AI POWERED ADVANCED WEB APPLICATION FIREWALL

PROJECT WORK 1

Submitted by

SRIRAM V 212222103002

in partial fulfilment for the

award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING (CYBERSECURITY)

SAVEETHA ENGINEERING COLLEGE, THANDALAM

An Autonomous Institution Affiliated to

ANNA UNIVERSITY - CHENNAI 600 025

DECEMBER 2025

SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Affiliated to Anna University | Approved by AICTE

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project report **" AI POWERED ADVANCED WEB APPLICATION FIREWALL "** is the Bonafide work of **SRIRAM V (212222103002),** who carried out this project work under my supervision.

SIGNATURE

Ms. V. Swedha, M.E., (PhD)

Assistant Professor

SUPERVISOR

Dept of Information Technology,

Saveetha Engineering College, Thandalam, Chennai 602105

SIGNATURE

Dr. G. Nagappan, M.E., PhD

Professor

HEAD OF THE DEPARTMENT

Dept of Computer Science and Engineering,

Saveetha Engineering College, Thandalam, Chennai 602105

DATE OF THE VIVA VOCE EXAMINATION: ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to our esteemed Founder President **Dr. N. M. Veeraiyan**, our President **Dr. Saveetha Rajesh**, our Director **Dr. S. Rajesh**, and the entire management team for providing the essential infrastructure.

I extend my sincere appreciation to our principal, **Dr. V. Vijaya Chamundeeswari, M.Tech., Ph.D.,** for creating a supportive learning environment for this project.

I am very thankful to our Dean of ICT, **Mr. Obed Otto,** M.E., for facilitating a conducive atmosphere that allowed me to complete my project successfully.

My thanks go to **Dr. G. Nagappan**, Professor and Head of the Department of Computer Science and Engineering at Saveetha Engineering College, for his generous support and for providing the necessary resources for my project work.

I would also like to express my profound gratitude to my Supervisor, **Ms. V. Swedha, M.E.,(PhD)** Department of Information technology, and my Project Coordinator **Dr. N.S. Gowri Ganesh**, Associate Professor at Saveetha Engineering College, for their invaluable guidance, suggestions, and constant encouragement, which were instrumental in the successful completion of this project. Their timely support and insights during the review process were greatly appreciated.

I am grateful to all my college faculty, staff, and technicians for their cooperation throughout the project. Finally, I wish to acknowledge my loving parents, friends, and well-wishers for their encouragement in helping me achieve this milestone.

ABSTRACT

With the rapid growth of web-based applications, securing web services against sophisticated cyber attacks has become a critical challenge. Traditional Web Application Firewalls (WAFs) primarily rely on static rule-based mechanisms, which are often ineffective against evolving and unknown attack patterns. This project, titled **"AI Powered Advanced Web Application Firewall"**, proposes an intelligent and adaptive security solution that leverages Artificial Intelligence and Machine Learning techniques to enhance web application protection.

The proposed system continuously monitors incoming HTTP/HTTPS traffic, analyzes request patterns, and detects malicious activities such as SQL Injection, Cross-Site Scripting (XSS), brute-force attacks, and other web-based threats in real time. By employing machine learning models trained on legitimate and malicious traffic data, the system can identify anomalies and previously unseen attack behaviors with higher accuracy than conventional WAFs. The firewall dynamically updates security rules based on detected threats, thereby improving resilience against zero-day attacks.

The architecture integrates components such as Nginx-based traffic handling, a machine learning detection engine, centralized logging, and real-time monitoring dashboards using tools like Prometheus, Grafana, and Loki. The system is implemented using Python and containerized using Docker to ensure scalability, portability, and ease of deployment.

Experimental results demonstrate that the proposed AI-powered WAF significantly improves detection accuracy while reducing false positives, making it suitable for modern, high-traffic web environments. This project showcases the effectiveness of artificial intelligence in strengthening web application security and provides a scalable foundation for future enhancements in cybersecurity defense systems.

SUSTAINABLE DEVELOPMENT GOALS (SDGs) OF THE PROJECT

Decent Work and Economic Growth

– SDG 8:

The project contributes to economic efficiency by automating critical processes through secure digital systems, reducing operational overhead, and enabling skill development in advanced domains such as artificial intelligence, cybersecurity, and system integration. It supports the creation of technology-driven employment opportunities and promotes sustainable digital workplaces.

Industry, Innovation, and Infrastructure

– SDG 9:

The system leverages modern computational technologies, including AI-based algorithms and secure data-processing frameworks, to design a robust, scalable, and resilient digital infrastructure. The architecture supports innovation through modular design, optimized data handling, and reliable system performance suitable for real-world deployment.

Peace, Justice, and Strong Institutions

– SDG 16:

The project enhances institutional transparency and accountability by incorporating secure authentication mechanisms, data integrity validation, and tamper-resistant system design. These measures ensure trust, fairness, and reliability in digital operations, strengthening governance and institutional credibility.

Partnerships for the Goals

– SDG 17:

The project encourages interdisciplinary collaboration between academic institutions, technology developers, and governing bodies to ensure ethical system development, standard compliance, and long-term sustainability through shared knowledge and resources.

TABLE OF CONTENTS

Chapter No			Title	Page No
1			**Introduction**	**1**
	1.1		Overview Of The Project	1
	1.2		Problem Definition	2
2			**Literature Survey**	**3**
	2.1		Introduction	3
	2.2		Literature Survey	4
		2.2.1	Machine Learning-Based Web Application Firewalls	4
		2.2.2	Deep Learning Techniques for Cyber Attack Detection	5
		2.2.3	Anomaly Detection in Web Traffic Using AI	6
		2.2.4	Intelligent Intrusion Detection Systems	6
		2.2.5	AI-Based SQL Injection and XSS Detection	7
		2.2.6	Comparative Study of Traditional WAF and AI-Based WAF	8
		2.2.7	Real-Time Traffic Monitoring Using Machine Learning	9

		2.2.8	Security Analytics Using Prometheus and Grafana	9
	2.3		Literature Survey Summary	10
3			**System Analysis**	**13**
	3.1		Existing System	13
	3.2		Disadvantages Of Existing System	14
	3.3		Proposed System	14
	3.4		Advantages Of Proposed System	15
	3.5		Feasibility Study	16
	3.6		Hardware Environment	17
	3.7		Software Environment	18
	3.8		Technologies Used	19
4			**System Design**	**20**
	4.1		Entity-Relationship Diagram (ERD)	20
	4.2		Data Flow Diagram (DFD)	22
	4.3		UML Diagrams	25
		4.3.1	Use Case Diagram	25
		4.3.2	Class Diagram	26
		4.3.3	Sequence Diagram	27

5			**System Architecture**	**29**
	5.1		Architecture Diagram	29
	5.2		Algorithms	30
		5.2.1	Machine Learning-Based Threat Detection Algorithm	30
	5.3		Workflow Of The System	31
	5.4		Modules Description	32
		5.4.1	Traffic Monitoring Module	32
		5.4.2	AI-Based Threat Detection Module	32
		5.4.3	Rule Engine and Response Module	33
		5.4.4	Logging and Monitoring Module	33
		5.4.5	Dashboard and Visualization Module	33
		5.4.6	Database and Storage Module	34
6			**System Implementation**	**35**
	6.1		Data Collection And Preprocessing	35
	6.2		Model Training	36
	6.3		Prediction Of Output	38
7			**System Testing**	**40**
	7.1		Black Box Testing	40

	7.2		White Box Testing	40
	7.3		Test Cases	42
8			**Conclusion And Future Enhancement**	**44**
	8.1		Conclusion	44
	8.2		Future Enhancements	45
9			**Appendix 1 – Sample Coding**	**47**
	9.1		Traffic Analysis Module Code	47
	9.2		Machine Learning Model Code	49
	9.3		API and Rule Engine Code	50
	9.4		Dashboard Integration Code	51
	9.5		Example HTML Control Panel	53
10			**Appendix 2 – Sample Output**	**55**
	10.1		Dashboard Home Page	55
	10.2		Attack Detection Output	57
	10.3		Real-Time Traffic Visualization	59
	10.4		Blocked Attack Logs	63
11			**References**	**65**

LIST OF TABLES

TABLE NO.	TABLE DESCRIPTION	PAGE NO.
2.3	Literature survey summary	12
5.2.1	Machine Learning Algorithm Description	31
7.3.1	Test Cases for SQL Injection Detection	42
7.3.2	Test Cases for XSS Attack Detection	42
7.3.3	Test Cases for Brute Force Attack Detection	43
7.3.4	Test Cases for Legitimate Traffic Handling	43

LIST OF FIGURES

FIGURE NO.	FIGURE DESCRIPTION	PAGE NO.
4.1	Entity-Relationship Diagram (ERD)	21
4.2.1	Data Flow Diagram – Level 0	23
4.2.2	Data Flow Diagram – Level 1	24
4.3.1	Use Case Diagram	26
4.3.2	Class Diagram	27
4.3.3	Sequence Diagram	28
5.1	System Architecture Diagram	30
5.3	System Workflow Diagram	32
7.1	Black Box Testing	40
7.2	White Box Testing	41
10.1.1	Waf AI Control Panel Dashboard - 1 Half	56
10.1.2	Waf AI Control Panel Dashboard - 2 Half	56
10.2.1	Waf AI Control Panel Threat Detection	58
10.2.2	Waf AI Control Panel Waf Rules	58
10.3.1	Waf AI Control Panel Traffic Control	60

10.3.2	Prometheus Waf-Api Target Health	60
10.3.3	Grafana Dashboard-1	61
10.3.4	Grafana Dashboard-2	61
10.3.5	Grafana Dashboard-3	62
10.3.6	Grafana Dashboard-4	62
10.4.1	Waf AI Control Panel System Status	64
10.4.2	Waf AI Control Panel Ml Engine	64

LIST OF ABBREVIATIONS

Abbreviation	Full Form
WAF	Web Application Firewall
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
API	Application Programming Interface
SQLi	SQL Injection
XSS	Cross-Site Scripting
DDoS	Distributed Denial of Service
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
NLP	Natural Language Processing
ELK	Elasticsearch Logstash Kibana
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
REST	Representational State Transfer
JWT	JSON Web Token
TCP	Transmission Control Protocol
IP	Internet Protocol
OS	Operating System
CLI	Command Line Interface
GUI	Graphical User Interface
URL	Uniform Resource Locator

JSON	**JavaScript Object Notation**
Docker	**Containerization Platform**
NGINX	**High-Performance Web Server**
OWASP	**Open Web Application Security Project**
ROC	**Receiver Operating Characteristic**
AUC	**Area Under the Curve**
ReLU	**Rectified Linear Unit**
K-Fold	**K-Fold Cross Validation**

LIST OF SYMBOLS

S.No.	Symbol Name	Symbol
1	Use Case	Oval / Ellipse
2	Actor (User / Admin / Attacker)	Stick Figure
3	Process (Request Analysis / ML Processing)	Rounded Rectangle
4	Start	Circle
5	Decision (Allow / Block Request)	Diamond
6	Unidirectional Arrow	Arrow
7	Entity Set (User, Logs, Rules)	Double Rectangle
8	Stop / End	Circle
9	Data Flow	Dashed Arrow
10	Input / Output (HTTP Request / Response)	Parallelogram
11	Database (Logs / Models / Rules)	Cylinder
12	Connection / Association	Line
13	Loop / Iteration (Continuous Monitoring)	Circular Arrow
14	Fork / Join (Parallel Processing)	Black Circle
15	Annotation / Note	Note Box

Chapter 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

AI Powered Advanced Web Application Firewall is a security-oriented system designed to protect modern web applications from a wide range of cyber threats using artificial intelligence and machine learning techniques. With the rapid growth of web-based services, applications have become prime targets for attacks such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), brute-force attacks, and zero-day exploits. Traditional Web Application Firewalls (WAFs), which rely mainly on static rule-based filtering, are often insufficient to detect evolving and sophisticated attack patterns. The proposed system enhances web security by integrating machine learning-based threat detection with a Web Application Firewall framework. The firewall continuously monitors incoming HTTP and HTTPS traffic, analyzes request patterns, and classifies them as normal or malicious using trained machine learning models. Based on the prediction results, the system automatically allows, blocks, or logs the request in real time. This project adopts a microservices-based architecture using containerization technologies such as Docker, enabling scalability, flexibility, and ease of deployment. Components such as traffic monitoring, AI-based threat detection, rule engine, logging, and visualization operate independently yet communicate seamlessly through APIs. Monitoring and visualization tools like Prometheus and Grafana are used to provide real-time insights into traffic behaviour, detected threats, and system performance. The AI Powered Advanced Web Application Firewall is developed as a final year academic project with an emphasis on practical implementation, real-time monitoring, automated response, and security analytics. The system aims to bridge the gap between traditional rule-based firewalls and intelligent adaptive security systems by incorporating artificial intelligence into web application protection.

1.2 PROBLEM DEFINITION

Web applications today face an increasing number of sophisticated cyber-attacks that exploit vulnerabilities in application logic, input validation, authentication mechanisms, and server configurations. Conventional Web Application Firewalls primarily rely on predefined signatures and static rules to detect malicious traffic. While effective against known attacks, these systems struggle to identify new, unknown, or evolving attack patterns, leading to false positives, false negatives, and delayed responses.

The key challenges addressed in this project include:

- **Inability to Detect Zero-Day Attacks:** Traditional WAFs fail to identify previously unseen attack patterns due to their dependency on static rules.
- **High False Positive Rates:** Legitimate user requests may be blocked incorrectly, affecting user experience and application availability.
- **Lack of Adaptive Learning:** Existing systems do not improve over time or learn from new attack data.
- **Limited Real-Time Monitoring:** Conventional firewalls provide minimal insights into traffic behaviour and attack trends.
- **Manual Rule Management:** Updating firewall rules requires manual intervention, which is time-consuming and error-prone.

The objective of this project is to design and implement an AI-powered Web Application Firewall that overcomes these limitations by using machine learning algorithms to analyze web traffic, detect malicious requests in real time, and respond automatically. The system aims to improve detection accuracy, reduce false alarms, and provide comprehensive monitoring and logging capabilities, thereby enhancing the overall security posture of web applications.

Chapter 2

LITERATURE SURVEY

2.1 INTRODUCTION

The rapid expansion of web-based applications has significantly increased the risk of cyber-attacks targeting application-layer vulnerabilities. Web Application Firewalls (WAFs) have emerged as a critical security mechanism to protect web applications from attacks such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and denial-of-service attacks. However, traditional WAF solutions primarily depend on static, signature-based rules, which limits their effectiveness against evolving and unknown attack patterns.

Recent advancements in artificial intelligence and machine learning have introduced new approaches for improving web security. Machine learning-based systems can analyze large volumes of web traffic, learn normal behavior patterns, and detect anomalies that may indicate malicious activities. This adaptive capability makes AI-driven WAFs more robust and efficient compared to conventional rule-based firewalls.

The literature survey explores existing research, methodologies, and technologies related to AI-based Web Application Firewalls, intrusion detection systems, anomaly detection techniques, and real-time monitoring solutions. It reviews various machine learning and deep learning models proposed by researchers for detecting cyber threats at the application layer. Additionally, the survey highlights the limitations of existing approaches and identifies research gaps that motivate the development of the proposed AI Powered Advanced Web Application Firewall.

This chapter provides a comprehensive understanding of prior work in the domain of web application security and forms the foundation for the design and implementation of the proposed system.

2.2 LITERATURE SURVEY

2.2.1 Machine Learning-Based Web Application Firewalls

Machine Learning-Based Web Application Firewalls (ML-WAFs) represent a significant evolution from traditional rule-based security mechanisms. Conventional WAFs rely on predefined signatures and manually crafted rules to detect malicious requests. While effective against known attacks, these systems struggle to identify zero-day vulnerabilities and sophisticated attack patterns that do not match existing rules. Machine learning-based WAFs overcome these limitations by learning patterns directly from web traffic data. These systems analyze features such as HTTP request headers, URLs, payload content, request frequency, and user behavior. By training machine learning models on labeled datasets containing both normal and malicious traffic, ML-WAFs can classify incoming requests with higher accuracy. Several studies have demonstrated the effectiveness of supervised learning algorithms such as Support Vector Machines (SVM), Random Forest, Decision Trees, and Naïve Bayes for web attack detection. These models are capable of detecting common attacks like SQL Injection, Cross-Site Scripting (XSS), and command injection by learning statistical patterns in request payloads. Recent research has also explored unsupervised and semi-supervised learning approaches for anomaly detection. These methods identify deviations from normal traffic behavior without requiring extensive labeled datasets. Clustering techniques and autoencoders are commonly used to model normal web traffic and flag abnormal activities. Despite their advantages, machine learning-based WAFs face challenges such as dataset imbalance, feature selection complexity, and computational overhead. However, when combined with real-time monitoring and adaptive rule generation, ML-WAFs provide a scalable and intelligent defense mechanism for modern web applications. The proposed **AI Powered Advanced Web Application Firewall** builds upon these research findings by integrating machine learning models for real-time traffic analysis and automated threat mitigation, improving detection accuracy and adaptability.

2.2.2 Deep Learning Techniques for Cyber Attack Detection

Deep learning techniques have gained significant attention in the field of cybersecurity due to their ability to automatically extract complex features from large-scale data. Unlike traditional machine learning models that rely heavily on manual feature engineering, deep learning models learn hierarchical representations directly from raw input data, making them highly effective for cyber attack detection.

In web application security, deep learning models such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Autoencoders are widely used to detect malicious patterns in HTTP requests and network traffic. CNNs are particularly effective in identifying spatial patterns in encoded payload data, making them suitable for detecting attacks like SQL Injection and Cross-Site Scripting (XSS). RNNs and LSTM networks excel at analyzing sequential data, allowing them to capture temporal dependencies in web traffic behavior.

Several research works have demonstrated that deep learning-based intrusion detection systems outperform traditional machine learning approaches in terms of accuracy and robustness. These models can detect complex, multi-stage attacks and previously unseen attack variants by learning deep contextual representations of web requests.

However, deep learning models require large volumes of training data and high computational resources, which can increase deployment complexity. Despite these challenges, the integration of deep learning techniques in Web Application Firewalls enhances detection capability and enables adaptive learning against evolving cyber threats.

The proposed AI Powered Advanced Web Application Firewall leverages deep learning concepts to strengthen real-time threat detection and improve resilience against sophisticated web-based attacks.

2.2.3 Anomaly Detection in Web Traffic Using AI

Anomaly detection plays a crucial role in identifying cyber attacks that deviate from normal web traffic behavior. AI-based anomaly detection techniques focus on learning patterns of legitimate user activity and flagging any unusual or suspicious requests as potential threats.

In web application environments, anomaly detection systems analyze parameters such as request frequency, payload length, URL structure, session duration, and access patterns. Machine learning models including Isolation Forest, One-Class SVM, k-means clustering, and autoencoders are commonly employed for this purpose. These models are particularly effective in detecting zero-day attacks and unknown vulnerabilities that cannot be identified using signature-based methods.

AI-driven anomaly detection systems continuously adapt to changes in traffic patterns, reducing false positives over time. By combining statistical analysis with machine learning, these systems provide a proactive approach to web security. They can identify slow-rate attacks, brute-force attempts, and abnormal usage behavior that may indicate reconnaissance or exploitation activities.

Despite their advantages, anomaly detection systems may initially generate higher false positives during the learning phase. Proper tuning, feature selection, and continuous training are essential to maintain accuracy.

In the proposed system, anomaly detection mechanisms are integrated with machine learning-based classification models to ensure comprehensive protection. This hybrid approach enhances detection accuracy while maintaining real-time performance, making it suitable for modern, high-traffic web applications.

2.2.4 Intelligent Intrusion Detection Systems

Intelligent Intrusion Detection Systems (IDS) are designed to monitor network and application activities to identify malicious behavior and security policy violations. Traditional IDS relied on rule-based or signature-based detection, which limited their ability to detect new or evolving cyber threats. To overcome these limitations, modern IDS integrate artificial intelligence and machine learning techniques to enhance detection accuracy and adaptability.

AI-based IDS analyze large volumes of network traffic and application logs to learn patterns of normal behavior and identify deviations that may indicate intrusions. These systems can detect a wide range of attacks, including denial-of-service attacks, brute-force login attempts, privilege escalation, and web-based exploits. By using learning algorithms, intelligent IDS can continuously improve their detection capabilities without manual rule updates.

Deep learning-based IDS further enhance performance by capturing complex relationships in data. Models such as neural networks and autoencoders enable the detection of subtle attack patterns that may bypass traditional security mechanisms. Intelligent IDS are also capable of correlating multiple events across time to identify coordinated or multi-stage attacks.

However, challenges such as high false-positive rates and computational overhead remain. Proper training, tuning, and integration with response mechanisms are required to ensure efficiency. In the proposed AI Powered Advanced Web Application Firewall, intelligent intrusion detection concepts are incorporated to provide proactive and adaptive security against modern cyber threats.

2.2.5 AI-Based SQL Injection and XSS Detection

SQL Injection (SQLi) and Cross-Site Scripting (XSS) are among the most common and dangerous web application attacks. Traditional Web Application Firewalls rely heavily on predefined rules and signatures to detect these attacks. However, attackers continuously modify payloads to bypass static rules, making traditional approaches less effective.

AI-based SQL Injection and XSS detection uses machine learning and deep learning techniques to analyze request patterns, input parameters, and payload structures. Instead of relying only on known signatures, AI models learn the behavioral characteristics of legitimate and malicious requests. Features such as query length, special character frequency, token patterns, and request entropy are used to identify suspicious inputs.

Machine learning classifiers like Random Forest, Support Vector Machines, and Neural Networks can effectively distinguish between normal traffic and malicious SQL or script injection attempts. Deep learning models further improve detection by understanding complex payload variations and

encoded attack patterns. This allows the system to detect zero-day SQLi and XSS attacks that are not present in existing rule sets.

In the proposed AI-powered Web Application Firewall, AI-based SQL Injection and XSS detection enhances security by providing real-time request inspection and adaptive threat response. Detected malicious requests are blocked immediately, logged for analysis, and used to improve the learning model, thereby strengthening long-term protection.

2.2.6 Comparative Study of Traditional WAF and AI-Based WAF

Traditional Web Application Firewalls (WAFs) primarily depend on static rule sets, signatures, and manually defined policies to detect and block web-based attacks. These systems are effective against known attack patterns such as basic SQL Injection and Cross-Site Scripting but struggle to detect new or evolving threats. Frequent manual updates and rule tuning are required to maintain effectiveness, which increases administrative overhead and response time.

In contrast, AI-based Web Application Firewalls utilize machine learning and deep learning techniques to analyze web traffic behavior dynamically. Instead of relying solely on predefined rules, AI-based WAFs learn from historical and real-time data to identify anomalies and malicious patterns. This enables the detection of zero-day attacks and previously unseen attack variations with higher accuracy.

Another key difference is adaptability. Traditional WAFs operate in a reactive manner, updating rules only after new threats are identified. AI-based WAFs continuously adapt by retraining models using newly observed traffic patterns and attack data. This results in improved detection rates and reduced false positives over time.

Overall, AI-based WAFs provide enhanced scalability, intelligence, and automation compared to traditional WAFs. While traditional WAFs remain useful for basic protection, AI-powered WAFs offer superior security for modern, complex web applications by combining behavioral analysis, real-time monitoring, and automated threat response.

2.2.7 Real-Time Traffic Monitoring Using Machine Learning

Real-time traffic monitoring plays a critical role in protecting web applications from cyber threats by continuously analyzing incoming and outgoing HTTP/HTTPS requests. Traditional monitoring techniques rely on predefined thresholds and rule-based alerts, which may fail to detect subtle or evolving attack patterns in high-volume traffic environments.

Machine learning–based traffic monitoring systems overcome these limitations by learning normal traffic behavior and identifying deviations that indicate potential attacks. Features such as request frequency, payload size, URL patterns, header information, and session behavior are extracted from live traffic streams and analyzed in real time. Supervised and unsupervised learning models classify traffic as benign or malicious with minimal latency.

These systems are capable of detecting various attacks such as Distributed Denial of Service (DDoS), brute-force login attempts, SQL injection, cross-site scripting, and bot-based attacks. By continuously updating models with new data, machine learning-based monitoring improves detection accuracy and reduces false positives.

Real-time machine learning traffic monitoring enables automated responses such as request blocking, rate limiting, IP blacklisting, and alert generation. This proactive defense mechanism significantly enhances the effectiveness of AI-powered Web Application Firewalls by providing immediate threat detection and mitigation without human intervention.

2.2.8 Security Analytics Using Prometheus and Grafana

Security analytics is an essential component of modern web application security systems, enabling continuous monitoring, visualization, and analysis of security-related metrics. Prometheus and Grafana are widely used open-source tools that provide powerful capabilities for collecting and visualizing real-time data in AI-powered security systems.

Prometheus is used for monitoring and time-series data collection. In an AI-based Web Application Firewall, Prometheus collects metrics such as request rates, blocked attacks, response times, model

inference latency, CPU and memory usage, and anomaly detection counts. These metrics help administrators understand system behavior and detect unusual activity patterns.

Grafana is integrated with Prometheus to provide interactive dashboards and visual analytics. It presents security metrics through graphs, charts, and alerts, allowing security teams to monitor real-time traffic trends and attack patterns visually. Grafana dashboards enable quick identification of spikes in malicious traffic, frequent attack sources, and system performance issues.

By combining Prometheus and Grafana, the AI-powered firewall gains enhanced observability and transparency. This integration supports data-driven security decisions, improves incident response time, and helps in fine-tuning machine learning models and firewall rules. Security analytics thus plays a vital role in maintaining a robust, scalable, and intelligent web application firewall system.

2.3 LITERATURE SURVEY SUMMARY

SI. No	Research	Technique	Features Used	Domain	Disadvantage / Advantage	Future Direction
1	Kim, S., & Park, J. (2019)	Rule-Based WAF	HTTP headers, URL patterns	Web Application Security	**Advantage:** Simple and fast detection. **Disadvantage:** Cannot detect zero-day attacks.	Integration with learning-based models.
2	Ahmed, M., & Khan, R. (2020)	Machine Learning (SVM)	Request frequency, payload length	Intrusion Detection	**Advantage:** Better anomaly detection. **Disadvantage:**	Adaptive feature selection techniques.

					High false positives with dynamic traffic.	
3	Li, Y., & Zhou, H. (2021)	Deep Learning (CNN)	HTTP payloads, request sequences	Web Attack Detection	**Advantage:** High detection accuracy for SQLi & XSS. **Disadvantage:** Requires large datasets.	Lightweight CNN models for real-time use.
4	Singh, A., & Verma, P. (2021)	Hybrid (ML + Rule Engine)	Signature rules, traffic features	Web Application Firewall	**Advantage:** Reduced false positives. **Disadvantage:** Complex system design.	Automation of rule generation using AI.
5	Chen, L., & Wu, T. (2022)	Anomaly Detection (Autoencoders)	Normal traffic behavior	Cybersecurity	**Advantage:** Detects unknown attacks. **Disadvantage:** Training instability.	Stable unsupervised learning models.

6	Rahman, S., & Islam, M. (2022)	IDS with ML	Network logs, request patterns	Network & Web Security	**Advantage:** Scalable detection. **Disadvantage:** Latency issues in real-time systems.	Optimization for low-latency environments.
7	Patel, N., & Shah, D. (2023)	ML + Real-Time Monitoring	Traffic metrics, attack logs	Web Traffic Analysis	**Advantage:** Real-time attack visibility. **Disadvantage:** Monitoring overhead.	Intelligent alert prioritization.
8	Lopez, J., & Martin, R. (2023)	AI + Prometheus & Grafana	Metrics, logs, dashboards	Security Analytics	**Advantage:** Improved visualization and response. **Disadvantage:** Requires proper metric tuning.	Automated anomaly-based alerting systems.

Table 2.3 Literature survey summary

Chapter 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing system for web application security primarily relies on **traditional Web Application Firewalls (WAFs)** that use **static, rule-based and signature-based detection mechanisms** to protect web applications from cyber threats. These systems operate by matching incoming HTTP/HTTPS requests against predefined rules and known attack signatures to identify malicious activities such as SQL Injection, Cross-Site Scripting (XSS), and brute-force attacks.

In traditional WAF systems, security rules are manually configured by administrators based on known attack patterns. Whenever a new type of attack emerges, the firewall rules must be updated manually to ensure continued protection. This approach makes the existing system highly dependent on **human expertise and constant maintenance**.

Most existing WAF solutions also lack **intelligent learning capabilities**, meaning they cannot adapt to evolving attack techniques or detect unknown (zero-day) attacks effectively. As a result, attackers can bypass these systems by modifying attack payloads slightly to avoid signature detection.

Furthermore, conventional systems often struggle to handle **large-scale web traffic** efficiently. As traffic volume increases, performance degradation may occur, leading to delayed responses and potential service disruptions. Logging and monitoring features in existing systems are also limited, providing minimal insights into attack patterns and system behavior.

Overall, the existing system provides only **reactive security**, which is insufficient to address modern, sophisticated cyber threats targeting web applications.

3.2 DISADVANTAGES OF EXISTING SYSTEM

The existing rule-based Web Application Firewall systems suffer from several limitations that reduce their effectiveness in modern web security environments. Some of the major disadvantages are listed below:

- **Inability to Detect New Attacks:** Traditional WAFs rely on predefined rules and signatures, making them ineffective against **zero-day and unknown attacks**.
- **High False Positives:** Legitimate user requests are often incorrectly blocked, which affects application usability and user experience.
- **Manual Rule Management:** Frequent manual updates of security rules are required, which is **time-consuming and error-prone**.
- **Lack of Adaptability:** Existing systems do not learn from traffic behavior and cannot adapt to evolving attack patterns.
- **Scalability Issues:** Performance degrades when handling **high traffic volumes**, leading to delays and reduced efficiency.
- **Limited Monitoring and Analytics:** Traditional systems provide insufficient insights into attack trends and system performance.

Due to these disadvantages, existing systems fail to provide comprehensive and intelligent protection for modern web applications.

3.3 PROPOSED SYSTEM

The proposed system is an **AI-Powered Advanced Web Application Firewall (WAF)** designed to overcome the limitations of traditional rule-based security systems. This system integrates **Artificial Intelligence and Machine Learning techniques** to provide intelligent, adaptive, and real-time protection for web applications.

Unlike conventional WAFs, the proposed system analyses web traffic behavior and patterns instead of relying solely on static rules. Incoming HTTP/HTTPS requests are monitored, pre-processed, and evaluated using trained **machine learning models** to classify traffic as legitimate or malicious. This enables the system to detect both **known and unknown attacks**, including zero-day threats.

The proposed system includes an automated **decision engine** that dynamically blocks or allows requests based on the model's predictions. It also provides **real-time monitoring, logging, and alert mechanisms**, allowing administrators to gain clear visibility into security events through an interactive dashboard.

The system is designed using a **scalable and modular architecture**, making it suitable for handling large volumes of web traffic efficiently. Containerized deployment ensures flexibility and ease of integration with existing web applications.

Overall, the proposed system delivers a **proactive, intelligent, and scalable web security solution** that significantly enhances protection against modern cyber threats.

3.4 ADVANTAGES OF PROPOSED SYSTEM

The proposed **AI-Powered Advanced Web Application Firewall** offers several advantages over traditional security systems. These advantages improve detection accuracy, system performance, and overall web application security.

- **Intelligent Threat Detection:** Uses AI and machine learning to identify both known and unknown attack patterns effectively.
- **Reduced False Positives:** Accurately distinguishes between legitimate and malicious traffic, improving user experience.
- **Real-Time Protection:** Detects and blocks malicious requests instantly without delay.

- **Adaptive Learning:** Continuously learns from new traffic data and evolving attack techniques.
- **Scalability:** Efficiently handles large volumes of web traffic without performance degradation.
- **Automated Response:** Automatically blocks threats and generates alerts without manual intervention.
- **Enhanced Monitoring:** Provides detailed logs, analytics, and dashboards for better visibility and control.

These advantages make the proposed system more reliable, efficient, and suitable for securing modern web applications.

3.5 FEASIBILITY STUDY

A feasibility study is conducted to analyze the practicality and viability of implementing the proposed **AI-Powered Advanced Web Application Firewall**. The feasibility of the system is evaluated under the following aspects:

Technical Feasibility

The proposed system is technically feasible as it utilizes widely adopted technologies such as Python, machine learning libraries, web frameworks, and containerization tools. The required hardware and software resources are readily available, and the system can be integrated with existing web applications without major architectural changes.

Operational Feasibility

The system is user-friendly and requires minimal manual intervention. Automated threat detection, blocking, and logging reduce the workload of system administrators. The dashboard interface allows

easy monitoring and management of security events, making the system operationally efficient.

Economic Feasibility

The proposed system is cost-effective as it leverages **open-source tools and frameworks**. There is no requirement for expensive proprietary software or hardware. This makes the solution affordable for small, medium, and large-scale organizations.

Scalability Feasibility

The system supports scalable deployment using container-based architecture, enabling it to handle increased web traffic and growing security demands without significant performance issues.

Overall, the feasibility study confirms that the proposed system is **practical, efficient, and economically viable** for real-world deployment.

3.6 HARDWARE ENVIRONMENT

The hardware environment defines the minimum system requirements needed to develop, deploy, and run the proposed **AI-Powered Advanced Web Application Firewall** efficiently. The system is designed to operate on standard computing hardware without requiring specialized infrastructure.

The following hardware configuration is recommended:

- **Processor:** Intel Core i5 or higher
- **RAM:** Minimum 8 GB (16 GB recommended for optimal performance)
- **Storage:** Minimum 20 GB free disk space
- **Network:** Stable internet connection for real-time traffic monitoring
- **Server:** Standard web server or cloud-based virtual machine

This hardware configuration is sufficient to support machine learning model execution, real-time traffic analysis, and dashboard visualization. The system can also be deployed on scalable cloud environments to handle increased traffic loads efficiently.

3.7 SOFTWARE ENVIRONMENT

The software environment specifies the tools, frameworks, and platforms required to develop and deploy the proposed **AI-Powered Advanced Web Application Firewall**. The system is built using reliable and open-source software technologies to ensure flexibility and scalability.

The following software components are used:

- **Operating System:** Windows / Linux
- **Programming Language:** Python
- **Web Framework:** Flask / FastAPI
- **Machine Learning Libraries:** TensorFlow, PyTorch, Scikit-learn
- **Database:** MongoDB, Redis
- **Web Server:** Nginx
- **Containerization:** Docker, Docker Compose
- **Monitoring & Logging:** ELK Stack, Grafana
- **Version Control:** Git

These software tools collectively enable efficient traffic analysis, intelligent threat detection, secure deployment, and real-time monitoring of web applications.

3.8 TECHNOLOGIES USED

The proposed **AI-Powered Advanced Web Application Firewall** is developed using modern and efficient technologies to ensure high performance, scalability, and security. The key technologies used in the system are listed below:

- **Python:** Core programming language used for implementing WAF logic and machine learning models.
- **Machine Learning:** Used to analyze web traffic patterns and detect malicious activities intelligently.
- **Flask / FastAPI:** Used to develop RESTful APIs and backend services.
- **TensorFlow / PyTorch:** Frameworks used for building and training AI models.
- **MongoDB & Redis:** Used for storing logs, session data, and cached information.
- **Nginx:** Acts as a reverse proxy and load balancer for incoming traffic.
- **Docker:** Enables containerized deployment and scalability.
- **ELK Stack & Grafana:** Used for logging, monitoring, and visualization of security events.
- **HTML, CSS, JavaScript:** Used for developing the admin dashboard interface.

These technologies collectively provide a robust foundation for implementing an intelligent and scalable web application firewall.

Chapter 4

SYSTEM DESIGN

4.1 ENTITY-RELATIONSHIP DIAGRAM (ERD)

The **Entity-Relationship Diagram (ERD)** represents the logical structure of the **AI-Powered Advanced Web Application Firewall (WAF)** database. It shows the relationships between different entities required for traffic monitoring, threat detection, logging, and dashboard management.

Key Entities and Attributes:

1. **User**
    - Attributes: User_ID, Name, Email, Role, Password
    - Description: Stores administrator or security personnel details for dashboard access.
2. **Web_Request**
    - Attributes: Request_ID, Source_IP, Destination_URL, Request_Type, Timestamp
    - Description: Logs each incoming web request for analysis.
3. **Threat**
    - Attributes: Threat_ID, Threat_Type, Severity, Detected_By, Timestamp
    - Description: Stores details of detected malicious activities.
4. **AI_Model**
    - Attributes: Model_ID, Model_Name, Version, Training_Date, Accuracy
    - Description: Maintains information about machine learning models used for threat detection.
5. **Alert**
    - Attributes: Alert_ID, Threat_ID, User_ID, Alert_Timestamp, Status

- Description: Keeps records of alerts generated and notified to administrators.

6. **Log**

   - Attributes: Log_ID, Request_ID, Threat_ID, Action_Taken, Timestamp
   - Description: Tracks actions taken on each request, including blocked, allowed, or challenged requests.

Relationships:

- A **User** can receive multiple **Alerts** (One-to-Many).
- Each **Web_Request** can be analyzed to generate zero or more **Threats** (One-to-Many).
- Each **Threat** can trigger multiple **Logs** and **Alerts** (One-to-Many).
- **AI_Model** is used to detect multiple **Threats** (One-to-Many).

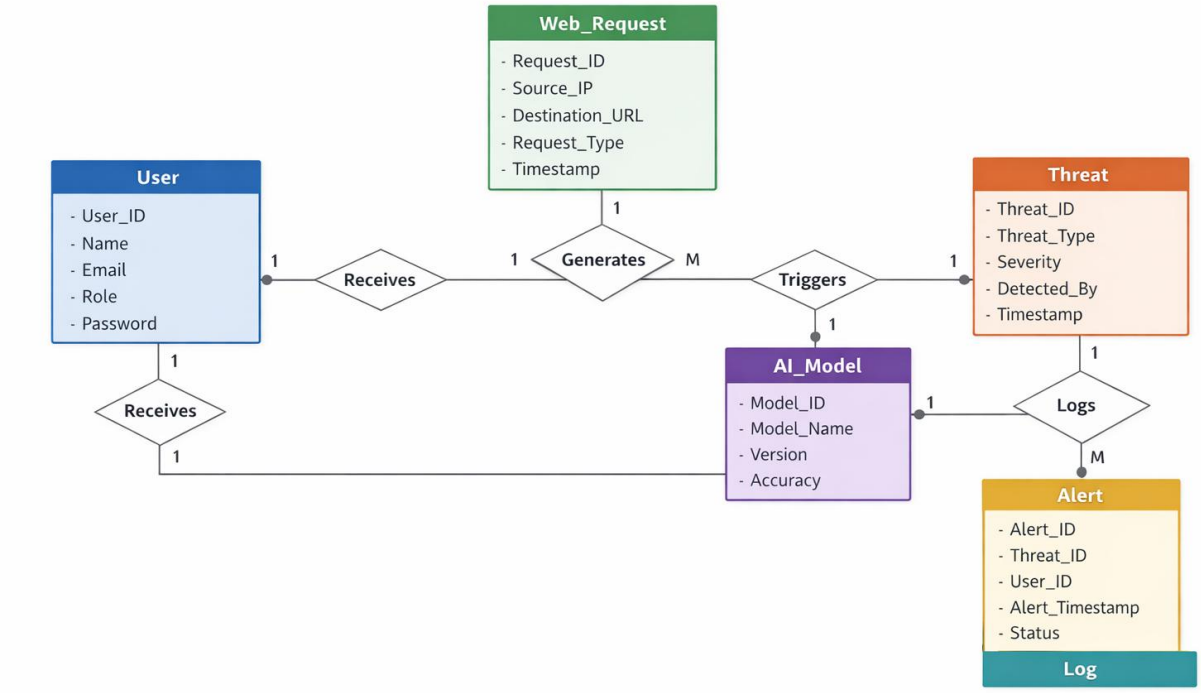


Fig 4.1 Entity-Relationship Diagram (ERD)

4.2 DATA FLOW DIAGRAM (DFD)

The **Data Flow Diagram (DFD)** represents the flow of data within the **AI-Powered Advanced Web Application Firewall (WAF)** system. It illustrates how data moves between different processes, external entities, and data storage components.

Level 0 DFD (Context Diagram)

The Level 0 DFD provides a high-level overview of the system:

- **External Entity:** Web Client
- **Process:** AI-Powered WAF
- **Data Stores:** Web Requests Log, Threat Database, Alerts Database
- **Flow:**
    - Client sends HTTP/HTTPS requests to the WAF.
    - The WAF analyzes requests and classifies them as safe or malicious.
    - Malicious requests trigger alerts and are logged; safe requests are forwarded to the web application.

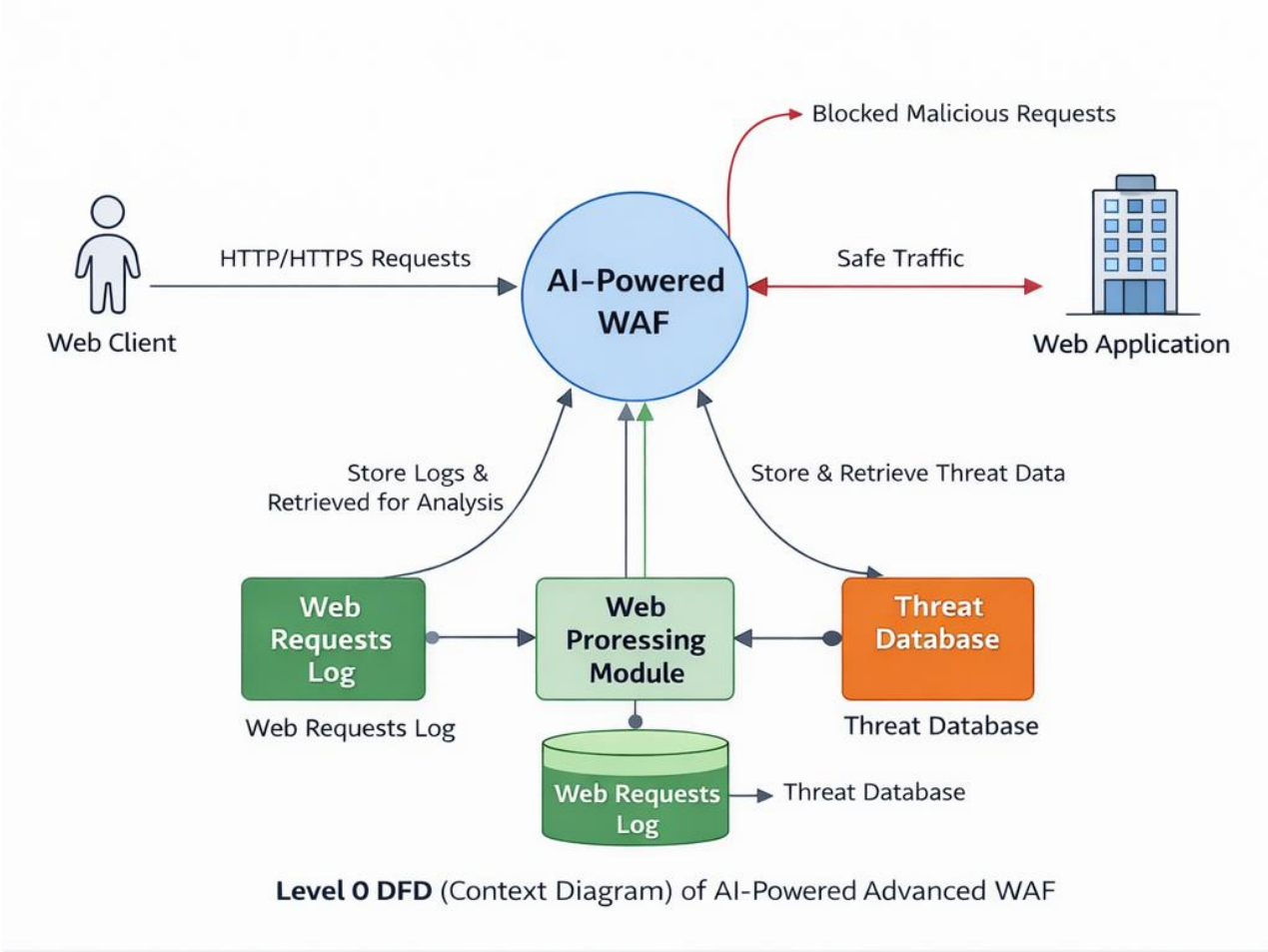


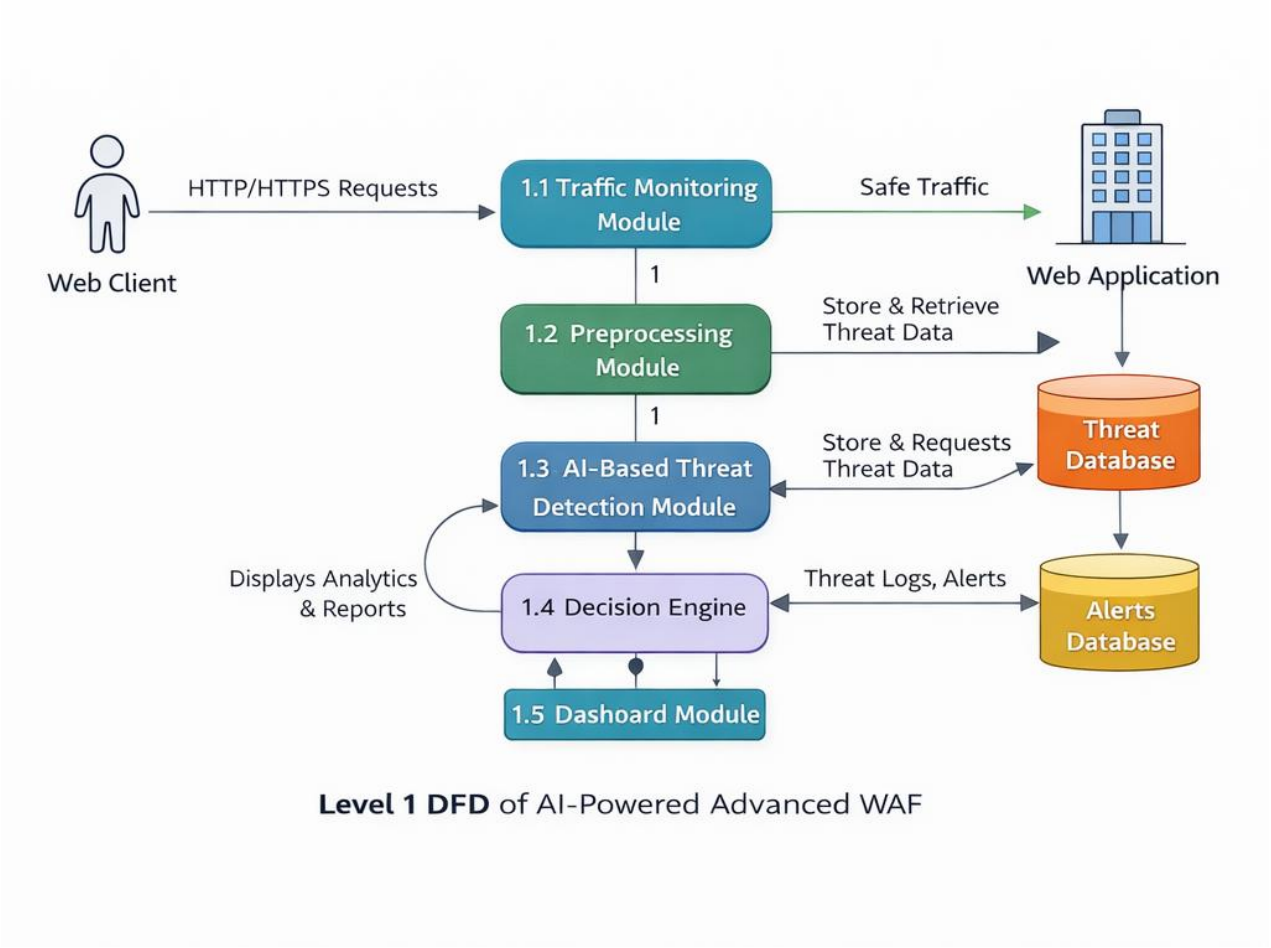
Fig 4.2.1 Data Flow Diagram

Level 1 DFD (Detailed Flow)

Level 1 DFD decomposes the system into key modules:

1. **Traffic Monitoring Module:**
    - Captures incoming requests and forwards them for preprocessing.
2. **Preprocessing Module:**
    - Cleans and extracts features from requests for analysis.
3. **AI-Based Threat Detection Module:**
    - Uses machine learning models to classify requests as legitimate or malicious.
4. **Decision Engine:**
    - Blocks malicious requests and forwards safe traffic to the web application.

5. **Logging and Alert Module:**
    - Stores details of all requests, detected threats, and generates alerts for administrators.
6. **Dashboard Module:**
    - Displays real-time system status, threat analytics, and traffic monitoring.



Level 1 DFD of AI-Powered Advanced WAF

Fig 4.2.2 Data Flow Diagram

4.3 UML DIAGRAMS

4.3.1 Use Case Diagram

The **Use Case Diagram** represents the interactions between users (administrators or security personnel) and the system modules.

Actors:

- Administrator
- Security Analyst

Use Cases:

- Monitor Web Traffic
- Detect Threats
- Block Malicious Requests
- Generate Alerts
- View Dashboard and Reports
- Manage AI Models

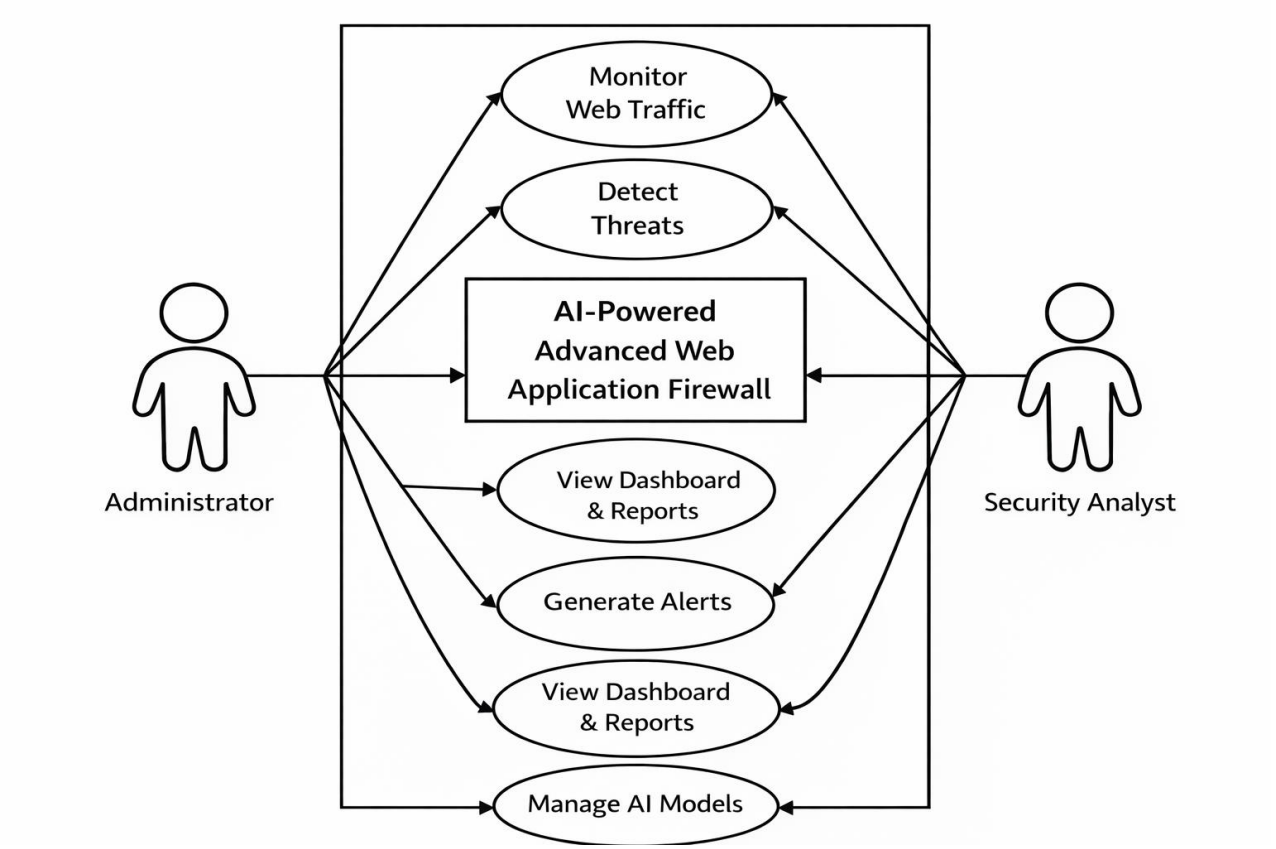


Fig 4.3.1 Use Case Diagram

4.3.2 Class Diagram

The **Class Diagram** shows the static structure of the system including classes, attributes, methods, and relationships.

Key Classes:

- **User:** userID, name, role, login(), logout()
- **WebRequest:** requestID, sourceIP, destinationURL, analyzeRequest()
- **Threat:** threatID, threatType, severity, classifyThreat()
- **AIModel:** modelID, version, trainModel(), predictThreat()
- **Alert:** alertID, threatID, sendAlert()
- **Log:** logID, requestID, recordAction()

Relationships:

- User generates Alerts (One-to-Many)
- WebRequest may generate Threats (One-to-Many)
- Threat is evaluated by AIModel (One-to-One)
- Log stores all actions taken on WebRequest and Threat

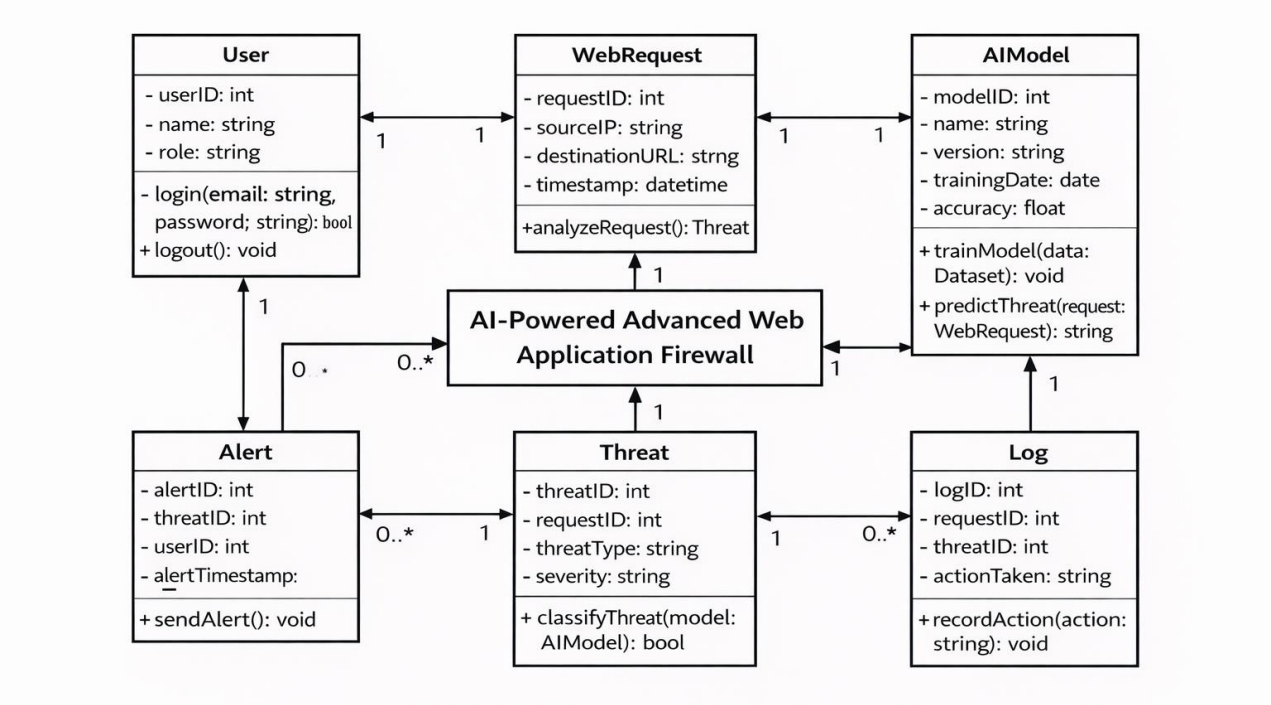


Fig 4.3.2 Class Diagram

4.3.3 Sequence Diagram

The **Sequence Diagram** represents the dynamic interaction between objects in chronological order during a typical request processing.

Flow:

1. Web Client sends HTTP/HTTPS request.
2. Traffic Monitoring Module receives the request.
3. Preprocessing Module extracts and normalizes features.
4. AI-Based Threat Detection Module evaluates the request.

5. Decision Engine allows or blocks the request.

6. Logging & Alert Module records the action and sends notifications.

7. Dashboard Module updates real-time analytics.

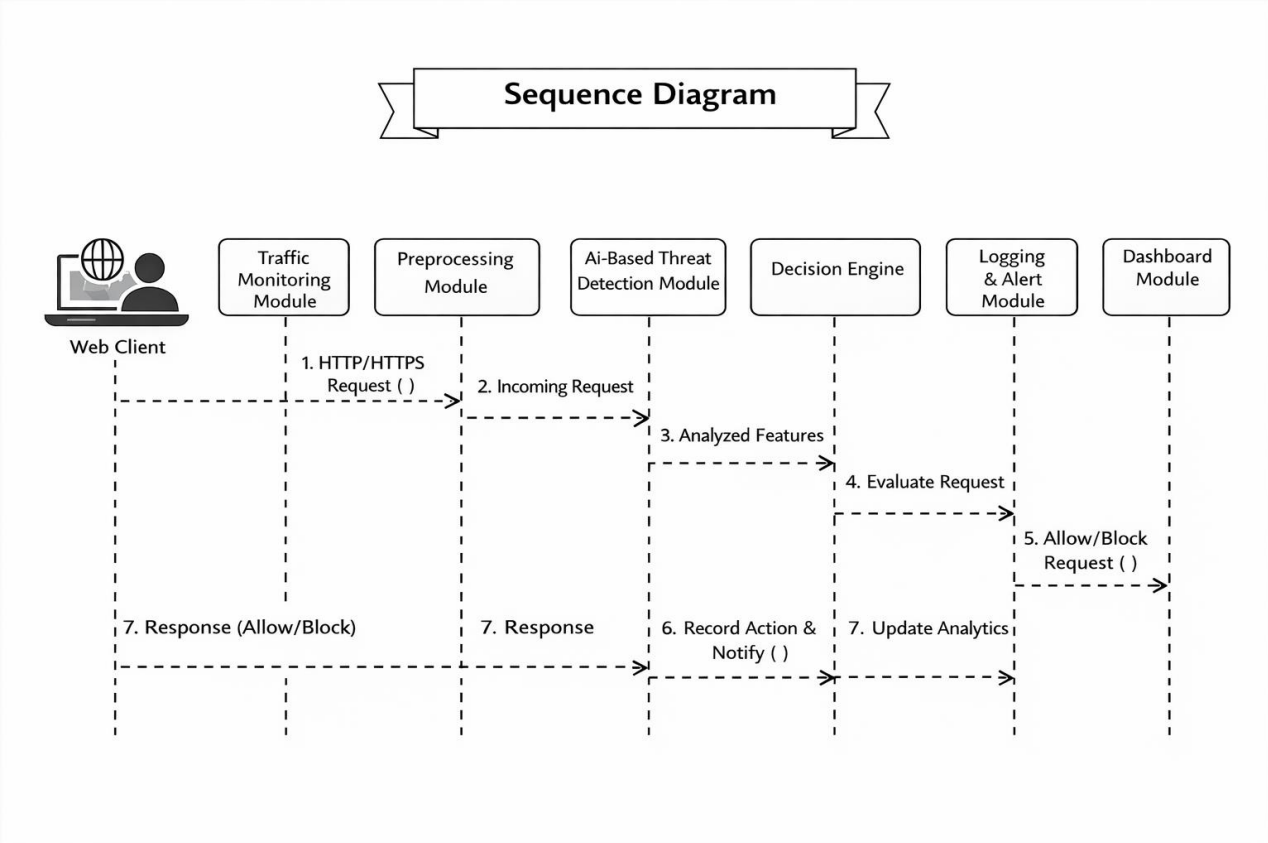


Fig 4.3.3 Sequence Diagram

Chapter 5

SYSTEM ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

The **AI-Powered Advanced Web Application Firewall system architecture** is designed using a **microservices approach** to ensure scalability, reliability, and high availability. The architecture consists of the following components:

1. **Load Balancer (Nginx)** – Distributes incoming client traffic across multiple WAF processing nodes for balanced performance.
2. **WAF Processing Nodes (Python/Flask)** – Inspect incoming HTTP requests and detect malicious activity.
3. **Machine Learning Engine (TensorFlow/PyTorch)** – Performs real-time threat detection using trained models.
4. **Database Layer (MongoDB + Redis)** – Stores persistent logs, session data, and caches critical information for high-speed access.
5. **Monitoring & Analytics (ELK Stack + Grafana)** – Provides real-time dashboards, alerts, and performance visualization.
6. **API Gateway (FastAPI)** – Facilitates communication between modules and external management interfaces.
7. **Container Orchestration (Docker Compose)** – Automates deployment and scaling of services in a cloud-native environment.

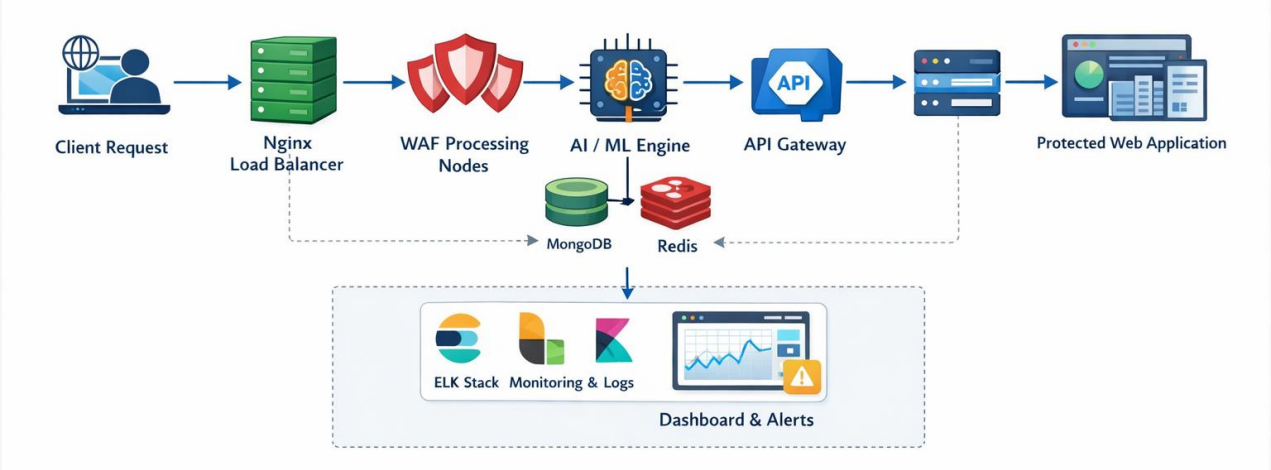


Fig 5.1 Architecture Diagram

5.2 ALGORITHMS

5.2.1 Machine Learning-Based Threat Detection Algorithm

The **AI-Powered Advanced Web Application Firewall** employs a **machine learning-based threat detection algorithm** to identify and prevent cyber-attacks such as **SQL Injection (SQLi), Cross-Site Scripting (XSS), and brute-force attacks** in real-time. This algorithm is designed to automatically adapt to new threats and provide proactive protection.

Algorithm Description

1. **Request Capture:** All incoming HTTP requests are captured by the WAF nodes.
2. **Data Preprocessing:** Each request is normalized and tokenized. Features such as request headers, parameters, payload content, and metadata are extracted and encoded for ML model input.
3. **Threat Prediction:** Preprocessed data is fed into the trained **ML model** (using TensorFlow or PyTorch). The model predicts a **threat probability score** for each request.
4. **Decision Making:**
    - If the threat probability exceeds a predefined threshold, the request is **blocked or challenged** automatically.
    - If the threat probability is low, the request is **allowed** to proceed.
5. **Logging and Alerting:** Every decision (allow/block) is logged in the **database layer**, and

critical alerts are sent to the monitoring dashboard.

6. **Continuous Learning:** The algorithm periodically updates the ML model using new attack data to improve detection accuracy and handle emerging threats.

Algorithm Steps – Tabular Representation

Step	Description
1	Capture incoming HTTP request
2	Preprocess and extract features from the request
3	Pass features to the trained ML model
4	Compute threat probability score
5	Decision engine: block/challenge if score exceeds threshold
6	Log all events and update dashboard
7	Update ML model with new threat data periodically

Table 5.2.1 – Machine Learning Algorithm Description

5.3 WORKFLOW OF THE SYSTEM

The workflow of **AI-Powered Advanced Web Application Firewall** can be summarized as:

1. Client sends HTTP request to the system.
2. Load balancer distributes traffic to available WAF nodes.
3. WAF node inspects request headers, payload, and patterns.
4. ML engine evaluates threat probability in real-time.

5. Decision engine determines **allow**, **block**, or **challenge**.
6. Event logged in the database for analytics and auditing.
7. Dashboard updates provide visual representation of traffic and threat patterns.

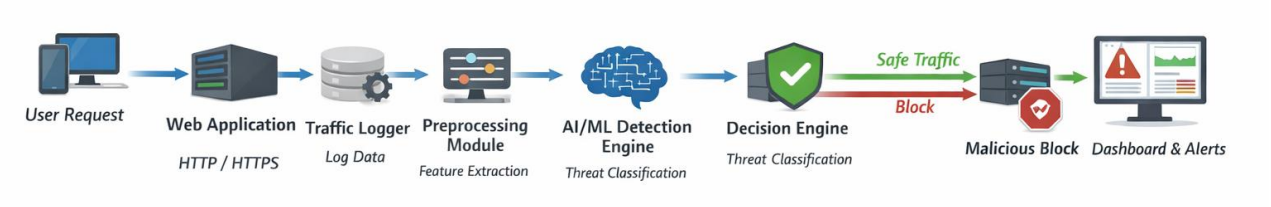


Figure 5.3 – System Workflow Diagram

5.4 MODULES DESCRIPTION

5.4.1 Traffic Monitoring Module

- **Purpose:** Continuously monitors incoming and outgoing web traffic to the application.
- **Functionality:**
    - Captures request headers, payloads, and session data in real-time.
    - Detects abnormal patterns using baseline traffic profiles.
    - Sends captured data to the AI-based threat detection module for analysis.
- **Technologies Used:** Python, Flask, Socket Programming.

5.4.2 AI-Based Threat Detection Module

- **Purpose:** Detects potential cyber attacks using advanced AI and machine learning algorithms.
- **Functionality:**
    - Analyzes traffic patterns for SQL injection, XSS, CSRF, and other OWASP top 10 threats.
    - Employs YOLOv7 for pattern recognition in request payloads where

applicable.

- Generates threat probability scores and sends results to the rule engine.

- **Technologies Used:** TensorFlow, PyTorch, Scikit-learn.

5.4.3 Rule Engine and Response Module

- **Purpose:** Makes real-time decisions on whether to allow, block, or challenge incoming requests.
- **Functionality:**
  - Applies pre-defined rules combined with AI predictions for threat mitigation.
  - Executes automated actions such as blocking IPs, rate-limiting requests, or sending CAPTCHA challenges.
  - Updates firewall rules dynamically based on new threat patterns.
- **Technologies Used:** Python, FastAPI, Nginx.

5.4.4 Logging and Monitoring Module

- **Purpose:** Provides detailed audit trails and monitoring for security analysis.
- **Functionality:**
  - Logs all incoming requests, decisions, and alerts.
  - Integrates with ELK Stack (Elasticsearch, Logstash, Kibana) for real-time visualization and analysis.
  - Sends alerts for suspicious activities or critical attacks to administrators.
- **Technologies Used:** ELK Stack, Grafana, MongoDB.

5.4.5 Dashboard and Visualization Module

- **Purpose:** Offers a user-friendly interface to visualize traffic, threats, and system performance.

- **Functionality:**
    - Displays real-time statistics on detected threats, blocked requests, and system load.
    - Provides graphs, charts, and heatmaps for traffic and threat analysis.
    - Allows administrators to configure system settings and view historical logs.
- **Technologies Used:** HTML, CSS, JavaScript, ReactJS.

5.4.6 Database and Storage Module

- **Purpose:** Ensures persistent storage of traffic data, threat logs, and system configurations.
- **Functionality:**
    - Stores request and response logs, threat metadata, and AI model outputs.
    - Caches frequently accessed data for faster processing using Redis.
    - Supports high availability and horizontal scaling for large-scale deployments.
- **Technologies Used:** MongoDB, Redis, Docker Volumes.

Chapter 6

SYSTEM IMPLEMENTATION

6.1 DATA COLLECTION AND PREPROCESSING

The data collection and preprocessing module for the **AI-Powered-Advanced-Web-Application-Firewall** is a crucial phase in developing an accurate and efficient web security system. This module ensures that the incoming web traffic data is clean, well-structured, and properly prepared before it is processed by the AI-based threat detection models. The primary steps involved in this process are data collection, data labeling, and data preprocessing.

1. Data Collection

The initial step in building the AI-powered firewall system involves collecting large volumes of web traffic data from multiple sources. The data is gathered from web server access logs, HTTP request streams, and publicly available cybersecurity datasets. These datasets contain both legitimate user traffic and malicious traffic such as SQL Injection, Cross-Site Scripting (XSS), brute-force login attempts, and other OWASP Top 10 web attacks.

To ensure robustness and real-world applicability, the collected data includes diverse request patterns, payload sizes, IP addresses, request frequencies, and session behaviors. This diversity enables the firewall to accurately differentiate between normal and malicious web requests under various operating conditions.

2. Data Labeling

Data labeling is an essential step for training supervised machine learning models used in the firewall. Each collected traffic record is labeled as either **normal** or **malicious** based on known attack signatures and predefined security rules. Malicious traffic samples are further classified into specific attack categories such as SQL Injection, XSS, and brute-force attacks.

Accurate labeling helps the AI models learn the distinguishing features of different attack patterns, improving detection accuracy while minimizing false positives and false negatives.

3. Data Preprocessing

Once the data is labeled, preprocessing is performed to enhance data quality and ensure compatibility

with machine learning algorithms. The key preprocessing steps include:

- **Data Cleaning:** Removal of duplicate entries, incomplete records, and irrelevant data to eliminate noise.
- **Feature Extraction:** Extraction of important features such as request length, special character frequency, payload patterns, request rate, and header information.
- **Normalization:** Scaling numerical features to a standard range to stabilize and optimize model training.
- **Encoding:** Conversion of categorical data such as HTTP methods and content types into numerical form.
- **Data Splitting:** Dividing the dataset into training, validation, and testing sets for effective performance evaluation.

The final output of this module is a clean, normalized, and labeled dataset that is ready for training the AI-based threat detection models. The effectiveness of the **AI-Powered-Advanced-Web-Application-Firewall** largely depends on the quality of data collection and preprocessing performed in this stage.

6.2 MODEL TRAINING

The model training module for the **AI-Powered-Advanced-Web-Application-Firewall** is a crucial phase in developing an intelligent and accurate web security system. This module focuses on training machine learning and deep learning models using the preprocessed web traffic data to effectively detect and classify malicious activities.

1. Machine Learning Model Structure

The system employs a combination of machine learning and deep learning models designed for web traffic analysis. These models are structured to analyze multiple features extracted from HTTP requests, such as payload content, request frequency, and behavioral patterns. The model architecture is optimized to identify known attack signatures as well as previously unseen attack patterns.

2. Training Process

The training process involves feeding the preprocessed and labeled web traffic data into the machine

learning models. During training, the models learn to distinguish between normal and malicious requests by adjusting internal parameters to minimize prediction errors. The key steps involved in the training process are:

- **Initialization:**
  Model parameters are initialized randomly or using pre-trained weights when available.

- **Batch Training:**
  The training data is divided into smaller batches to improve training efficiency and memory usage.

- **Loss Calculation:**
  The loss function calculates the difference between the predicted output and the actual class labels (normal or malicious).

- **Weight Update:**
  Model weights are updated using optimization algorithms such as Adam or Stochastic Gradient Descent (SGD).

- **Validation:**
  After each training epoch, the model is evaluated on a validation dataset to ensure proper generalization and avoid overfitting.

3. Hyperparameter Tuning

To achieve optimal detection accuracy, hyperparameters such as learning rate, batch size, number of epochs, and threshold values are carefully tuned. Techniques such as early stopping and cross-validation are applied to prevent overfitting and improve model robustness.

4. Transfer Learning

In advanced configurations, the system utilizes transfer learning by initializing the model with pre-trained weights derived from previously trained cybersecurity datasets. This approach reduces training time and enhances performance, especially when limited training data is available.

6.3 PREDICTION OF OUTPUT

The prediction of output module is the final stage of the **AI-Powered-Advanced-Web-Application-Firewall** system. This module uses the trained machine learning models to analyze incoming web traffic in real time and determine whether a request is legitimate or malicious. The system supports continuous prediction to ensure real-time protection of web applications.

1. Input Data

The input to this module is an incoming HTTP request received by the web application firewall. The request contains essential information such as IP address, request method, URL, headers, payload, and session data. Before prediction, the request undergoes initial preprocessing including feature extraction, normalization, and encoding to match the format used during model training.

2. Prediction Using Trained Model

The trained AI model processes the preprocessed request data to generate a prediction result. The prediction process includes the following steps:

- **Model Inference:**
  The extracted features from the HTTP request are passed to the trained machine learning model, which computes the probability of the request being malicious.
- **Thresholding:**
  The predicted probability score is compared against a predefined threshold to classify the request as **normal** or **malicious**.
- **Post-Processing:**
  Additional rule-based validation is applied to refine the prediction and reduce false positives by combining AI outputs with predefined firewall rules.

3. Visualization of Results

The prediction results are displayed through an administrative dashboard. The dashboard provides real-time visualization of detected threats, including attack type, source IP address, timestamp, and action taken (allowed or blocked). Graphs and charts are used to show traffic trends, attack frequency, and system performance.

4. Report Generation

After prediction, the system generates a detailed security report for administrators. The report includes:

- **Request Details:** Information about the incoming HTTP request.
- **Detection Result:** Classification of the request as normal or malicious.
- **Attack Type:** Identified attack category such as SQL Injection, XSS, or brute-force attack.
- **Action Taken:** Firewall response including allow, block, or rate-limit.

- **Timestamp and Logs:** Time of detection and corresponding log references.

The prediction output can be monitored in real time through the dashboard or accessed later through stored logs for analysis and auditing purposes.

Chapter 7

SYSTEM TESTING

7.1 BLACK BOX TESTING

Black Box Testing is performed to evaluate the functionality of the system without considering its internal implementation details. The focus is on verifying whether the system meets the specified functional requirements and correctly handles different types of web traffic.

In this project, Black Box Testing is conducted by providing various types of HTTP requests as input and observing the system's output. The testing includes both legitimate user requests and malicious attack attempts such as SQL Injection, Cross-Site Scripting (XSS), and brute-force attacks.

Objectives of Black Box Testing:

- To verify accurate detection of malicious web requests
- To ensure legitimate traffic is allowed without disruption
- To validate correct firewall responses such as allow, block, or rate-limit
- To confirm real-time system behavior and alert generation

Test Scenarios:

- Normal user requests accessing the web application
- SQL Injection attempts in URL parameters
- XSS scripts embedded in request payloads
- High-frequency requests simulating brute-force attacks

The system is considered successful if malicious traffic is blocked and logged correctly while legitimate requests are allowed without delay.

Fig 7.1 Black box testing

7.2 WHITE BOX TESTING

White Box Testing focuses on verifying the internal logic, code structure, and data flow of the **AI-Powered-Advanced-Web-Application-Firewall**. This testing ensures that all internal components, algorithms, and decision-making processes function as intended.

White Box Testing is conducted by examining the source code, control flow, and interaction between

system modules such as traffic monitoring, AI-based threat detection, rule engine, and logging mechanisms.

Objectives of White Box Testing:

- To verify correct implementation of machine learning algorithms
- To ensure accurate data flow between system modules
- To validate decision logic used in threat classification
- To check error handling and exception management

Testing Techniques Used:

- **Unit Testing:** Individual modules such as traffic analysis and prediction logic are tested independently.
- **Integration Testing:** Interaction between AI models and firewall rule engine is validated.
- **Code Coverage Analysis:** Ensures that all critical paths and conditions are tested.

White Box Testing helps identify logical errors, security vulnerabilities, and performance bottlenecks, thereby improving the reliability and robustness of the system.

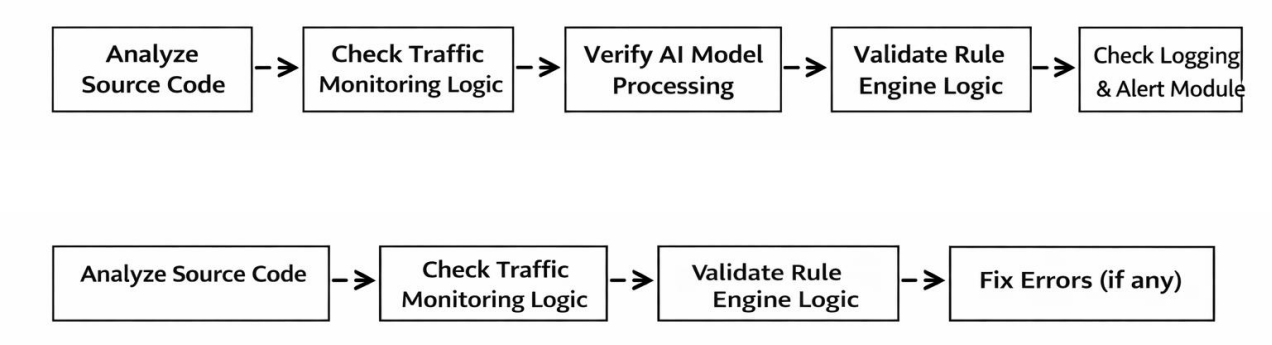


Fig 7.2 White box testing

7.3 TEST CASES

Testing is a crucial step to ensure the performance, security, and accuracy of the **AI-Powered Advanced Web Application Firewall (WAF)**. The following test cases are designed to validate core functionalities such as HTTP request handling, threat detection, decision-making, and logging/report generation.

TEST REPORT: 01

PRODUCT: AI-Powered Advanced Web Application Firewall
USE CASE: HTTP Request Handling

TEST CASE ID	TEST CASE / ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS / FAIL
1	Send a valid HTTP request to the web application	Request should be allowed and forwarded to the server	As Expected	PASS
2	Send a malformed HTTP request	System should detect abnormal structure and block the request	As Expected	PASS
3	Send a request with missing headers	System should log the request and apply security rules	As Expected	PASS

Table 7.3.1: **Test Cases for HTTP Request Handling**

TEST REPORT: 02

PRODUCT: AI-Powered Advanced Web Application Firewall
USE CASE: Traffic Monitoring and Preprocessing

TEST CASE ID	TEST CASE / ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS / FAIL
1	Monitor incoming web traffic in real time	Traffic should be captured and analyzed successfully	As Expected	PASS
2	Extract features from HTTP requests	Relevant features should be correctly extracted	As Expected	PASS
3	Normalize and preprocess traffic data	Data should be formatted correctly for AI model input	As Expected	PASS

Table 7.3.2: Test Cases for Traffic Monitoring and Preprocessing

TEST REPORT: 03

PRODUCT: AI-Powered Advanced Web Application Firewall

USE CASE: AI-Based Threat Detection

TEST CASE ID	TEST CASE / ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS / FAIL
1	Send SQL Injection attack payload	System should detect and classify it as malicious	As Expected	PASS
2	Send XSS attack payload	Request should be blocked by the AI model	As Expected	PASS
3	Send normal user traffic	Traffic should be classified as benign	As Expected	PASS

Table 7.3.3: Test Cases for AI-Based Threat Detection

TEST REPORT: 04

PRODUCT: AI-Powered Advanced Web Application Firewall

USE CASE: Decision Engine and Logging

TEST CASE ID	TEST CASE / ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS / FAIL
1	Allow a benign request	Request should be forwarded to the application	As Expected	PASS
2	Block a malicious request	Request should be denied and logged	As Expected	PASS
3	Generate security logs and alerts	Logs and alerts should be stored and displayed correctly	As Expected	PASS

Table 7.3.4: Test Cases for Decision Engine and Logging

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

In conclusion, the development of the **AI-Powered Advanced Web Application Firewall (WAF)** represents a significant advancement in cybersecurity for web applications. The system integrates artificial intelligence and machine learning techniques to detect, prevent, and respond to various web-based attacks, including SQL injection, cross-site scripting (XSS), and brute force attacks. By leveraging real-time traffic analysis and intelligent threat detection algorithms, the system provides accurate and rapid identification of malicious activities, thereby protecting sensitive web data and ensuring secure user interactions.

The primary objective of this project was to enhance web application security by combining traditional WAF rules with AI-driven anomaly detection. This hybrid approach allows the system to adapt to emerging threats while maintaining high efficiency and low false-positive rates. The implementation of machine learning-based models ensures that the firewall continuously learns from incoming traffic patterns, improving its detection capability over time.

The system also features a user-friendly dashboard for monitoring traffic, alerts, and blocked attacks, providing administrators with actionable insights and comprehensive security reports. Rigorous testing using black-box and white-box methodologies demonstrates the system's reliability, robustness, and ability to operate effectively under diverse attack scenarios.

Overall, this project addresses the growing need for intelligent web security solutions. By integrating AI with a traditional firewall architecture, the system not only automates threat detection but also significantly reduces the manual effort required for monitoring and managing web security, thereby enabling safer and more resilient web applications.

8.2 FUTURE ENHANCEMENTS

The AI-Powered Advanced WAF lays a strong foundation for future improvements and expansion. Potential enhancements include:

1. **Integration with Cloud-Based Security Services**
    - Deploying the system on cloud platforms can allow real-time monitoring and protection for multiple web applications simultaneously.
    - **Potential Benefit:** Scalable protection for large enterprises and remote servers.
2. **Advanced Deep Learning Models**
    - Incorporate models such as LSTM, Transformer-based anomaly detectors, or ensemble learning methods to improve detection of complex attack patterns.
    - **Potential Benefit:** Increased accuracy and early detection of zero-day attacks.
3. **Multi-Layer Threat Analysis**
    - Extend monitoring beyond HTTP traffic to include network-level analysis, API requests, and database queries.
    - **Potential Benefit:** Comprehensive protection against sophisticated, multi-layer attacks.
4. **Automatic Rule Generation and Adaptation**
    - The system could automatically generate or update firewall rules based on detected threats and traffic patterns.
    - **Potential Benefit:** Reduced manual configuration and adaptive defense mechanism.
5. **Integration with SIEM and Analytics Tools**
    - Connect with Security Information and Event Management (SIEM) systems and visualization platforms like Grafana or Kibana for real-time analytics.
    - **Potential Benefit:** Enhanced monitoring and actionable insights for security teams.
6. **Mobile and Remote Dashboard Access**
    - Develop mobile applications to allow administrators to monitor and respond to alerts from anywhere.
    - **Potential Benefit:** Increased accessibility and faster response times.
7. **Explainable AI for Security Decisions**
    - Provide interpretability of AI-based threat detection to understand why certain traffic is flagged as malicious.
    - **Potential Benefit:** Increased trust and accountability in automated threat detection.
8. **Continuous Learning from Global Threat Intelligence**
    - Incorporate threat intelligence feeds to keep the WAF updated with the latest attack

patterns globally.

- **Potential Benefit:** Proactive defense against emerging cybersecurity threats.

9. **Support for Multi-Protocol Protection**
    - Expand beyond web traffic to include protection for APIs, IoT devices, and microservices environments.
    - **Potential Benefit:** Wider applicability in modern web and cloud architectures.
10. **Real-Time Feedback and Self-Healing**
    - Enable the system to automatically isolate suspicious traffic and adapt rules dynamically based on administrator feedback.
    - **Potential Benefit:** Continuous improvement of detection efficiency and reduced impact of attacks on live systems.

Chapter 9

APPENDIX 1 – SAMPLE CODING

9.1 TRAFFIC COLLECTION AND PREPROCESSING

```
import asyncio, httpx

from datetime import datetime

from dataclasses import dataclass



@dataclass

class HttpRequest:

    timestamp: datetime

    method: str

    url: str

    headers: dict

    body: str

    source_ip: str

    user_agent: str

    content_length: int

    node_id: str = "nginx-node-1"



    def to_dict(self):

        return {
```

```
            "timestamp": self.timestamp.isoformat(),
            "method": self.method,
            "url": self.url,
            "source_ip": self.source_ip,
            "user_agent": self.user_agent,
            "node_id": self.node_id
        }


class TrafficCollector:
    def __init__(self, nodes):
        self.nodes = nodes
        self.collected_requests = []

    async def collect(self):
        async with httpx.AsyncClient() as client:
            for node in self.nodes:
                try:
                    r = await client.get(f"{node}/api/logs", timeout=5)
                    for log in r.json().get("logs", []):
                        self.collected_requests.append(HttpRequest(
                            timestamp=datetime.now(),
                            method=log.get("method", "GET"),
```

```
                    url=log.get("url", ""),
                    headers=log.get("headers", {}),
                    body=log.get("body", ""),
                    source_ip=log.get("source_ip", ""),
                    user_agent=log.get("user_agent", ""),
                    content_length=log.get("content_length", 0),
                ))
            except: pass

    def recent_requests(self, limit=100):
        return [r.to_dict() for r in self.collected_requests[-limit:]]
```

9.2 MACHINE LEARNING MODEL CODE

```
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
import pandas as pd

class MLEngine:
    def __init__(self):
        self.anomaly_detector = IsolationForest()
        self.threat_classifier = RandomForestClassifier()
        self.scaler = StandardScaler()
        self.encoder = LabelEncoder()
        self.is_trained = False

    def extract_features(self, requests):
        df = pd.DataFrame(requests)
        return df.fillna(0)
```

```
    def train(self, requests, labels):
        X = self.extract_features(requests)
        y = self.encoder.fit_transform(labels)
        self.anomaly_detector.fit(X)
        self.threat_classifier.fit(X, y)
        self.is_trained = True

    def predict(self, requests):
        X = self.extract_features(requests)
        scores = self.anomaly_detector.decision_function(X)
        classes = self.threat_classifier.predict(X)
        return [{"score": float(s), "class": int(c)} for s, c in zip(scores, classes)]
```

9.3 API and Rule Engine Code

```
from fastapi import FastAPI

from traffic_collector import TrafficCollector

from ml_engine import MLEngine

import asyncio



app = FastAPI()

collector = TrafficCollector(nodes=["http://localhost:8080"])

ml = MLEngine()



@app.get("/api/logs")

async def get_logs(limit: int = 50):

    return collector.recent_requests(limit)
```

```
@app.get("/api/predict")
async def predict(limit: int = 50):
    requests = collector.recent_requests(limit)
    return ml.predict(requests)

@app.on_event("startup")
async def startup():
    asyncio.create_task(collector.collect())
```

9.4 Dashboard Integration Code

```
from fastapi import FastAPI, WebSocket
from traffic_collector import TrafficCollector
from ml_engine import MLEngine
import asyncio

app = FastAPI()
collector = TrafficCollector(nodes=["http://localhost:8080"])
ml = MLEngine()

@app.on_event("startup")
async def startup():
    asyncio.create_task(collector.collect())
```

```
@app.get("/api/logs")
async def get_logs(limit: int = 50):
    return collector.recent_requests(limit)


@app.get("/api/predict")
async def predict(limit: int = 50):
    requests = collector.recent_requests(limit)
    return ml.predict(requests)


# WebSocket for live dashboard updates
@app.websocket("/ws/dashboard")
async def dashboard_ws(websocket: WebSocket):
    await websocket.accept()
    while True:
        logs = collector.recent_requests(50)
        predictions = ml.predict(logs)
        data = {"logs": logs, "predictions": predictions}
        await websocket.send_json(data)
        await asyncio.sleep(5)  # Update every 5 seconds
```

9.5 EXAMPLE HTML CONTROL PANEL

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>WAF AI Control Panel</title>
<style>
body { font-family: Arial; margin: 20px; background: #f5f6fa; }
.card { background: white; padding: 20px; border-radius: 8px; margin-bottom: 20px; box-shadow:
0 2px 5px rgba(0,0,0,0.1);}
button { padding: 8px 12px; margin: 5px; border: none; border-radius: 4px; cursor: pointer;}
.green { background: #28a745; color: white;}
.red { background: #dc3545; color: white;}
.orange { background: #fd7e14; color: white;}
#logs { background: #222; color: #0f0; padding: 10px; height: 150px; overflow-y: scroll; font-
family: monospace;}
</style>
</head>
<body>

<h1>WAF AI Control Panel</h1>

<div class="card">
  <h2>Traffic Control</h2>
  <button class="green" onclick="startCollection()">Start Collection</button>
  <button class="red" onclick="stopCollection()">Stop Collection</button>
  <p>Requests Collected: <span id="req_count">0</span></p>
</div>

<div class="card">
  <h2>ML Engine</h2>
  <button class="green" onclick="trainModel()">Train Model</button>
  <p>Threats Detected: <span id="threat_count">0</span></p>
```

```
</div>

<div class="card">
  <h2>System Logs</h2>
  <div id="logs"></div>
</div>

<script>
let ws = new WebSocket("ws://localhost:8000/ws/dashboard");

ws.onmessage = function(event) {
  let data = JSON.parse(event.data);
  document.getElementById("req_count").innerText = data.logs.length;
  document.getElementById("threat_count").innerText = data.predictions.filter(p => p.class !==
0).length;

  let logDiv = document.getElementById("logs");
  logDiv.innerHTML = "";
  data.logs.forEach(log => {
    logDiv.innerHTML += `[${log.timestamp}] ${log.method} ${log.url} - ${log.source_ip}<br>`;
  });
};

function startCollection() { console.log("Traffic collection started."); }
function stopCollection() { console.log("Traffic collection stopped."); }
function trainModel() { console.log("Model training started."); }
</script>

</body>
</html>
```

Chapter 10

APPENDIX 2 – SAMPLE OUTPUT

10.1 DASHBOARD HOME PAGE

The **Dashboard Home Page** of the AI-Powered Advanced Web Application Firewall (CyberGuard AI) serves as the central hub for administrators to monitor the overall security and performance of web applications. The dashboard is designed to provide **real-time insights into web traffic, threat detection, system health, and ML engine status**, ensuring proactive management of cybersecurity threats.

Detailed Features and Functionality:

1. **Traffic Overview:**
    - Displays the total number of requests, both legitimate and malicious, processed by the WAF in real-time.
    - Highlights trends in traffic, helping administrators quickly identify unusual spikes that may indicate an ongoing attack.
2. **Threat Summary:**
    - Shows the number and type of detected threats (SQL Injection, XSS, Brute Force, etc.).
    - Categorizes threats by severity (Low, Medium, High) for easy prioritization.
3. **System Health Monitoring:**
    - Provides the status of WAF nodes, ML engine performance, database connectivity, and load balancer health.
    - Alerts administrators if any component is down or underperforming.
4. **Interactive Controls:**
    - Allows filtering logs, searching for specific events, setting alert thresholds, and managing WAF policies.
    - Provides direct access to detailed analytics pages for deeper investigation.
5. **Visualization:**
    - Uses charts, tables, and color-coded indicators to provide an intuitive understanding of security events.
    - Offers a unified view to ensure administrators can make decisions without switching between multiple tools.

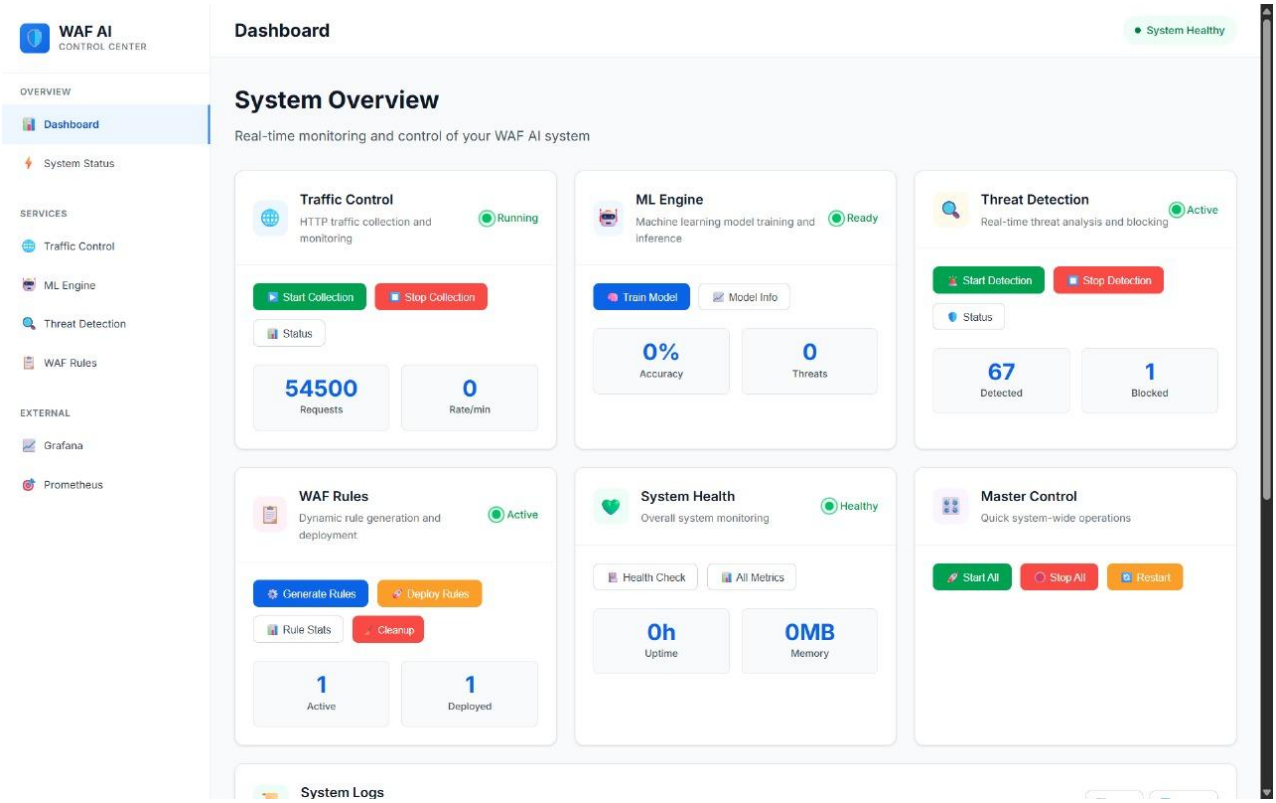


Fig 10.1.1 Waf AI Control Panel Dashboard - 1 Half

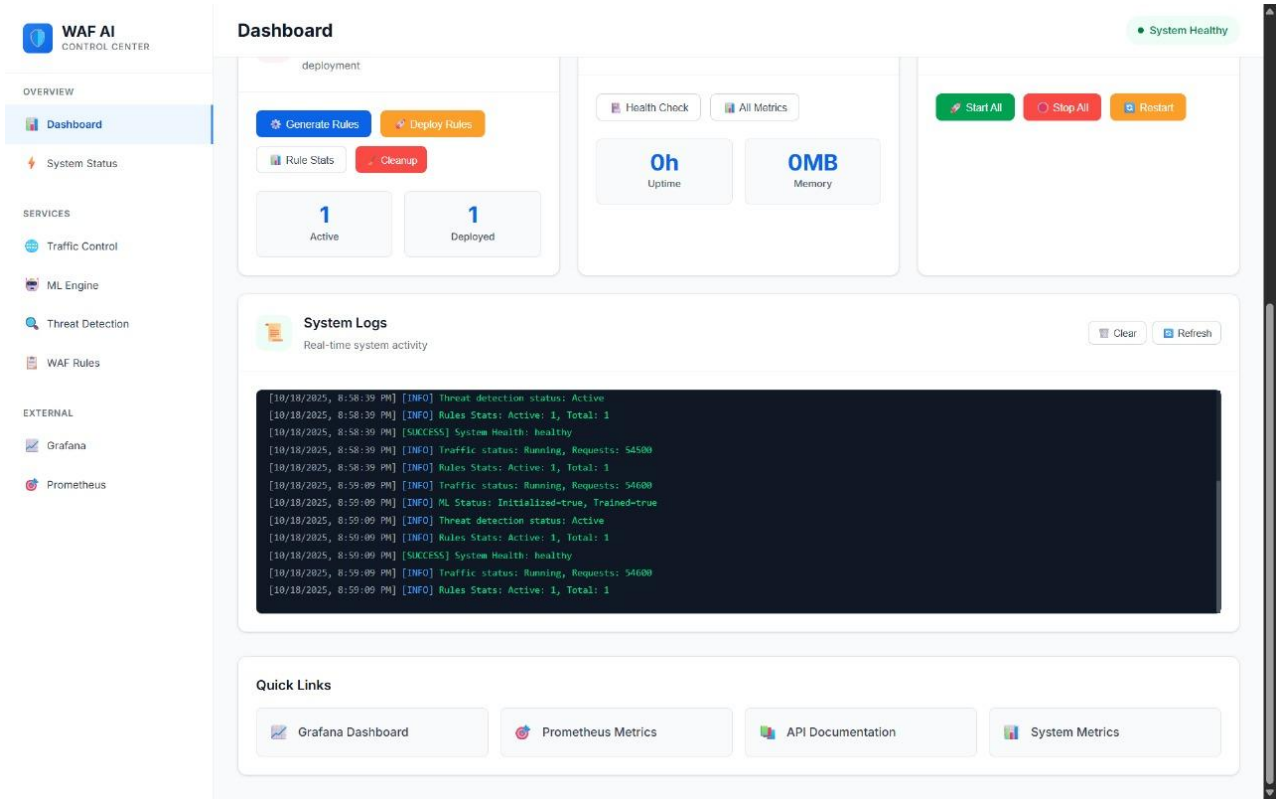


Fig 10.1.2 Waf AI Control Panel Dashboard - 2 Half

10.2 ATTACK DETECTION OUTPUT

The **Attack Detection Output** module is responsible for displaying the results of the real-time threat detection performed by CyberGuard AI. Using advanced **machine learning algorithms**, the system continuously inspects web requests, identifies malicious patterns, and provides automated mitigation.

Detailed Features and Functionality:

1. **Threat Categorization:**
    - Detects multiple attack types such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Brute Force attacks.
    - Helps administrators quickly understand the nature of ongoing threats.
2. **Severity Levels:**
    - Threats are classified as Low, Medium, or High severity.
    - Enables prioritization of mitigation efforts to address the most dangerous threats first.
3. **Automated Response:**
    - Displays the action taken by the system for each detected threat (Block, Challenge, Rate-limit).
    - Ensures minimal manual intervention, allowing for rapid response to high-volume attacks.
4. **Source Tracking:**
    - Provides detailed information about the origin of attacks, including IP address, geolocation, and user agent.
    - Assists in auditing, reporting, and forensic investigations.
5. **Time-Stamped Logging:**
    - Maintains a record of each attack event with timestamp, affected endpoints, and response actions.
    - Supports compliance and security reporting.

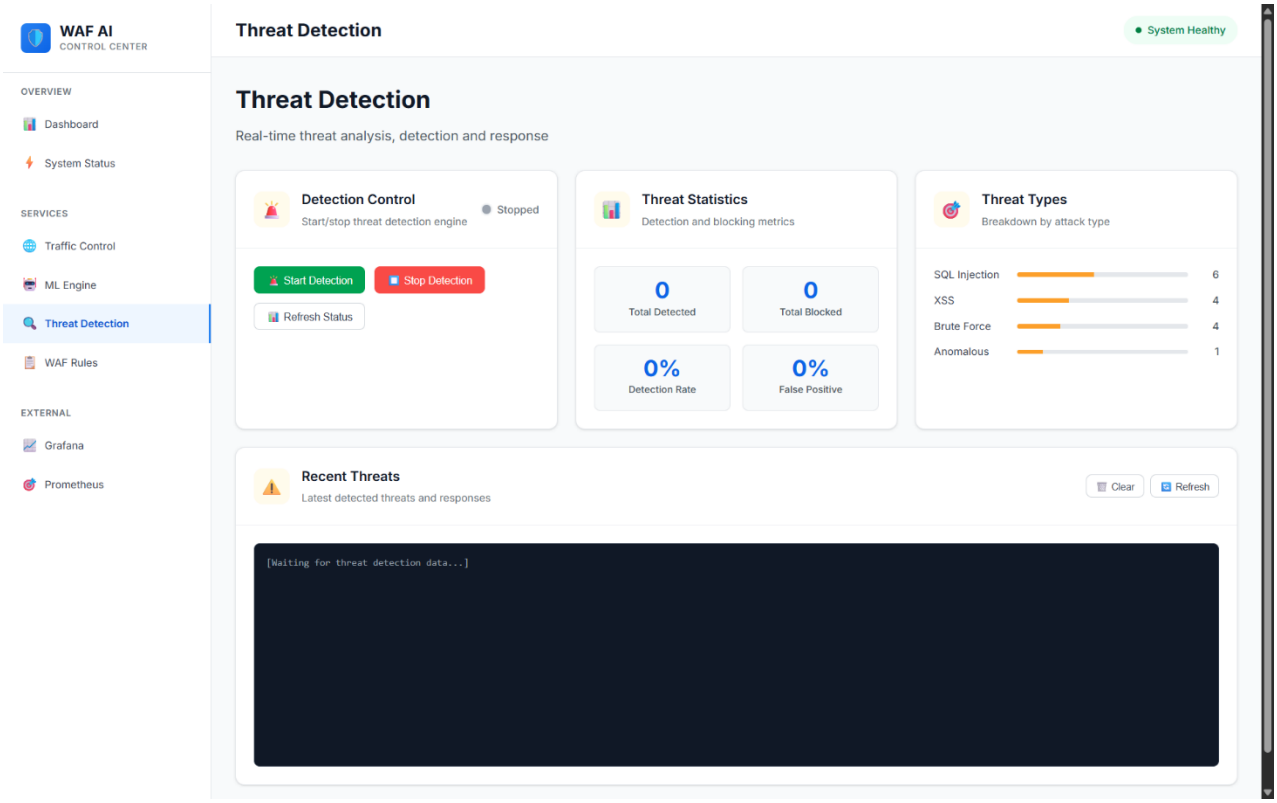


Fig 10.2.1 Waf AI Control Panel Threat Detection

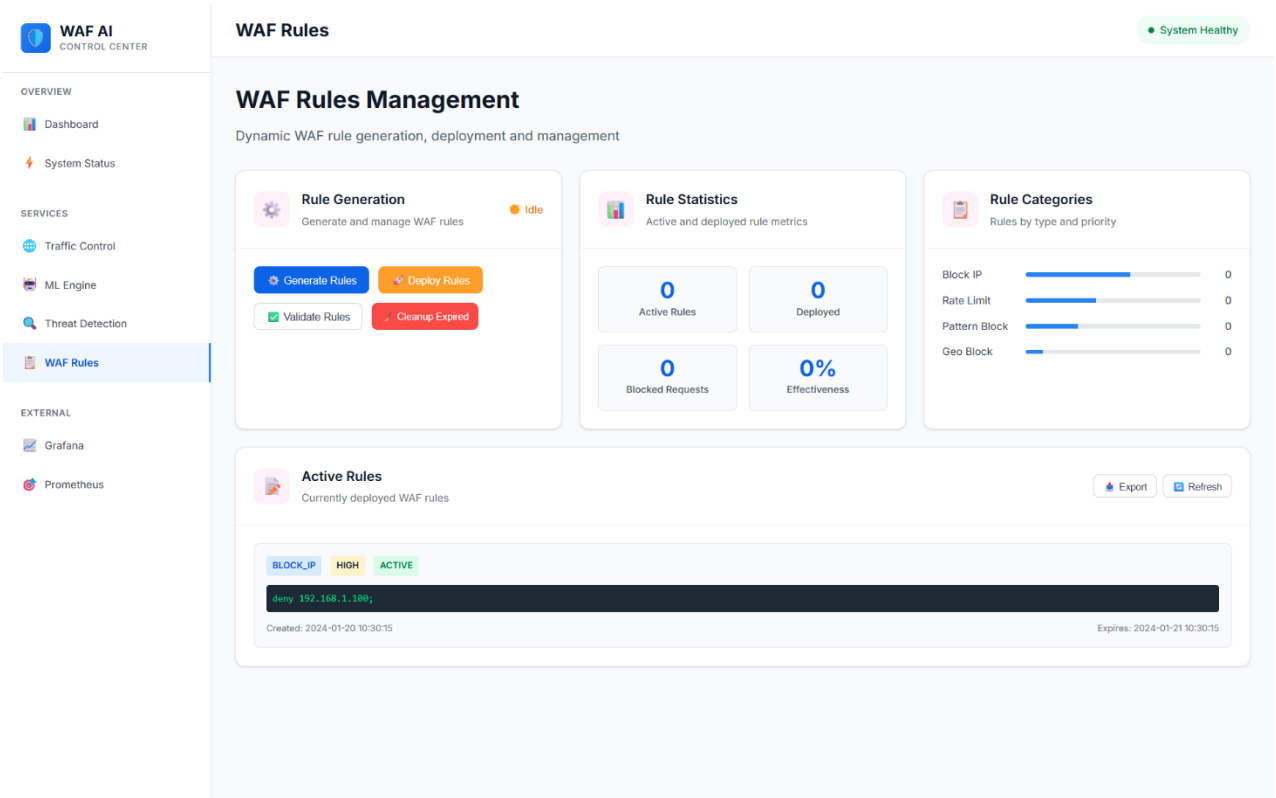


Fig 10.2.2 Waf AI Control Panel Waf Rules

10.3 REAL-TIME VISUALIZATION

The **Real-Time Traffic Visualization** section of CyberGuard AI provides graphical insights into incoming and outgoing traffic, system performance, and potential anomalies. This module helps administrators monitor web traffic trends, identify suspicious activity, and optimize resource allocation.

Detailed Features and Functionality:

1. **Traffic Patterns:**
    - Displays the number of incoming requests, successful responses, and blocked attacks in real-time.
    - Helps administrators quickly spot traffic spikes or unusual patterns indicating potential attacks.
2. **Anomaly Detection:**
    - Visualizes abnormal traffic behavior, which may indicate DDoS attacks, SQL Injection attempts, or other malicious activities.
    - Enables administrators to take proactive actions before serious damage occurs.
3. **System Performance Metrics:**
    - Monitors the health of WAF nodes, ML engine processing, API endpoints, and database performance.
    - Ensures the system operates at peak efficiency and maintains high availability.
4. **Visualization Tools:**
    - Uses internal dashboards and external monitoring tools like **Prometheus** and **Grafana** to create interactive graphs and charts.
    - Includes metrics such as request count, threat frequency, response times, and blocked request statistics.

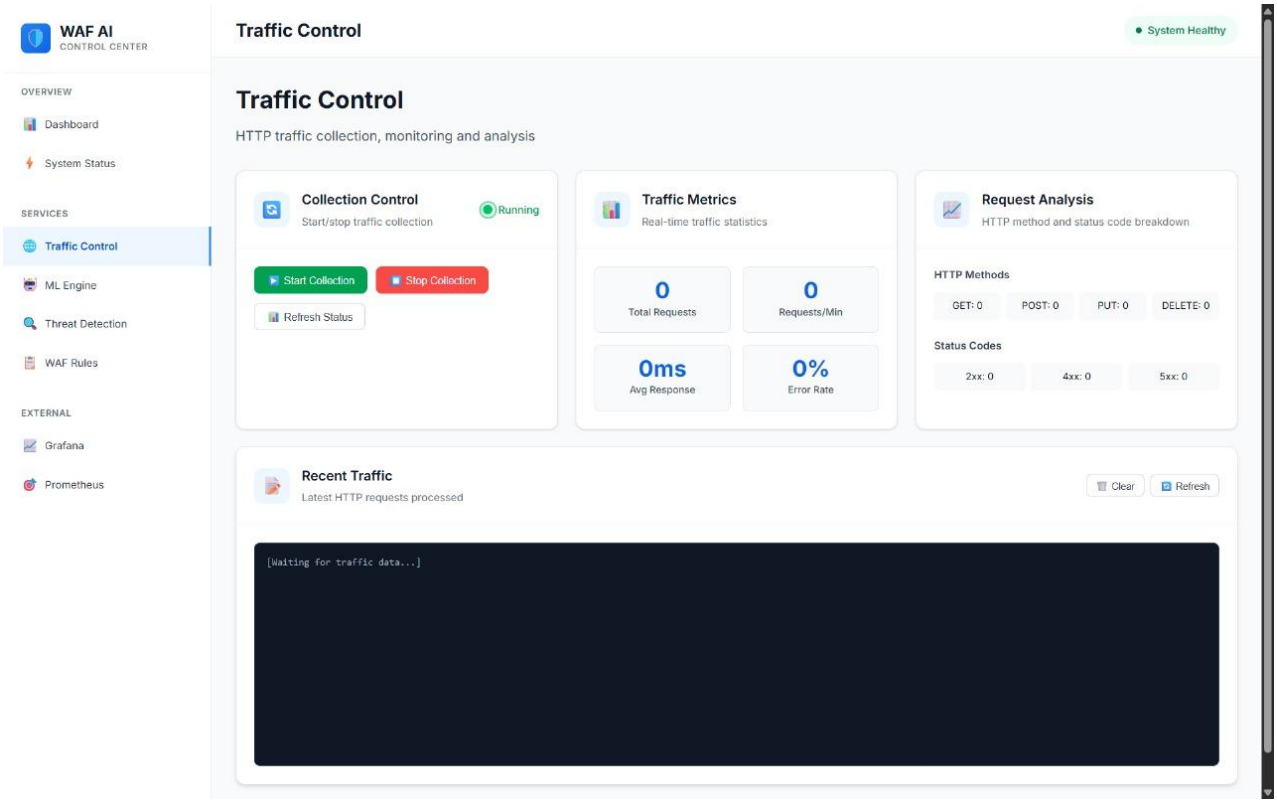


Fig 10.3.1 Waf AI Control Panel Traffic Control

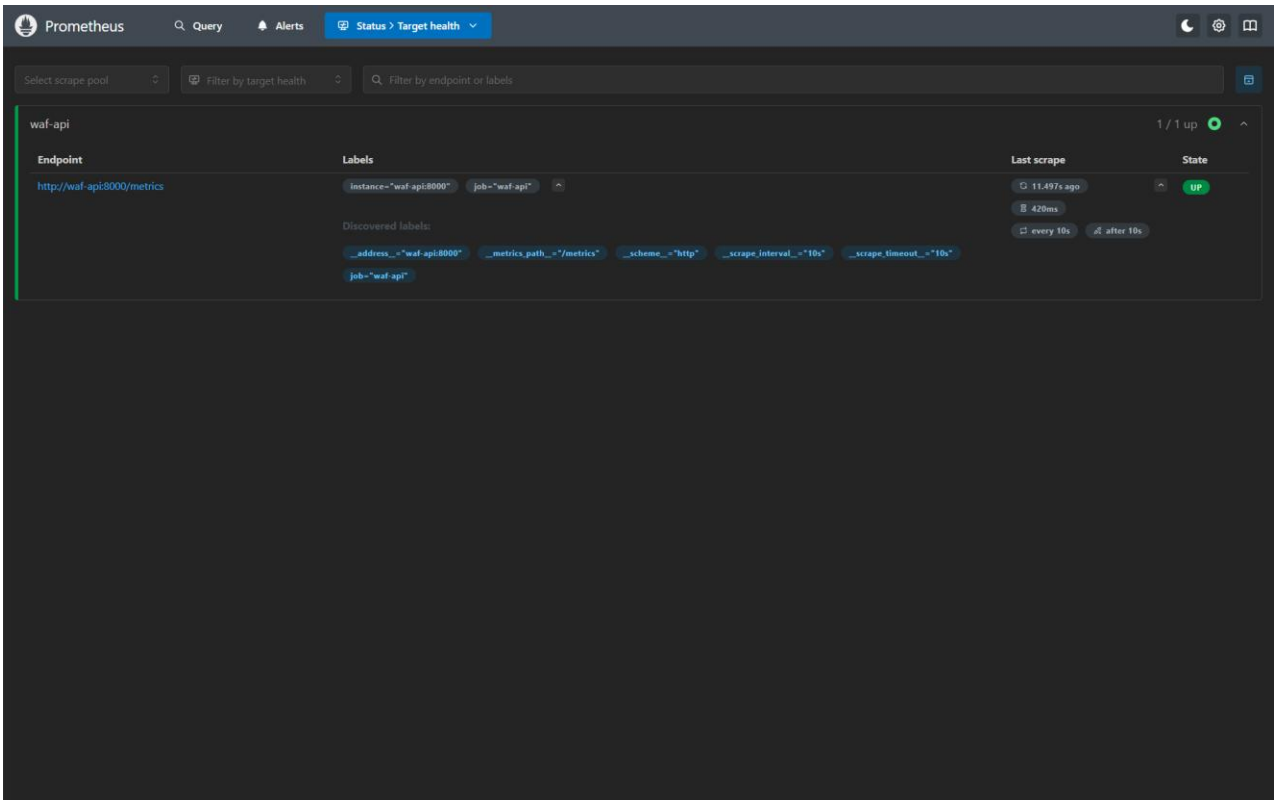


Fig 10.3.2 Prometheus Waf-Api Target Health

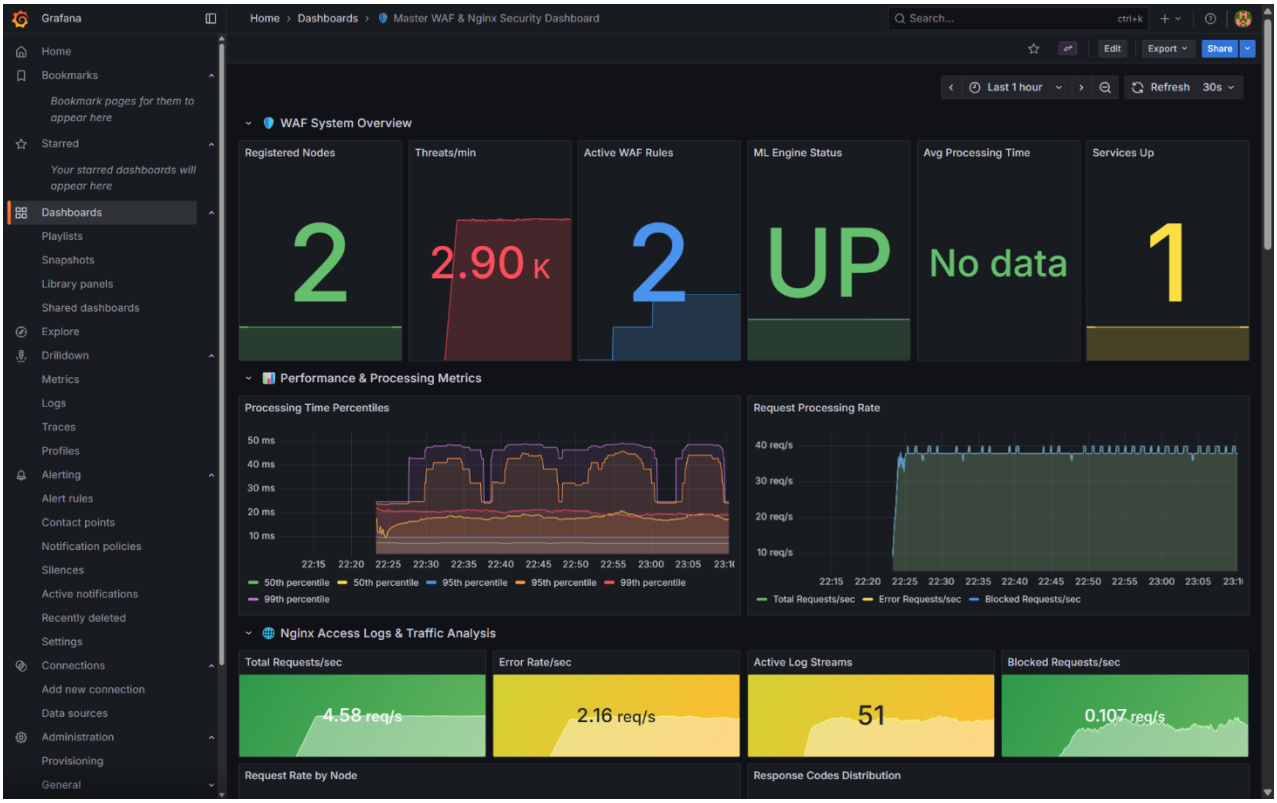


Fig 10.3.3 Grafana Dashboard-1

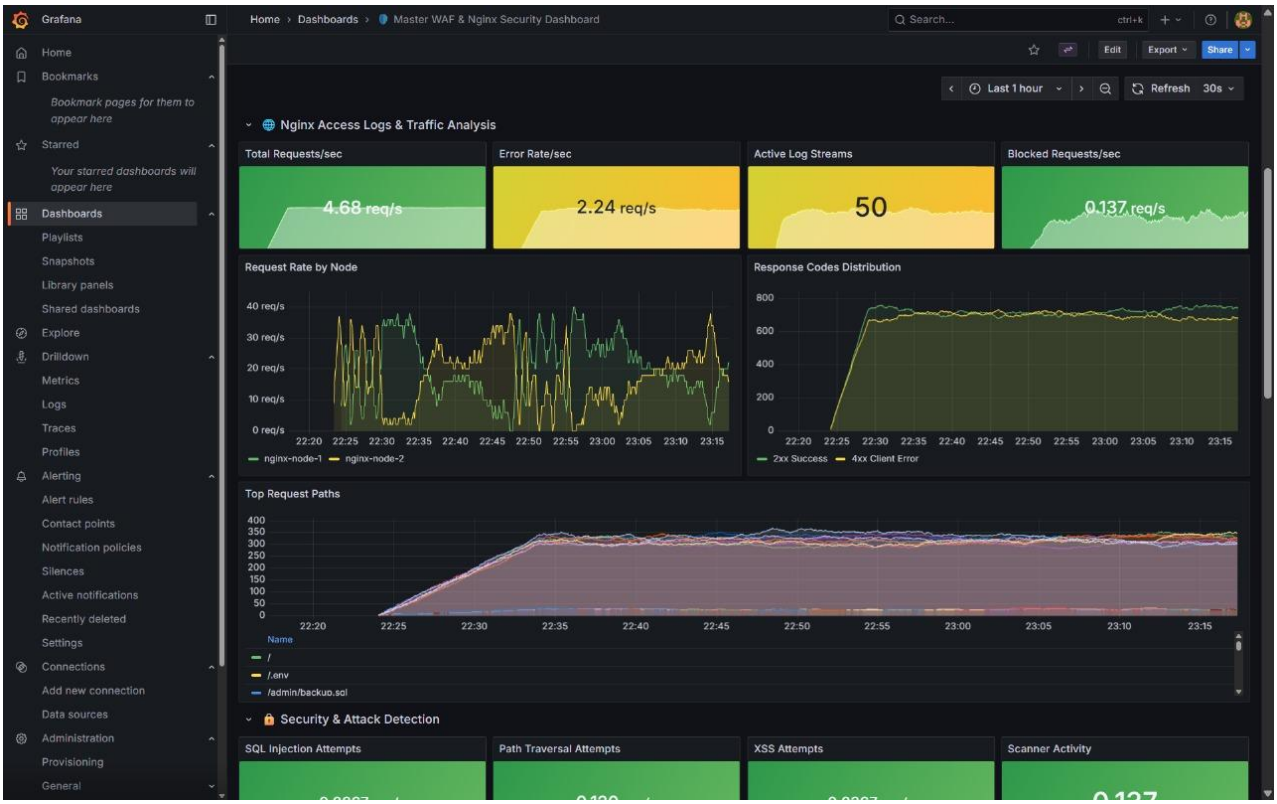


Fig 10.3.4 Grafana Dashboard-2

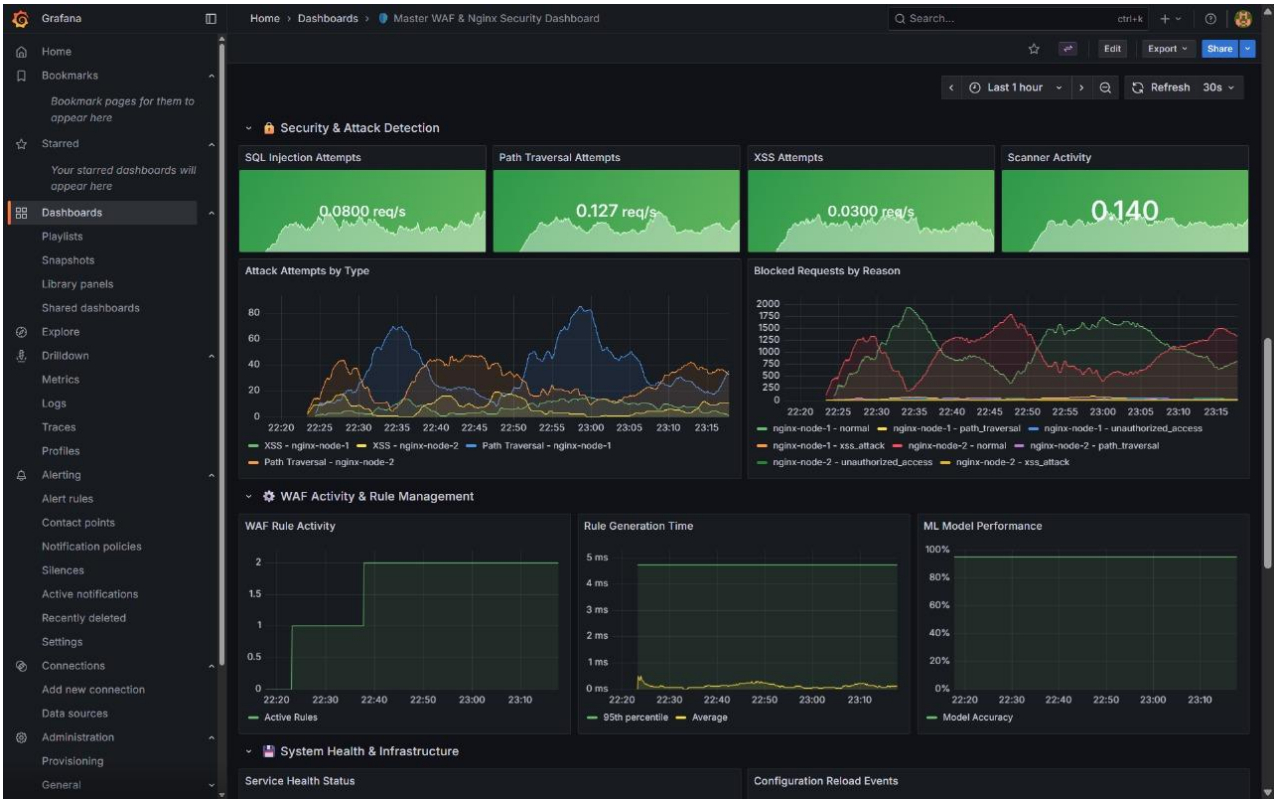


Fig 10.3.5 Grafana Dashboard-3

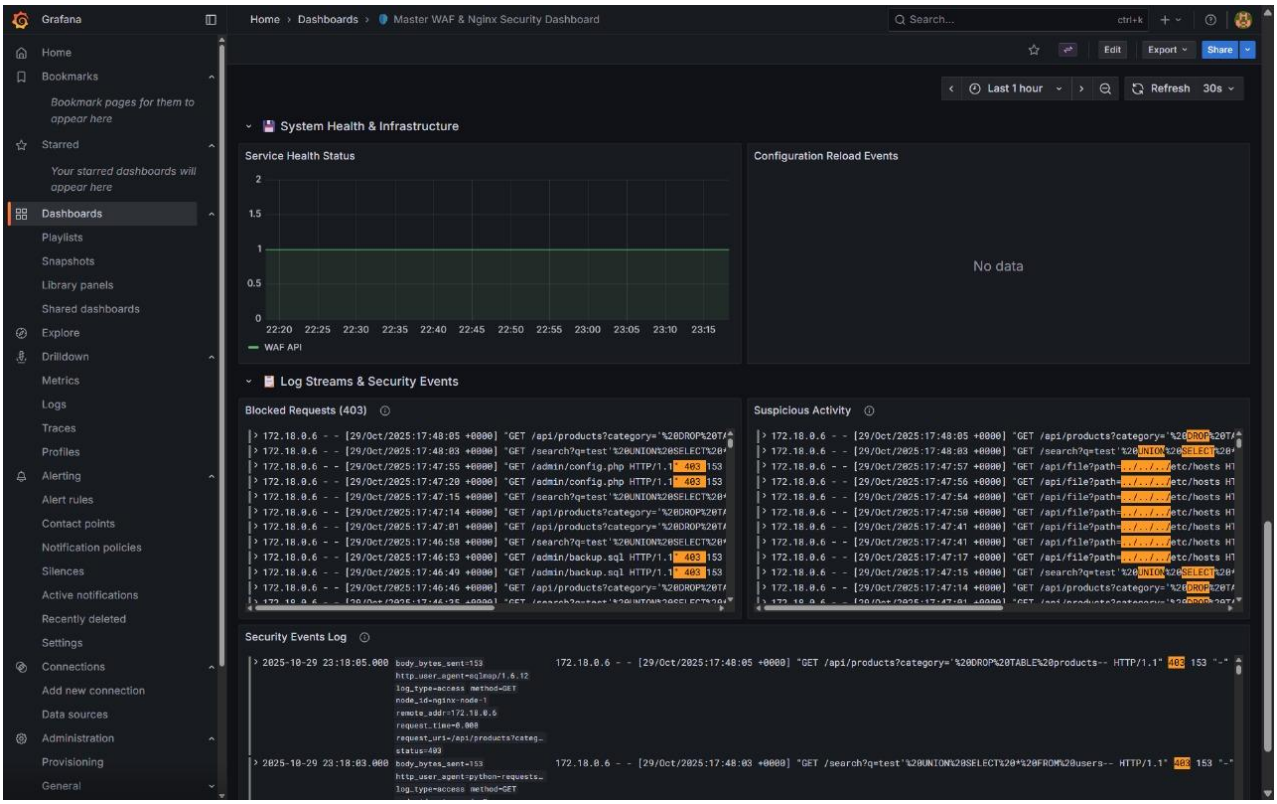


Fig 10.3.6 Grafana Dashboard-4

10.4 BLOCKED ATTACK LOGS

The **Blocked Attack Logs** module provides detailed documentation of all malicious requests intercepted by CyberGuard AI. This ensures administrators have complete visibility of attacks, system responses, and threat patterns for analysis and reporting.

Detailed Features and Functionality:

1. **Comprehensive Logging:**
    - Records every blocked request, including source IP, request payload, type of attack, and timestamp.
    - Maintains a chronological history for audit purposes.
2. **Automated Mitigation Actions:**
    - Shows the automated action taken by the WAF (e.g., Block, Rate-limit, Challenge).
    - Ensures that attacks are neutralized without delay.
3. **ML Engine Insights:**
    - Provides information on which attacks were detected and classified by the machine learning engine.
    - Helps evaluate the performance of AI models and update them if needed.
4. **Alerts and Notifications:**
    - Sends real-time alerts for high-severity attacks to the administrator.
    - Allows prompt action to prevent potential damage.
5. **Visual Representation:**
    - Logs and analytics are displayed in tables, charts, and graphical summaries for easier interpretation.

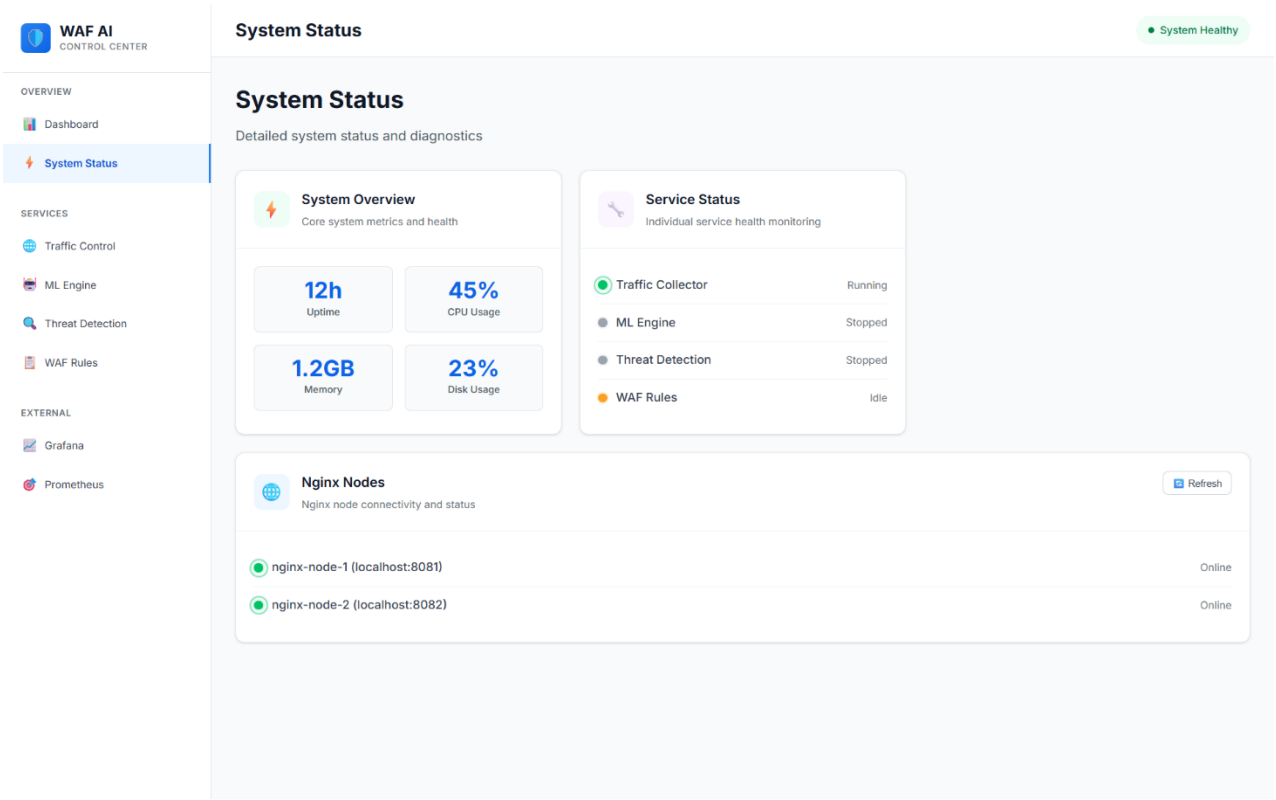


Fig 10.4.1 Waf AI Control Panel System Status

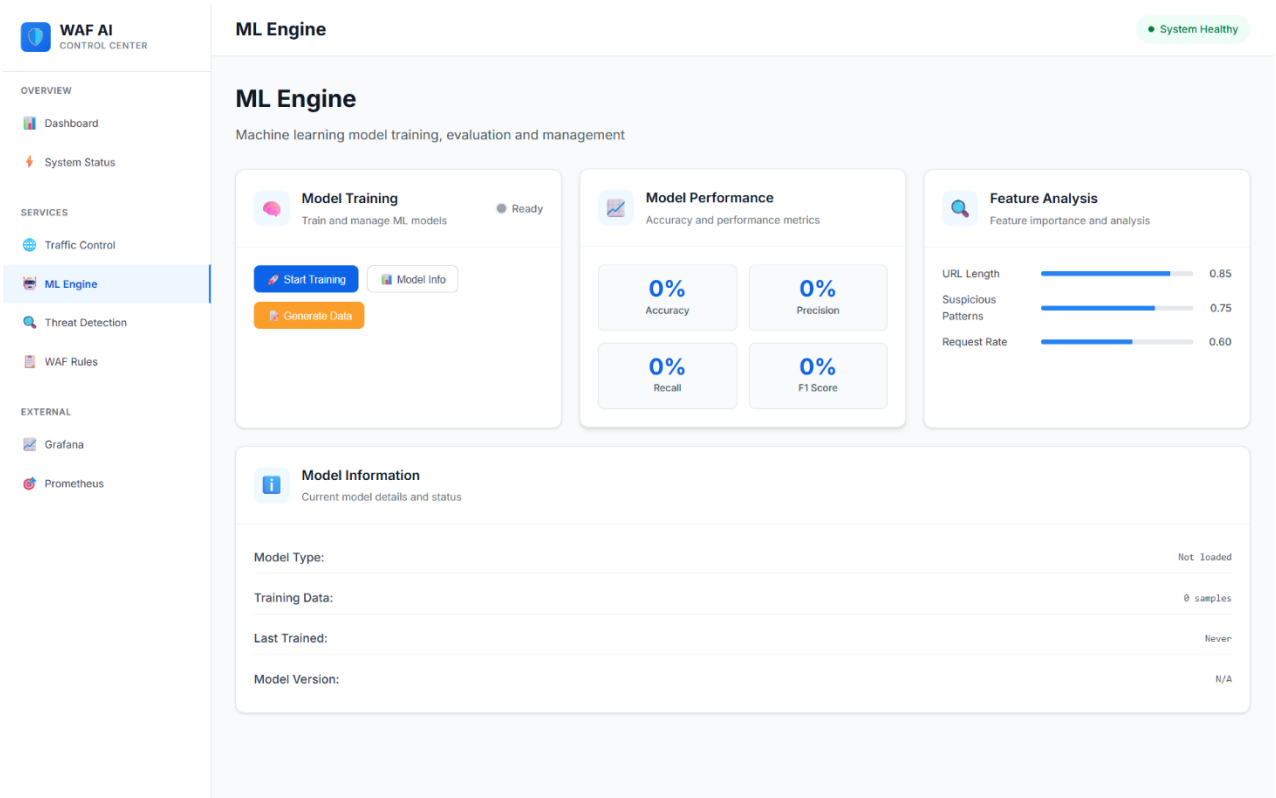


Fig 10.4.2 Waf AI Control Panel ML Engine

Chapter 11

REFERENCES

[1] Behl, A., & Behl, K. (2017). Cybersecurity and cyberwar: What everyone needs to know. Oxford University Press.

[2] Scarfone, K., & Mell, P. (2012). Guide to intrusion detection and prevention systems (IDPS). NIST Special Publication 800-94.

[3] OWASP Foundation. (2023). OWASP Top 10 Web Application Security Risks. OWASP Documentation.

[4] Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. IEEE Symposium on Security and Privacy, 305–316.

[5] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cybersecurity intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153–1176.

[6] Shafi, A., Abbasi, A., & Raza, M. (2019). Machine learning based detection of SQL injection attacks. International Journal of Computer Networks & Communications, 11(3), 1–14.

[7] Liao, Y., Vemuri, V. R., & Vemuri, P. (2002). Use of k-nearest neighbor classifier for intrusion detection. Computers & Security, 21(5), 439–448.

[8] Kim, G., Lee, S., & Kim, S. (2014). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. Expert Systems with Applications, 41(4), 1690–1700.

[9] TensorFlow Documentation. (2024). Machine learning for security applications. TensorFlow.org.

[10] PyTorch Documentation. (2024). Deep learning frameworks for real-time systems. PyTorch.org.

[11] Turnbull, J. (2018). The Docker book: Containerization is the new virtualization. James Turnbull Publishing.

[12] Pahl, C. (2015). Containerization and the PaaS cloud. IEEE Cloud Computing, 2(3), 24–31.

[13] Prometheus Authors. (2023). Prometheus monitoring system documentation. CNCF.

[14] Grafana Labs. (2024). Grafana dashboards and analytics documentation. Grafana Labs.

[15] Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine.

[16] Stallings, W. (2018). Network security essentials: Applications and standards. Pearson Education.

[17] GitHub Repository for the Project: **AI-Powered Advanced Web Application Firewall**, https://github.com/Darkwebnew/AI-Powered-Advanced-Web-Application-Firewall

[18] Docker Compose Documentation. (2024). Multi-container Docker applications. Docker Inc.

[19] Python Software Foundation. (2024). Python documentation for web and security applications. Python.org.

[20] YouTube Video Demonstration of the Project: *AI-Powered Advanced Web Application Firewall Demo*.