APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 2 (2025) 586-604
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

RETINA: Network Intrusion Detection System using Machine Learning Approach

Muhammad Zainurain Mohd Zain[1], Isredza Rahmi A Hamid[1*]

1 *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rahmi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.02.032

Article Info

Received: 21 July 2025
Accepted: 19 November 2025
Available online: 30 November 2025

Keywords

Intrusion detection system, XGBoost, Machine learning, DoS attacks, Brute force attacks

Abstract

In today's digital age, protecting networks from cyberattacks is crucial because it pose significant risks to sensitive data, financial assets, and organizational reputation. However, traditional Network Intrusion Detection Systems (NIDS) often face challenges such as inefficiency, limited accuracy, and difficulty handling real-time data. To address these issues, we proposed a Real Time Anomaly (RETINA) system using machine learning approach. RETINA will be focusing on real-time packet analysis and threat classification to detect Dos attack and brute force attack. The development follows the Object-Oriented Analysis and Design (OOAD) methodology. The project employs XGBoost algorithm and dataset from CICIDS 2017 for threat classification, Scapy library for real-time packet capture, and a web-based interface for monitoring and visualization. The project primarily benefits network administrators by enabling them to identify and respond to security threats proactively. The outcome of this project will highlight the potential of intelligent algorithms in advancing intrusion detection systems. The final developed system could detect Denial of Service (DoS) and Brute Force Attack and significantly low 0.07% False Positive Rate.

1. Introduction

Network Intrusion Detection system (NIDS) are critical components in cybersecurity, serving to defend networks against cyber threats. With increasing numbers of cyber threats targeting both corporate and home networks, investing in effective NIDS is crucial for protecting sensitive information and maintaining regulatory compliance. According to a report by Check Point, organizations worldwide experienced a 75% increase in weekly cyberattacks in Q3 2024, averaging 1,876 attacks per week per organization[1]. As the cybersecurity landscape continues to evolve, organizations must prioritize innovative approaches like NIDS to safeguard their critical assets against emerging threats.

Traditional NIDS such as Snort, rely on predefined rules and signatures to detect known attack patterns, which limit their ability to adapt to emerging threats[2]. They often suffer from high false positive and false negative rates, resulting in inconsistent detection accuracy. Additionally, the system generates many alerts for non-threatening situations[3]. To address these limitations, the integration of artificial intelligence (AI), particularly machine learning (ML), into NIDS has gained significant attention. ML-based NIDS can analyze large volumes of network traffic data, identify patterns, and detect anomalies with higher precision and efficiency, reducing the likelihood of false alarms [3]. The objectives of this project are as follows:

i. to design a Network Intrusion Detection system using machine learning approach.
ii. to develop Network Intrusion Detection system by utilizing network traffic analysis to detect malicious attacks.
iii. to perform alpha and beta testing on the proposed system in terms of system functionality and user acceptance.

We propose Real Time Anomaly (RETINA) system using Machine Learning approach utilizing the XGBoost algorithm to enhance detection accuracy. The system architecture integrates Scapy library for real-time packet capture and flow analysis with FastAPI serving as the backend framework, providing REST API functionality and WebSocket support for seamless real-time communication. The system aims to detect and classify malicious activities such as Denial of Service (DoS) and Brute Force attacks [6]. The CIC-IDS2017 dataset serves as the primary source for training and testing the models. The project also includes the development of a user-friendly interface for visualizing network traffic and alerting administrators. This system focuses on real-time detection within home network environments. This work employs an object-oriented analysis and design model. By addressing the challenges of traditional NIDS, this work aims to contribute to a more accurate and efficient approach to network security [7].

The remainder of this paper is organized as follows; Section 2 discusses related work. Section 3 describes the methodology, Section 4 explains the design and analysis, Section 5 presents the implementation and result and lastly, Section 6 presents the conclusion and future work of the project.

2. Related Work

This section discusses intrusion detection systems, type of network attacks, network intrusion detection techniques, machine learning algorithm and existing network intrusion detection system (NIDS).

2.1 Intrusion Detection System

Intrusion detection involves monitoring events on a network or computer system and analysing them for indications of potential threats or breaches of usage policies and standard security protocols [5]. By spotting all the possible security breaches, intrusion detection systems (IDS) provide organisations with the ability to defend their digital infrastructure. IDS is crucial because it acts as an early warning system, helping organizations identify and respond to cyber threats before they escalate into significant incidents [4]. It enhances the overall security posture by providing continuous monitoring and threat intelligence, which is essential in an environment where attacks are increasingly sophisticated.

2.1.1 Network Based Intrusion Detection System (NIDS)

Network Based Intrusion Detection System (NIDS) is a security technology used to monitor network traffic for suspicious or malicious activity by analysing data packets that travel through a network [5]. NIDS offer a wide range of detection capabilities. They often integrate signature-based and anomaly-based methods to analyse data thoroughly and boost detection rates [8]. When identifying unusual activity, the anomaly-based method breaks down the data into requests and responses, comparing them against known attack patterns [5]. This layered approach ensures that threats are accurately identified before data is transmitted to its destination. NIDS are designed to protect the whole network, not just a single host like Host Intrusion Detection System (HIDS). NIDS continuously monitors live network traffic, making it more suitable for detecting real-time attacks like DoS and brute force. In contrast, HIDS often relies on logs and post-event analysis, which might delay response times.

2.1.2 Host Intrusion Detection System (HIDS)

Host-based Intrusion Detection System (HIDS) observe a host's properties and activities to possible security threats. It monitors data such as network traffic, system logs, and file access and modifications [5]. Basically, HIDS is a security tool that monitors and analyses the activities of a specific host, like a computer, server, or virtual machine, to detect suspicious behaviour or policy violations. Unlike NIDS, which focuses on traffic across the network, HIDS operates on individual devices to detect potential intrusions. The main drawback of HIDS is more HIDS need to be installed as the network expands. So, as the number of hosts increases, managing them all becomes challenging [6].

2.2 Network Attack

Networks attack is an attempt to steal, disable, destroy, alter, or gain unauthorized access to an asset. Network attacks can lead to network services slowing down, becoming temporarily unavailable, or experiencing

prolonged downtime. This makes it essential for users and network administrators to detect these attacks early to prevent potential damage to the system.

2.2.1 Denial of Service (DoS)

Denial of Service (DoS) attacks are often grouped into two main types which are Weaknesses-based Attacks and Flooding Attacks[4]. Both attacks aim to disrupt services and stop regular users from accessing the resources. Weaknesses-based Attacks take advantage of flaws or weak points in internet protocols or systems. For example, attackers misuse parts of protocols like Internet Control Message Protocol (ICMP), Hypertext Transfer Protocol (HTTP), and Transmission Control Protocol (TCP) to overload a system and cause it to fail or slow down. In contrast, Flooding Attacks involve the attacker sending huge amounts of traffic to the target. This overloads the system's resources such as memory, processing power, and bandwidth causing it to drop normal traffic as it tries to handle the flood of data. Both types of attacks can cause serious interruptions to services, showing the importance of having strong detection and protection methods in place. DOS attacks also work by flooding systems, servers, and/or networks with traffic to overload resources[7].

2.2.2 Brute Force Attack

A brute force attack is a hacking technique where an attacker systematically tries every possible combination of credentials (like passwords or encryption keys) to gain unauthorized access to a system, network, or account[8] This method relies on trial and error. While it is simple, it can be effective, especially if the target uses weak or commonly used passwords. Brute force attacks resemble legitimate network traffic, making it difficult to defend an organization that rely mainly on perimeter-based security solutions a major challenge The most common brute force attacks use a password dictionary with millions of possible words to try [9]. A successful brute force attack can grant hackers access to data, applications, and resources, and may also act as a gateway for additional attacks. Several signs may indicate a brute force attack, such as multiple failed login attempts from the same Internet protocol (IP) address, repeated attempts with different usernames from one IP, or login attempts for a single account coming from various IP addresses[9]. Other warning signs include failed logins using usernames or passwords in alphabetical order, logins with a referring Uniform Resource Locator (URL) from someone's email or chat client, high bandwidth use within a single session, and many failed login attempts [9].

2.3 Network Based Intrusion Detection Techniques

Network Based Intrusion Detection Systems (NIDS) play a crucial role in detecting unauthorized access or malicious activities within a network. Two main techniques employed are signature-based and anomaly-based.

2.3.1 Signature Based

Signature-based IDS detects attacks based on specific patterns such as the number of bytes or several 1s or the number of 0s in the network traffic. The system compares the signatures of network traffic packets with those in the database to see if there is a match, helping to detect any known attacks [6]. Known attacks are easier to detect because the observed pattern can be matched with an existing signature in the database. However, detecting new attacks is more difficult since their signatures have not been created yet.

Signature-based systems can recognize specific patterns associated with known DoS attacks, such as SYN floods or UDP floods, by comparing incoming traffic to established signatures in their databases[10]. If a signature for a particular DoS attack is triggered repeatedly within a short timeframe, it indicates ongoing malicious activity targeting the network Detection of traffic from IP addresses that are known to be associated with previous DoS attacks can be flagged as suspicious.

2.3.2 Anomaly Based

Anomaly detection monitors network traffic over time to create a model of normal network behaviour. If any unusual activity is detected that deviates from this normal model, it is considered an intrusion or attack. Anomaly detection uses methods like statistical analysis, machine learning, and knowledge-based techniques to build these models. These methods can identify unusual behaviour with reasonable accuracy [6]. The effectiveness of anomaly detection depends on the quality of the training data. If the training data accurately represents real network data, the detection method will perform better. Therefore, in this work a machine learning algorithm will be utilized to design an anomaly-based network intrusion detection system.

A large increase in traffic that is different from the established baseline can indicate a potential DoS attack in anomaly-based detection. For example, SYN flood attacks generate a significant spike in network traffic, which results in a noticeable increase in incoming connections[10]. Timely monitoring of this surge is crucial for early detection. Other signs might be a high number of connection attempts or incomplete connections (like half-open

connections), which could show that an attacker is trying to overload the server[10]. Any activity that doesn't match normal user behaviour, such as logins from strange locations or at odd times, might also trigger alerts for possible DoS attacks. Monitoring systems may also notice unusual resource usage, like spikes in CPU or memory, which could suggest that the service is being overwhelmed by harmful traffic[11].

2.4 Machine Learning Algorithm

Machine learning algorithms are essential components in the field of cybersecurity, especially in detecting network intrusions. Different types of machine learning algorithms, such as XGBoost, Random Forest, and J48, offer various strengths in identifying network behavior. Among these, XGBoost was chosen due to its use of gradient boosting techniques that optimize both bias and variance, often leading to better overall accuracy than Random Forest and J48.

2.4.1 Extreme Gradient Boosting (XGBoost)

XGBoost[12], which stands for Extreme Gradient Boosting, is a fast and powerful machine learning algorithm based on gradient boosting. Gradient boosting works by combining many small decision trees to create a strong model that makes better predictions. XGBoost improves this method by being faster, able to run in parallel, and includes features like regularization to avoid overfitting, handling missing data, and balancing uneven classes. It also allows users to set custom loss functions and uses built-in cross-validation to check model accuracy. In this case, important features related to different types of attacks are used with the XGBoost model. The algorithm keeps improving by reducing errors in each round using loss functions like mean squared error (for predicting numbers) or log loss (for classification). Because of its speed and ability to find complex patterns, XGBoost is widely used for detecting intrusions in large datasets[13]

2.5 Existing Network Based Intrusion Detection System (NIDS)

This section discusses various NIDS such as Snort, Suricata and Zeek.

2.5.1 Snort

Snort is an open-source network intrusion detection and prevention system created by Martin Roesch in 1998[14]. It is developed and maintained by SourceFire, which was later acquired by Cisco. Snort is renowned for its signature-based detection capabilities, which excel in identifying known threats. However, it tends to generate higher false positive rates compared to Suricata and Zeek[14]. This can lead to unnecessary resource allocation for investigating alerts that may not represent actual threats. Rules can be specified to detect certain content in packet payload or other features found in the packet headers. When a rule is matched against a packet, Snort can take actions such as Alert, Log packet, ignore packet or Drop packet. Snort is in its original form and does not include a native Graphical User Interface (GUI)[15]. However, there are several third-party tools and frameworks available that provide GUI interfaces for managing and visualizing Snort's functionality. According to Sadargari Viharika and Nalangudi Balaji[16], Snort exhibited a false positive rate of 5.1%, highlighting one of its key limitations in accurately distinguishing malicious activity from legitimate traffic.

2.5.2 Suricata

Suricata is a network intrusion detection system (NIDS) developed by the Open Information Security Foundation (OISF)[14]. Suricata performs comparably or better than Snort in terms of detection accuracy while maintaining lower false positive rates. Its multi-threaded architecture allows it to handle high-volume traffic more efficiently than Snort. It uses existing rule sets of signature-based features to monitor network traffic and sends alerts when it detects suspicious activities. Some common rule sets include Emerging Threats, Emerging Threats Pro, and Sourcefire's VRT. Suricata also supports Lua scripting to detect complex threats and can identify various anomalies in the traffic it examines. It is designed to integrate with other network security tools. Suricata is a multithreaded engine, which allows it to process network traffic quickly and efficiently by distributing the workload based on processing needs. Suricata works much the same as Snort but is more scalable as its analyzer supports multi-threading. It is also faster than Snort but consumes more resources. Suricata also has been recorded producing lower false positive than Snort which is 2.5%[17].

2.5.3 Zeek

Zeek formerly known as Bro is an open-source, network-based intrusion detection system (NIDS) that passively monitors network traffic for suspicious activity and potential attacks [18]. It was created in 1998 by Vern Paxson at the Lawrence Berkeley National Lab and the International Computer Science Institute. Zeek uses a mix of signature-based and anomaly-based methods to detect threats. Zeek works differently from Snort and Suricata as Zeeks analyzer consists of an event engine that is triggered upon different events in the traffic. The events are passed to Zeeks script land where traffic features are further analyzed using Zeeks own scripting language which produces log files and notifications about suspicious or interesting activity. However, Nadia Niknami and Emily Inkrott conducted an experiment to analyse the effectiveness of Zeek against several attack[19]. The results indicate that both IDSs can detect an attack on the network when there is a port scanning attack. However, they do not consider this kind of traffic to be malicious. As a result, Zeek produces False Negative (FN) results for such malicious traffic. Additionally, Zeek does not have GUI and to work with Zeek one should have good grip on shell commands whereas Snort and Suricata has a GUI which makes it more accepted[20].

2.6 Comparison Between Existing System and Proposed System

The existing NIDS, such as Snort, Suricata, and Zeek shared several similarities with the proposed system RETINA. All systems including RETINA support anomaly-based detection techniques which enabling the identification of unusual network behaviour. Additionally, all systems provided real-time detection capabilities, ensuring timely identification and response to threats. In terms of user interface, RETINA offer a Graphical User Interface (GUI) for ease of use, whereas Suricata, Zeek and Snort relies on a Command-Line Interface (CLI) which requires more technical expertise. However, there are notable differences between the existing systems and the proposed system RETINA. Suricata and Zeek combine both signature-based and anomaly-based detection, while RETINA exclusively focuses on anomaly detection. Furthermore, RETINA leverages the XGBoost machine learning algorithm to enhance its detection accuracy and has a significantly lower false positive rate (0.07%) compared to other NIDS.Table 1 shows the comparison between existing NIDS such as Snort, Suricata, and Zeek with RETINA.

Table 1 *The Comparison of Existing System with the Proposed System*

Tools / Features	Snort	Suricata	Zeek	RETINA
Detection technique	Signature-based	Hybrid	Hybrid	Anomaly-based
Machine learning algorithm	No	No	No	Yes (XGBoost)
Real time Detection	Yes	Yes	Yes	Yes
User Interface	CLI	CLI	CLI	GUI
False positive rate	5.1%[16]	2.5%[17]	9.3%[17]	0.07%

3. Methodology

Object-Oriented Analysis and Design (OOAD) provides a structured approach and methodological foundation for developing RETINA system. OOAD encourages organizing the system into reusable components, called objects, which enhances code clarity and maintainability.

Object-Oriented System Development (OOSD) life cycle as shown in Fig. 1 consists of five phases that are Requirements, Design, Implementation, Verification, and Maintenance. In the Requirements Phase, specifications related to functionality, performance, hardware and software compatibility are gathered to define the project scope. In the Design Phase, these requirements are translated into architectural models including class, use case, and sequence diagrams which focus on integrating machine learning model (XGBoost). The Implementation Phase involves dataset preprocessing, feature selection, model training, and GUI development for real-time monitoring. The Verification Phase ensures the system meets its functional goals through unit, integration, and performance testing in terms of precision and recall, and simulated attacks for validation. Finally, the Maintenance Phase addresses bug fixes, model retraining, performance monitoring and system updates to maintain long-term reliability and adaptability to evolving threats.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 586-604

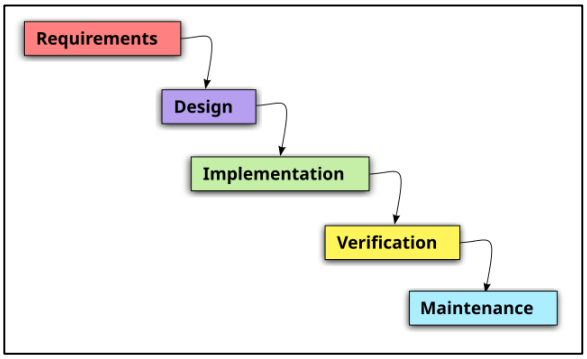

Fig.1 *Phases in object-oriented analysis and design (OOSD) life cycle*[21]

3.1 Software Requirements

In the development the system, it is essential to consider the software requirements needed to aid in the development process. The development of the RETINA requires specialized software. The specifications for the software used to create this system are laid forth in Table 2.

Table 2 *Software Requirements*

No	Software	Description
1	Operating System	Windows 10
2	Web Server and Framework	FastAPI and Uvicorn
3	Traffic Capture	Scapy
4	Database Engine	SQLite with SQLAlchemy 2.0.23 ORM
5	Core Programming Language	Python 3.7+
6	Integrated Development Environment	Google Colab and Visual Studio Code

4. Analysis and Design

This section explains the components of the analysis and design phase. This phase includes the user, functional and non-functional requirements, system architecture, machine learning process, feature extraction, and the Unified Modeling Language (UML) diagram.

4.1 System Architecture

Fig. 2 illustrates the RETINA system workflow, which functions as a continuous real-time detection and response loop. It begins with Scapy capturing live network packets, which are then processed to extract 50 network flow features such as packet sizes, timing, ports, and protocols. These extracted features are fed into a pre-trained XGBoost machine learning model, which has been trained on labeled datasets of both benign and malicious traffic. The model evaluates the incoming packet flows using learned patterns to classify each as either benign or malicious. This prediction is made by the model based on the statistical and behavioral characteristics of the flow. If malicious activity is detected, the system automatically executes a three-step response. First action, user can block the source IP. Second actions it logs full attack details into an SQLite database for analysis and forensics and third action, the system sending a real-time alert to the web dashboard through WebSocket for immediate notification. After processing each packet (whether benign or malicious), the system loops back to capture and analyze the next packet, creating a continuous cycle of real-time threat detection and automated response.

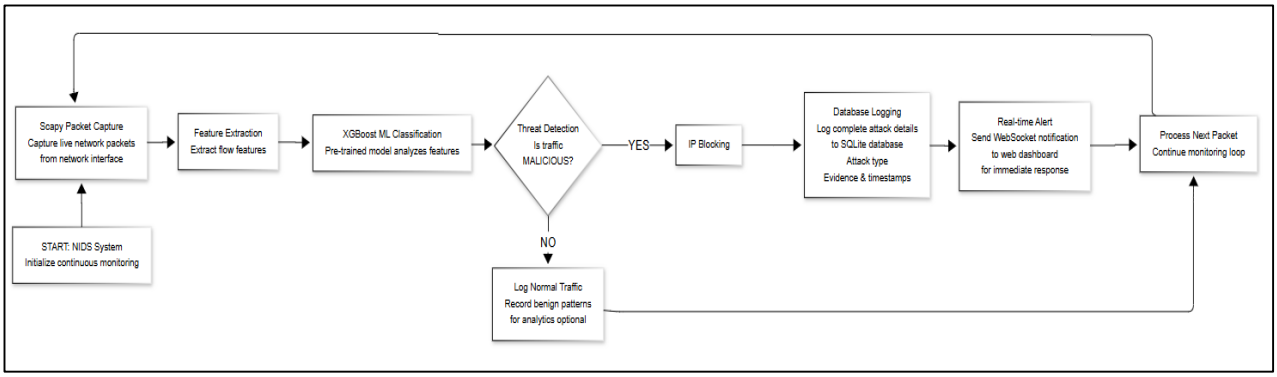

Fig.2 *RETINA System architecture*

4.2 Machine Learning Process

Fig. 3 shows the machine learning process for RETINA that begin with a dataset containing network traffic data which include various features such as packet size, protocol type, flow duration, and traffic flags. This data is then subjected to data preprocessing, where the raw data is cleaned and prepared for analysis. Tasks in this step include removing irrelevant or redundant data, converting categorical variables into numerical formats, handling missing values, and normalizing or scaling features for consistency. Next, the dataset requires data balancing where class distribution is analysed using Synthetic Minority Over-sampling Techniques (SMOTE) to oversample rare attack classes. This to ensure that the dataset is balanced and the model does not bias toward the majority class. After balancing, feature selection is performed where Random Forest classifier is initially trained to evaluate the important of each feature. The top features are selected based on their important scores, reducing dimensionality and focusing on the most relevant attributes. The balanced and feature selected data is then fed into the XGBoost Classifier stage. Here the XGBoost algorithm is trained to classify network traffic either benign or malicious.

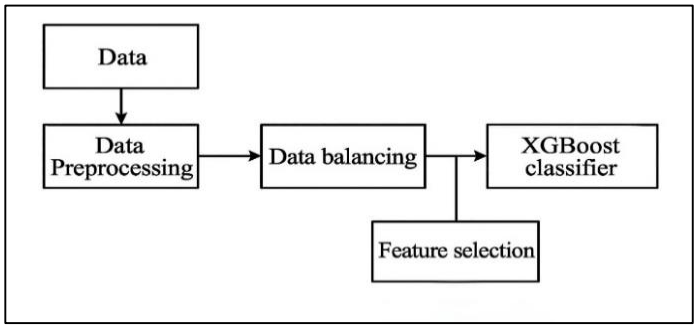

Fig.3 *Machine learning process on training model of the RETINA*

4.3 Feature Selection

Feature selection is the process of identifying and selecting the most relevant input variables (features) from a dataset that contribute the most to the prediction variable or output of interest. It helps in improving the performance of machine learning models by reducing overfitting, enhancing accuracy, and decreasing computational cost. In this study, feature selection was carried out to reduce the dimensionality of the CICIDS2017 dataset, which originally contained 78 features. A Random Forest algorithm was utilized to compute the importance scores for each feature based on how effectively they contribute to distinguishing between normal and malicious traffic. Based on these scores, the top 50 most significant features were selected for further analysis and model training, as detailed in Appendix A.

4.4 Requirement Analysis

Requirement analysis is the process of identifying and understanding the needs and desires of users for an application to be developed or updated. It involves analysing, documenting, validating, and managing the software or system requirements to ensure that the final system meets the user's needs and functions as intended. This includes both functional and non-functional requirements.

4.4.1 Functional Requirements

Functional requirements define the system's capabilities and outline what it should accomplish. It also describes the system's characteristics and behaviour that enable users to perform their tasks effectively. These requirements are crucial as they specify how the system processes user inputs and delivers the expected outcomes. Table 3 lists the functional requirements of RETINA system.

Table 3 *Functional requirements for RETINA system*

No	Functional Requirement	Description
1	Network Traffic Monitoring	The system must continuously monitor network traffic in real time
2	Real time analysis	The system must be able to analyse and classify the attack type once threat activity detected
3	Alert Generation	The system must generate alerts for detected anomalies or threats, providing details such as the type of the threat
4	Traffic Data Logging	The system will log onto network traffic and alert information for future analysis, auditing, and training of the system

4.4.2 Non-Functional Requirements

Non-functional requirements define the system's characteristics and serve as evaluation criteria for its operation rather than focusing on specific behaviours include usability, reliability, scalability and performance. These attributes are essential for ensuring the system's usability and effectiveness. Table 4 outlines the non-functional requirements of RETINA system.

Table 4 *Non-Functional Requirements for RETINA system*

No	Non-Functional Requirement	Description
1	Usability	The system's interface should be user's friendly and require minimal practice
2	Reliability	The system must detect the attack without any error
3	Scalability	The system should scale effectively to adapt the increased usage
4	Performance	The run time of the system should not take a long time to produce analysis result

4.5 Use Case Diagram

The use case diagram illustrates how users interact with the system to accomplish specific tasks. It provides a clear view of the user's perspective during their interaction with the system. Fig. 4 presents the use case diagram for the proposed system, detailing the interactions between the user and the system. In this application, the actor is the user, who can monitor network traffic, receive real-time alert, block/unblock malicious IP address and view detected attack logs stored in database.

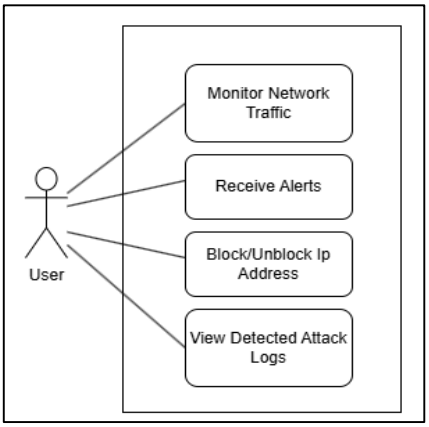

Fig.4 *Use Case Diagram of RETINA*

4.6 Activity Diagram

Activity diagrams are a flowchart that depicts the flow of activities or actions within a system. it is often used to illustrate the steps involved in a single operation or process and show how the events in that process are related to one another. Fig. 5 illustrates the user activity diagram of RETINA system. The process begins with the system capturing live network traffic data, which is then analysed using the XGBoost classifier to determine whether the traffic is normal or potentially malicious. If no anomalies are detected, the traffic is labelled as benign and is subsequently displayed as part of the normal network activity. However, if an anomaly is identified, the system proceeds to send alerts to notify administrators or monitoring systems about the detected attack. The event is then recorded in the database for logging and analysis purposes. Next, the system checks whether any action is required to block or unblock the associated IP address based on the nature of the threat. If action is taken, it is updated accordingly. Otherwise, the process continues to display the network traffic. RETINA system demonstrates an automated and intelligent flow of intrusion detection, alert generation, response handling, and real-time monitoring and effectively supporting proactive network security.

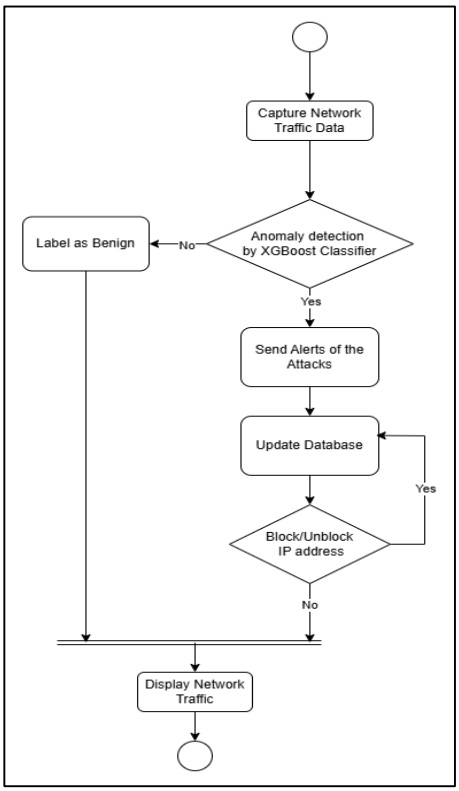

Fig.5 *Activity Diagram of RETINA* system

4.7 Sequence Diagram

The sequence diagram in Fig. 6 illustrates how objects interact with each other in a particular sequence to accomplish a specific task. The process begins automatically when the system is running. So, there is no need for the user to manually activate the system. Upon entering the dashboard through the interface, the user can immediately view network traffic data. Simultaneously, the traffic monitoring and data collection component is already capturing live network traffic and forward it to the XGBoost model for analysis. The XGBoost model examines the incoming traffic data and detects any anomalies that may indicate malicious activity. If an anomaly is identified, the system sends the detection results to the database, which stores the information for logging and future reference. The interface then displays alerts and attack logs to inform the user of any detected threats. This automated setup ensures that intrusion detection runs in real-time as soon as the system is operational, providing continuous monitoring and immediate feedback on potential network threats.

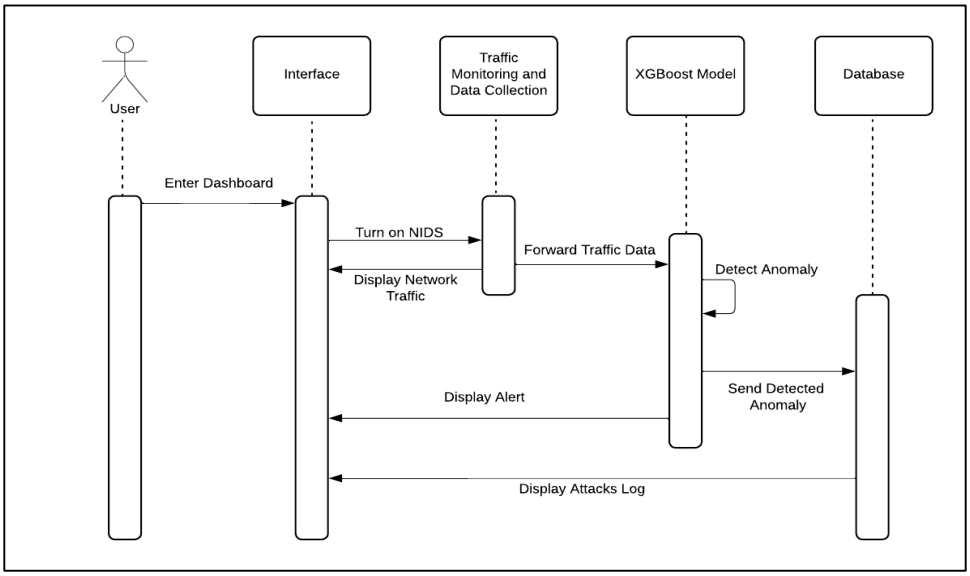

Fig.6 *User Sequence Diagram of RETINA*

4.8 User Interface Design

The RETINA system's main page shows in Fig. 7. The system will start capturing and monitoring network traffic automatically once it is running. The first interface is the main dashboard, which serves as a central hub for monitoring real time network traffic. It provides a traffic overview graph indicating normal activity and anomalies, a detailed network log with timestamps, intrusion alert, attacking IP addresses and network statistics.

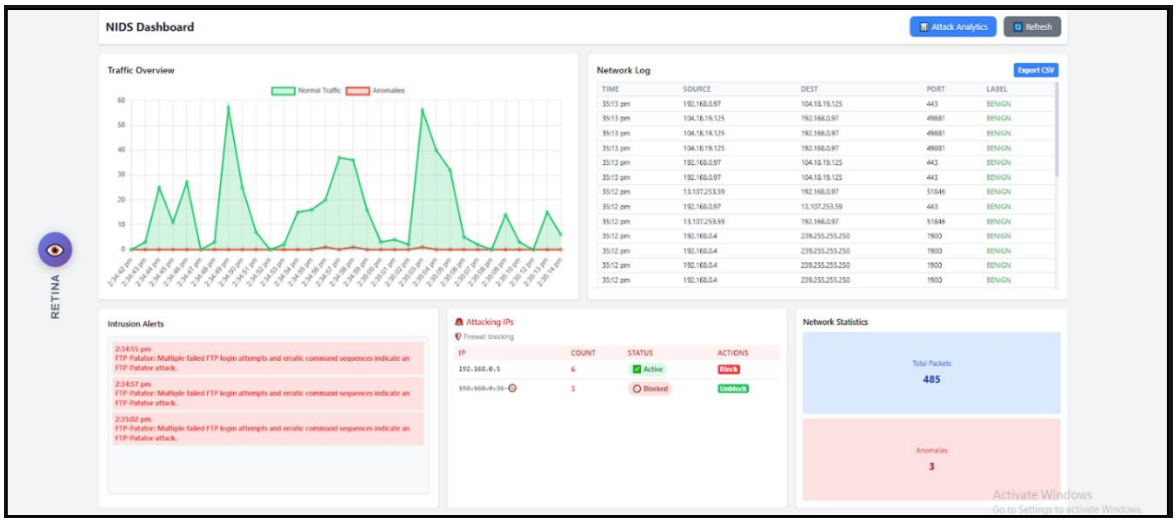

Fig.7 *Main page of RETINA*

Fig. 8 shows RETINA Attack Analytics page. It provides an overview of network attack data, including sections for Top Attacking IPs, Top Targeted Ports, Blocked IP Addresses, and Recent Attacks. There is also an IP Analysis panel that allows users to input and analyze specific IP addresses for threat intelligence. Users can filter recent attacks by time and type and refresh the data using the provided buttons for real-time updates. The layout emphasizes quick access to key threat information for network security monitoring.

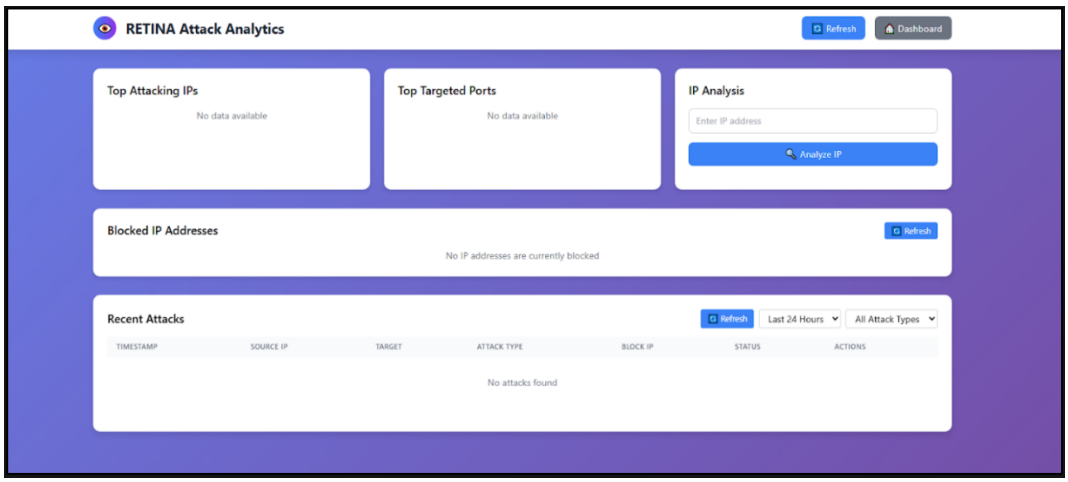

Fig.8 *Attack Analytics page of RETINA*

4.9 Test Plan

The test plan for this system was created as a guide for validating and ensuring the functionality of the RETINA as intended. Table 5 illustrate the test plans for all the functionalities included in the proposed system. The testing process was carried out by a group of dedicated home network users, ensuring a thorough evaluation of the system's performance

Table 5 *Test Plan of RETINA*

No	Test Case	Expected Output	Actual Result
1	Home Page Functionality Test	An alert message will display and ask the user to complete the login form.	Pass/Fail
2	Attack Detection and Alerting test	Verifies that the system accurately detects attacks and generates real-time alerts.	Pass/Fail
3	Attack Analytics Page functionality Test	Assesses that the analytics page displays detailed visual summaries of detected attacks.	Pass/Fail
4	Blocking/Unblocking IP address test	Verifies whether users can successfully block and unblock suspicious IP addresses from the interface.	Pass/Fail
5	Usability test	Evaluate the overall user experience, such as navigation	Pass/Fail
6	Performance test	Assesses the system's speed, responsiveness under various scenarios	Pass/Fail

5. Implementation and Result

The system implementation phase includes the process and results model training for intrusion detection. This section will also explain testing of the system and the training result.

5.1 WebSocket Configuration

WebSocket is a communication protocol that provides full-duplex communication channels over a single Transmission Control Protocol (TCP) connection. Unlike traditional Hyper Text Transfer Protocol (HTTP) requests which follow a request-response pattern, WebSocket enables real-time, bidirectional communication between a client (web browser) and server. Once a WebSocket connection is established through an initial HTTP handshake, both the client and server can send data to each other at any time without the overhead of HTTP headers. This makes WebSocket ideal for applications requiring real-time updates monitoring dashboards where instant data transmission is crucial.

Fig. 9 shows WebSocket serves as the backbone for real-time monitoring by creating a persistent connection between the backend detection engine and the frontend dashboard. When a client connects to the /ws endpoint, the server accepts the WebSocket connection and immediately checks if the packet capture thread is running. If not, it spawns a new daemon thread to begin monitoring network traffic. The system then enters a continuous

loop where it processes network flows from the packet queue using the Intrusion Detector class to perform machine learning-based threat analysis.

```python
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    if not hasattr(app.state, "capture_thread") or not app.state.capture_thread.is_alive():
        app.state.capture_thread = threading.Thread(target=start_packet_capture, daemon=True)
        app.state.capture_thread.start()

    try:
        while True:
            try:
                features = packet_queue.get_nowait()
                analysis = IntrusionDetector(features)
                message = {
                    "features": features,
                    "prediction_label": analysis.prediction_label,
                    "intrusion_type": analysis.intrusion_type,
                    "severity": analysis.severity,
                    "evidence": analysis.evidence,
                    "timestamp": datetime.now().isoformat()
                }
                app.state.stats.add_detection(message)

                # Log to database if it's an attack
                if analysis.prediction_label != "normal":
                    attack_data = {
                        'src_ip': features.get('src_ip', 'Unknown'),
                        'dst_ip': features.get('dst_ip', 'Unknown'),
                        'intrusion_type': analysis.intrusion_type,
                        'prediction_label': analysis.prediction_label,
                        'severity': analysis.severity,
                        'is_blocked': check_if_ip_should_be_filtered(features.get('src_ip', ''))
                    }

                    attack_id = db_service.log_attack(attack_data, features, analysis.evidence)
                    message["attack_id"] = attack_id

                await websocket.send_text(json.dumps(message))
                logging.info(f"Detected: {json.dumps(message)}")
```

Fig 9 *WebSocket*

5.2 Model Training

Fig. 10 presents the code that performs a multi-class classification task using the XGBoost classifier on a CIC-IDS2017 dataset. First, the dataset is split into training and testing sets using the train_test_split function from Scikit-learn, with 70% of the data used for training and 30% for testing. The XGBoost model is configured for multi-class classification using the 'multi:softmax' objective, meaning it predicts the class labels directly. The model parameters include 100 trees (n_estimators), a maximum depth of 10 per tree, a learning rate of 0.1, and a fixed random state for reproducibility. After training the model on the training data, predictions are made on the test data. The performance of the model is then evaluated using a classification report, which displays precision, recall, and F1-score for each class. Additionally, a confusion matrix is generated and visualized using a heatmap to better understand how well the model distinguishes between different classes.

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_balanced, test_size=0.3, random_state=42)

model = XGBClassifier(n_estimators=100,
                      objective='multi:softmax',
                      num_class=len(np.unique(y_balanced)),
                      learning_rate=0.1,
                      max_depth=10,
                      random_state=42)
model.fit(X_train, y_train)
```

```python
cm=confusion_matrix(y_test, y_pred)
labels = le.classes_
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for CIC-IDS2017 (XGBoost)')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

Fig 10 *Model training process using the XGBoost algorithm*

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	680919
DoS Attack	1.00	1.00	1.00	114159
FTP-Patator	1.00	1.00	1.00	15106
SSH-Patator	1.00	1.00	1.00	14949
accuracy			0.99	962959
macro avg	0.96	0.96	0.96	962959
weighted avg	0.99	0.99	0.99	962959

Average False Positive Rate (FPR): 0.07%
Average False Negative Rate (FNR): 3.69%

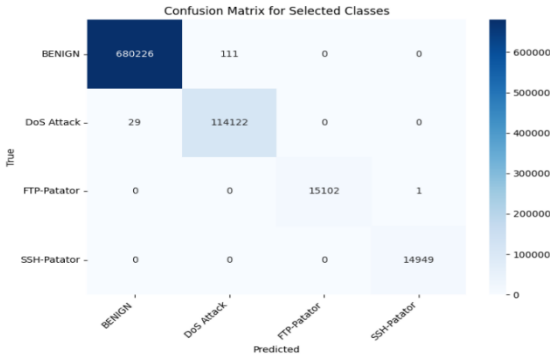

Fig 11 *Evaluation Result of the trained XGBoost model*

Fig. 11 presents the results of a classification model evaluated using various metrics such as accuracy, precision, recall, F1-score, and a confusion matrix. The model was tested on a dataset containing 962,959 samples, representing different types of network traffic, including both benign activity and malicious attacks such as DoS, FTP-Patator, and SSH-Patator. The overall accuracy of the model is 99%, indicating that it correctly classified most of the samples. Precision and recall are both reported as 1.00 across all classes except for minor deviations in the BENIGN and DoS Attack categories, where some misclassifications occurred. This results in similarly high F1-scores, confirming that the model maintains a strong balance between precision and recall. The confusion matrix on the right side of the figure provides a visual summary of the model's predictions. It shows that the model correctly identified 680,226 BENIGN samples, misclassifying only 111 of them as DoS attacks. Similarly, 114,132 DoS Attack samples were correctly identified, with just 29 being misclassified as BENIGN. For the FTP-Patator and SSH-Patator classes, classification performance was nearly perfect, with only one sample misclassified in each case. These results highlight the model's ability to distinguish accurately between normal and attack traffic. Additionally, the model achieves a very low average False Positive Rate (FPR) of 0.07%, which is important for minimizing the incorrect classification of benign traffic as malicious. The False Negative Rate (FNR) is also kept low at 3.69%, indicating that only a small proportion of actual attacks were missed. Such low error rates are essential for effective network intrusion detection, ensuring both high detection rates and minimal disruption due to false alarms. Overall, the model demonstrates excellent performance and reliability in detecting and classifying network traffic.

5.3 IP Blocking

```python
def block_ip_firewall(ip_address):
    """Block an IP address using Windows Firewall via netsh."""
    rule_name_in = f"NIDS_Block_IN_{ip_address.replace('.', '_')}"
    rule_name_out = f"NIDS_Block_OUT_{ip_address.replace('.', '_')}"

    # Block incoming traffic from the IP
    command_args_in = [
        "advfirewall", "firewall", "add", "rule",
        f"name={rule_name_in}",
        "dir=in",
        "action=block",
        f"remoteip={ip_address}",
        "protocol=any"
    ]

    # Block outgoing traffic to the IP
    command_args_out = [
        "advfirewall", "firewall", "add", "rule",
        f"name={rule_name_out}",
        "dir=out",
        "action=block",
        f"remoteip={ip_address}",
        "protocol=any"
    ]

    # Execute both commands
    success_in, message_in = execute_netsh_command(command_args_in)
    success_out, message_out = execute_netsh_command(command_args_out)

    if success_in and success_out:
        with threat_data_lock:
            blocked_ips.add(ip_address)

        # Save to database for persistence
        save_blocked_ip_to_db(ip_address)

        logging.info(f"Successfully blocked IP: {ip_address} (both inbound and outbound)")
        return True, f"IP {ip_address} blocked successfully"
    else:
        logging.error(f"Failed to block IP {ip_address}")
        return False, f"Failed to block IP"
```

Fig 12 *IP Blocking Function*

The block_ip_address function blocks both incoming and outgoing traffic to a specified IP address on a Windows machine by creating firewall rules using the netsh command as shown in Fig. 14. It constructs two rule names (for inbound and outbound), builds corresponding command arguments to block all protocols to/from the IP, and executes them using a helper function. If both rules are successfully applied, the IP is added to an in-memory set (protected by a thread lock) and saved to a database for persistence, with success logged. If either command fails, an error is logged, and the function returns failure.

5.4 Packet Capture

Fig. 13 shows Packet Capture and Processing uses Scapy's packet sniffing capabilities with callback functions. This to intercept live network traffic, filters out already-blocked IPs to prevent processing loops, extract network flow keys for packet aggregation, and feed processed features into the analysis queue for immediate threat detection, creating a complete end-to-end pipeline from raw packets to actionable security intelligence.

```
def packet_callback(packet):
    """Callback used by Scapy to process each captured packet."""
    if IP in packet:
        ip = packet[IP]

        # Skip processing packets from blocked IPs
        if check_if_ip_should_be_filtered(ip.src):
            logging.debug(f"Filtered packet from blocked IP: {ip.src}")
            return

        flow_key = update_flow(packet)
        if flow_key is None:
            return
        flow_packets = flow_table.get(flow_key, [])
        features = extract_flow_features(flow_packets)

        # Add IP information for the network log
        features["src_ip"] = ip.src
        features["dst_ip"] = ip.dst

        packet_queue.put(features)

def start_packet_capture():
    sniff(prn=packet_callback, store=0, stop_filter=lambda _: shutdown_event.is_set())
```

Fig. 13 *Packet Capture and Processing*

5.5 Testing and Verification

This section explains how the RETINA system is tested on its function, overall performance, compatibility, and user acceptance. This is to ensure that the developed system is aligned with the requirements and objectives mentioned in the Requirement Gathering and Analysis phase.

5.5.1 System Testing

The testing phase of RETINA intends to assess the efficiency and efficacy of the machine learning model in real-world network situations. The RETINA system was tested against DoS attacks and Brute Force attacks, where the attacks were simulated in a controlled virtual environment using VirtualBox. The victim machine ran on Windows 10, while the attacker machine was on Kali Linux. Attack tools such as hping3 and patator were used from the Kali Linux machine to launch DoS and FTP brute force attacks respectively targeting the Windows 10 system. The results show that the RETINA system successfully detected all the attacks. Additionally, more tests were performed with the system running in a normal traffic situation (without performing any cyberattack) in a local network, for a total of around 4 hours of workload. This to test whether any problems occurred during execution and to spot any false positive results. During the experiments very minor false positives were identified.

```
┌──(zainurain㉿Kali)-[~]
└─$ sudo hping3 -S --flood -V -p 80 192.168.0.97
```

Fig 15 *hping3*

```
┌──(zainurain㉿Kali)-[~]
└─$ patator ftp_login host 192.168.9.97 user=FILE0 password=FILE1 0=usernames.txt 1=passwords.txt -x ignore:code=530
```

Fig 16 *patator*

Fig. 15 shows the hping3, a network testing tool to perform a flood attack against the IP address 192.168.0.97. This type of command would have generated a high volume of SYN packets to overwhelm the target system's ability to respond, which is a form of denial-of-service attack. Fig. 16 shows the patator, a multi-purpose brute-

forcing tool to attempt a brute force attack against a File Transfer Protocol (FTP) service. This was a dictionary attack that systematically tried different username and password combinations from the provided wordlists against the FTP server. Table 6 shows the testing results of RETINA against Dos and Brute Force attack.

Table 6 *Testing Result of RETINA*

No	List of Attack	Result
1	FTP Brute Force Attack	True Positive
2	SSH Brute Force Attack	True Positive
3	DoS attack	True Positive

Table 7 displays the test plan of the RETINA system. The test plan was after the development of the system. The Test Plan serves as a guide for validating and ensuring the effectiveness of the RETINA system in a systematic manner. It tested the functionality with different environments.

Table 7 *Test Plan for RETINA*

No	Test Case	Expected Output	Actual Result
1	Home Page Functionality Test	An alert message will display and ask the user to complete the login form	Pass
2	Attack Detection and Alerting test	Verifies that the system accurately detects attacks and generates real-time alerts.	Pass
3	Attack Analytics Page functionality Test	Assesses that the analytics page displays detailed visual summaries of detected attacks.	Pass
4	Blocking/Unblocking IP address test	Verifies whether users can successfully block and unblock suspicious IP addresses from the interface.	Pass
5	Usability test	Evaluate the overall user experience, such as navigation	Pass
6	Performance test	Assesses the system's speed, responsiveness under various scenarios	Pass

5.5.2 User Acceptance Testing

A total of 10 respondents from Universiti Tun Hussein Onn Malaysia and daily home network users had responded to the User Acceptance Testing (UAT) form. Table 8 shows the user's acceptance testing result using Google Form.

Table 8 *User Acceptance Testing Form of RETINA*

Question	Description	Actual		
		Agree	Disagree	Neutral
1	The system's interface is user friendly and easy to navigate	10		
2	All the buttons function as expected and perform their intended actions correctly	10		
3	The system continuously monitors network traffic in real time	10		
4	The system provides clear and understandable alerts when intrusions are detected	9		1
5	The system effectively analyses and classifies the attack type once intrusion activity detected, but there is room for improvement in handling certain variant of network attack or emerging threats	8		2
6	The system demonstrates reasonable effectiveness in blocking and unblocking attacking IP address.	10		

Table 8 *(cont.)*

Question	Description	Actual		
		Agree	Disagree	Neutral
7	The system effectively logs all detected attack with its detail for future analysis	10		
8	The system's performance in terms of speed and responsiveness is satisfactory	9		1
9	I would recommend RETINA system to others to monitor their home network	10		

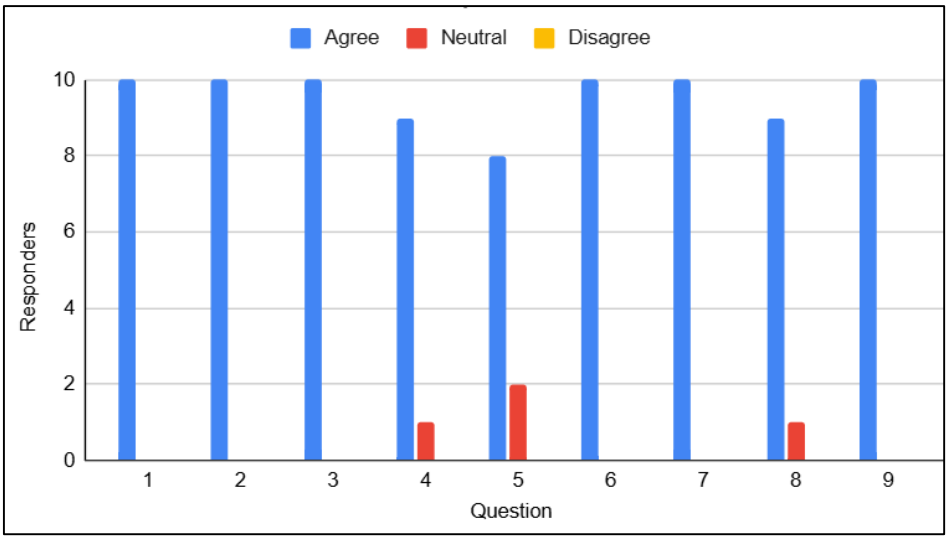

Fig 15 User Acceptance result using Google Form

Fig. 15 shows most of the respondents gave a positive result from question 1 to question 9. This showcase that the developed RETINA system functions and is usable. The only disagreement from the respondent is where the user thinks the system could detect wider attack types as the system currently only focuses on detecting Denial of Service (DoS) and Brute Force Attack.

Conclusion

The RETINA system has successfully achieved all the objectives outlined in this project. It was designed using an object-oriented approach and developed by employing the XGBoost algorithm, trained on the CICIDS-2017 dataset for effective detection of Denial of Service (DoS) and brute force attacks. The system was implemented with alpha and beta testing involving the target users. RETINA addresses the problem statement by overcoming common challenges in traditional NIDS, such as high false positives that lead to reduced trust. The system's integration with a user-friendly interface ensures accessibility and operational transparency for home network administrators. Future work could focus on extending the system's capabilities to handle a wider variety of attack types, optimizing its performance for larger and more complex networks, and exploring the integration of advanced machine learning techniques, such as deep learning, for better adaptability and scalability.

Acknowledgment

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *M. Z. M. Zain, I. R. A. Hamid;* ***data collection:*** *M. Z. M. Zain, I. R. A. Hamid;* ***analysis and interpretation of results*** *M. Z. M. Zain, I. R. A. Hamid;* ***draft manuscript preparation:*** *M. Z. M. Zain, I. R. A. Hamid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] P. A. Perdigão, N. M. Coelho, and J. C. Brás, "AI-Driven Threats in Social Learning Environments - A Multivocal Literature Review," *ARIS2 - Advanced Research on Information Systems Security*, vol. 5, no. 1, pp. 4–37, May 2025, doi: 10.56394/aris2.v5i1.60.

[2] I. A. Mahar, W. Libing, G. A. Rahu, Z. A. Maher, and M. Y. Koondhar, "Feature Based Comparative Analysis of Traditional Intrusion Detection System and Software-Defined Networking Based Intrusion Detection System," in *International Conference on Engineering Technologies and Applied Sciences: Shaping the Future of Technology through Smart Computing and Engineering, ICETAS 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ICETAS59148.2023.10346497.

[3] J. Mijalkovic and A. Spognardi, "Reducing the False Negative Rate in Deep Learning Based Network Intrusion Detection Systems," *Algorithms*, vol. 15, no. 8, Aug. 2022, doi: 10.3390/a15080258.

[4] B. H. Ali, N. Sulaiman, S. A. R. Al-Haddad, R. Atan, and S. L. M. Hassan, "DDoS Detection Using Active and Idle Features of Revised CICFlowMeter and Statistical Approaches," in *ICOASE 2022 - 4th International Conference on Advanced Science and Engineering*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 148–153. doi: 10.1109/ICOASE56293.2022.10075591.

[5] M. Ozkan-Okay, R. Samet, O. Aslan, and D. Gupta, "A Comprehensive Systematic Literature Review on Intrusion Detection Systems," 2021, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2021.3129336.

[6] S. Razdan, H. Gupta, and A. Seth, "Performance Analysis of Network Intrusion Detection Systems using J48 and Naive Bayes Algorithms," in *2021 6th International Conference for Convergence in Technology, I2CT 2021*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021. doi: 10.1109/I2CT51068.2021.9417971.

[7] N. Lim, J. Jie, K. Amin, and M. Sukri, "Personal Home Network Intrusion Checking Tool (PacAN)," *Applied Information Technology And Computer Science*, vol. 2, no. 2, pp. 115–126, 2021, doi: 10.30880/aitcs.2021.02.02.008.

[8] A. A. Hamza and R. J. surayh Al-Janabi, "Detecting Brute Force Attacks Using Machine Learning," in *BIO Web of Conferences*, EDP Sciences, Apr. 2024. doi: 10.1051/bioconf/20249700045.

[9] S. K. Wanjau, G. M. Wambugu, and G. N. Kamau, "SSH-Brute Force Attack Detection Model based on Deep Learning," *International Journal of Computer Applications Technology and Research*, vol. 10, no. 01, pp. 42–50, Jan. 2021, doi: 10.7753/ijcatr1001.1008.

[10] "What Is a Denial of Service (DoS) Attack? - Palo Alto Networks." Accessed: Nov. 14, 2024. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos

[11] H. S. Obaid, "Denial of Service Attacks: Tools and Categories." [Online]. Available: www.ijert.org

[12] G. S. Fuhnwi, M. Revelle, and C. Izurieta, "Improving Network Intrusion Detection Performance : An Empirical Evaluation Using Extreme Gradient Boosting (XGBoost) with Recursive Feature Elimination," in *2024 IEEE 3rd International Conference on AI in Cybersecurity, ICAIC 2024*, Institute of Electrical and Electronics Engineers Inc., 2024. doi: 10.1109/ICAIC60265.2024.10433805.

[13] N. Iftikhar, M. U. Rehman, M. A. Shah, M. J. F. Alenazi, and J. Ali, "Intrusion Detection in NSL-KDD Dataset Using Hybrid Self-Organizing Map Model," *CMES - Computer Modeling in Engineering and Sciences*, vol. 143, no. 1, pp. 639–671, 2025, doi: 10.32604/cmes.2025.062788.

[14] A. A. E. Boukebous, M. I. Fettache, G. Bendiab, and S. Shiaeles, "A Comparative Analysis of Snort 3 and Suricata," in *2023 IEEE IAS Global Conference on Emerging Technologies, GlobConET 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/GlobConET56651.2023.10150141.

[15] N. Ziadah Harun and F. Sains Komputer dan Teknologi Maklumat, "Distributed Denial of Service (DDoS) Detection Tool Using Artificial Neural Network (ANN) for Homelab," *Applied Information Technology And Computer Science*, vol. 4, no. 2, pp. 139–158, 2023, doi: 10.30880/aitcs.2023.04.02.009.

[16] S. Viharika and N. Balaji, "Enhancing Intrusion Detection and Cloud Security by Integrating Snort with Advanced AI Techniques for Improved Accuracy and Threat Mitigation," 2024. [Online]. Available: https://www.jisem-journal.com/
[17] G. K. Bada, W. K. Nabare, and D. K. K. Quansah, "Comparative Analysis of the Performance of Network Intrusion Detection Systems: Snort, Suricata and Bro Intrusion Detection Systems in Perspective," *Int J Comput Appl*, vol. 176, no. 40, pp. 39–44, Jul. 2020, doi: 10.5120/ijca2020920513.
[18] A. Tiwari, S. Saraswat, U. Dixit, and S. Pandey, "Refinements In Zeek Intrusion Detection System," in *8th International Conference on Advanced Computing and Communication Systems, ICACCS 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 974–979. doi: 10.1109/ICACCS54159.2022.9785047.
[19] N. Niknami, E. Inkrott, and J. Wu, "Towards Analysis of the Performance of IDSs in Software-Defined Networks," in *Proceedings - 2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems, MASS 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 787–793. doi: 10.1109/MASS56207.2022.00124.
[20] Dhanashri Ashok Bhosale and Vanita Manikrao Mane, *Comparative Study and Analysis of Network Intrusion Detection Tools*. IEEE, 2020.
[21] wikiwand, "Object-oriented analysis and design - Wikiwand." Accessed: Nov. 30, 2024. [Online]. Available: https://www.wikiwand.com/en/articles/Object-oriented_analysis_and_design

Appendix A

No	Feature	Description
1	destination_port	Transport-layer port at the flow's destination host
2	init_win_bytes_backward	Bytes advertised in the initial TCP window (destination → source)
3	min_seg_size_forward	Smallest TCP segment size observed (source → destination)
4	total_length_of_bwd_packets	Sum of payload bytes in all backward-direction packets
5	init_win_bytes_forward	Bytes advertised in the initial TCP window (source → destination)
6	max_packet_length	Largest packet size (bytes) in the whole flow
7	fwd_packet_length_max	Largest packet size (bytes) in the forward direction
8	flow_iat_mean	Mean inter-arrival-time between any two packets in the flow
9	flow_iat_max	Maximum packet inter-arrival-time in the flow
10	bwd_packets/s	Average backward packets transmitted per second
11	flow_packets/s	Average packets per second for the whole flow
12	fwd_iat_min	Minimum inter-arrival-time between forward packets
13	avg_bwd_segment_size	Average TCP segment size in the backward direction
14	total_length_of_fwd_packets	Sum of payload bytes in all forward-direction packets
15	flow_iat_std	Standard deviation of packet inter-arrival-times in the flow
16	average_packet_size	Mean packet size (bytes) across the flow
17	bwd_packet_length_min	Smallest packet size (bytes) in the backward direction
18	fwd_iat_std	Std.dev. of inter-arrival-time between forward packets
19	bwd_packet_length_max	Largest packet size (bytes) in the backward direction
20	flow_bytes/s	Average bytes per second for the whole flow
21	subflow_bwd_bytes	Avg. bytes per backward sub-flow (between SYN/FIN pairs)
22	subflow_fwd_bytes	Avg. bytes per forward sub-flow
23	flow_duration	Total lifetime of the flow
24	bwd_header_length	Total header-bytes of backward packets
25	packet_length_mean	Mean packet size (bytes) across the flow
26	fwd_iat_max	Maximum inter-arrival-time between forward packets
27	fwd_packets/s	Average forward packets transmitted per second
28	fwd_iat_total	Cumulative inter-arrival-time of forward packets
29	bwd_packet_length_mean	Mean packet size (bytes) in the backward direction
30	fwd_header_length.1	Duplicate of fwd_header_length

31	fwd_iat_mean	Mean inter-arrival-time between forward packets
32	flow_iat_min	Minimum packet inter-arrival-time in the flow
33	packet_length_std	Standard deviation of packet sizes in the flow
34	packet_length_variance	Variance of packet sizes in the flow
35	fwd_packet_length_std	Std.dev. of packet sizes in the forward direction
36	avg_fwd_segment_size	Average TCP segment size in the forward direction
37	total_backward_packets	Total number of backward-direction packets
38	fwd_packet_length_mean	Mean packet size (bytes) in the forward direction
39	subflow_fwd_packets	Avg. packets per forward sub-flow
40	act_data_pkt_fwd	Count of forward packets that carry ≥ 1 byte of data
41	fwd_header_length	Total header-bytes of forward packets
42	total_fwd_packets	Total number of forward-direction packets
43	bwd_packet_length_std	Std.dev. of packet sizes in the backward direction
44	subflow_bwd_packets	Avg. packets per backward sub-flow
45	active_min	Shortest continuous active period before the flow went idle
46	psh_flag_count	Total packets (either direction) with PSH flag set
47	idle_mean	Mean length of idle periods in the flow
48	bwd_iat_std	Std.dev. of inter-arrival-time between backward packets
49	bwd_iat_min	Minimum inter-arrival-time between backward packets
50	bwd_iat_total	Cumulative inter-arrival-time of backward packets