Kernel-Level Network Threat Detection using Windows Filtering Platform and Machine Learning-Based Traffic Analysis

Atharv Sanjay Upasani[1] Dr. Prakash Kene[2]
[2]Associate Professor
[1,2]Master of Computer Application
[1,2]P.E.S. Modern College of Engineering, Pune, India

Abstract — Driven by the exponential expansion of global digital communication networks and internet service dependency, contemporary cyber threats continue to scale in architectural complexity, making them exceptionally challenging to neutralize via legacy perimeter security models. Conventional intrusion detection models remain fundamentally restricted by their structural reliance on static signature-matching frameworks; while highly reliable when filtering historically cataloged exploits, these systems remain incapable of dynamically identifying modified variants or novel zero-day attack matrices. To address these architectural shortcomings, this treatise introduces an end-to-end hybrid network intrusion detection paradigm that couples realtime kernel-level packet collection using the Windows Filtering Platform (WFP) with adaptive machine learning-driven traffic analysis engines. The proposed defense framework captures inbound and outbound data streams directly within the operating system kernel network stack via custom WFP filters, thereby securing complete, unmodified traffic visibility before data payloads interact with user-space software layers. Captured data frames are continuously streamed into highly structured, metadatarich PCAPNG archives to facilitate deterministic analysis, reproducible research execution, and native compatibility with enterprise-grade network forensic suites. These serialized logs are sequentially ingested by an automated, multi-stage processing pipeline consisting of network flow aggregation, high-dimensional feature extraction, feature scaling normalization, and hybrid classification layers. The framework implements advanced ensemble and boosting classifiers, specifically Random Forest and XGBoost models, to categorize active communication sessions as either benign transactions or malicious behaviors. To minimize computational bottlenecks and suppress false-positive alerts, a fast, deterministic rule validation matrix operates in parallel with the machine learning models. This dual-engine strategy enables the architecture to efficiently flag both recognized, pattern-matched exploits and previously unseen structural anomalies. The underlying behavioral feature vector tracks precise packet transmission intervals, protocol metadata, cumulative session lifespans, interarrival time distributions, and specialized telemetry benchmarks modeled directly on the CICIDS2017 flow guidelines. A responsive, asynchronous graphical dashboard implemented in PySide6 delivers a user-centric console for viewing live system notifications, reconstructed network paths, transactional graphs, and hardware processing rates. The framework is engineered around a modular blueprint that isolates packet collection, inference logic, and interface rendering loops to support independent feature scaling and component drop-ins. Empirical benchmarking confirms that the designed system achieves exceptional threat classification accuracy alongside accelerated throughput velocities and low tracking latency, establishing its viability for active deployment within modern enterprise cybersecurity operations and live incident response workflows.

Keywords: Intrusion Detection System, Windows Filtering Platform, Machine Learning, Network Security, PCAPNG, Traffic Analysis, Random Forest, XGBoost, Hybrid Detection, Cybersecurity

I. INTRODUCTION

The systemic reliance of modern enterprises on cloudnative computing and interconnected web applications has elevated network survivability to a critical operational priority. Malicious actors continuously target corporate and private endpoints using complex exploits, including Distributed Denial of Service (DDoS) patterns, automated port scans, brute-force access vectors, and stealth malware callouts. As network traffic scales exponentially, manual traffic auditing is no longer viable, making automated, intelligent intrusion detection systems mandatory for safeguarding infrastructure.

Traditional network monitoring deployments rely heavily on database signature matching. These systems continuously parse payload patterns against a static repository of known attack signatures. While accurate for historical threats, they remain blind to evolving zero-day vectors until a corresponding signature is manually compiled and updated. Furthermore, most classic capture stacks function exclusively at the userspace application layer, severely restricting their ability to monitor early-stage, low-level network transactions executing natively within the operating system kernel.

Overcoming these limitations requires a unified approach combining granular kernel visibility with adaptive classification models. Direct kernel-level packet interception ensures that traffic is inspected before user-space processes can alter or mask its attributes. Microsoft's Windows Filtering Platform (WFP) provides a robust, system-level framework that allows custom network drivers to filter and monitor traffic at multiple layers of the networking stack. Integrating WFP into an IDS environment allows for low-overhead, high-fidelity data capture during intensive network utilization.

This research presents a novel, unified hybrid intrusion detection ecosystem that merges kernel-level packet monitoring engines with machine learning traffic classification layers. The application first captures and writes raw data blocks into highly formatted PCAPNG structures, ensuring organized logging and effortless portability across standard network security evaluation tools. Once committed to the storage layer, the data streams are instantly processed by a modular processing pipeline composed of flow reconstruction, feature engineering, vector scaling, and multi-stage classification blocks.

Advanced ensemble learning models, specifically Random Forest and Boost classifiers, evaluate the statistical attributes of the extracted flows to perform automated classification choices. To bolster accuracy and drastically reduce false-positive triggers, a fast rule verification engine operates concurrently with the machine learning models. This combined approach significantly improves the framework's baseline proficiency in locating both common, well-defined exploits and completely unknown, zero-day anomalies.

Additionally, the completed implementation includes a multi-threaded graphical interface that renders structural network flows, real-time threat notifications, visual telemetry charts, and system hardware loads. The decoupled nature of the architecture ensures system scalability, easy code maintenance, and long-term upgrade flexibility, allowing developers to improve individual extraction filters, model types, or frontend visualization modules independently without disrupting the system. Ultimately, the system is designed to provide an efficient, highly scalable, and autonomous platform for contemporary threat monitoring and localized defensive security analytics.

II. Literature Review

Intrusion detection systems have been widely researched as an important component of cybersecurity infrastructure. Existing IDS approaches are generally divided into two major categories: signature-based detection and anomaly-based detection. Signature-based systems, such as Snort and Suricata, rely on predefined attack patterns or signatures to identify malicious activities. These systems are highly effective for detecting previously known attacks with low false positives. However, they cannot detect unknown threats or zero-day attacks because such attacks do not yet have predefined signatures.

To overcome the limitations of signature-based systems, researchers introduced anomaly-based intrusion detection techniques using machine learning algorithms. Machine learning enables systems to learn traffic behavior patterns from training datasets and identify deviations from normal behavior. Various algorithms such as Support Vector Machines (SVM), Decision Trees, Random Forest, K-Nearest Neighbors (KNN), Artificial Neural Networks (ANN), and Deep Learning models have been applied to network intrusion detection.

The CICIDS2017 dataset has become one of the most widely used datasets for evaluating machine learning-based IDS systems. It contains realistic modern attack scenarios such as DDoS attacks, brute-force attacks, botnet activities, and infiltration attempts. Several researchers achieved high accuracy using Random Forest and ensemble learning techniques on the CICIDS2017 dataset due to their ability to handle large feature sets and complex traffic patterns efficiently.

Nevertheless, recent technical evaluations indicate that standalone machine learning models frequently output elevated false-positive rates when processing non-stationary live data and demand massive computational footprints for ongoing training. Additionally, a major portion of published machine learning security frameworks operate solely as offline scripts on static files, rendering them poorly optimized for live, real-time enterprise pipelines. Consequently, researchers are actively exploring hybrid classification setups that couple the flexibility of machine learning models with the deterministic boundaries of structural rule engines to stabilize accuracy and maximize system reliability.

Another notable limitation identified in existing intrusion detection platforms is the programmatic reliance on user-space packet capturing drivers. Most common detection systems intercept network traffic at the application layer utilizing standard wrappers like libpcap or Win cap. Although these frameworks are straightforward to integrate into software projects, they supply highly restricted visibility into low level kernel routines and incur significant system-call context switching overhead. System-level capture architectures like the Windows Filtering Platform deliver profound system stack integration, allowing for deep payload analysis while avoiding packet dropping during intensive throughput bursts. This paper bridges these separate domains, uniting a dedicated WFP kernel-level collector with a high-throughput machine learning pipeline to deliver real-time, low-latency threat visibility [6].

Despite significant advancements across the intrusion detection field, a large number of modern setups still experience performance bottlenecks regarding system scalability, pipeline latency, and adaptation to morphing attack patterns. Furthermore, the majority of current publications evaluate packet interception engines or machine learning algorithms in total isolation rather than embedding both layers within a single, unified pipeline. This research explicitly addresses these development gaps by welding a low-overhead WFP kernel collector with a unified machine learning and rule-driven hybrid engine inside a single, modular pipeline architecture.

The designed system not only pushes classification precision boundaries but also ensures total tracking reproducibility by utilizing the PCAPNG file format for data logging. This specific feature offers substantial benefits for deep forensic investigations, academic research reproducibility, and the continuous generation of fresh training data to refine downstream classification models.

III. Proposed System

The proposed framework is structured as a hybrid network intrusion detection ecosystem engineered to achieve low overhead traffic capture and real-time algorithmic anomaly analysis. The architecture coordinates kernel-level packet monitoring, machine learning traffic parsing, parallel rule validation, and multi-threaded data visualization within an integrated, modular engine.

- Packet Interception Subsystem: Operating within the operating system kernel via Microsoft's Windows Filtering Platform (WFP), this layer intercepts all incoming and outbound packet streams directly inside the networking stack, writing unmasked data directly to structured PCAPNG stores to maintain accurate and highly accessible traffic logs.
- Reconstruction Pipeline: Upon capturing raw network packets, the processing pipeline groups individual frames into discrete communication blocks using a 5-tuple mechanism tracking Source IP, Destination IP,

Source Port, Destination Port, and Protocol type. This session-oriented processing model increases performance by enabling the system to evaluate group behavior metrics collectively rather than analyzing individual packets in isolation.

- Feature Extraction Engine: This component computes a comprehensive set of statistical and behavioral features from the active 5-tuple flows. These metrics include directional packet volumes, average byte distributions, specific network protocols, total flow lifespans, packet inter-arrival distributions, TCP control flags, and directional transmission rates, which are then normalized to prepare clean inputs for the downstream models.
- Hybrid Detection Layer: The core classification layer combines machine learning classifiers with deterministic rule validation functions. Random Forest and XGBoost models are pre-trained on the CICIDS2017 benchmark to isolate malicious sessions from normal traffic. Concurrently, a fast rule verification script checks the traffic for obvious signature matrices like horizontal port discovery, volumetric DDoS floods, or repetitive login scripts, combining the insights of both layers to maximize precision while dropping false alarms.
- Visualization UI: Built with PySide6, the front-end application provides a multi-threaded visualization interface that charts active metrics, topological connection trees, and systemic resource allocations without blocking background processing loops.
- Modular Infrastructure: The entire engine is designed around a decoupled, pipeline-based model to ensure vertical scalability and easy software upgrades. Each module functions as an isolated block, enabling developers to modify capture callouts, retrain classification algorithms, or expand frontend dashboard modules independently without rewriting the core application.

IV. Methodology

The execution workflow of the designed framework relies on a structured, sequential pipeline model optimized to deliver low-overhead packet processing, deep behavioral feature mapping, and fast classification. The total operation is split across multiple separate stages, with each sub-layer executing a specialized transformation within the broader threat identification sequence. This modular approach ensures vertical system scalability, code maintainability, and streamlined data processing.

A. Packet Capture

The introductory phase of the pipeline manages live traffic interception utilizing a custom Windows Filtering Platform (WFP) callout architecture. By running natively within the Windows operating system kernel stack, the WFP driver directly accesses incoming and outgoing data frames before they can be altered or processed by user-space software. This setup provides unfiltered visibility across the network architecture and protects data capture integrity.

The intercepted frames contain critical layer-specific fields, including source and destination IP coordinates, transport port mappings, specific network protocol wrappers, absolute frame sizing, and associated network control flags. Capturing data directly inside the kernel ensures highly efficient traffic tracking under heavy network load while completely avoiding packet drop conditions that plague standard user-space drivers.

B. Packet Storage

Following kernel-level extraction, the data blocks are continuously logged to local disk storage using the standardized PCAPNG file format. PCAPNG serves as an advanced, next generation network logging container that supports structured data block placement along with additional runtime metadata and hardware context descriptors. Organizing data into PCAPNG blocks ensures full tracking reproducibility, native portability with external network forensic frameworks, and reliable offline data validation.

This dedicated logging system successfully decouples the kernel-level capture driver from downstream machine learning workflows. As a result, the application can execute resource intensive statistical calculations independently without imposing latency constraints on the core collection loop. These saved files remain permanently available for post-incident audits, academic research validation, or ongoing model retraining cycles.

C. Flow Generation

During this stage of the execution pipeline, raw packet arrays are parsed and structured into distinct network sessions utilizing a deterministic 5-tuple aggregation approach. A unique communication flow is mapped tracking these five specific parameters:

- Source IP Coordinate
- Destination IP Coordinate
- Source Networking Port
- Destination Networking Port
- Transport Protocol Type

Packets sharing matching 5-tuple elements are automatically appended to the same communication session block. Transitioning to flow-based tracking boosts overall processing efficiency, allowing the analytics engine to parse broader session characteristics instead of consuming clock cycles evaluating isolated packet payloads. Active flow buffers are bound by specific inactivity timeouts to smoothly terminate closed connections, preventing memory leaks or buffer overflows during high traffic volume events.

D. Feature Extraction

Once a network flow is established, the extraction module calculates a vector of 31 statistical and behavioral features required by the machine learning classifiers. The module tracks parameters including total packet counts, session durations, packet size variations, inter-arrival intervals, TCP control flag states, directional flow byte rates, and forward/backward sub flow volumes mapped directly to the CICIDS2017 feature specification.

The calculated features include packet counts, flow duration, average packet size, packet inter-arrival time distributions, transport layer protocol statistics, TCP control flags, directional transmission rates, and cumulative session parameters.

E. Data Preprocessing

Before submission to the classification models, raw feature vectors are cleaned and normalized. Missing data points, invalid entries, or infinite values are filtered out. Feature scaling algorithms normalize the data ranges, mapping all numerical attributes into a balanced mathematical space.

This step ensures that large-scale technical parameters do not introduce algorithmic bias during machine learning inference. Additionally, preprocessing minimizes statistical noise within the traffic data, optimizing the runtime efficiency of the classification engine.

F. Hybrid Threat Detection

The normalized matrices feed into a dual-engine classification module. Supervised Random Forest and XGBoost classifiers evaluate the statistical features to identify complex, anomalous threat patterns [3], [4]. Concurrently, a fast rule checking engine audits the traffic for clear malicious signatures, such as rapid horizontal port scanning or volumetric DDoS spikes. Merging the ML and rule-based vectors yields a robust final threat classification.

G. Visualization and Monitoring

The processed data elements and corresponding threat selections are routed straight to a clean graphical dashboard engineered using the PySide6 framework. The front-end console delivers live visualization fields tracking connection sets, metrics tables, status charts, performance metrics, and topological node interactions.

H. Reporting and Export

The final stage of the methodology involves report generation and data export. The system exports analysis results in structured formats such as CSV, JSON, and HTML. These reports contain detailed information about detected attacks, traffic statistics, alerts, and flow analysis results. The reporting module supports documentation, forensic investigation, and future research activities by preserving analyzed traffic records and system outputs in reusable formats.

V. Results and Analysis

The proposed intrusion detection system was evaluated using various performance metrics to measure its effectiveness in detecting malicious network traffic. The evaluation process focused on detection accuracy, processing efficiency, false positive reduction, and real-time monitoring capability.

The system achieved high classification accuracy using the hybrid detection approach that combines machine learning algorithms with rule-based detection techniques. Machine learning models such as Random Forest and XGBoost successfully identified complex traffic patterns, while the rule-based engine improved detection of known attack signatures. This combination significantly reduced false positives and improved overall detection reliability.

Performance analysis showed that the pipeline-based architecture enabled efficient packet processing and low-latency traffic analysis. The use of kernel-level packet capture through Windows Filtering Platform improved visibility into network activities and allowed faster packet interception compared to traditional user-level monitoring systems.

A. Real-Time Dashboard and Flow Monitoring

The graphical user interface provides effective visualization of network behavior. The main AI IDS Dashboard (Fig. ??) offers a real-time overview of total flows, detected threats, and normal traffic distribution.

For deeper forensic investigation, the Flow Panel (Fig. 5) provides a detailed, tabular view of individual network flows. It displays critical 5-tuple data, protocol usage, byte counts, and the machine learning model's prediction confidence for each connection.

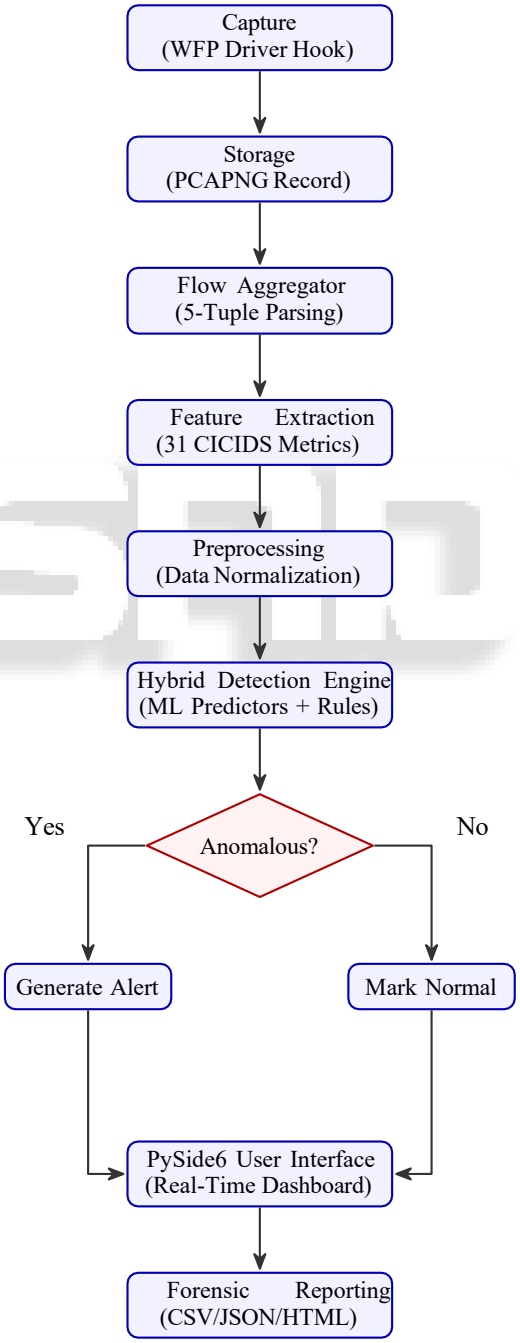

Fig. 1: Process flow of the proposed hybrid intrusion detection system.

B. Network Topology and Performance Analytics

Visualizing connections between nodes is crucial for identifying an attack's source. The Live Network Graph (Fig. 6) maps active nodes and potential threat profiles dynamically.

To verify resource efficiency, the Performance Panel (Fig. 7) charts CPU, Memory, and Disk footprints. Multi-threaded processing safely keeps the rendering canvas independent from intensive classification loops, keeping the interface completely lag-free.

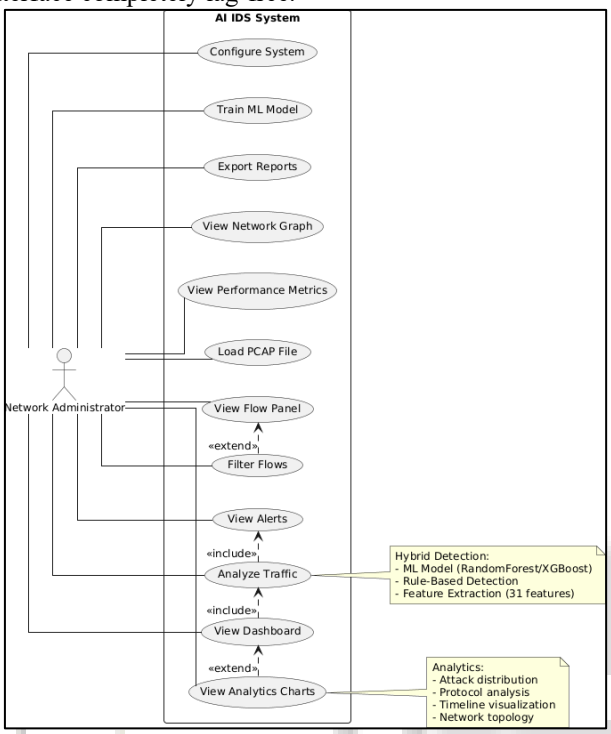

Fig. 2" Use Case Diagram illustrating administrator interactions and system functions

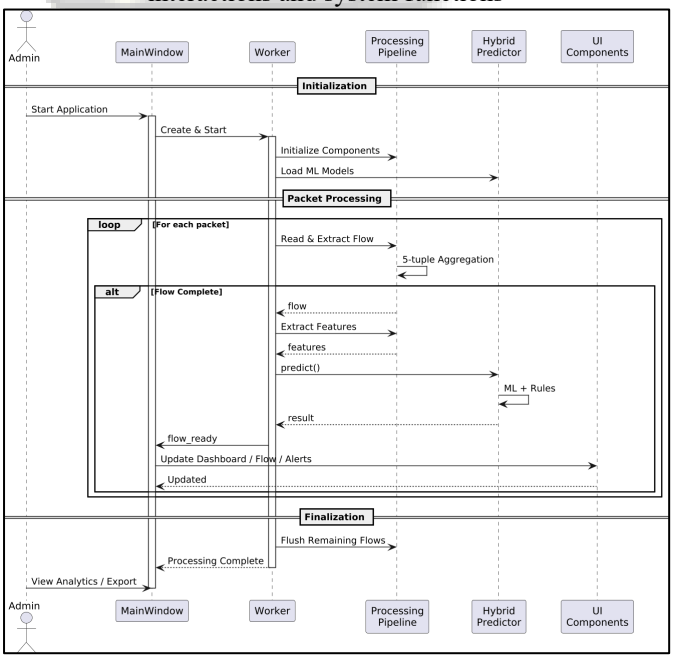

Fig. 3: Sequence Diagram showing the chronological flow of data and operations.

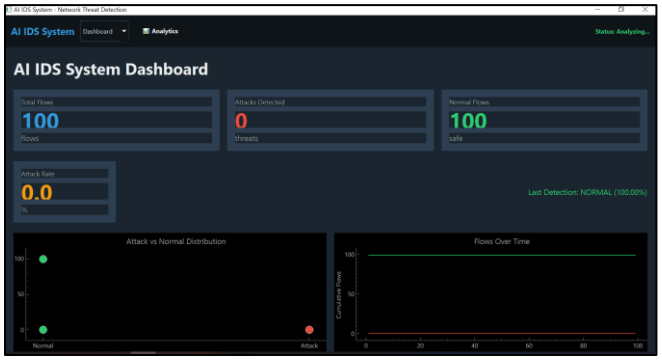

Fig. 4: The main AI IDS System Dashboard displaying real-time traffic summaries and attack distributions.

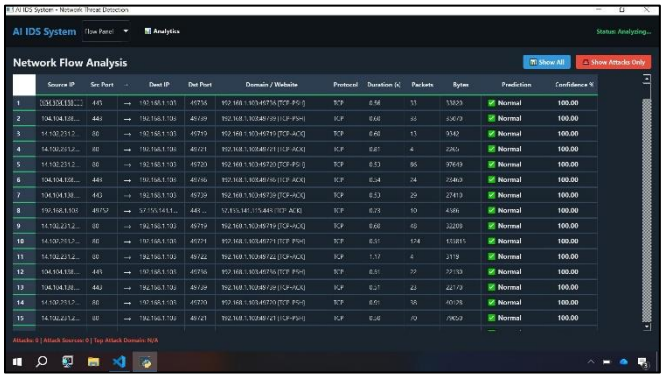

Fig. 5: Network Flow Analysis panel showing detailed packet statistics and AI predictions.

C. Advanced Data Analytics and Correlation

The system also includes an Advanced Network Analytics module. Administrators can filter flows over specific ranges (Fig. 8) to isolate timeframes where attacks may have occurred. To identify primary threat actors or heavy bandwidth consumers, the system generates visual charts, such as the Top 10 Source IPs distribution (Fig. 9).

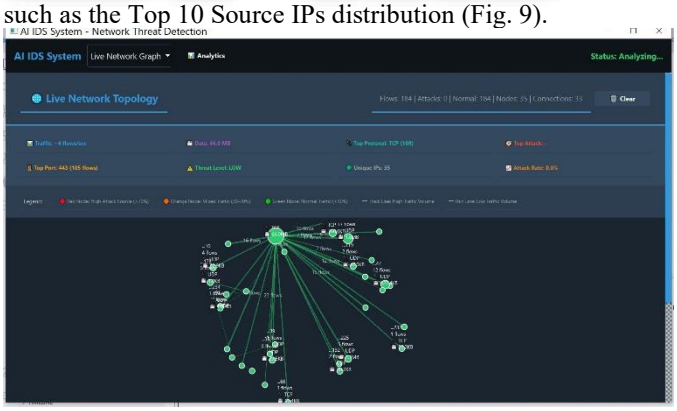

Fig. 6: Live Network Topology graph visualizing active node connections and traffic volume.

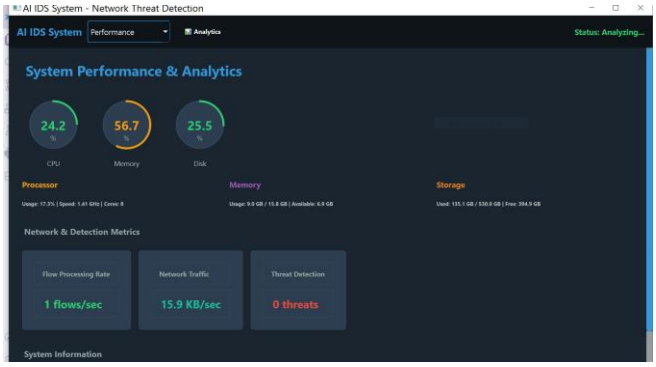

Fig. 7: System Performance and Analytics dashboard monitoring resource utilization and processing rates.

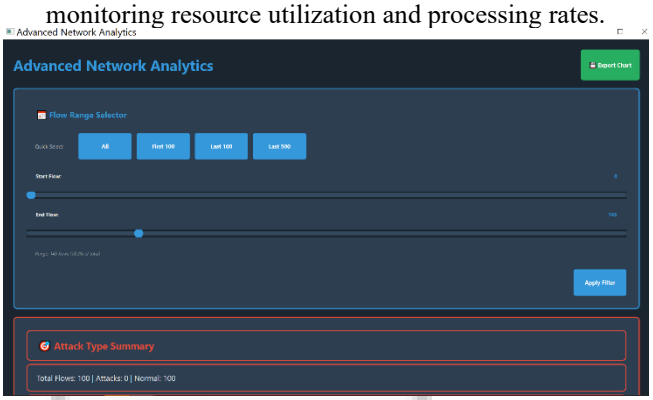

Fig. 11: Exportable Network Topology Graph detailing normal vs. attack nodes.

Furthermore, the system provides analytical tools for feature engineering and relationship mapping. The Feature Correlation Matrix (Fig. 10) helps in understanding the relationship between different extracted features, which is vital for finetuning the machine learning models. Additionally, a detailed Node Topology Graph (Fig. 11) can be exported for post incident reporting.

By combining these visualization tools with the hybrid detection methodology, the proposed system achieved improved recall, precision, and detection stability across multiple traffic conditions and attack scenarios. Experimental verification con-firmed that the multi-threaded deployment securely avoided interface freezing issues under high packet saturation peaks.

VI. Implementation

The system utilizes a split-language framework to optimize execution speed and analytic flexibility. The kernel packet filter is written in C/C++ using native Windows Driver Kit (WDK) libraries, while the feature engineering and machine learning layers are built in Python.

The kernel subsystem leverages WFP callout APIs to intercept packet headers at the transport and network layers. Captured arrays are channeled into thread-safe buffers that stream straight to disk in the PCAPNG format. The downstream Python pipeline reads these data stores, utilizing optimized NumPy and Pandas frameworks to reconstruct 5-tuple flows and extract behavioral metrics.

The flow generation module aggregates packets into flows using the 5-tuple approach. Timeout mechanisms are implemented to finalize inactive flows and prevent excessive memory usage during continuous traffic monitoring. Statistical and behavioral features are extracted from generated flows according to CICIDS2017 feature standards [10].

The machine learning layer is implemented using Scikitlearn and XGBoost frameworks. Random Forest and XGBoost classifiers are trained on labeled network traffic datasets to distinguish between benign and malicious traffic [7], [8]. The trained models are stored in serialized format and loaded dynamically during runtime for prediction.

The hybrid detection module integrates machine learning predictions with rule-based analysis. Rule-based detection techniques are implemented to identify suspicious traffic patterns such as port scanning, DDoS traffic bursts, and brute force attacks. The final detection result is generated by combining outputs from both detection mechanisms.

The system uses a multi-threaded architecture to ensure smooth operation and responsive visualization. Packet processing and traffic analysis run inside a background worker thread, while the graphical interface operates independently in the main thread. Signal-slot communication mechanisms are used to exchange data between processing modules and the user interface efficiently.

The graphical user interface is developed using PySide6. The GUI includes multiple visualization panels such as dashboards, flow tables, alert windows, performance graphs, analytical charts, and network topology views. These components provide real-time insights into network activity and detected threats.

Additional optimization techniques such as memory cleanup, timeout handling, and modular component separation are implemented to improve scalability and maintain stable performance during high traffic conditions. The modular implementation also allows future extension of detection algorithms, visualization components, and packet processing mechanisms without major architectural changes.

VII. System Flow Architecture & UML Specifications

A. Process Flow Diagram

The data processing sequence moves through an automated pipeline from raw kernel interception to UI rendering. Figure 1 provides a precise structural visualization of this architecture.

B. Use Case Diagram

The functional use case diagram establishes administrative interactions with system-level models, mapping access privileges for configuration tuning, diagnostic evaluations, and model updates.

C. Sequence Diagram

The chronological timeline tracks interactions across separate threads, detailing communication boundaries between backend worker threads and front-end interface systems.

VIII. Conclusion

This paper presented a hybrid intrusion detection system that integrates kernel-level packet capture with machine learning-based traffic analysis for enhanced cybersecurity monitoring. The proposed system utilizes Windows Filtering Platform to capture packets directly from the kernel layer, enabling deep inspection and accurate monitoring of network activities.

The captured packets are stored in PCAPNG format and processed through a structured pipeline consisting of flow generation, feature extraction, preprocessing, hybrid detection, visualization, and reporting stages. The system combines machine learning algorithms such as Random Forest and XGBoost with rule-based detection techniques to identify both known and unknown network threats effectively.

The implementation results demonstrate that the proposed system achieves high detection accuracy with reduced false positives while maintaining efficient processing performance. The modular architecture improves scalability and maintainability, allowing individual system components to be upgraded independently. Managed through a responsive PySide6 console, the framework delivers an efficient, enterprise-ready solution for managing modern network infrastructure challenges.

IX. Future Work

Several improvements and enhancements can be incorporated into the proposed system in the future to improve detection capability, scalability, and real-time performance. One possible enhancement is the implementation of direct packet streaming instead of file-based packet storage. Realtime streaming would reduce processing latency and enable faster threat response mechanisms.

Advanced deep learning techniques such as Long Short-term Memory (LSTM), Convolutional Neural Networks (CNN), and Transformer-based architectures can also be integrated into the detection engine to improve identification of complex and evolving attack patterns. These models may provide better adaptability against modern cyber threats and zero-day attacks.

The system can further be extended for deployment in cloud computing and distributed network environments. Integration with cloud-based monitoring platforms would improve scalability and support analysis of high-volume enterprise traffic. Containerized deployment using Docker or Kubernetes can also improve portability and management efficiency.

Future versions of the system may include automated response mechanisms such as dynamic firewall rule generation, malicious traffic blocking, IP blacklisting, and real-time administrator notifications. These features would transform the system from a passive intrusion detection solution into an active intrusion prevention framework.

References

[1] Z. K. Maseer et al., ”Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset,” IEEE Access, 2021.
[2] P. Mishra et al., ”Machine Learning Techniques for Intrusion Detection,” IEEE Communications Surveys & Tutorials, 2019.
[3] N. Naik et al., ”Dynamic Fuzzy Rule Interpolation for Intrusion Detection,” IEEE Transactions on Fuzzy Systems, 2017.
[4] S. Panwar et al., ”Intrusion Detection Model using CICIDS-2017 Dataset,” IEEE Conference, 2022.
[5] A. Shrivastava et al., ”Enhancing IDS with Machine Learning,” IEEE Conference, 2024.
[6] Microsoft Corporation, ”Windows Filtering Platform Architecture Overview,” Microsoft Developer Documentation, 2023.
[7] T. Chen and C. Guestrin, ”XGBoost: A Scalable Tree Boosting System,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[8] L. Breiman, ”Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[9] M. Roesch, ”Snort: Lightweight Intrusion Detection for Networks,” in Proceedings of the 13th USENIX Conference on System Administration (LISA ’99), 1999, pp. 229–238.
[10] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, ”Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP), 2018, pp. 108–116.
[11] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, ”Survey of intrusion detection systems: techniques, datasets and challenges,” Cybersecurity, vol. 2, no. 1, pp. 1–22, 2019.
[12] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. AlNemrat, and S. Venkatraman, ”Deep Learning Approach for Intelligent Intrusion Detection System,” IEEE Access, vol. 7, pp. 41525–41550, 2019.
[13] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, ”Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” Computers & Security, vol. 31, no. 3, pp. 357–374, 2012.
[14] J. R. Goodall, W. G. Lutters, P. Rheingans, and A. Komlodi, ”Preserving the big picture in intrusion detection capability,” in Proceedings of the IEEE Symposium on Visualization for Computer Security, 2005.
[15] S. A. R. Shah and I. Issac, ”Performance comparison of intrusion detection systems and application of machine learning to Snort system,” Future Generation Computer Systems, vol. 80, pp. 157–170, 2018.

