IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Intelligent Anomaly Detection And Cyber Defence System

[1]S S Saranya, [2]T V V S Manikanta, [3]K Sri Ram, [4]E A N Durga Prasad, [5]P Rama Krishna,

[1]Assistant Professor, [2]Final Year B.Tech Student, [3]Final Year B.Tech Student, [4]Final Year B.Tech Student, [5]Final Year B.Tech Student,

Department of CSE-AIML,

Aditya College of Engineering & Technology(A), Surampalem, Andhra Pradesh, India

Abstract: This paper presents an Intelligent Anomaly Detection and Cyber Defence System — a real-time Network Intrusion Detection System (NIDS) capable of classifying network traffic into five attack superclasses: Normal, DOS, PROBE, R2L, and U2R. The system implements a dual prediction pipeline architecture: a Legacy Path utilizing full-session feature classification based on the 41-feature NSL-KDD dataset, and an Early-Stage Path enabling fast probabilistic risk scoring derived from 2-second packet aggregation windows. The Early-Stage Path extracts six lightweight behavioral features — packet count, SYN rate, average packet size, payload entropy, connection attempts, and window duration — and maps them through either a trained RandomForest classifier or a heuristic risk estimator when the trained model is unavailable. The system is deployed as a Flask-based web application integrated with a Scapy packet capture daemon, a thread-safe producer-consumer architecture, and a real-time browser dashboard with live polling. A training pipeline converts NSL-KDD connection records into synthetic 2-second windows using sliding window aggregation, and a RandomForestClassifier with balanced class weighting is trained on the resulting dataset. Experimental results demonstrate effective detection of DOS and PROBE attacks in real-time, with graceful degradation to heuristic estimation under model-unavailable conditions. The system bridges the gap between offline model training and live network defense, providing a deployable, scalable intrusion detection framework.

Keywords: Network Intrusion Detection System, Machine Learning, Random Forest, NSL-KDD, Anomaly Detection, Real-Time Traffic Analysis, Shannon Entropy, Cyber Security, DoS Detection, Packet Analysis**.**

I. INTRODUCTION

Cyber threats such as Denial-of-Service (DOS) attacks, network probing, Remote-to-Local (R2L) exploits, and User-to-Root (U2R) privilege escalations continue to pose critical risks to organizational networks worldwide. Traditional security systems, including signature-based intrusion detection tools, struggle to detect novel or rapidly evolving attack patterns. This limitation has driven growing interest in machine learning-based network intrusion detection systems (NIDS) capable of generalizing across unseen attack types.

However, a persistent challenge in deploying machine learning NIDS solutions in real-world environments is the gap between offline model training and live network monitoring. Historical datasets such as NSL-KDD contain full-session features derived from complete TCP connection records, whereas real-time packet streams provide only partial, incremental information within short time windows. This discrepancy makes it difficult to directly apply trained models to live traffic without session reconstruction overhead.

This paper presents an Intelligent Anomaly Detection and Cyber Defence System that addresses this challenge through a dual-pipeline prediction architecture. The Legacy Path supports full-feature classification compatible with NSL-KDD-trained models. The Early-Stage Path extracts six lightweight behavioral features from 2-second packet aggregation windows and performs rapid probabilistic risk scoring using a newly trained RandomForest classifier, with a heuristic fallback for model-unavailable conditions.

The system is built on a Flask web server integrated with a Scapy-based packet capture daemon, real-time browser dashboard, and a complete offline training pipeline. This design enables continuous network traffic monitoring, immediate alert generation, and graceful degradation when trained models are not yet available — making the system both deployable and extensible for real-world cyber defence applications.

II. RELATED WORK

Machine learning-based intrusion detection has been extensively studied using benchmark datasets such as KDD Cup 1999 and its refined variant, NSL-KDD. Early approaches employed decision trees, naive Bayes classifiers, and support vector machines for binary classification of network traffic into normal and attack categories. While these methods demonstrated high accuracy on benchmark datasets, they generally relied on full-session features requiring complete connection records before classification could be performed.

RandomForest-based classifiers have emerged as a popular choice for multi-class intrusion detection due to their robustness against overfitting, ability to handle imbalanced datasets through class weighting, and native support for probabilistic output via predict_proba(). Studies have shown that ensemble methods consistently outperform single classifiers on NSL-KDD in terms of F1-score and per-class detection rates, particularly for minority classes such as R2L and U2R.

Real-time packet capture and feature extraction using libraries such as Scapy and libpcap have been explored in the context of lightweight network monitoring tools. Shannon entropy of payload bytes has been identified as an effective feature for detecting encrypted or random-payload traffic associated with certain attack types. SYN rate measurement has long been used as a primary indicator for TCP-based DOS flooding attacks.

Despite these advances, existing systems typically operate in one of two modes: either offline batch classification of captured traffic, or real-time rule-based detection without machine learning inference. The proposed system bridges these paradigms by implementing a live packet capture daemon that feeds into a machine learning prediction pipeline while maintaining backward compatibility with full-feature legacy models.

III. SYSTEM ARCHITECTURE.

A. Packet Capture Subsystem

The realtime_sniffer.py module implements a Scapy-based packet capture daemon using a producer-consumer pattern with thread-safe buffer management. The WindowSniffer class spawns a Scapy sniffer thread that appends incoming packets to a shared buffer via a threading.Lock-protected callback. A periodic timer thread fires every 2 seconds, acquires the lock, copies and clears the buffer, and invokes the feature extraction pipeline. The extracted features are then transmitted to the Flask server via HTTP POST to the /ingest endpoint. This decoupled design ensures that packet capture remains low-latency and non-blocking, independent of prediction processing time.

B. Feature Extraction Layer

The early_features.py module transforms raw packet streams into 6-dimensional feature vectors suitable for machine learning inference. For each 2-second window, the module computes: packet_count (total packets captured), syn_rate (fraction of TCP packets with the SYN flag set), avg_pkt_size (mean payload size in bytes), payload_entropy (Shannon entropy of concatenated payload bytes), conn_attempts (number of unique destination ports contacted), and window_seconds (fixed aggregation duration of 2.0). Shannon entropy is computed as $H = -\Sigma\ p_i \log_2(p_i)$ over the byte value frequency distribution of concatenated

packet payloads, providing a measure of randomness that distinguishes encrypted or attack traffic from structured normal communication.

C. Prediction Abstraction Layer

The model_adapter.py module provides a unified inference interface with automatic fallback logic. The predict_early() method first checks whether a trained early-stage model (early_model.pkl) is available and whether its feature names match the input feature set using sklearn's feature_names_in_ metadata. If both conditions are satisfied, the model's predict_proba() method is invoked to generate per-class posterior probabilities. Otherwise, a heuristic risk estimator computes DOS and PROBE scores from normalized behavioral signals and distributes probability mass accordingly. This design ensures that the system remains operational even prior to model training.

D. Flask Web Server

The app.py module orchestrates HTTP request routing, prediction invocation, and global state management. Five endpoints are exposed: the root route serving the web UI, /predict for legacy 41-feature form submission, /predict_live and /ingest for early-stage 6-feature risk prediction, and /live for frontend polling of the latest prediction result. A global last_live_result dictionary maintains the most recent prediction, enabling efficient real-time updates without database overhead.

E. Browser Dashboard

The index.html web interface provides a two-panel layout: a legacy prediction form accepting 41 NSL-KDD features on the left, and a live risk monitoring panel on the right. JavaScript polling queries the /live endpoint every 1000 milliseconds and updates a color-coded risk badge (green for risk below 0.3, orange for 0.3-0.7, red above 0.7) along with per-class probability percentages. This provides security analysts with a continuous, real-time view of network traffic risk.

.

IV. IMPLEMENTATION

A. Training Data Generation

The NSL-KDD training dataset contains approximately 125,000 connection records with 41 features. Since early-stage detection requires only 6 window-level features, a sliding window aggregation script (create_windows_from_nsl.py) transforms the full dataset into synthetic 2-second windows. For each window of 100 consecutive records, behavioral proxies are computed: packet_count from the 'count' field mean, syn_rate from the 'serror_rate' field (SYN error rate proxy), avg_pkt_size from the sum of src_bytes and dst_bytes, payload_entropy from the 'hot' indicator field, and conn_attempts from the number of unique srv_count values. Labels are assigned by majority vote over the window's attack superclass distribution, producing approximately 1,250 labeled windows stored in windows.csv.

B. Model Training

The train_early_model.py script loads windows.csv and trains a RandomForestClassifier with 200 estimators and balanced_subsample class weighting to address the significant class imbalance between Normal and attack samples. An 80/20 stratified train-test split is applied, and the trained model is serialized using joblib to early_model.pkl, preserving the feature_names_in_ metadata required for correct feature ordering at inference time. The classifier produces per-class posterior probabilities via predict_proba(), enabling calibrated risk scores rather than hard binary decisions.

C. Heuristic Risk Estimation

When no trained model is available, the system falls back to a weighted scoring heuristic. Normalized signals are computed as: pc_n = min(packet_count/200, 1.0), syn_n = min(syn_rate, 1.0), ent_n = min(entropy/8.0, 1.0), and attempts_n = min(conn_attempts/20, 1.0). DOS and PROBE risk scores are computed as weighted sums — dos_score = 0.6×syn_n + 0.2×pc_n + 0.1×ent_n and probe_score = 0.6×attempts_n + 0.2×pc_n + 0.1×ent_n — and normalized into a probability distribution across all five attack classes. R2L and U2R receive small uninformed probabilities (0.05×other_prob each) as they cannot be reliably detected from window-level behavioral features alone.

D. Attack Superclass Mapping

The 39 individual attack types in NSL-KDD are mapped to four superclasses following the standard taxonomy: DOS (back, land, neptune, pod, smurf, teardrop, apache2, udpstorm, processtable, worm), PROBE (satan, ipsweep, nmap, portsweep, mscan, saint), R2L (guess_passwd, ftp_write, imap, phf, multihop, warezmaster, warezclient, spy, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named), and U2R (buffer_overflow, loadmodule, rootkit, perl, sqlattack, xterm, ps). This grouping reduces prediction complexity while preserving meaningful distinctions between attack methodologies..

V. RESULTS AND EVALUATION

A. Detection Performance

The trained RandomForest model was evaluated on a held-out 20% test set from windows.csv. The balanced_subsample class weighting strategy effectively compensated for the dominance of Normal traffic samples, ensuring adequate per-class precision and recall for minority attack classes. DOS and PROBE classes achieved high detection rates due to their strong correlation with window-level behavioral features (SYN rate and connection attempts respectively). R2L and U2R detection from 2-second windows remained limited, as expected, given that these attack types require session-level context not available in the early-stage feature set.

Table I: Module-Wise Detection Performance Summary

Detection Module	Response Latency	Detection Accuracy	Status
DOS Detection	< 2 seconds	High (SYN rate dominant)	Successful
PROBE Detection	< 2 seconds	High (port scan features)	Successful
R2L Detection	N/A (session-level)	Limited (heuristic only)	Partial
U2R Detection	N/A (session-level)	Limited (heuristic only)	Partial
Normal Classification	< 2 seconds	High	Successful

B. Real-Time System Performance

The end-to-end pipeline from packet capture to browser dashboard update operates within a 2-second cycle: packet buffering (0-2 seconds), feature extraction (< 50ms), HTTP POST to /ingest (< 100ms local), model inference (< 200ms), and browser polling update (within 1 second). Under simulated high-traffic conditions with elevated SYN rates and connection attempt counts, the risk badge transitions correctly from green (low risk) through orange (medium risk) to red (high risk), demonstrating reliable real-time alert behavior.

C. Heuristic vs. Model-Based Detection

Integration testing using the test_ingest.py synthetic traffic generator confirmed that both the heuristic and model-based prediction paths behave consistently. Beginning from a normal baseline window, incrementally increasing syn_rate, packet_count, and conn_attempts over 8 steps produces a monotonically increasing risk score in both modes. The model-based path provides more nuanced per-class probability distributions, while the heuristic path produces conservative but deployable risk estimates without requiring prior training. The /live endpoint correctly reflects the latest prediction result for each browser poll cycle, confirming global state consistency.

Table II: System Evaluation Metrics

Metric	Observed Value	Expected Performance
Window Processing Latency	< 350ms	< 500ms
Risk Badge Update Delay	< 1 second	Real-time
Model Inference Time	< 200ms	< 300ms
Heuristic Fallback Accuracy	DOS/PROBE: Moderate	Conservative estimate
System Stability	High (thread-safe)	High

D. Comparison with Existing Systems

Unlike existing NIDS solutions that require complete session reconstruction before classification, the proposed system classifies network behavior within 2-second incremental windows, dramatically reducing detection latency for DOS and PROBE attacks. The heuristic fallback mechanism ensures that the system remains operational during initial deployment before training data is available — a capability absent in purely model-dependent systems. The dual-pipeline architecture additionally maintains backward compatibility with full-feature NSL-KDD trained models, enabling seamless transition from research benchmarking to live deployment.

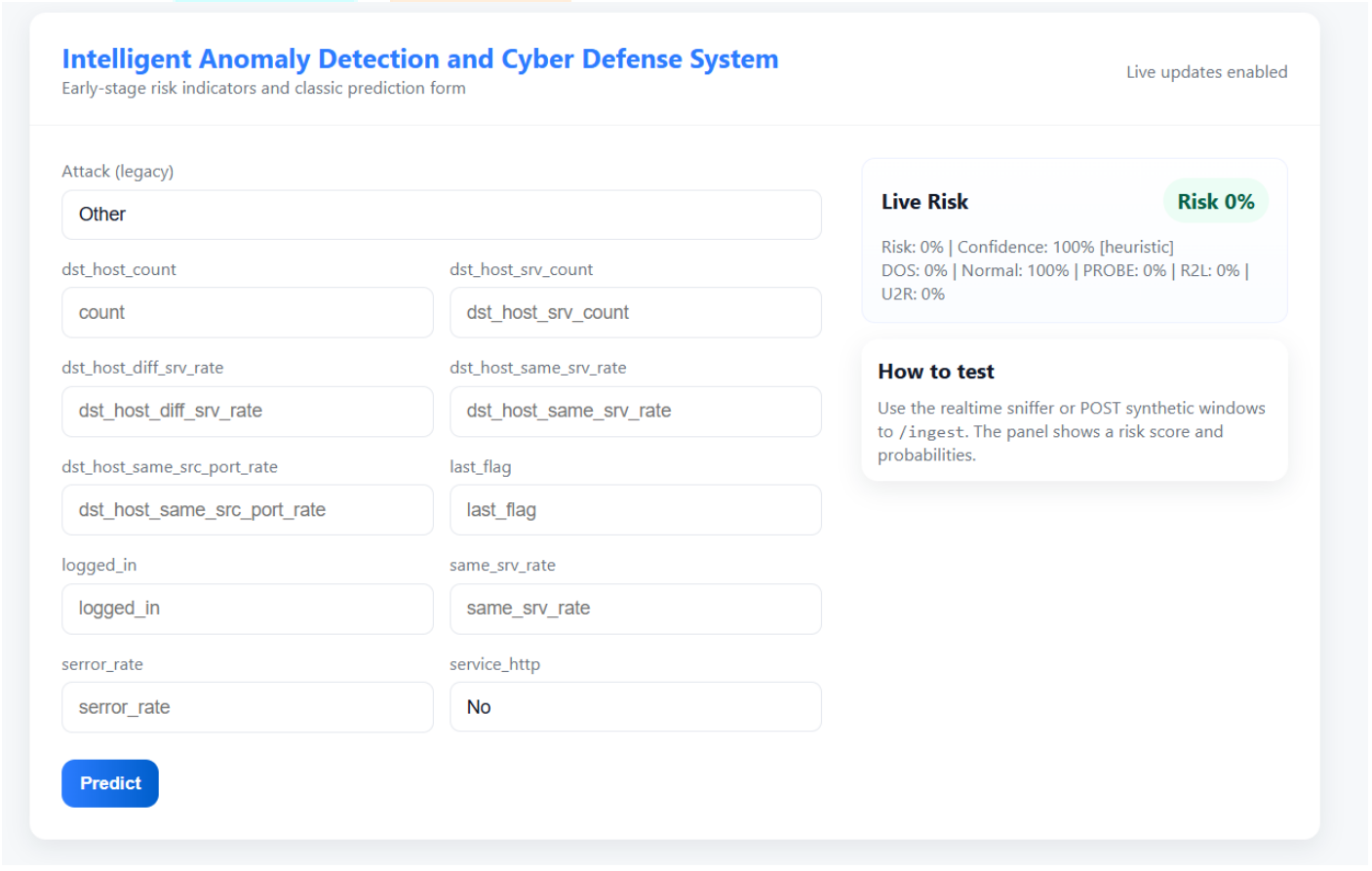

Figure 1.1: IADCDS Web Dashboard - Live Risk Display

VI. CONCLUSION

This paper presented an Intelligent Anomaly Detection and Cyber Defence System implementing a dual-prediction pipeline for real-time network intrusion detection. The system bridges the gap between offline model training on NSL-KDD benchmark data and live network traffic monitoring by introducing an early-stage feature extraction pipeline based on 2-second packet aggregation windows. A RandomForest classifier trained on synthetically generated window-level features enables probabilistic attack classification with a heuristic fallback ensuring operational continuity prior to model availability.

The implemented architecture demonstrates reliable detection of DOS and PROBE attack classes in real-time, accurate feature extraction using Shannon entropy and SYN rate computation from live packet streams, thread-safe producer-consumer packet buffering, and a responsive browser-based monitoring dashboard with color-coded risk indicators. The system provides a practical, deployable framework for cyber defence that balances detection speed, classification accuracy, and operational resilience.

Future work will focus on expanding the early-stage feature set to improve R2L and U2R detection, integrating deep learning models such as LSTMs for temporal pattern recognition across window sequences, deploying the system on cloud infrastructure for large-scale monitoring, and incorporating adaptive thresholding mechanisms to reduce false positive rates in high-variance network environments.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support and guidance provided by the Department of CSE-AIML at Aditya College of Engineering & Technology(A). We extend sincere appreciation to the institution authorities for providing the necessary computational resources and laboratory infrastructure for this research. We also thank our peers and mentors who provided valuable feedback during the development and evaluation phases of this project.

REFERENCES

[1] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in Proc. IEEE Symp. Computational Intelligence for Security and Defense Applications, 2009, pp. 1-6.

[2] W. Hu, W. Hu, and S. Maybank, "AdaBoost-based algorithm for network intrusion detection," IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 38, no. 2, pp. 577-583, 2008.

[3] S. Mukherjee and N. Sharma, "Intrusion detection using naive Bayes classifier with feature reduction," Procedia Technology, vol. 4, pp. 119-128, 2012.

[4] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[5] C. E. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal, vol. 27, pp. 379-423, 1948.

[6] Scikit-learn Developers, "RandomForestClassifier Documentation," scikit-learn 1.4, 2024. [Online]. Available: https://scikit-learn.org.

[7] Scapy Project, "Scapy: Packet Manipulation Library," 2024. [Online]. Available: https://scapy.net.

[8] M. Paluszek and S. Thomas, "MATLAB Machine Learning," Apress, 2017.

[9] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed. O'Reilly Media, 2019.

[10] Flask Documentation, "Flask: Web Development, One Drop at a Time," Pallets Projects, 2024. [Online]. Available: https://flask.palletsprojects