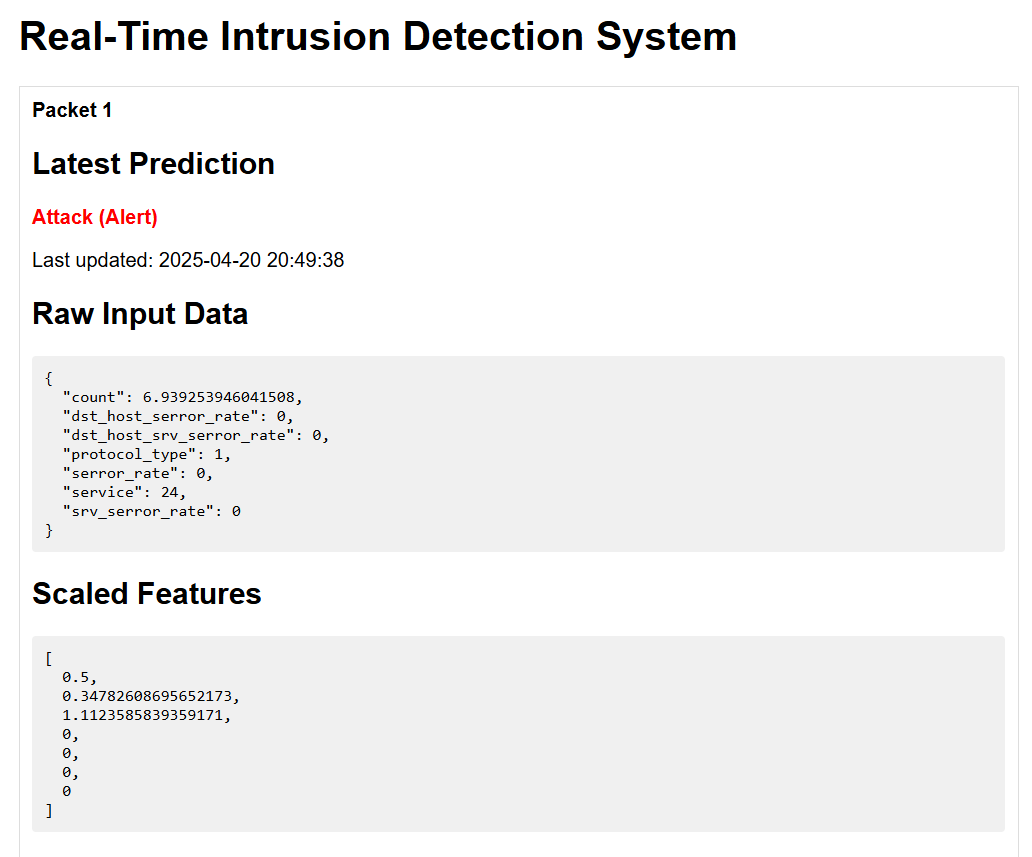
The system is designed as a modular, three-layered architecture to ensure efficient packet processing and real-time presentation. The components are as follows:

* Packet Capture Layer: This layer employs Scapy to intercept live network traffic, utilizing the sniff function with an "ip" filter to focus on IP-based packets. Executed in a separate thread, it prevents interference with the Flask server, ensuring uninterrupted capture and feeding packets into the processing pipeline.
* Processing Layer: This layer manages the core analytics, performing feature extraction from packet headers (e.g., TCP flags for serror\_rate) and aggregated window data, followed by model prediction using a pre-trained Logistic Regression model. A queue.Queue handles asynchronous data flow, storing results for the presentation layer, though it may face performance challenges during high traffic or complex computations.
* Presentation Layer: This layer features a Flask web server that renders an HTML dashboard, updated every second via JavaScript polling. The dashboard displays packet history, sorted by timestamp, with color-coded predictions (red for "Attack," green for "Normal"), presenting raw features, scaled values, and probabilities in a structured format for user accessibility.

1. **Results & Discussion:**

**Explain the Result Obtained**

The IDS successfully captured and classified network packets in real-time. Logs from April 20, 2025, show examples like {'protocol\_type': 'tcp', 'service': 'https', 'count': 12, 'serror\_rate': 0.25, ...}, with some packets flagged as "Attack (Alert)" due to high serror\_rate (e.g., 0.75), indicating potential SYN floods. The dashboard now retains packet history, reducing the "Waiting for data..." issue, as implemented in the latest UI update.



However, challenges include slow updates during low traffic, with the UI lagging behind packet arrival, and warnings for unknown services (e.g., https) due to encoder limitations.

1. **Conclusion**

This project has successfully developed a functional real-time Intrusion Detection System (IDS) that leverages machine learning to detect network intrusions with notable efficacy. By utilizing Scapy for packet capture, a Logistic Regression model trained on the KDD dataset for classification, and a Flask-based dashboard for visualization, the system provides both historical and current insights into network security. The dashboard’s recent enhancement to retain packet history addresses previous display delays, enabling continuous monitoring of traffic patterns, such as those indicated by high serror\_rate values in the logs, and offering actionable alerts for potential threats.

**Limitations**

Despite these achievements, the project faces several limitations that impact its performance and scope:

* **Incomplete Feature Extraction**: The system currently extracts only a subset of features (e.g., protocol\_type, count, serror\_rate), lacking critical metrics such as duration, src\_bytes, and dst\_bytes, which are essential for comprehensive intrusion detection and may reduce detection accuracy for certain attack types.
* **Dependency on Pre-trained Data**: The reliance on a pre-trained model limits adaptability to new or evolving threats not represented in the KDD dataset, potentially leading to missed detections if the training data becomes outdated.
* **Performance Issues During Low Traffic**: The system exhibits slow updates and frequent "Waiting for data..." messages during periods of low network activity, attributed to the 1-second prediction interval and insufficient packet accumulation in the sliding window.
* **Lack of Payload Analysis**: The absence of deep packet inspection (e.g., analyzing packet payloads) restricts the detection of application-layer attacks, such as SQL injection or command-and-control traffic.
* **Scalability Constraints**: The current design, with a fixed window size and single-threaded processing, may struggle to handle high-volume traffic, leading to potential bottlenecks in real-world deployments.

**Future Work**

To address these limitations and enhance the system’s capabilities, the following areas are proposed for future development:

* **Enhancing Feature Extraction with Session Tracking**: Implement session-based analysis to calculate duration, src\_bytes, and dst\_bytes by tracking packet flows over time, improving the model’s ability to detect prolonged attacks like data exfiltration.
* **Optimizing Prediction Speed**: Reduce the prediction interval (e.g., to 0.5 seconds) or parallelize model processing to minimize delays, ensuring timely updates even during low traffic, potentially using multi-threading or GPU acceleration.
* **Integrating Payload Analysis**: Incorporate payload inspection using libraries like dpkt or pyshark to detect application-layer threats, such as malicious commands or anomalous data patterns, expanding the system’s threat coverage.
* **Dynamic Model Updating**: Develop a mechanism to retrain or fine-tune the model with real-time data, reducing dependency on pre-trained datasets and enhancing adaptability to new attack vectors.
* **Improving Scalability**: Increase window size dynamically based on traffic volume or deploy a distributed architecture to handle high-throughput networks, ensuring robust performance in large-scale environments.
* **Adding Alert Mechanisms**: Integrate automated notifications (e.g., email or SMS) for critical alerts, enabling proactive response by network administrators.

1. **References**

1. Jehuty4949. (n.d.). *NSL\_KDD/KDDTrain+.txt*. Retrieved from [<https://github.com/Jehuty4949/NSL_KDD/blob/master/KDDTrain+.txt>]  
   *Note: Dataset hosted on GitHub, used to train the Logistic Regression model for intrusion detection in this project.*
2. Npcap. (n.d.). *Npcap: WinPcap for Windows*. Retrieved from [<https://npcap.com/#download>]  
   *Note: Required for enabling packet sniffing capabilities through Scapy on Windows.*
3. Scapy Documentation. (n.d.). *Scapy: Packet crafting for Python*. Retrieved from [<https://scapy.readthedocs.io/>]  
   *Note: Used for live packet capture and analysis in the dynamic intrusion detection module.*