Real-time SQL Injection Detection and Prevention System Based on Reverse Proxy Server

Project Submitted to University of Science and Technology in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering in Cyber security

from University of Science and Technology , Taiz.

Submitted by:

Khaled Abdulsattar Ahmed and Nafea Ali Ghaleb

Under the Guidance of:

Eng. Abdulaziz Taher

Assistant Professor

Cyber security

Engineering and computing Department

Engineering and computing College

University of Science and Technology, Taiz , Yemen

May 2025

University of Science and Technology, Taiz

Engineering and computing College

Engineering and computing Department

Bachelor of Engineering in Cyber security

May 2025

Certified Bonafide Project Work done By:

Khaled Abdulsattar Ahmed and Nafea Ali Ghaleb

Project Manager **Head of Department**

Submitted for the Project Evaluation and Viva-Voce held at the **Engineering and computing Department Taiz** , on ……..

CERTIFICATE

This is to certify that the project entitled **“** **”** submitted to the **Cyber Security Program ,Engineering and computing Department , Engineering and computing College** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Engineering in Cyber security** is a record of original final project work done by ……student[ID] during the period October 2021 to May 2025 of his study in the **Cyber Security Program ,Engineering and computing Department** , under my internal supervision and guidance of **Dr. Akram Alhammadi ,Assistant professor** of **Engineering and computing Department** .

Signature of the Guide

DECLARATION

We, **Khaled Abdulsattar Ahmed and Nafea Ali Ghaleb**, hereby declare that the project, entitled submitted to the **Cyber Security Program ,Engineering and computing Department**, **Engineering and computing College** , in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Engineering in Cyber Security** is a record of original project work done by us during the period October 2021 to May 2025 under the guidance of **Dr. Akram Alhammadi, Assistant Professor , Engineering and computing Department , Engineering and computing College.**

Signature of the Candidate

Dedicated To

To those who have been our compass on life's journey,

instilling in us a love of knowledge and the pursuit of excellence,

teaching us to face life's challenges with patience and steadfastness

and to adhere to the noble values that elevate humanity.

To our parents, we dedicate this project, which we have accomplished thanks to your continued support and encouragement.

You are the support and assistance we cannot do without.

We ask God to reward you abundantly for your efforts on our behalf.

To those who have illuminated the path of knowledge with their wisdom and patience

and instilled in us a love of knowledge and the pursuit of excellence,

To our distinguished professors.

We dedicate this humble work to you, hoping it will be admired and accepted by you.

You are the beacon that guides us on our academic journey.

We ask God to reward you abundantly for your great efforts. To our colleagues who have shared this journey with us,

and supported us in times of hardship and fatigue,

To our dear colleagues,

We dedicate to you this work, accomplished with a spirit of teamwork.

You are our partners in success and witnesses to our journey.

I hope that we will always remain united in achieving our ambitions.

In conclusion, we ask God to make this work sincerely for His sake,

to benefit our beloved country, and to open for us the doors of success and prosperity.

ACKNOWLEDGEMENT

First and foremost, I praise Allah Almighty for His blessings and for facilitating the completion of this work. I would like to express my sincere gratitude to my family members who have always been by my side, enduring the pressures of study and research with me, and providing me with psychological and material support with all love and dedication. I also thank those honorable individuals who extended their helping hands during this period, including all professors, doctors, and teaching assistants with whom I communicated, especially my esteemed supervisor, Engineer Abdulaziz Taher, who spared no effort in providing assistance to me, encouraging me to research and succeed, motivating me, and strengthening my determination. May Allah reward him greatly, and I extend to him all appreciation, gratitude, and thanks. I also thank all those in charge of the university, especially Dr. Nael Al-Haddad, Director of the Engineering and Computing Branch. May Allah guide them to all that is good for their dedication to the college students at various levels. All praise is due to Allah, Lord of the worlds.

Khaled Abdulsattar Ahmed & Nafea Ali Ghaleb

Acknowledgment

First and foremost, I praise Allah Almighty for His blessings and for facilitating the completion of this work. I would like to express my sincere gratitude to my family members who have always been by my side, enduring the pressures of study and research with me, and providing me with psychological and material support with all love and dedication. I also thank those honorable individuals who extended their helping hands during this period, including all professors, doctors, and teaching assistants with whom I communicated, especially my esteemed supervisor, Engineer Abdulaziz Taher, who spared no effort in providing assistance to me, encouraging me to research and succeed, motivating me, and strengthening my determination. May Allah reward him greatly, and I extend to him all appreciation, gratitude, and thanks. I also thank all those in charge of the university, especially Dr. Nael Al-Haddad, Director of the Engineering and Computing Branch. May Allah guide them to all that is good for their dedication to the college students at various levels. All praise is due to Allah, Lord of the worlds.

INDEX OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Despite the greatness and advantages of the Internet, it hides flaws that negatively affect its performance. One of the most prominent of these flaws is "Internet fragility," which refers to weaknesses and security vulnerabilities in network systems. "Internet fragility" is one of the fundamental problems, as it may lead to data modification and theft. Many web applications rely on storing, retrieving, and updating data in databases as needed. Additionally, servers hosting these applications retrieve information from databases and fulfill user requests, which poses a significant risk to the security of applications and data, making them vulnerable to a range of attacks, most notably SQL injection attacks (SQLIA). These are considered among the most dangerous security vulnerabilities for websites, known for 25 [1],[5] years and still classified as one of the most dangerous vulnerabilities to this day, ranking third in the OWASP Top 10 [3] ,list for2024.

In this project, we will create an integrated security system to protect against this attack, which will be based on a reverse proxy [4] ,[10] that acts as a firewall or intermediary between web applications and the client to prevent direct client access. This system will be implemented by deploying a reverse proxy linked to a scanning service based on modern security technologies to intercept and examine requests before passing them to the main server.

Chapter 1:

Introduction

1.1 Introduction

Web applications have experienced rapid growth in recent years, evolving faster than anticipated just a few years ago. Today, billions of transactions are conducted online with the help of various web applications, providing the advantage of access from anywhere in the world and offering services that are accessible to virtually everyone. Through these applications, enormous amounts of information are distributed daily, which may include private and confidential data stored in the web server's database along with other website-related data. However, this can significantly impact data security; an insecure design of a web application may allow the injection of malicious instructions or updates to the backend database, leading to the theft of sensitive user data or causing severe damage. In the worst cases, an attacker may gain complete control of the application, resulting in system destruction or corruption. This technique, where malicious SQL statements are injected into inputs, is known as SQL Injection attack [12] , [1], one of the most dangerous security [3] ,vulnerabilities exploited to leak sensitive information, gain unauthorized access, and cause financial losses to individuals and companies. Despite this vulnerability being known for over 25 years and the existence of many commercial and open-source tools claiming to eliminate it, it still ranks third in the OWASP Top 10 [16], list of web application security vulnerabilities for 2024, [12], due to the lack of a comprehensive solution to address new and evolving attacks and provide immediate protection before they occur. Based on this, a system has been developed that integrates several techniques to detect and prevent [6], this type of attack, including its advanced forms, and provide immediate alerts when suspicious activity is detected.

1.2 Project Challenges (Current System)

Despite the widespread use of traditional security systems in many organizations, these systems face numerous challenges, including:

1. Lack of Real-time Protection:

that traditional protection systems lack the capability for immediate detection of advanced SQL injection attacks, which delays the response to threats.

2. Management Complexity:

Current systems require continuous human intervention to adjust settings and process alerts, leading to resource depletion. The lack of user-friendly interfaces makes management difficult for non-specialists.

3. Lack of Integration Between Security Tools:

Current systems lack integration between detection and protection tools; many tools work independently without coordination, creating vulnerabilities that attackers can exploit.

4. Absence of Immediate Alerts**:**

Traditional systems often do not provide immediate notifications when attacks are detected, leading to delays in taking necessary measures to limit damage.

5. Traditional protection systems face challenges in integration with existing infrastructure, which increases the complexity of deployment and maintenance processes (Kumar & Singh, 2023)

1.3 Research Objectives

The research aims to achieve several main objectives to enhance database security, including:

1. Building an integrated security system that reduces risks and increases confidence in the technical systems used.

2. development of a protection system, this project aims to develop a system that combines real-time detection techniques.

3. Reducing the technical burden, meaning that a system will be designed to operate independently without the need for modifications to the source code.

4. Adapting to evolving threats by integrating machine learning technology to predict new attacks of this type.

5. Create a mechanism to send immediate alerts to administrators when suspicious activity is detected via channels such as email or text messages.

6. Developing an easy-to-use user interface that allows even non-experts to manage and operate the system.

1.4 Project Scope

Functional boundaries:

The scope of the project is limited to developing a reverse proxy server-based protection system that aims to detect and prevent SQL Injection attacks on web applications only, and does not include other cyber attacks such as XSS or DDoS, an approach that most previous studies have indicated .

Spatial Boundaries:

The project is implemented in a laboratory environment first, with experimental tests using attack simulation tools (such as SQLMap and Burp Suite) before considering deploying the system in a production environment.

The source code of the target applications is not modified; instead, the solution is applied as an intermediary system (reverse proxy) added to the existing system structure.

Temporal boundaries:

Due to time and resource constraints, the testing phase will be limited to specific attack scenarios, which may not cover all types of potential scenarios in wide production environments.

1.5 Project methodology

For our project, we will use a Hybrid Methodology that combines elements from Waterfall and Agile approaches, with a focus on iterative and incremental development. This choice is based on the complex nature of security systems, the high security requirements, and the need for gradual development and continuous testing.

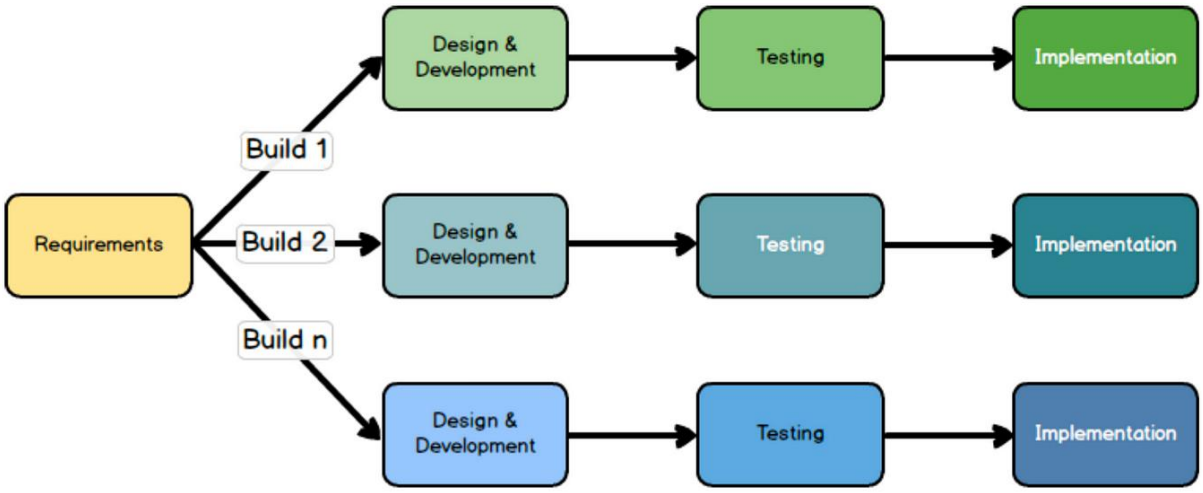

Figure 0- 1 Incremental model

1.5.1 Analysis and Data collection phase

Literature and reference study**:**

Previous research and studies related to SQL Injection attacks were reviewed [16], analyzing traditional and modern methods of detection. The focus was on the strengths and weaknesses of current systems [10], to identify research gaps that our project seeks to fill.

Security requirements analysis**:**

Current systems were analyzed to identify security vulnerabilities and issues relate to the inability to detect immediately and integrate protection tools. System requirements were also gathered through extensive study in this field and reviewing related studies and research.

1.5.2 Design phase

General system architecture design:

The system will be designed as an integrated solution based on using a reverse proxy to receive and examine requests before they reach the main server. The design is divided into the following components:

-Reverse proxy: To intercept requests facing the web application and forward them to the scanning unit.

- Scanning unit: combines signature-based detection and machine learning (anomaly-based detection) techniques to analyze requests and detect abnormal patterns.

- Control and management interface: For controlling the activation and deactivation of protection, adding basic configuration settings, and displaying reports and system monitoring.

Preparation of Illustrative Diagrams:

Flowcharts and Data Flow Diagrams will be drawn to illustrate how data moves between different system components, showing the steps that requests go through from reception to final decision-making.

1.5.3 Implementation phase

Software system development:

We will use python programming language with integration of NGINX web server, and develop scanning units using signature-based technology and machine learning algorithms to classify requests and determine risk levels.

Management interface development:

An interactive user interface will be designed to facilitate settings adjustment and report review for the administrator.

Component Integration:

All units (reverse proxy, scanning unit, and management interface) are integrated into one comprehensive system, ensuring their compatibility, integration, and smooth data exchange between them.

1.5.4 Testing and Evaluation Phase

Performance and Reliability Testing:

The system will be tested using specialized tools such as SQLMap and Burp Suite to simulate SQL Injection attacks. Performance metrics such as response time and detection accuracy will be measured. Afterward, the protection system will be tested on a simple web application or a test environment such as (metasploitable).

Results Analysis:

Test data will be collected and analyzed to evaluate the effectiveness of the system compared to traditional methods. This will identify improvement points and make necessary adjustments to system units, their settings, and the technology used.

1.5.5 Documentation Phase

All steps of our project will be documented from the analysis and design phase to testing and evaluation, with the preparation of a detailed report including diagrams and other materials, formulating conclusions about the effectiveness of our system, and providing future recommendations to improve it and increase the level of protection.

1.6 Feasibility Study

1.6.1 Technical Feasibility

The technologies used in our project (such as NGINX and machine learning algorithms) are based on open-source and reliable tools in the security field, ensuring the availability of continuous support and updates.

The system can be integrated with infrastructure systems without the need for radical modifications to application code.

Efficiency and Performance:

The system is designed to work as an intermediary system (Reverse Proxy) that allows real-time examination of requests without significantly affecting response time.

Using a hybrid approach that combines signature-based detection and anomaly-based detection helps reduce the false positive rate and increase detection accuracy.

1.6.2 Economic Feasibility

Protecting sensitive data and mitigating cyber breaches will help reduce the potential costs associated with data loss. A study by the Ponemon Institute (2024) indicates that the average cost of a data breach resulting from SQL injection attacks is $4.35 million, highlighting the economic importance of effective security systems.

1.6.3 Operational Feasibility

Ease of Management and Integration:

The system provides a simplified management interface that enables monitoring of security activities and easy adjustment of settings.

Operational feasibility can be achieved by designing the system as an intermediate unit that does not require radical changes to existing applications (Garcia & Martinez, 2022).

Scalability:

The system is designed to be scalable in the future to cover other types of attacks or to increase the ability to handle a larger number of requests without affecting performance.

The possibility of developing additional modules or improving machine learning algorithms based on data and experiences gained from practical application.

1.7 Project Building Tools

Programming Languages and Frameworks:

Python: Used to develop the core of the system, including request analysis and implementation of machine learning algorithms.

React/JavaScript: frameworks for building the management interface and providing web services through RESTful APIs.

Web Server and Reverse Proxy:

NGINX: Works as a web server and reverse proxy to receive requests and direct them to the scanning unit before they reach the main server.

Database:

MySQL or PostgreSQL: Used to store request logs and system reports, allowing tracking and later analysis of activities.

Machine Learning Tools:

TensorFlow / Keras / scikit-learn: Libraries used to develop machine learning models for analyzing request behavior, detecting abnormal patterns, and reducing false positives.

Security Testing Tools:

SQLMap: A tool for simulating SQL Injection attacks and testing the effectiveness of the system in detecting and preventing these attacks.

1.8 Time plan:

The timeline is considered one of the fundamental elements in project management. It represents an organized and detailed vision of the various phases and activities that will be implemented throughout the project period, with specification of the expected timeframe for each phase and activity. The timeline functions as a roadmap that guides the work team toward achieving project objectives within the designated time and with the available resources . In our project, we represented the timeline in two ways: (PERT chart).

Project Timeline:

The following timeline outlines the phases and tasks for the our system project. The timeline is divided into five main phases: Analysis and Data Collection, Design, Implementation, Testing and Evaluation, and Documentation.

Project Timeline Table:

Phase	Tasks	Duration
1. Analysis and Data Collection Phase	- Literature review on SQL injection attacks - Study of existing detection and prevention techniques - Requirements gathering and analysis - Data collection for training and testing	4 weeks
2. Design Phase	- System architecture design - Reverse proxy server configuration - Detection module design - Prevention module design - User interface design	5 weeks
3. Implementation Phase	- Reverse proxy server	6 weeks

	implementation - Detection module implementation - Prevention module implementation - Integration of components - Initial testing	
4. Testing and Evaluation Phase	- Testing with SQLMap and Burp-suite -Performance metrics measurement - Testing on test environment (metasploitable) - System optimization based on test results	3 weeks
5. Documentation Phase	- Documentation of all project phases -Preparation of detailed report - Formulation of conclusions - Future recommendations	2 weeks

Table 1 Project Timeline Table

PERT chart:

The PERT (Program Evaluation and Review Technique) chart is one of the most important project management diagrams. It is used to determine the time required to complete a specific task or activity. Additionally, it works to organize and coordinate tasks and improves the decision-making process by identifying the critical path for project tasks and activities.

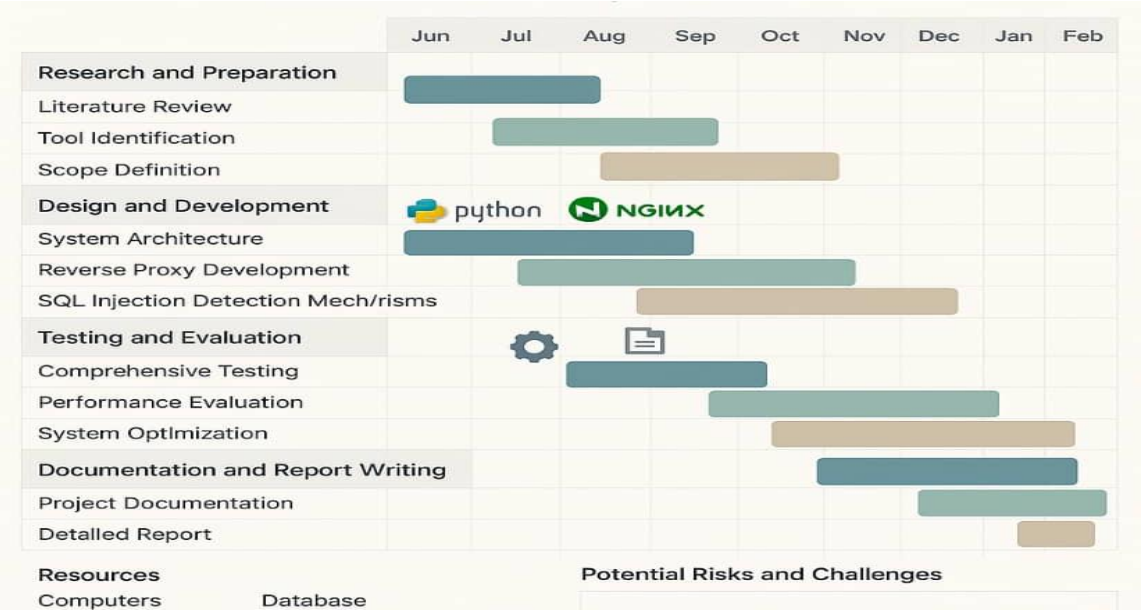

Figure 0- 2 PERT chart

Chapter 2 : Literature Review

2.1 Introduction

This chapter aims to provide a comprehensive theoretical background explaining the nature of SQL Injection attacks and the evolution of security protection methods for web applications. It also includes a critical review of previous studies and research that addressed the topic of detecting SQL Injection attacks, highlighting the research gaps that the proposed project seeks to fill. This review serves as a basis for understanding the methodology and techniques that will be adopted later in the project.

2.2 Historical Overview

SQL Injection (SQLI) attack is one of the oldest and most dangerous vulnerabilities in web application development. These attacks first appeared in the late 1990s with the development of dynamic applications that rely on databases. As reliance on databases to store sensitive information such as user data and financial transactions increased, this vulnerability emerged to become one of the most common attacks. In the first decade of the 21st century, SQL injection attacks became one of the most dangerous attacks on web applications, with reports of government and commercial database breaches leading to the leakage of huge amounts of sensitive information. In 2008, SQL injection attack activities peaked, accounting for about 80% of reported breaches. Even today, SQL injection attacks still rank high in the(OWASP foundation, 2021) OWASP Top 10 list, accounting for more than 40% of attacks targeting web applications according to recent security reports.

SQL injection attacks have led to some of the largest data breaches in history, confirming the urgent need for strong application security. Here are some prominent real-world examples:

In 2008, Heartland Payment Systems Breach (2008) Heartland Payment Systems, a leading payment processing company, suffered a breach that exposed approximately 130 million credit and debit card numbers. Attackers exploited an SQL injection vulnerability to infiltrate the company's network, resulting in one of the largest recorded data breaches.

In July 2012, Yahoo! Voices Data Breach (2012) Yahoo! Voices fell victim to an SQL injection attack that compromised nearly 450,000 user accounts. Hackers exploited security vulnerabilities in Yahoo's database servers to obtain unencrypted usernames and passwords, highlighting the risks of inadequate input validation.

In 2015, TalkTalk Data Breach (2015) UK telecommunications company TalkTalk suffered an SQL injection attack, resulting in the exposure of personal data for approximately 160,000 customers. Attackers exploited security vulnerabilities in the company's website pages, causing significant financial and reputational damage.

In 2020, Freepik and Flaticon Breach (2020) Microsoft revealed that an SQL injection attack led to the leakage of 8.3 million user records from its Freepik and Flaticon platforms. Attackers exploited a security vulnerability in Flaticon, highlighting the risks associated with third-party components in the software supply chain.

In 2022, WooCommerce Plugin Vulnerability (2022) a serious SQL injection vulnerability was discovered in the WooCommerce Dropshipping by OPMC plugin for WordPress. This unauthenticated SQL injection flaw, which received a critical CVSS score of 9.8, allowed attackers to access sensitive information from the database without authentication.

In 2024, Boolka Cyberthreat Deploys BMANAGER Trojan (2024) a threat actor known as 'Boolka' was observed compromising websites through SQL injection attacks to deploy a standard trojan called BMANAGER. This campaign demonstrated the sophisticated tactics of cybercriminals who exploit SQL injection to spread malware.

These incidents highlight the ongoing threat posed by SQL injection attacks and the importance of implementing robust security measures, including regular code reviews, input validation, and the use of advanced security tools to detect and prevent such vulnerabilities

2.3 Evolution of Protection Systems

With the emergence of challenges resulting from SQL injection attacks, protection systems have gone through several generations [11],[9] :

2.3.1 First Generation: Simple Input Validation:

Programmers used programming language functions (such as isNumeric() in PHP or Java) to verify the type of inputs before inserting them into an SQL query. The processing was handled by excluding unsafe characters (quotation marks, semicolons, comments).

Weaknesses**:**

Non-comprehensive blacklists: attackers could bypass them by encoding characters or using alternative expressions (/**/ instead of space)

High False Negatives: Many new attacks are not detected.

2.3.2 Second Generation: Whitelisting and ORM:

Defining precise permissions for inputs (only letters and numbers in the username, for example) and preventing the entry of any character not specified in the list.

Weaknesses**:**

Does not cover all scenarios (such as injection in DDL commands or when using complex dynamic queries). May increase application complexity and performance in certain cases.

2.3.3 Third Generation: Web Application Firewalls (WAF):

Acts as an intermediary in front of the application, receiving all requests and applying ready-made (ModSecurity) or custom rules.

Weaknesses**:**

High false positives: Hinders user experience.

False Negatives: In facing customized or advanced attacks that do not conform to ready-made rules.

Need for periodic setup and maintenance of rules.

2.3.4 Fifth Generation: Hybrid and Intelligent Systems:

Modern systems combine signature-based detection techniques and behavioral analysis to increase detection accuracy and reduce false alarms.

2.4 System Definition

The proposed system is a security solution that relies on a Reverse Proxy server as an intermediate layer in front of the web server to intercept requests directed to it and route them to the scanning service to examine these requests before they reach the main server directly. The system primarily aims to protect databases from SQL injection attacks through:

Immediate Detection: Using machine learning algorithms and signature technology to detect suspicious requests.

Execution Prevention: Blocking requests containing harmful instructions before they reach the database.

2.5 System Importance

The proposed system offers innovative and comprehensive solutions to current security challenges, and its importance is manifested in the following points:

1. Protection of Sensitive Data and Ensuring Its Integrity:

Protecting databases containing sensitive information (such as customer data, financial records, and personal information). Preventing unauthorized access, data modification, and leakage, which enhances user confidence.

2. Immediate detection and prevention of attacks:

Using signature-based and anomaly detection techniques to discover malicious requests as soon as they arrive.

Isolating incoming requests through a reverse proxy server and preventing the execution of harmful commands before they reach the database.

3. Flexibility and ease of integration with existing systems:

Designing the system to integrate seamlessly with existing infrastructure with only minor modifications.
Supporting diverse environments (local and cloud), enhancing its scalability.

4. Adapting to evolving threats:

Confronting modern attacks that use advanced techniques (such as encryption and obfuscation) through machine learning algorithms that continuously update protection models.

5. Improving operational performance:

Reducing the impact of attacks on system performance and improving operational efficiency.

6. Enhancing organization reliability and security:

Helping comply with international security standards such as GDPR and ISO 27001, enhancing the organization's reputation and customer trust.

7. Reducing costs associated with cyber attacks:

Reducing losses resulting from data breaches, system recovery, and reducing reputational damage and legal fines.

8. Contributing to future security research:

Providing a model that can be developed as an open-source platform to enhance research and innovation in database security and expand the scope of protection to include other attacks such as XSS and DDoS

2.6 System Components

The proposed system consists of several main elements that work in an integrated manner to ensure system security:

- **Reverse Proxy Server:**

Acts as a firewall in front of the main server to prevent direct access to the server and reduce the risk of executing harmful commands. It intercepts requests and sends them to the scanning service, and based on the returned value, decides whether to pass or block them [8], [2].

- **Scanning Service:**

Consists of:

Signature-based Detection: Relies on matching requests with a database of known attack patterns or based on the libinjection library that does this.

Anomaly-based Detection: Uses machine learning algorithms to analyze behavioral patterns and detect deviations from normal behavior.

Alert Unit: Sends immediate notifications to administrators when suspicious activity is detected, including detailed reports about the nature of the attack and its source (such as IP address and execution time).

- **Management Interface:**

Allows administrators to monitor the system and activate or deactivate protection, make adjustments easily, with visual reports to track performance and analyze attacks.

2.7 Software Used

Python**:** For building the system due to its efficiency and availability of powerful libraries such as TensorFlow and Flask.

Flask**:** For building the API for the scanning service.

NGINX: For use as a Reverse Proxy server [2], and managing request routing and dynamic configurations using Lua.

MySQL**:** For storing data and analyzing requests.

React/JavaScript: For creating an advanced and secure web user interface.

Postman: For testing API during development.

2.8 Previous Studies

In October 2021, researcher Mohammed Nasereddin A Systematic Review of SQL Injection Attack Detection and Prevention Techniques (2021) and colleagues conducted a comprehensive systematic review of SQL injection attack detection and prevention techniques. The study examines various techniques used in this field and categorizes them into main classes, while analyzing the strengths and weaknesses of each approach. The research discusses signature-based techniques, anomaly-based techniques, and hybrid techniques, focusing on the evolution of these methods over time.

strengths:

This study provides a comprehensive and integrated overview of SQL injection attack detection and prevention, covering a wide range of techniques and methods used in this field. It presents a systematic classification that helps researchers and developers understand current and future trends. The strength of the study lies in its comparative analysis of different techniques, providing a solid foundation for making informed decisions about the best approaches to implement in various contexts. Additionally, the study offers valuable insights into current challenges and future directions in web application security.

weaknesses:

Despite its comprehensiveness, the study lacks practical and experimental application of the mentioned techniques, focusing primarily on theoretical review. The study doesn't offer specific solutions to future challenges, merely pointing them out. Furthermore, it doesn't deeply address emerging technologies such as deep learning and reinforcement learning in detecting SQL

injection attacks, which limits its ability to keep pace with the latest developments in this rapidly changing field.

In 2021, researcher Yasser Abdel Malik Using Machine Learning to Improve SQL Injection Attack Detection (2021) conducted a study focusing on developing an enhanced model for detecting SQL injection attacks using machine learning techniques to analyze behavioral patterns in SQL queries. The study addressed the problem faced by traditional protection methods that rely on syntax analysis or predefined rules, making them vulnerable to advanced attacks. The researcher proposed a model that extracts semantic features from SQL queries using machine learning techniques to improve attack detection accuracy.

strengths:

This study is distinguished by presenting an innovative approach that relies on extracting semantic features from data, thereby surpassing the limitations faced by traditional methods that depend on fixed rules. This approach gives the system greater ability to understand the context of queries and identify suspicious patterns with higher accuracy. The study also presents dynamic verification algorithms that ensure practical system performance in real operating environments, balancing detection accuracy with performance efficiency. The proposed model is also characterized by its ability to adapt to some new attack patterns through behavioral analysis, which represents a notable advancement compared to traditional systems with fixed rules.

weaknesses:

Despite the numerous advantages, the study suffers from some important limitations. Most notably, it is limited to analyzing previously documented attacks, which restricts its ability to deal with completely new or complex attacks that use sophisticated techniques for camouflage. Additionally, the study did not adequately address the problem of false positives, which can negatively impact user experience and system performance. Finally, the study lacked extensive testing in real production environments with high traffic, raising questions about the model's applicability in complex practical scenarios.

In 2012, researchers S. Fauzi Hidayat and Angela Geetha Using Reverse Proxy Server and Data Sanitization Techniques to Combat Attacks (2012) presented a study focused on developing a reverse proxy server that sanitizes inputs before sending them to the main server using dedicated algorithms for data sanitization. The study addressed the problem of SQL injection attacks and proposed a method for detecting and preventing them. The researchers implemented a system based on a reverse proxy server that uses an algorithm to sanitize inputs with the goal of reducing SQL injection attacks. The system tested on standard applications showed significant improvement in detecting and reducing these attacks.

strengths:

This study is distinguished by presenting a practical approach that relies on using a reverse proxy server as an intermediate protection layer, providing a non-intrusive solution that can be applied to existing applications without the need to modify source code. The study achieved a high detection rate of up to 100% in testing scenarios, demonstrating the effectiveness of the proposed approach in controlled testing environments. The study was also characterized by using signature verification techniques to analyze inputs and filter attacks, providing an effective protection layer against known attacks. Additionally, the study presented a scalable framework that can be developed to accommodate other types of attacks in the future.

weaknesses:

dspite the promising results, the study suffers from some important limitations. Most notably, it is limited to small environments and was not tested in high-traffic scenarios, raising questions about its applicability in real production environments. The study also did not include behavioral analysis algorithms to cover unknown attacks, limiting its ability to adapt to new and evolving attack patterns. Additionally, the study dates back to 2012, making it relatively outdated in the field of cybersecurity.

In 2025, researchers Hari Krishna A Hybrid Approach for SQL Injection Attack Detection Using Machine Learning Techniques (2025) and colleagues presented a study proposing a hybrid approach for detecting SQL injection attacks using a combination of machine learning techniques. The proposed approach combines Naive Bayes algorithm, Long Short-Term Memory (LSTM), and Random Forests to improve the accuracy of attack detection. These algorithms work sequentially, where data is extracted and analyzed through the successive application of Naive Bayes and LSTM algorithms, and their outputs are then used to feed the Random Forest classifier, resulting in improved accuracy in identifying potential threats.

strengths:

This study is distinguished by its innovative approach that combines multiple machine learning techniques, leveraging the strengths of each algorithm while overcoming their individual weaknesses. This integration leads to a significant improvement in detection accuracy compared to traditional methods. The use of LSTM gives the system the ability to understand context and sequence in data, which is crucial for detecting complex SQL injection attacks. The comprehensive tests conducted demonstrate the effectiveness of this approach, indicating its potential for application in real environments. Additionally, the study opens new horizons for applying integrated machine learning techniques in the field of cybersecurity in general.

weaknesses:

The study primarily focused on attack detection rather than prevention, leaving a gap in the area of proactive prevention. Finally, the model has not been sufficiently tested in real production environments, raising questions about its performance under real-world conditions.

In 2023, researchers John Irongu and colleagues presented Artificial Intelligence Techniques for SQL Injection Attack Detection (2023) a study exploring the use of artificial intelligence techniques for detecting SQL injection attacks. The study proposes a model based on advanced machine learning techniques, focusing on input string validation as an initial anomaly detector using pattern matching for SQL queries. The model was tested on one type of SQL injection attack, namely repetition attacks, and the model's performance was measured using three main

classifiers: Support Vector Machine (SVM) with 98.36% accuracy, K-Nearest Neighbors (KNN) with 96.29% accuracy, and Random Forests with 97.50% accuracy.

strengths:

This study is characterized by its innovative approach combining input validation techniques and advanced machine learning algorithms, providing an additional layer of protection against SQL injection attacks. The empirical results show very high accuracy in detecting repetition attacks, indicating the effectiveness of the proposed approach. The comparison between different classification algorithms provides valuable insights into their relative performance in the context of SQL injection attack detection. Additionally, the study suggests the possibility of integrating the proposed detection mechanism with Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), opening new horizons for integration with existing security systems.

weaknesses:

Despite the promising results, the study suffers from some limitations. First, the model was tested on only one type of SQL injection attack (repetition attacks), raising questions about its performance with other more complex types of attacks. Second, the study did not provide sufficient details on how to implement the model in real production environments.

Chapter3:system analysis

3.1 Introduction

this chapter focuses on the systematic analysis of the system requirements and the methodical gathering of specifications needed to develop an effective SQL injection detection and prevention system. The requirements gathering process is a critical phase in the software development lifecycle that ensures the final product meets the needs of users and stakeholders while addressing the security challenges identified in previous chapters. Through careful analysis of user needs, technical constraints, and security objectives, this chapter establishes the foundation for the system design and implementation phases that follow

3.2 Requirements Gathering

requirements gathering for this system will involve understanding the core functionalities needed for real-time detection, request interception, alerting, and administrative control. We gathered requirements through stakeholder interviews and analysis of security best practices (e.g. OWASP guidelines). From these discussions, we defined the functional needs (admin operations, detection behavior) and non-functional goals (performance, security, usability).

3.2.1 Functional Requirements

3.3.1 User (Administrative) Requirements

Login**:** The dashboard requires secure admin authentication before access.

Start/Stop inspection Service: The owner can start or stop the Python inspection engine via the UI.

Configure protected site: The owner can set or update the IP address and port of the backend web application.

View alert logs: The dashboard displays a list of logged SQL injection alerts (including

timestamp, request details, attacker IP).

Set Alert Notifications: The owner can enter a phone number and/or email; on detection, alerts (SMS/email) will be sent to these contacts.

Select detection method: The owner can toggle the inspection mode among signature only, anomaly-only, or both.

Change Password: The owner can change the admin password within the dashboard interface

3.3.2 System Requirements

HTTP iequest interception: NGINX must proxy all incoming HTTP/HTTPS requests to the inspection service.

Request inspection: The Python service analyzes each request's parameters and body for SQL injection.

Signature-Based detection: The service maintains a database of known SQLi patterns (e.g. common keywords, regexes) and flags any matching requests.

Anomaly-Based Detection: The service learns normal request characteristics (profiles) and flags unusual queries.

Forward or block: If a request is clean, the inspection service returns HTTP 200 to NGINX, which then forwards it to the protected site. If an injection is detected, the service returns HTTP 403; NGINX blocks the request and does not forward it.

Alerting: On each detected injection, the service triggers an alert mechanism (e.g. calls an SMS gateway or email API) to notify the configured contact immediately.

Logging: Each blocked attempt is recorded in a database (audit log) with details (timestamp, request data, detection method). Logs must be write-protected and include enough data to trace the event

Configuration storage: The system persistently stores the protected site's IP/port, the alert contacts, and detection method settings.

Dashboard control: The dashboard UI interacts with the backend to apply configuration changes (e.g. updating settings, restarting the service) and fetch logs for display. Authentication: Access

to the dashboard and system operations requires valid admin credentials; password data must be handled securely (e.g. hashing).

3.4 Non-Functional Requirements

3.4.1 User (Administrative) Requirements

Usability: The dashboard interface should be intuitive and responsive, with clear feedback on actions (e.g. confirmation when the inspection service starts).

Reliability**:** The admin must be able to rely on the dashboard being available (e.g. quick startup after crashes). The UI should handle errors gracefully.

Customization: Notification preferences (SMS/email) should be easy to modify. The system should allow the admin to tune detection without redeploying code.

3.4.2 System Requirements

performance**:** The inspection service must handle peak traffic loads with minimal added latency.. Throughput and latency targets should ensure user requests remain fast

scalability: the design should allow scaling (e.g. running the service on multiple cores or machines). The system should "adapt as your organization grows" by handling increasing request rates.

Security: The system must be secure by design. All communications should use encryption (HTTPS for the dashboard; secure channels for SMS/email APIs). The admin password and any stored credentials must be hashed/encrypted. The inspection service should run with least privilege. NGINX and the host OS should be hardened (restrict ports, keep software up-to-date). Input validation must be robust to prevent bypass (e.g. ensure URL-decoding before analysis). The log database should prevent unauthorized access or tampering.

Reliability/Availability: The system should have high availability for continuous protection. Possible measures include running NGINX in active-passive mode or using clustering for the inspection service. The service should fail-safe (e.g. if the inspection engine crashes, NGINX could default to blocking all requests or fail open if explicitly configured).

Maintainability: The code should be modular (separate detection logic, logging, and alerting components) to ease updates. Use of Python and common libraries should facilitate future development. Configuration (e.g. signature lists) should be easy to update without code changes.

Auditability: Detected events and admin actions should be auditable. Logs should be stored securely (e.g. encrypted) and retained per policy. As one best practice advises, retain logs long enough to meet compliance (e.g. 90+ days).

Compliance: If applicable, the system should meet relevant regulations (e.g. data retention rules). For instance, logging user actions and protecting logs are often compliance requirements.

3.5 Machine Learning Model Development Methodology:

This Section details the methodology for developing a machine learning model by employs specific natural language processing and classification techniques to differentiate between benign and malicious SQL queries, focusing on Term Frequency-Inverse Document Frequency (TF-IDF) for feature extraction and Logistic Regression for classification.

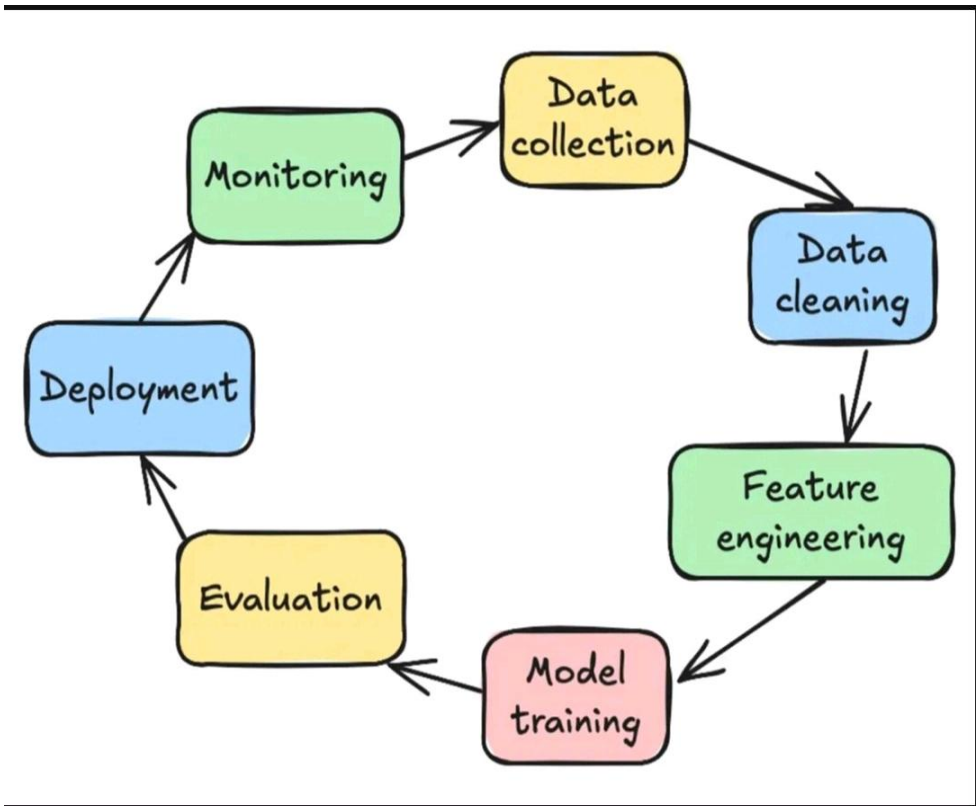

Figure 0- 1

The initial step involves acquiring a suitable dataset containing SQL queries. For this study, the "SQL Injection Dataset" sourced from Kaggle was utilized. This dataset is specifically designed for training and evaluating models for SQLi detection. It contains a total of 30,905 SQL query records, each labeled as either malicious (SQL injection) or legitimate (benign). The dataset includes 19,537 malicious queries (Label = 1) and 11,368 legitimate queries (Label = 0). An example of a malicious query entry is `select * from users where id = '1' or @1 = 1 union select 1,version() -- 1`. The dataset was reported as clean, with no missing values, making it ready for preprocessing and model training [3]. The quality and representativeness of this dataset are fundamental to the model's effectiveness [3].

3.3 Data Preprocessing

Raw data requires preparation before it can be used for model training.

3.3.1 Data Cleaning

While the acquired dataset was reported as clean, a standard verification step ensures data consistency. This typically involves handling any potential missing or incomplete records. Queries lacking either the SQL string or its corresponding label (benign/malicious) would be removed to ensure only complete data points are used [1].

3.3.2 Label Encoding

The categorical labels (benign/malicious) need to be converted into a numerical format suitable for machine learning algorithms. This involves encoding 'benign' as 0 and 'malicious' as 1, aligning with the dataset's structure [1].

3.4 Feature Engineering: TF-IDF

Machine learning models require numerical input, necessitating the transformation of textual SQL queries into numerical vectors.

3.4.1 Rationale for Feature Engineering

Raw text cannot be directly processed by algorithms like Logistic Regression. Feature engineering techniques convert text into a quantitative format that the model can understand and learn from [4].

3.4.2 TF-IDF Calculation

Term Frequency-Inverse Document Frequency (TF-IDF) is employed here. It calculates a weight for each word (term) in a query, reflecting its importance within that query relative to the entire dataset (corpus). Term Frequency (TF) measures word frequency within a query, while Inverse Document Frequency (IDF) down-weights common words appearing across many queries [1].

3.4.3 Vector Representation

The TF-IDF process results in each SQL query being represented as a numerical vector. Each element in the vector corresponds to a word in the dataset's vocabulary, holding the calculated TF-IDF score for that word in the specific query [1].

3.5 Model Selection: Logistic Regression

After feature engineering, a classification algorithm is chosen to learn the patterns distinguishing malicious from benign queries.

3.5.1 Algorithm Choice

Logistic Regression is selected for this task. It is a well-established statistical model suitable for binary classification problems and has been applied effectively in SQLi detection [1, 4]. Its interpretability and efficiency are advantageous.

3.5.2 Model Mechanism

Logistic Regression models the probability of a query being malicious using the sigmoid function applied to a linear combination of the input features (TF-IDF scores). It learns weights for each feature during training to make predictions [4]. A threshold (commonly 0.5) is used on the output probability to classify the query.

3.6 Model Implementation

This phase involves preparing the data splits and training the selected model.

3.6.1 Data Splitting

The preprocessed dataset (TF-IDF vectors and labels) is divided into training and testing sets. A standard split (e.g., 80% training, 20% testing) is often used. Stratification ensures that the proportion of benign and malicious queries is similar in both sets, preventing biased evaluation [5].

3.6.2 Model Training

The Logistic Regression classifier is trained using the training set. The model learns the relationship between the TF-IDF features and the query labels (benign/malicious) by optimizing its internal weights [1].

3.5.7 Model Evaluation

Assessing the trained model's performance on unseen data is critical.

3.5.7.1 Evaluation Metrics

Standard classification metrics are used for evaluation, based on the confusion matrix elements: True Positives (TP - correctly identified malicious queries), True Negatives (TN - correctly identified benign queries), False Positives (FP - benign queries incorrectly identified as malicious), and False Negatives (FN - malicious queries incorrectly identified as benign) [2, 5].

Accuracy: Measures the overall proportion of correct predictions (both malicious and benign).

Equation: Accuracy = (TP + TN) / (TP + TN + FP + FN)`

Precision:Measures the proportion of queries classified as malicious that are actually malicious. High precision indicates a low false positive rate.

Equation: Precision = TP / (TP + FP)

Recall (Sensitivity): Measures the proportion of actual malicious queries that were correctly identified by the model. High recall indicates a low false negative rate.

Equation: Recall = TP / (TP + FN)

F1-Score: Provides a balanced measure between Precision and Recall, calculated as their harmonic mean. It is useful when class distribution is uneven.

Equation: `F1-Score = 2  (Precision Recall) / (Precision + Recall)`

These metrics provide different perspectives on the model's correctness and its ability to identify malicious queries accurately while minimizing misclassifications [2, 5].

3.7.2 Performance Assessment

The model's performance is measured using the reserved testing set and the metrics defined above. This provides an unbiased estimate of how well the model generalizes to new, previously unseen SQL queries [3].

3.5 Use Case Diagram

3.5.1 Administrator use case diagram

This use case diagram outlines how the Admin interacts with the control panel. It covers key actions: logging in, starting or stopping the service, configuring the protected backend address, selecting inspection modes, setting up alert notifications, viewing detected attacks, and changing the account password.By mapping these interactions, we ensure real-time SQL injection detection.

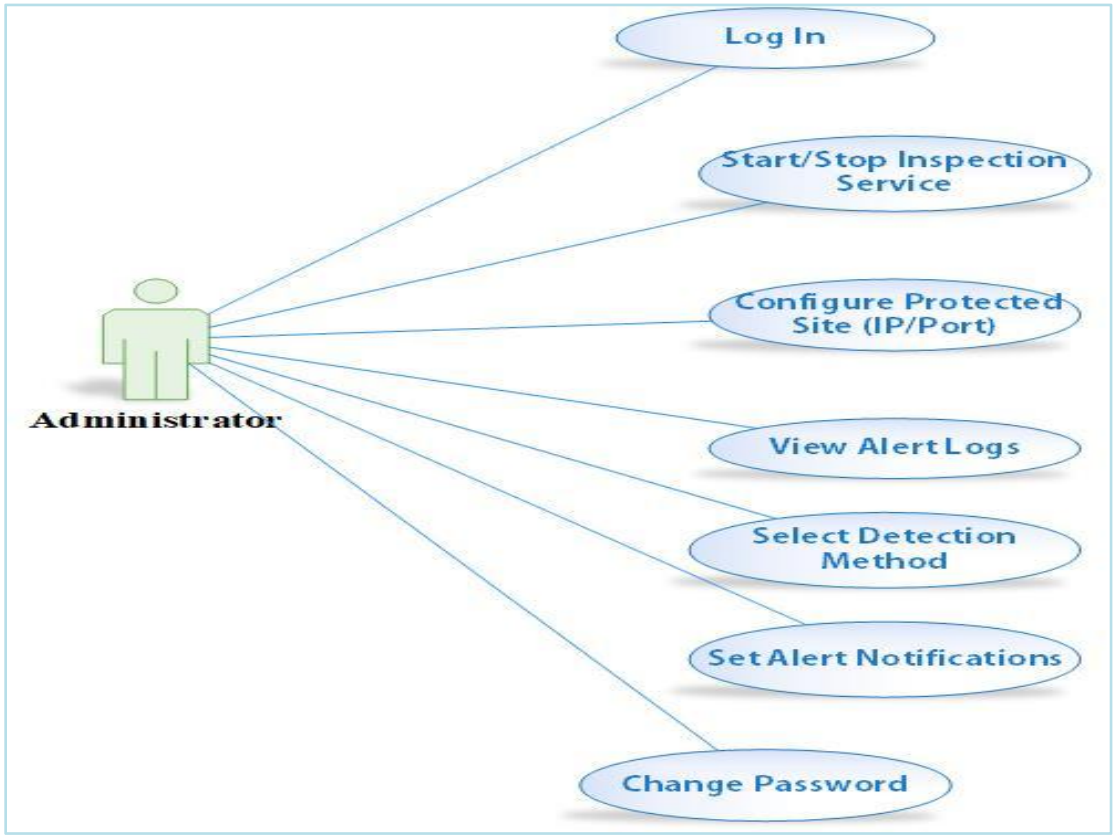

Figure 0- 2 Administrator use case diagram

3.5.2 Client Use case

This use case diagram outlines how the client interacts with the website

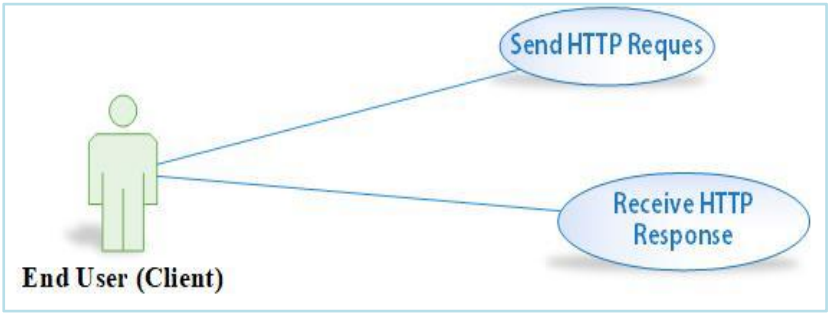

Figure 0- 3 Client Use case

3.5.3 System use case

This use case diagram outlines how the system interacts.

NGINX intercepts all incoming HTTP/HTTPS requests and forwards them to the Inspection Service for real-time inspection.

The Inspection Service analyzes each request (using signature and/or anomaly detection), returns HTTP 200 for safe traffic or HTTP 403 for detected SQL injections, logs any attacks in the database, and sends immediate SMS/email notifications when a threat is found.

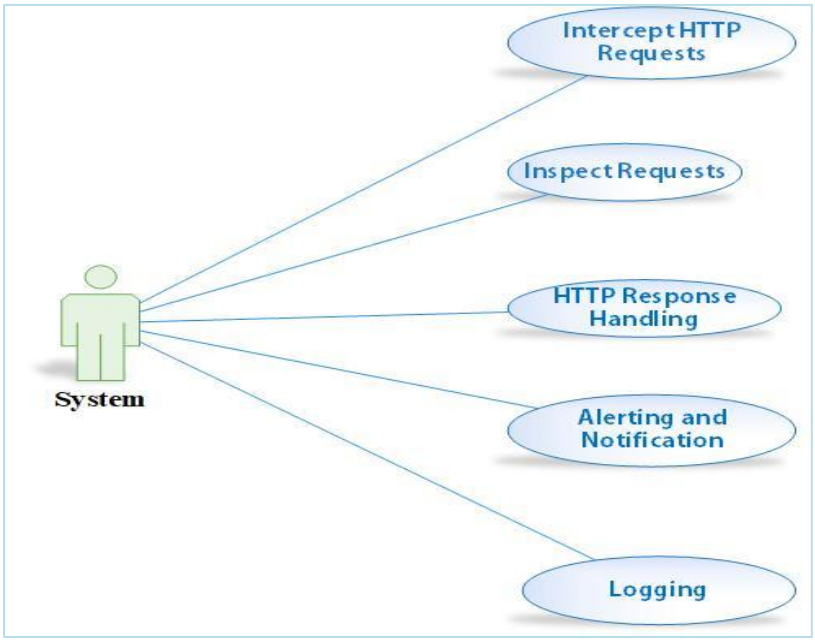

Figure 0- 4 System use case

Actor	Description
Administrator	Uses the dashboard to configure and control the system.
End User (Client)	Sends HTTP requests to the protected website. (This request is automatically handled by the system.)
System	Internally performs Inspect Request

Table 2 Use Case Diagram

3.6 Sequence diagram

This sequence diagram illustrates how the Admin interacts with the Dashboard, backend services, The Admin submits credentials to login which invokes AuthService and the Database to verify the UserAccount before granting access. When the Admin starts the service or requests alert logs, Dashboard forwards the commands to ServiceController or AlertLogDAO, respectively, triggering Inspection Service to run or retrieving stored AlertLog entries from the Database for display.

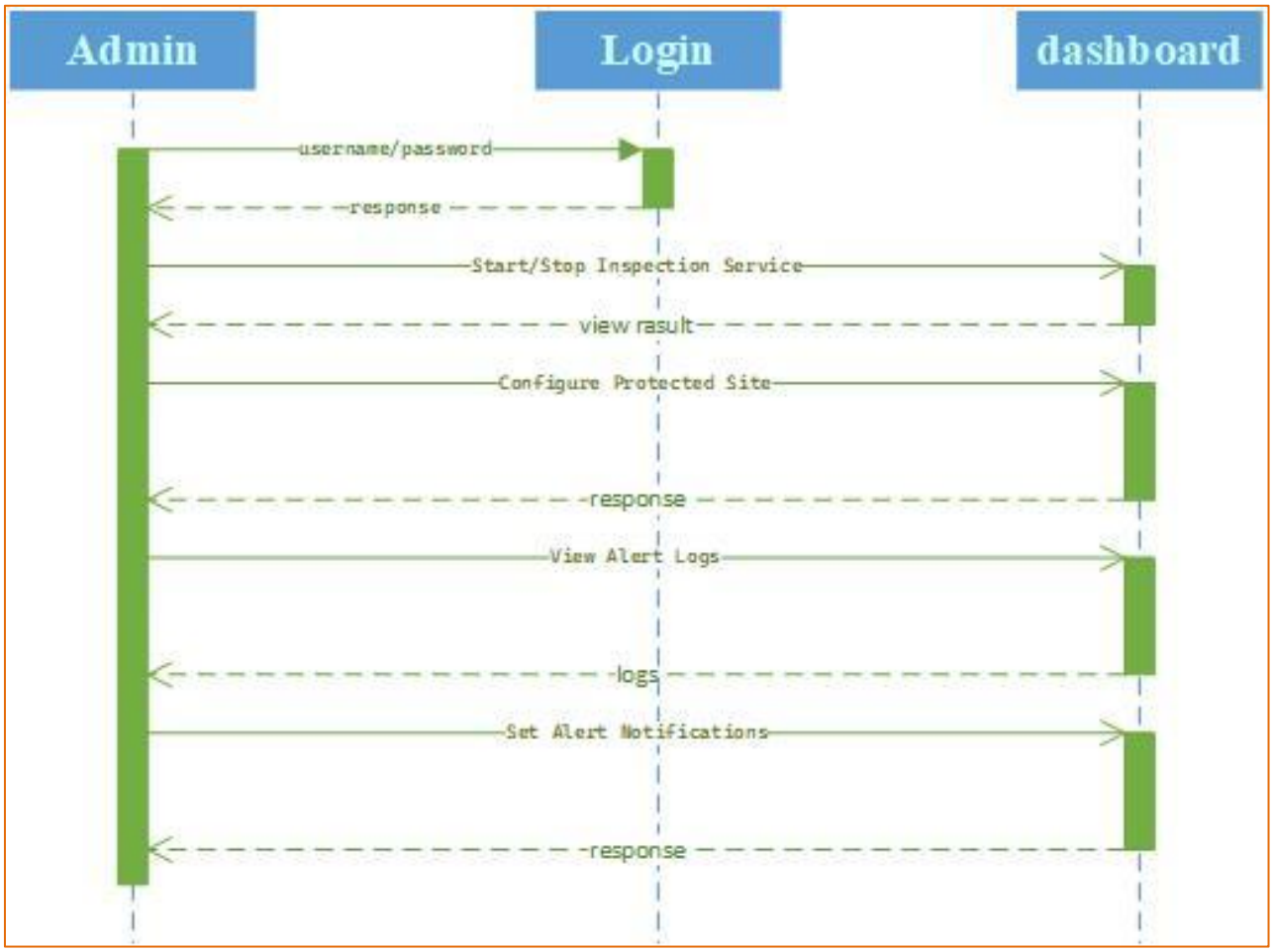

Figure 0- 5 Sequence diagram

3.7 Activity diagram

3.7.1 Administrator Activity diagram

The activity diagram shows the Admin's interaction with the Inspection Service: the Admin logs in, and once authenticated, the Admin can perform the specified functions.

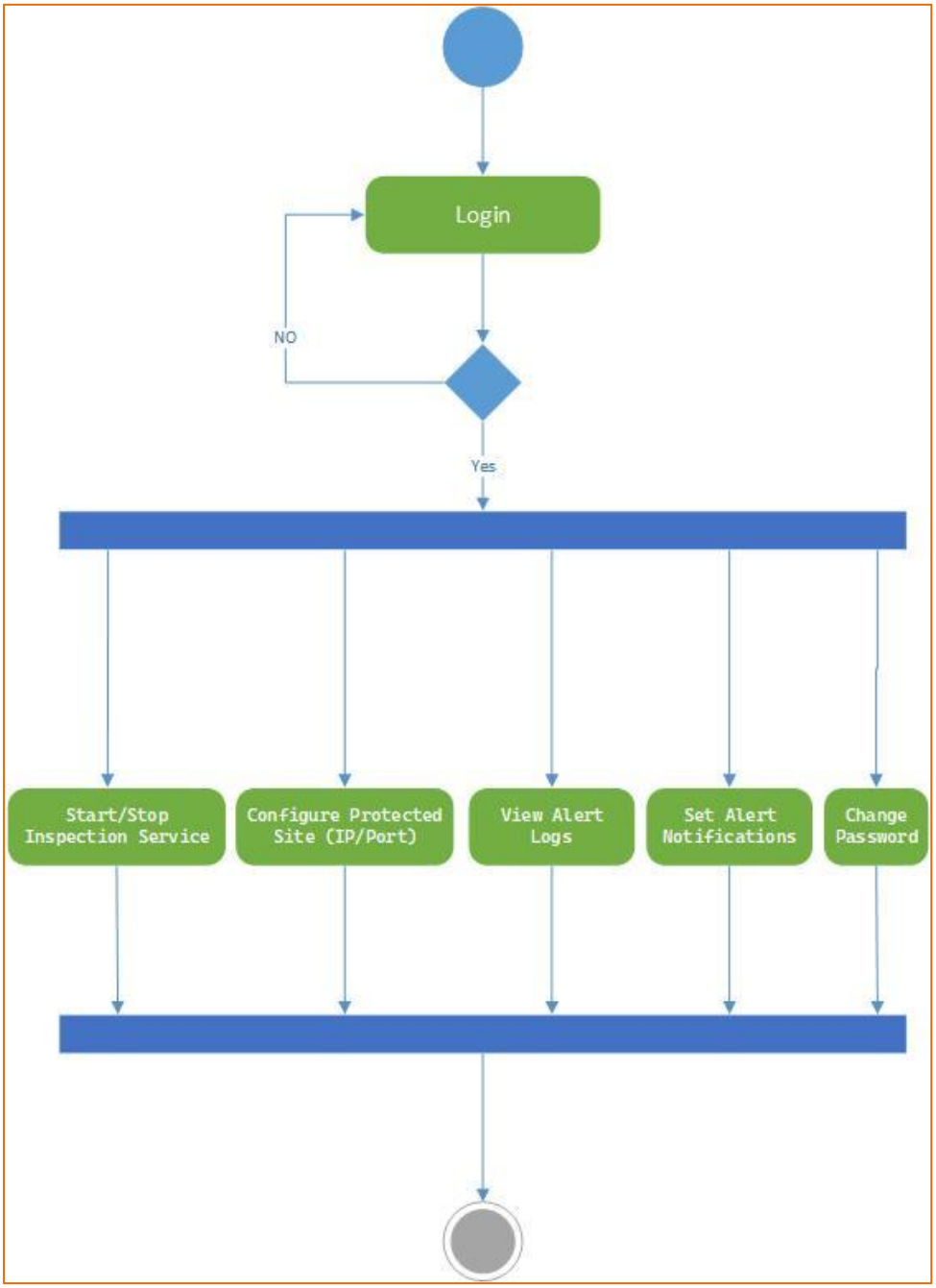

Figure 0- 6 Administrator Activity diagram

3.7.2 System Activity diagram

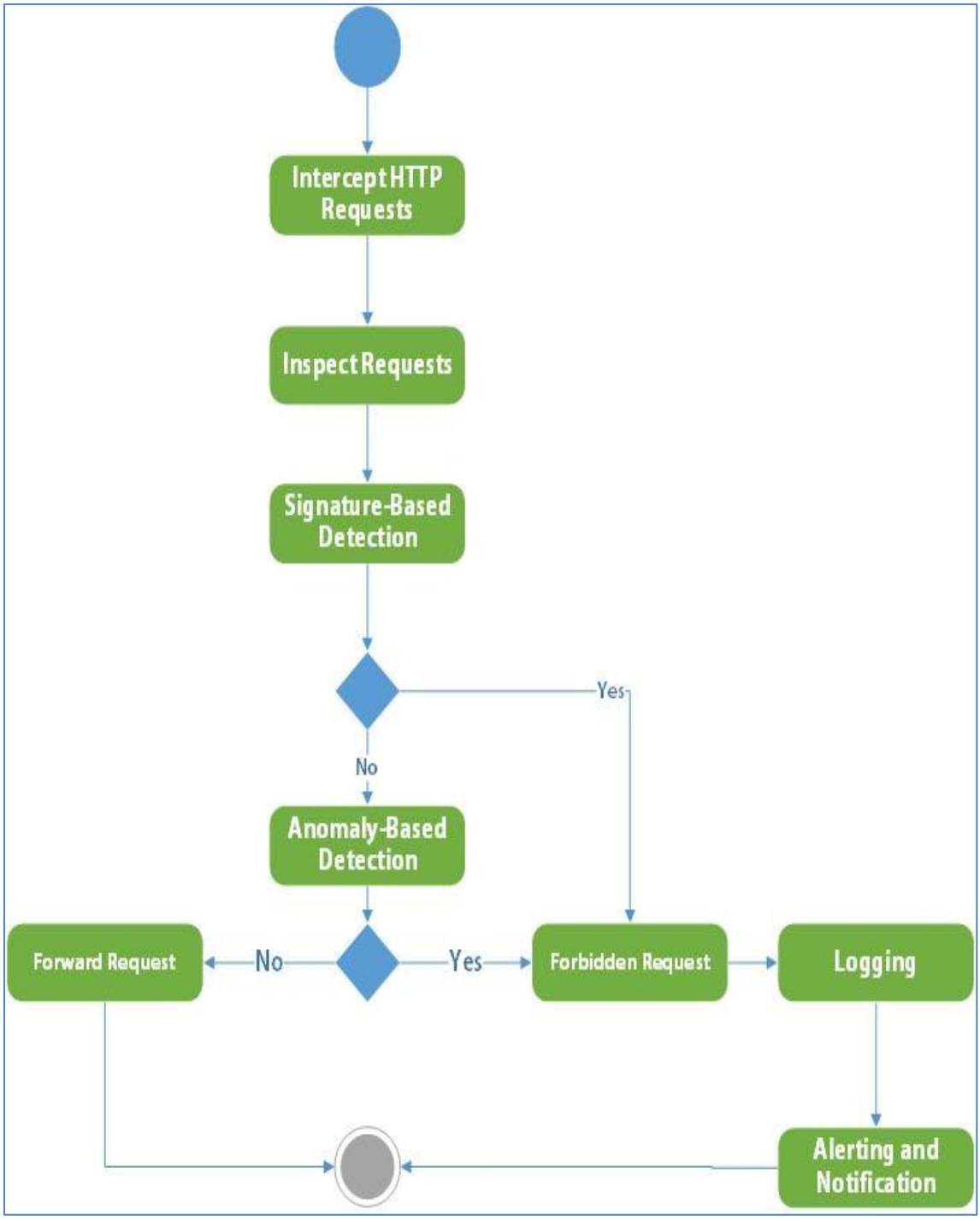

Figure 0- 7 System Activity diagram

3.8 Class diagram

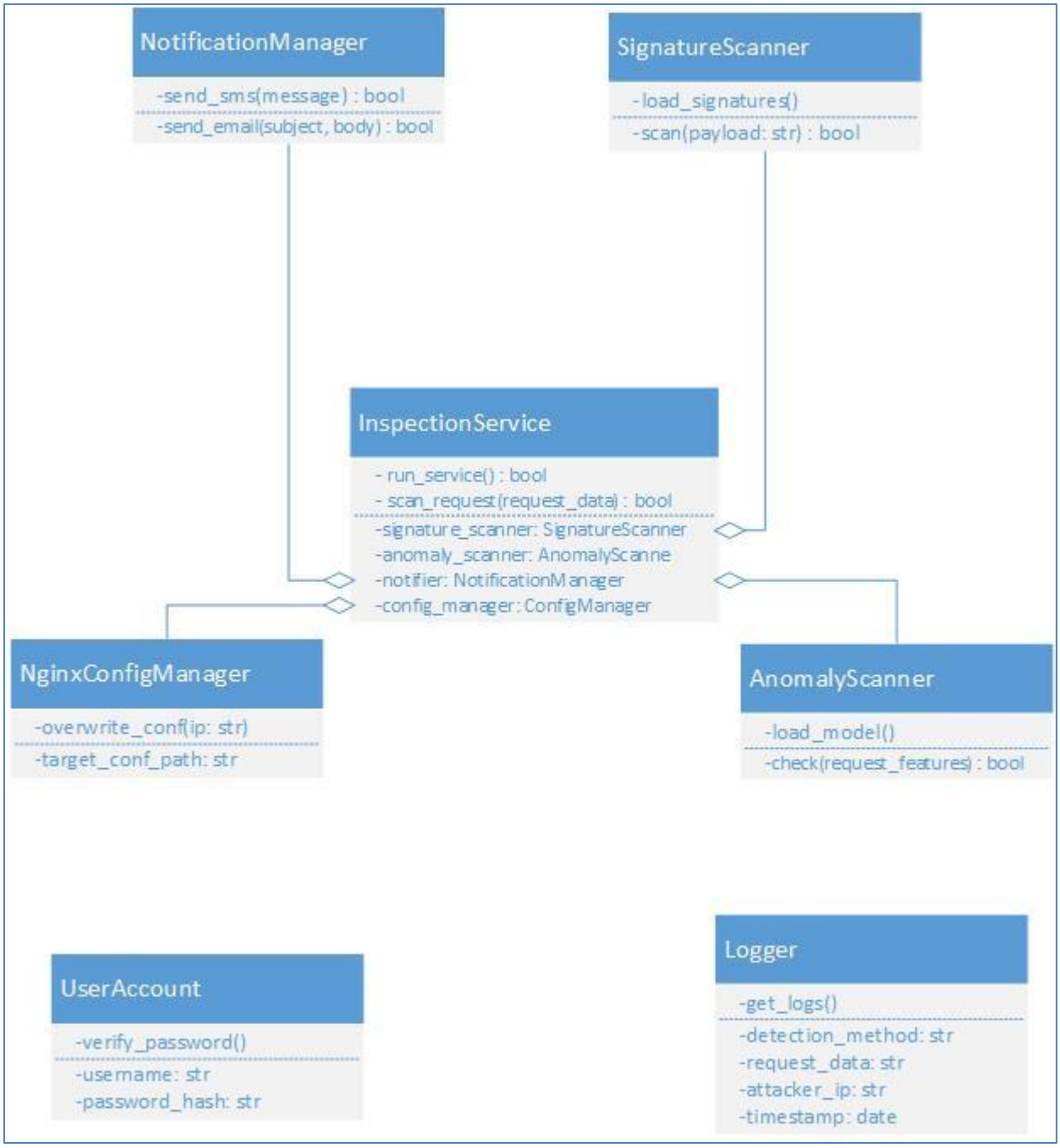

Figure 0- 8 Class diagra

3.9 Requirements Description

Function	Description
Login	The system shall authenticate the administrator using a username/password. Passwords must be stored hashed (e.g. bcrypt) to prevent compromise. On login, the user establishes a secure session for accessing the dashboard.
Start/Stop Inspection Service	Through the dashboard, the administrator can start or stop the Python inspection service. This may trigger system commands or signal the service process. The UI must reflect the current state (running/stopped) and confirm successful action. If the service is stopped, NGINX should reject (or queue) all incoming requests to avoid unprotected traffic
Configure Protected Site (IP/Port)	The administrator provides the IP address and port of the real web application. The system validates this input (correct format, reachability) and updates its configuration. Upon change, the system ensures that NGINX's proxy_pass target is updated so that clean traffic is forwarded to the correct location.
View Alert Logs	The dashboard displays the log of detected injection attempts. Each log entry should include at least: timestamp, client IP, request details (e.g. URL, payload), detection method (signature or anomaly), and action taken. This view should allow filtering/sorting by time or severity. The backend query to the database for logs must be efficient and secured against injection (use parameterized queries).
Set Alert Notifications	The administrator can enter a phone number and/or email address to receive alerts. The system validates these (proper format). When a SQL injection is detected, the inspection service sends a notification (e.g. via an email or SMS API). The system must handle failures (e.g. retry or log if an alert could not be sent).

Select Detection Method:	The administrator chooses among Signature, Anomaly, or Both. The system shall activate the corresponding logic: if Signature only, it ignores anomaly logic; if Anomaly only, it relies on behavioral profiling; if Both, it flags any request caught by either method. Changing this setting should take effect without restarting the server (dynamic reconfiguration).
Change Password	The administrator can change their dashboard password. This action must require entry of the old password (to prevent unauthorized change), validate the new password strength, and then store the new password securely (e.g. with a strong hash and salt).

Table 3 Requirements Description

Function	Description
Intercept HTTP Requests	NGINX is configured as a reverse proxy (using the proxy_pass directive) to send all incoming requests to the Python service . This means the system always examines every client request before it reaches the actual web application. The system must handle all standard HTTP methods (GET, POST, etc.) and possibly HTTPS (using SSL termination at NGINX).
Inspect Requests	Upon receiving a proxied request, the Python service parses the input (parameters, headers, body) and applies detection logic. For signature detection, it compares against a list of known malicious patterns (e.g. common SQL meta-characters or keywords). For anomaly detection, it compares request features to a learned profile of normal traffic. If neither method flags the request, the service responds with HTTP 200 to NGINX, indicating safe passage. If an injection is detected, the service responds with HTTP 403, and logs/alerts are triggered
Signature-Based Detection	The system should include or allow updates of a signature database. Known SQL injection patterns (like ' OR '1'='1 , UNION SELECT , comment sequences, etc.) must be matched quickly. This rule set must be maintained (regular updates) because signature-based detection is static

	and needs current threat intelligence
Anomaly-Based Detection	The system should maintain a statistical or ML-based model of normal requests (IP source, URL frequency, parameter entropy, etc.). Deviations beyond a threshold (e.g. unusually long query strings or unexpected SQL syntax) trigger alerts. Over time, the model should be updated with normal traffic (avoiding learning malicious behavior). This allows detecting new or obfuscated injection attempts that signatures miss
HTTP Response Handling	NGINX passes through the Python service's HTTP status. A 200 OK from the service causes NGINX to proxy the request to the protected site. A 403 Forbidden from the service causes NGINX to immediately return 403 to the client and not forward the request. Optionally, NGINX may serve a custom error page on 403 to inform users that their request was blocked.
Alerting and Notification	When the service detects an injection, it should immediately generate an alert record and attempt notification. This may involve calling an external API (e.g. Twilio REST API to send SMS, or an SMTP server to send email). The requirement implies configurable contact info, so the service retrieves the admin's chosen phone/email from its config and sends a message including details of the event. If the alert cannot be sent, the failure should be logged for administrator review

Table 4 System Requirements Description

Function	Description
Logging	All injection detections (and optionally all requests) must be logged to a secure database. Each log entry should include key context: timestamp, requesting IP, request URI and payload (sanitized as needed), which detection rule triggered, and action taken. This follows audit log best practices: logs help “identify suspicious activity and potential security breaches” The log storage must prevent tampering (e.g. write-once tables or append-only files) and support queries by the dashboard.
Configuration and Persistence	The system must save settings (site IP/port, alert contacts, chosen detection mode) persistently (e.g. in a configuration file or database). These settings should be loaded on startup and whenever changed. Misconfiguration (like an invalid IP) should generate an error and not be applied until fixed.
Service Control	The backend inspection service should run as a background daemon. The dashboard’s start/stop actions interface with the OS (for example, via systemd, supervisor, or a direct subprocess call) to manage this service. The system should ensure that NGINX and the service stay in sync (e.g. if the service stops, NGINX should detect the failure and either refuse connections or attempt to restart it).
Security Controls	The system should implement input validation for all interfaces. The dashboard should prevent CSRF and enforce session timeouts. The inspection engine itself should be hardened to avoid being crashed by large inputs (e.g. limit request size). Access to the database should be secured (use parameterized queries, restrict user privileges). All components should run under non-root accounts. Regular backups of logs and configurations are recommended to ensure recovery from failure.

Table 5 System Requirements Description

chapter4:Design

4.1 Introduction

This chapter represents the fundamental foundation upon which our system development process will be built. The importance of this chapter lies in converting the functional and non-functional requirements that we have previously identified into a tangible and applicable design. In this chapter, all aspects of the proposed system design will be comprehensively reviewed and documented. The chapter will begin by presenting the general architecture of the proposed system, explaining the main components and the interactive relationship between them, including the role of the reverse proxy server and the attack detection unit. Then, we will move on to database design and design of user interfaces necessary for system management and performance monitoring, as well as the design of informative messages and error messages through which the system interacts with the user or administrator. reports and queries necessary for monitoring system performance and analyzing detected attacks will be designed. Finally, the design of programming procedures and core algorithms on which the system depends for its operation will be detailed

4.2 System Architecture

In designing this system, I adopted a multi-layered architecture, as shown in Figure (10), specifically designed to efficiently detect SQL injection attacks and provide proactive protection for targeted web applications. This architecture consists of several key components that integrate and interact with each other to ensure the desired goals are achieved:

1. Interface and Interception Layer (Nginx - Reverse Proxy): The first entry point to the system is a web server acting as a reverse proxy. I chose Nginx technology to perform this vital role. This server receives all HTTP requests directed to the protected application and systematically passes them to the central processing layer for the necessary security checks before allowing them to reach their final destination. To ensure the full context of the request is available at the inspection layer, the reverse proxy was configured to enrich requests with custom headers containing essential information such as the user's original IP address and the actual request data. Based on the inspection results it receives from the processing layer, the reverse proxy makes the final decision to either block the suspicious request or allow the legitimate request to pass.

2. Central Processing Layer (Web Application - Flask): The web application developed using the Flask framework represents the backbone of the system and the platform for executing core logic. This application was built with a focus on flexibility and scalability, and divided into separate logical units to organize responsibilities. The application includes multiple Application Programming Interfaces (APIs), most notably an endpoint dedicated to receiving requests from the interception layer and subjecting them to the screening process. It also provides other interfaces for managing system configuration, reviewing security logs, and controlling the protection service status. Additionally, the application includes a specialized unit for managing authentication processes and access control for administrative users, directly interacts with the data layer to store and retrieve vital information, relies on the screening unit to perform security analysis, and uses alert mechanisms to notify administrators of suspicious activities.

3. Security Screening and Analysis Unit: A specialized unit was designed to implement the core logic of the SQL injection detection process. This unit analyzes data received from the central processing layer and returns an assessment indicating the level of risk associated with the request.

4. Data Layer (Database): To ensure data continuity and efficient management, a relational database was utilized. Object-Relational Mapping (ORM) technology represented by the SQLAlchemy library was used to abstract the interaction with the database, facilitating data handling and providing flexibility in choosing the underlying database engine (whether SQLite for development or more powerful systems for production). This database stores information about administrative users, detailed security logs, and system configuration settings.

5. Presentation Layer (User Interface): A Graphical User Interface (GUI) built on standard web technologies (HTML and React templates) was provided, allowing authorized administrative users to interact with the system easily. This interface provides functions for secure login, monitoring detected attack logs, modifying various system settings (such

as service status, alert policies, and protected server details), in addition to managing user accounts.

Data Flow and Control Mechanism: The process begins with an HTTP request arriving from the end user to the interface layer (Nginx), which passes it to the dedicated screening endpoint in the central processing layer (Flask). This layer extracts relevant data and passes it to the security screening unit. After analysis, if the request is identified as a potential attack, the processing layer logs the incident details in the database, sends an alert (if the policy is enabled), and returns a blocking response to the interface layer. If the request is legitimate, the processing layer returns an allow response. Based on this response, the interface layer either blocks the request or passes it to the targeted original server. In parallel, the administrator can use the user interface to monitor and configure the system, directly affecting the behavior of the processing layer and the database.

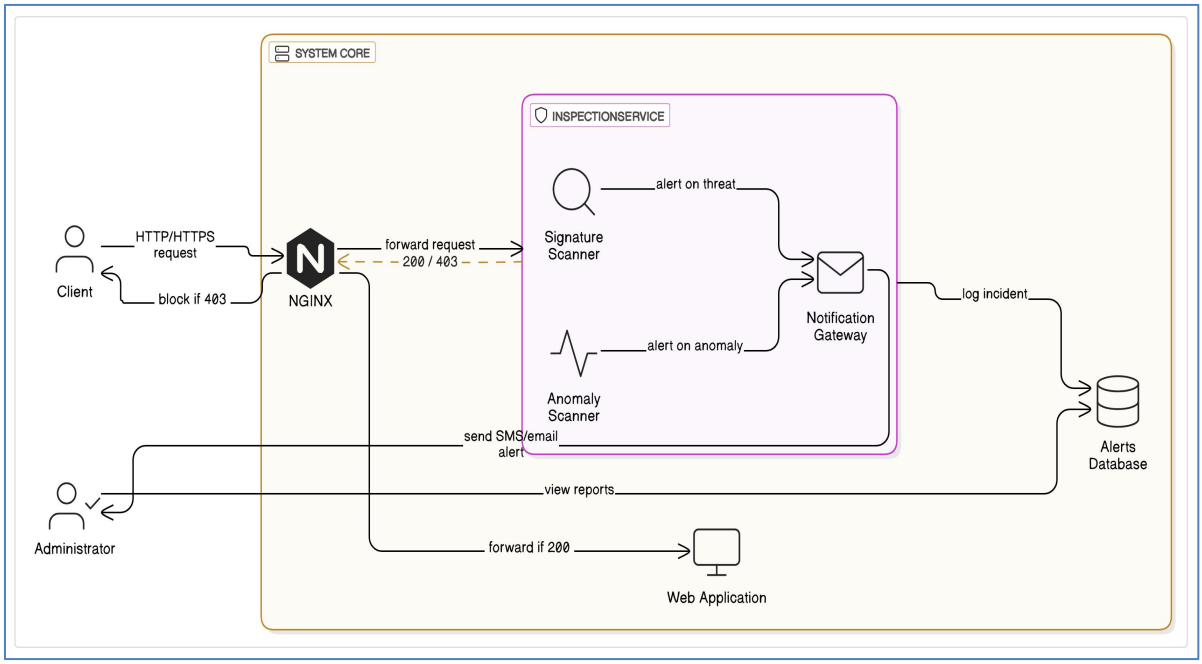

Figure 0- 1 System Architecture

4.3 Design system tables

To ensure efficient and secure management of system data, I designed a relational data model implemented using Object-Relational Modeling (ORM) technology via SQLAlchemy. This

approach allows for structured programmatic manipulation of data structures and facilitates adaptation to various database engines. The model consists of three carefully designed basic tables:

1. The User table: This table, detailed in Table (6), is dedicated to managing the identities of users authorized to access the system. It includes essential fields for authentication and account management, including a unique user ID, unique identification data (username and email), and a critical field for storing a secure password hash using strong and recognized hash algorithms. This design ensures credential protection and provides a solid foundation for access control mechanisms.

Field Name	Description	Data Type	Size	Notes / Constraints
id	User identifier	INTEGER	–	PRIMARY KEY, AUTOINCREMENT
username	Username	VARCHAR	64	NOT NULL, UNIQUE
email	Email address	VARCHAR	120	NOT NULL, UNIQUE
password_hash	Password hash	VARCHAR	256	NULLABLE

Table 6 Users table

2. Log Table: This table, detailed in Table (7), serves as a comprehensive security audit log for the system. It is designed to document every suspicious activity or attack detected by the scanning mechanisms. Each log entry includes rich contextual information, such as the unique identifier of the event, the method or rule that led to the detection, a copy of the suspicious request data, the source IP address, and the exact timestamp of the event. This design provides a detailed historical record that can be used for security analysis, incident response, and system effectiveness assessment.

Field Name	Description	Data Type	Size	Notes / Constraints
id	Log identifier	INTEGER	–	PRIMARY KEY, AUTOINCREMENT

detection_method	Detection method used	VARCHAR	50	NOT NULL
request_data	Raw request data	TEXT	–	NOT NULL
ip	Detection score/value	FLOAT	–	NOT NULL
timestamp	Detection timestamp	DATETIME	–	DEFAULT CURRENT_TIMESTAMP

Table 7 Registration table

3. Configuration table: To provide the flexibility to configure system behavior, the settings table is designed, as shown in Table (8). This table stores a set of parameters that the administrator can modify, such as the contact details of the protected server, the operational status of the protection service, and the policies to activate the various alert mechanisms and the identification of recipients. This table enables centralized and flexible management of system configuration, enabling it to adapt to the requirements of changing operating environments. This database-organized design, based on the specific ORM technology and tables, provides a strong foundation for efficient and secure system data management, and effectively supports basic monitoring, training and protection functions.

Field Name	Description	Data Type	Size	Notes / Constraints
id	Configuration identifier	INTEGER	–	PRIMARY KEY,AUTOINCREMENT
server_ip	Protected server IP address	VARCHAR	50	NOT NULL, DEFAULT '127.0.0.1'
server_port	Protected server port	INTEGER	–	NOT NULL, DEFAULT 80
service_active	Flag indicating if service is active	BOOLEAN	–	NOT NULL, DEFAULT FALSE
email_alerts	Email alerts enabled	BOOLEAN	–	NOT NULL, DEFAULT

	flag			TRUE
sms_alerts	SMS alerts enabled flag	BOOLEAN	–	NOT NULL, DEFAULT FALSE
email_recipient	Recipient email for alerts	VARCHAR	120	NULLABLE
phone_number	Recipient phone number for SMS alerts	VARCHAR	20	NULLABLE

Table 8 Configuration table

4.4 System screen design

In an effort to provide a seamless and efficient user experience for managing the SQL detection and prevention system, I designed a set of graphical user interfaces that allow the administrator to have complete control over the system with ease. The design focused on clarity and easy access to essential information and functions. Below is a review of the main screens I designed:

Login Screen: This screen is a secure login to the system control panel. As shown in Figure (11), I designed it to be simple and straightforward, requiring the administrator to enter basic credentials (username or email and password). Input validation mechanisms are included, and clear error messages are included if the authentication process fails. The design of this screen ensures that access to sensitive system functions is restricted to authorized users only, enhancing security.

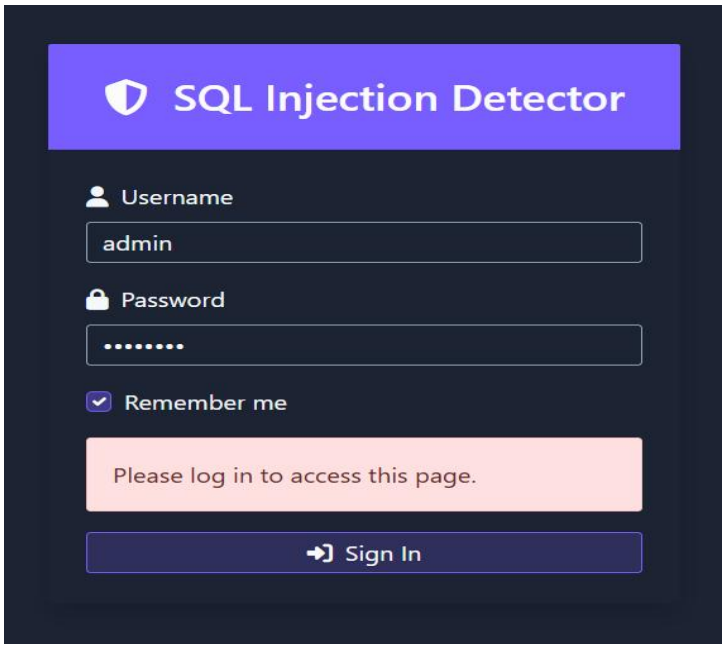

Figure 0- 2Administrator login interface

2.Dashboard:

After successful login, the administrator is directed to the main dashboard. This screen is designed to provide a quick overview of the system's status and latest activity. It typically displays a summary of the latest attacks detected, key performance indicators, or the status of the protection service (activated/disabled). Items are logically organized for immediate monitoring, with quick links to more detailed sections such as attack logs or the settings page.

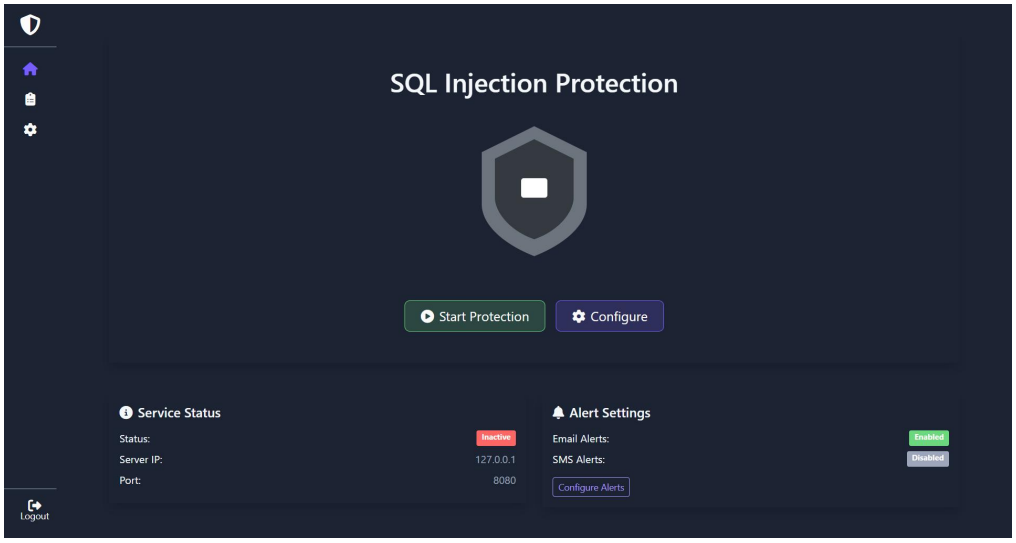

Figure 0- 3 Main control interface

Configuration window where you set the IP/port on which the web application runs in order to connect the reverse proxy with this setting As shown in Figure (13) .

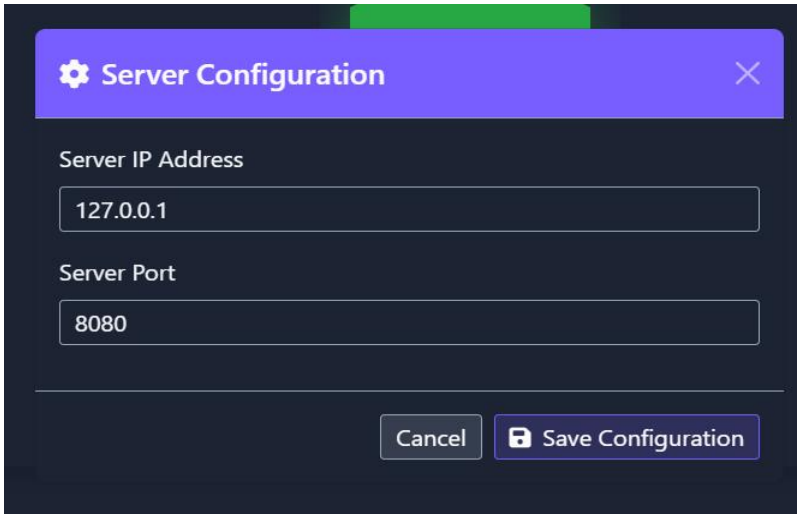

Figure 0- 4 Proxy Configuration Settings Window

Attack Logs Page:

This screen is central for monitoring and analyzing detected suspicious activity. It is designed to display a detailed list of all logged attacks, with basic information for each attack, such as the time stamp, source IP address, and the type of attack detected or the rule that was activated. Searching, filtering, and sorting mechanisms are provided to facilitate the analysis of large volumes of logs. Through this screen, the administrator can gain a deep understanding of the threats to the protected application.

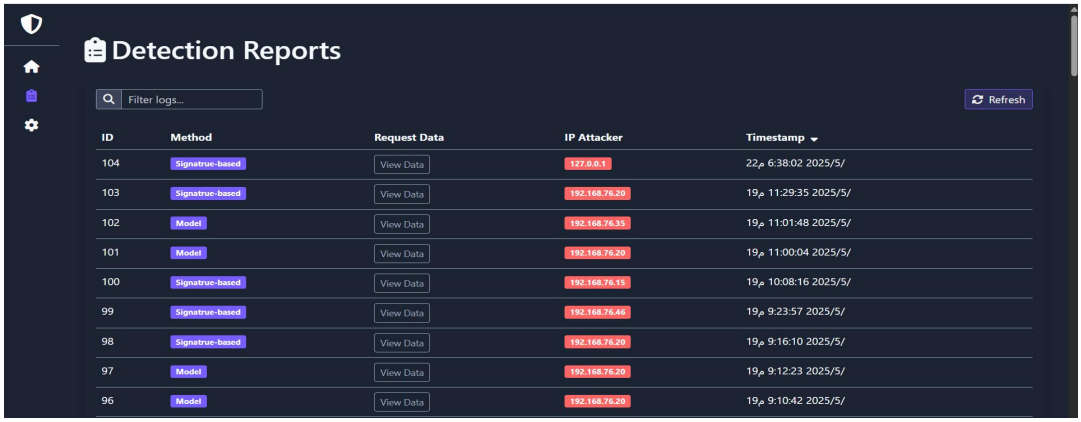

Figure 0- 5 Attack Reports and Information Window

Alerts and Password Settings Window

This screen gives the administrator complete control over configuring system behavior. I've grouped the various settings into a single, organized interface, including the ability to specify the IP address and port of the protected server, enable or disable the overall protection service, and configure alert mechanisms (such as enabling email alerts and specifying the recipient's address). The forms used in this screen are designed to be clear and easy to use, with input validation before saving to ensure system stability.

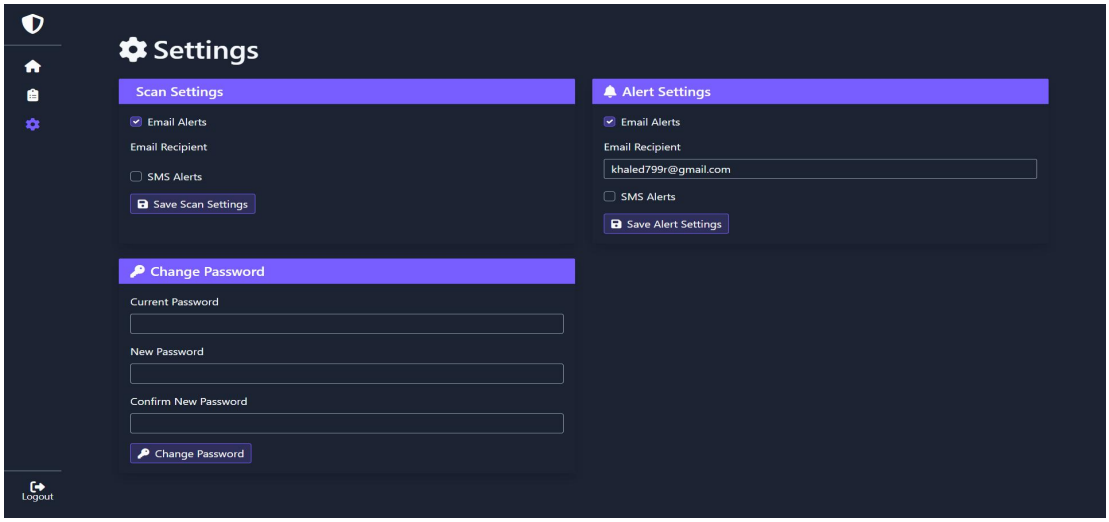

Figure 0- 6 Settings window

4.5 Design of system explanatory messages

Figure 0- 7

In Figure (18), an explanatory message appears in the upper right part of the screen, within a green bar to mark it as a success message. The text "Service started successfully" is an example of clarity and brevity, directly and immediately informing the administrator that the action they performed (starting the service) was successful and that the system is now in an active state. This is also confirmed by changing the service status to "Active" in the "Service Status" section below. This type of immediate feedback enhances the user's confidence in the system and assures them that their commands have been executed.

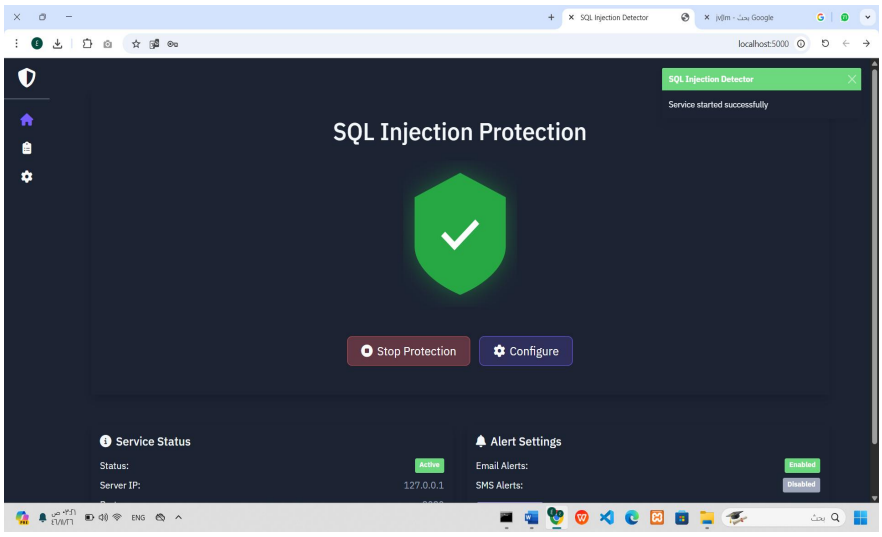

Figure 0- 8

Figure (19) illustrates the interface status after the administrator saves changes to the settings page (for example, after modifying the alert settings or the protected server address). Again, an explanatory message appears in a green bar in the upper right, reading "Configuration saved successfully." This message provides immediate confirmation to the administrator that the changes they made have been saved and successfully applied to the system. The design of this message is consistent with the principles of clarity, brevity, and immediate feedback, ensuring that the administrator clearly knows the outcome of their operations.

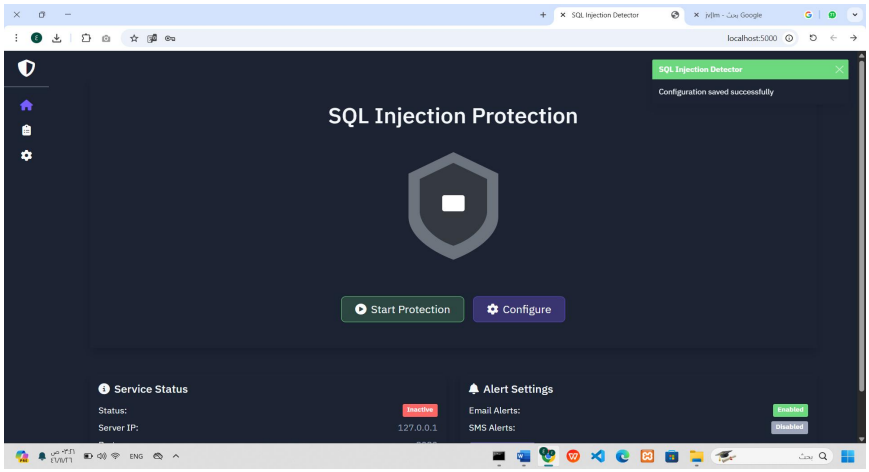

Figure 0- 9

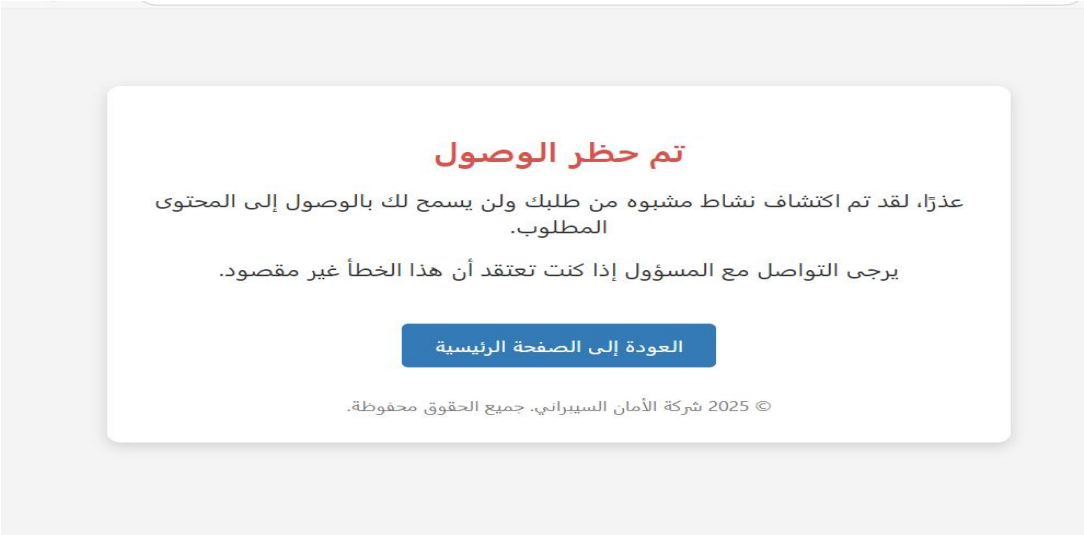

Figure 0- 10 Black page

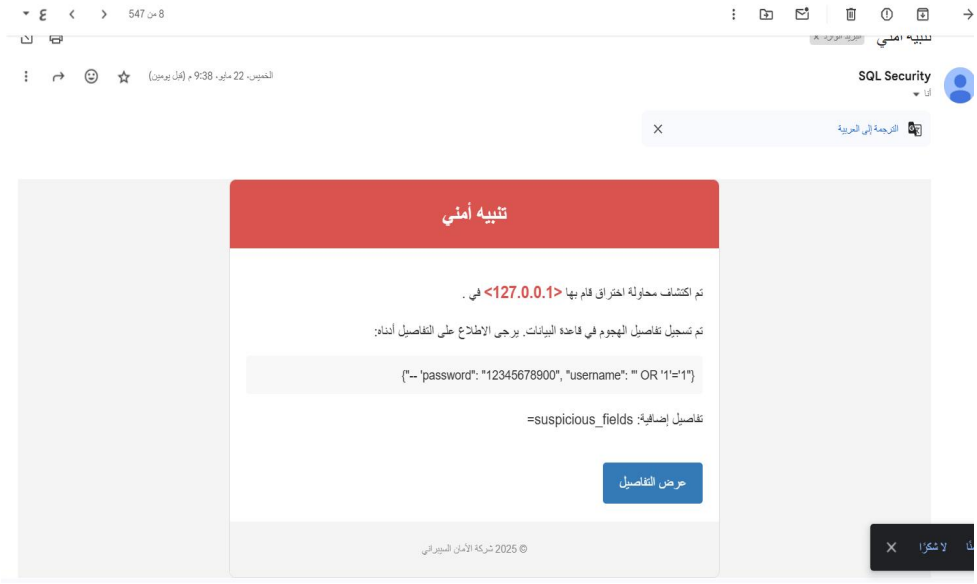

Figure 0- 11 Warning message

4.6 Design reports and inquiries

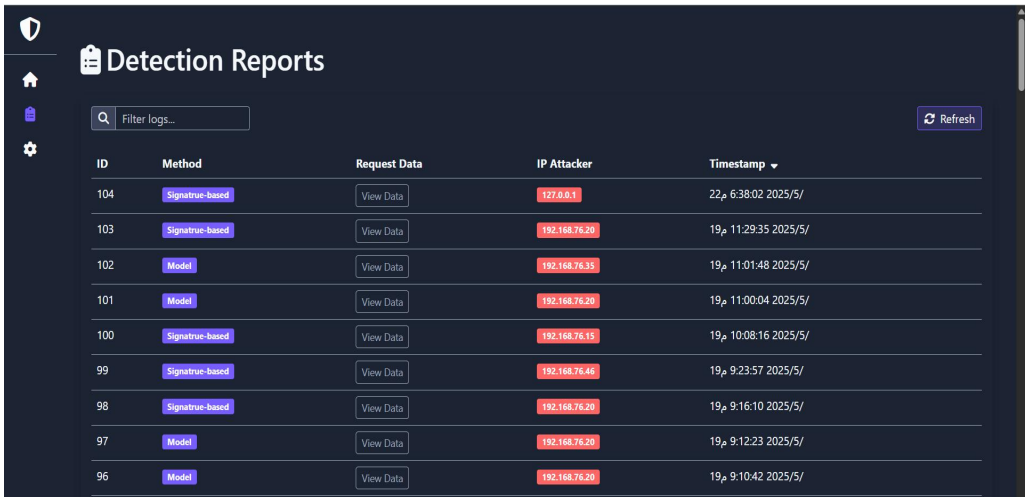

Figure 0- 12 Attack Inquiry Window

4.7 Design of software procedures and algorithms in the system

Signature-Based Detection (SBD) - As the diagram in Figure (23) shows, this mechanism relies on comparing incoming request data with a JSON file containing known signatures for SQL injection attacks. Or it relies on the libinjection library. If the request data matches one of the signatures, or if the libinjection library detects this signature, the request is considered malicious, an alert is raised, and appropriate action is taken (such as blocking). This method represents the first line of defense against common and known attacks.

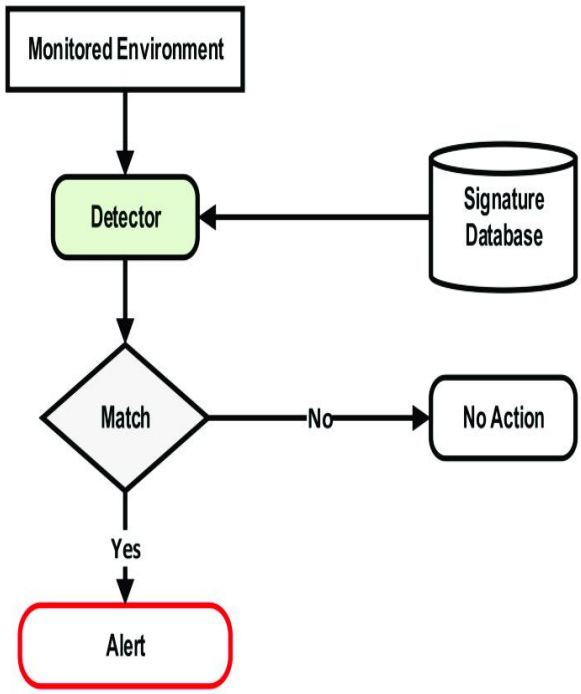

Figure 0- 13 Signature-Based Detection

Anomaly-Based Detection (ABD) – As shown in the diagram in Figure (), this mechanism aims to identify activities that deviate from the expected normal behavior of the application or users. The current activity is compared to a "baseline" that represents normal behavior. If the deviation from this baseline exceeds a predefined threshold, the activity is considered anomalous and an alert is raised. This method enables the detection of new or unknown attacks that may not be contained in the signature database. In addition to these two detection mechanisms, the system design includes other essential procedures such as: Request interception and analysis: This is done via the reverse proxy layer (Nginx) and the central processing layer (Flask) to examine each request. Suspicious request handling: This includes logging, alerting (if enabled), and blocking. Good request handling: This includes allowing the request to pass to the target application. Authentication and access control algorithm: To secure access to the control panel. Configuration management procedure: To allow the administrator to configure system behavior. These procedures and algorithms are designed to ensure effective and comprehensive protection against SQL injection attacks, with Maintain ease of management and monitoring by the administrator

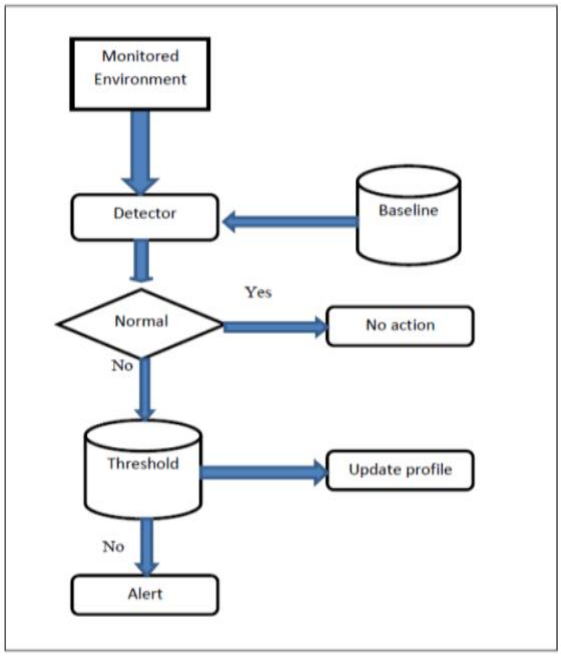

Figure 0- 14 Anomaly-Based Detection

Chapter 5: System testing and performance evaluation

5.1 Introduction

System testing and performance evaluation is a critical phase in the development lifecycle of any software system, particularly security systems like the one we have developed in this project. This chapter aims to verify the effectiveness of the system and its ability to achieve the objectives for which it was designed, through designing and implementing a set of systematic tests in a realistic simulation environment.Special focus will be placed on the system's ability to detect common SQL injection attacks launched using known penetration testing tools such as sqlmap, and prevent them from reaching the targeted backend server. In addition, the system's performance will be evaluated in terms of response speed and potential impact on user experience.

The testing process follows a scientific methodology that begins with preparing a controlled testing environment, then defining a clear testing plan that includes diverse testing scenarios (basic testing without protection, and testing with the protection system in place). After executing the tests, results are collected and accurately analyzed to evaluate the extent of the system's success in achieving its objectives, and to identify potential strengths and weaknesses.

All testing steps and results will be transparently documented, supported by visual evidence such as screenshots and system logs, to provide a comprehensive and objective evaluation of the system's performance.

5.2 Preparing the Test Environment

To ensure reliable and repeatable results, an isolated test environment was set up that simulates a real-world running environment for web applications. The test environment consists of the following components:

Target Machine

System: A virtual machine running Metasploitable2 was used. This system is known to intentionally contain several security vulnerabilities, including vulnerable web applications such as the Distributed Vulnerable Web Application (DVWA), making it an ideal target for testing the effectiveness of security systems.

Network: The virtual machine's network adapter was configured to operate in bridge mode, allowing it to obtain a unique IP address within the local network, as shown in Figure (5.1), making it accessible to other devices on the same network.

Target Application: The DVWA application shown in Figure (5.2), included in Metasploitable2, was used as the primary target for testing, specifically the SQL injection test page.

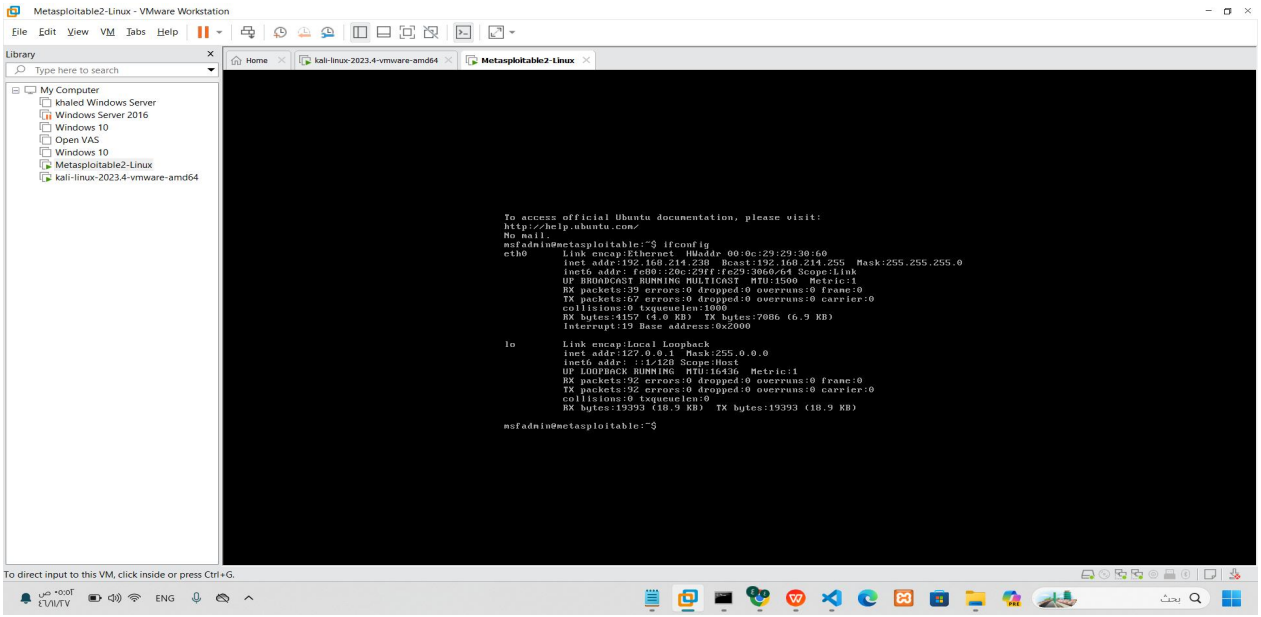

Figure 0- 15
metasploitable2

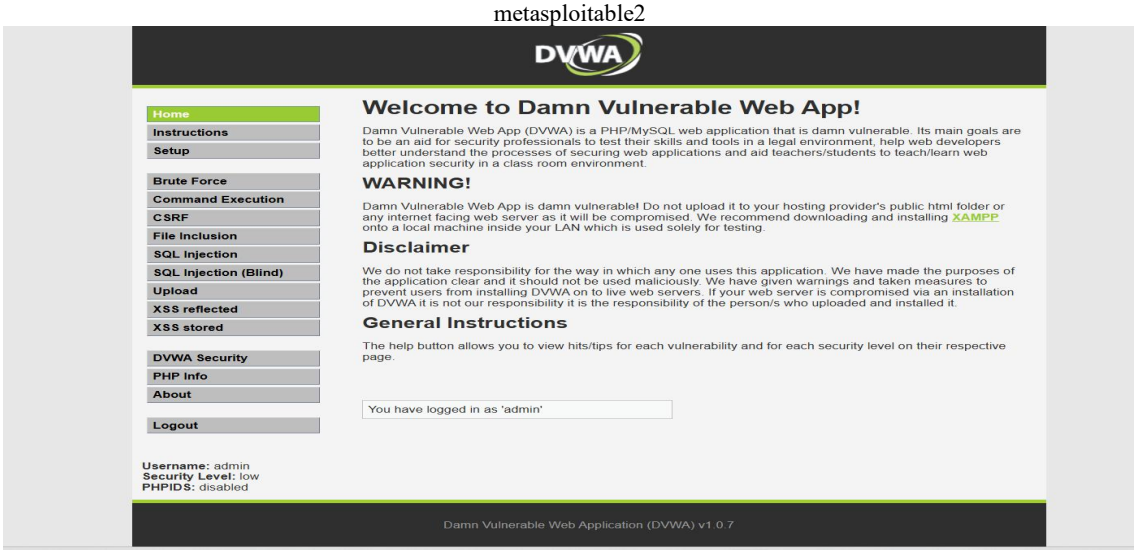

Figure 0- 16 Dvma

Attacking Machine:

System: Another virtual machine running Kali Linux or another Linux distribution equipped with the necessary penetration testing tools was used.

Tools: SQLmap, a widely used open-source penetration testing tool for automating the detection and exploitation of SQL injection vulnerabilities, was installed.

Network: It was also configured to operate in bridge mode on the same local network as the targeted machine, as shown in Figure (5.3).

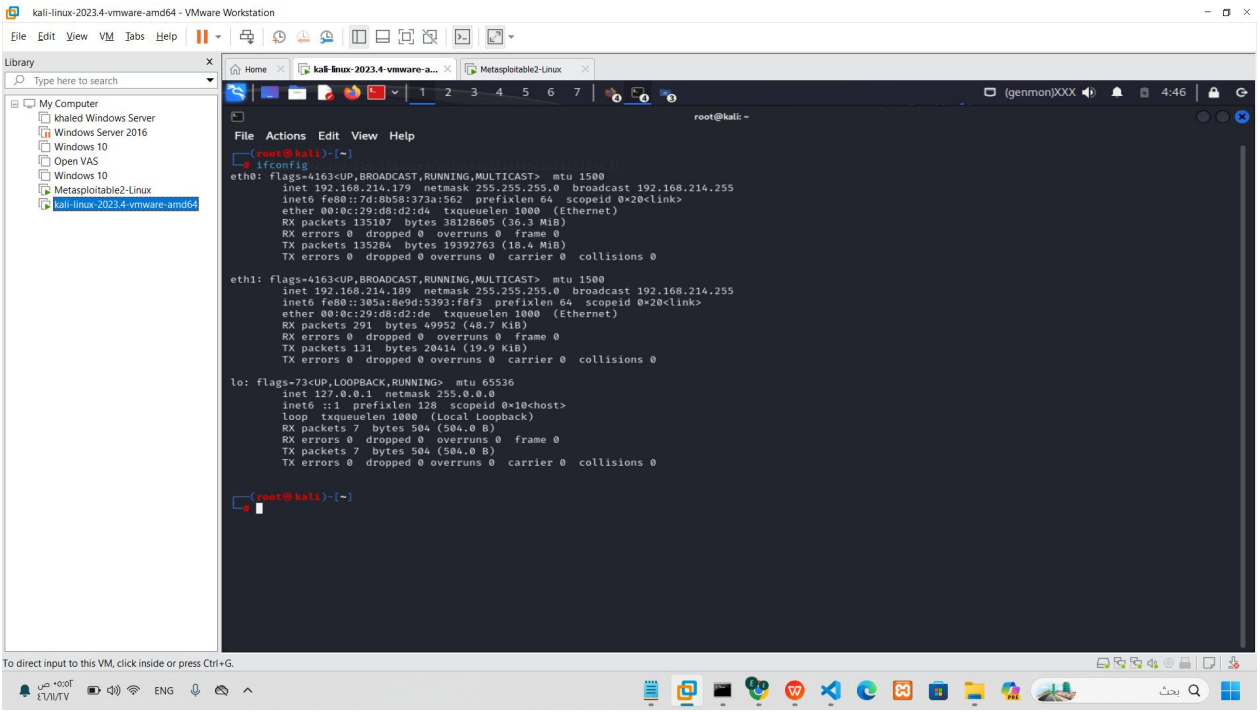

Figure 0- 17 Attacking machine

Protection System:

Components: The system consists of the Nginx reverse proxy server with lua-nginx-module, an inspection service written in Python using the Flask framework, and the libinjection library.

Deployment: The Flask inspection service was run on the developer's machine or a separate virtual machine. Nginx was configured to work as a reverse proxy that receives requests directed

to the target machine (Metasploitable2), passes them first to the Flask inspection service via Lua, and then makes a decision to block or pass based on the response from the inspection service. Configuration: The protection system's Graphical User Interface (GUI) was used to configure Nginx to listen on a specific port and direct traffic to the IP address and port of DVWA on Metasploitable2.

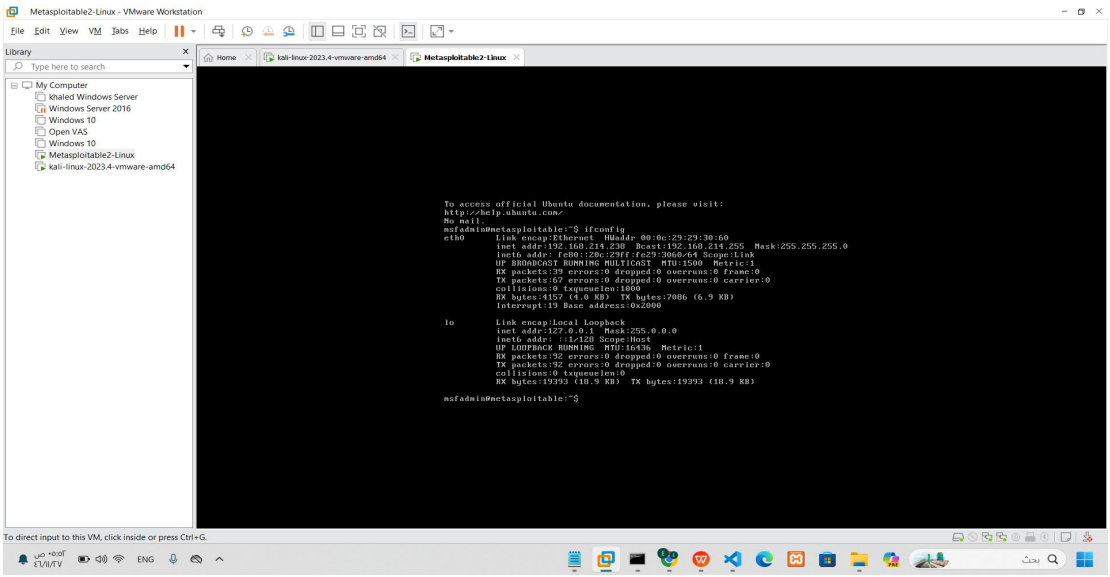

Figure 0- 18 Inspection service

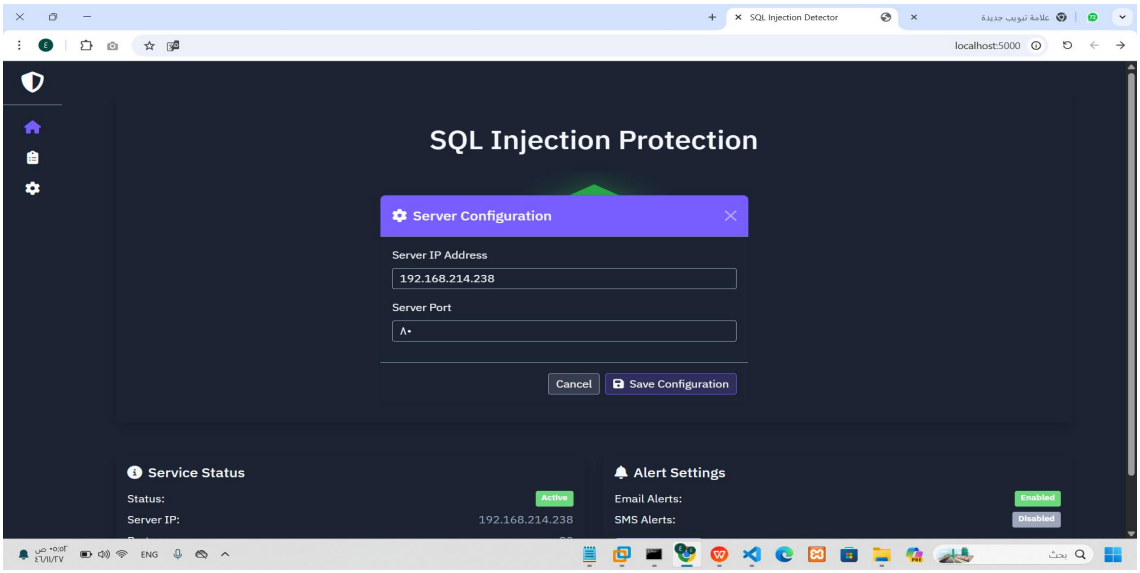

Figure 0- 19 Proxy configuration

5.3 Testing Methodology and Plan

A comparative testing methodology was adopted to evaluate the effectiveness of the protection system. The plan includes implementing two main scenarios:

5.3.1 Baseline Test

In this scenario, a SQL injection attack using sqlmap is launched directly against the DVWA application on Metasploitable2 without the protection system in place (i.e., the DVWA is accessed directly from the attack machine). The objectives of this test are:

- To confirm the presence of an exploitable SQL injection vulnerability in the DVWA.

- To identify the commands and payloads that sqlmap successfully used to exploit the vulnerability.

- To record the baseline results, which will later be compared to the test results with the protection application.

5.3.2 Protection Effectiveness Test

In this scenario, the same SQL injection attack is re-executed using sqlmap, but this time the attack is routed through the protection system (i.e., targeting the IP address and port of Nginx, which acts as a reverse proxy). The objectives of this test are:

- To evaluate the system's ability to detect malicious sqlmap payloads.

- To verify the system's success in preventing malicious requests from reaching the DVWA.

- Monitoring and recording alerts and reports generated by the protection system.

Evaluation Metrics

- Detection Rate: The percentage of attacks that the system successfully detects.

- Block Rate: The percentage of attacks that the system successfully prevents from reaching the target.

- False Positives: The number of legitimate requests incorrectly classified as attacks.

- False Negatives: The number of actual attacks that the system failed to detect or block.

- Response Time: A measure of any noticeable delay that the protection system may add to the response time of legitimate requests.

This comprehensive testing approach ensures a comprehensive assessment of the effectiveness of the protection system in real-world attack scenarios, while providing measurable metrics for performance evaluation.

5.4 Test Implementation

The tests were executed step by step as follows:

5.4.1 Baseline Test Implementation

1. Accessing DVWA: A web browser was opened on the attack machine and direct access was made to the IP address and port of DVWA on Metasploitable2.2. Login and Cookie Acquisition: Login was performed to DVWA (username: admin, password: password) and the security level was set to Low. The SQL Injection page was accessed as shown in Figure (5.5), and the browser's developer tools were used to obtain the cookie value (Session ID).

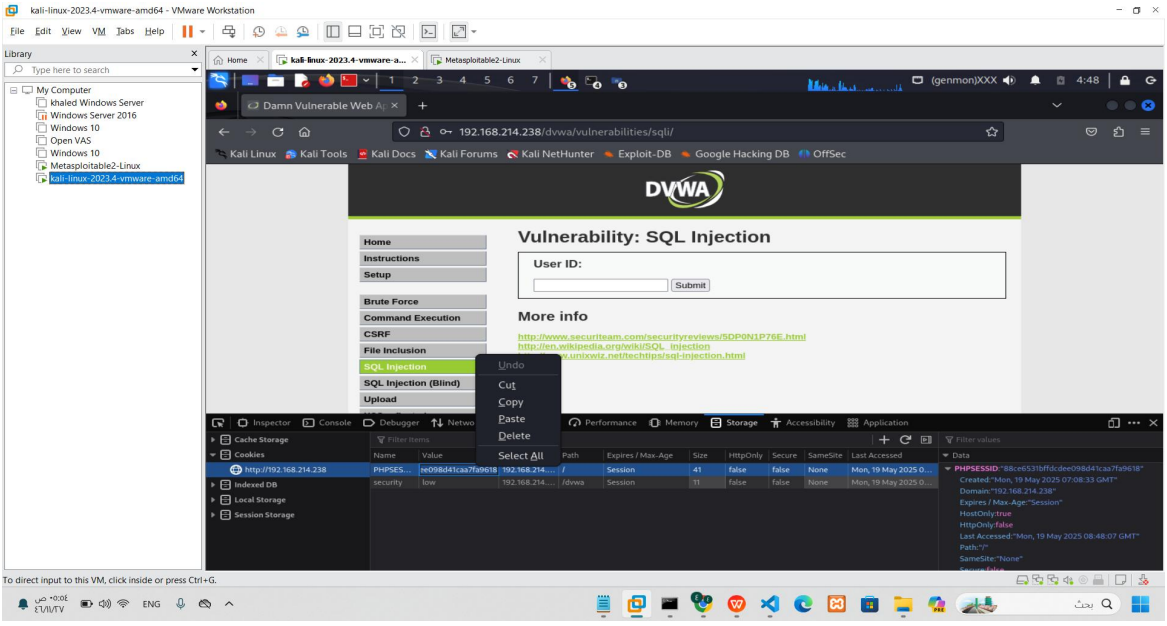

Figure 0- 20 Preparing cookies

3. Running sqlmap: A terminal was opened on the attack machine and the basic sqlmap command was executed to target DVWA directly, specifying the URL of the target page, the injectable parameter (id), and the cookie value. Example of the command:

sqlmap -u http://<192.168.214.238>/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#--cookie="security=low; PHPSESSID=<SESSION_ID>" --batch --dbs

- -u: Specifies the target URL.

- --cookie: Specifies the necessary cookies for the session.

- --batch`: Runs sqlmap in non-interactive mode using default settings.

- --dbs`: Requests sqlmap to attempt to enumerate available database names.

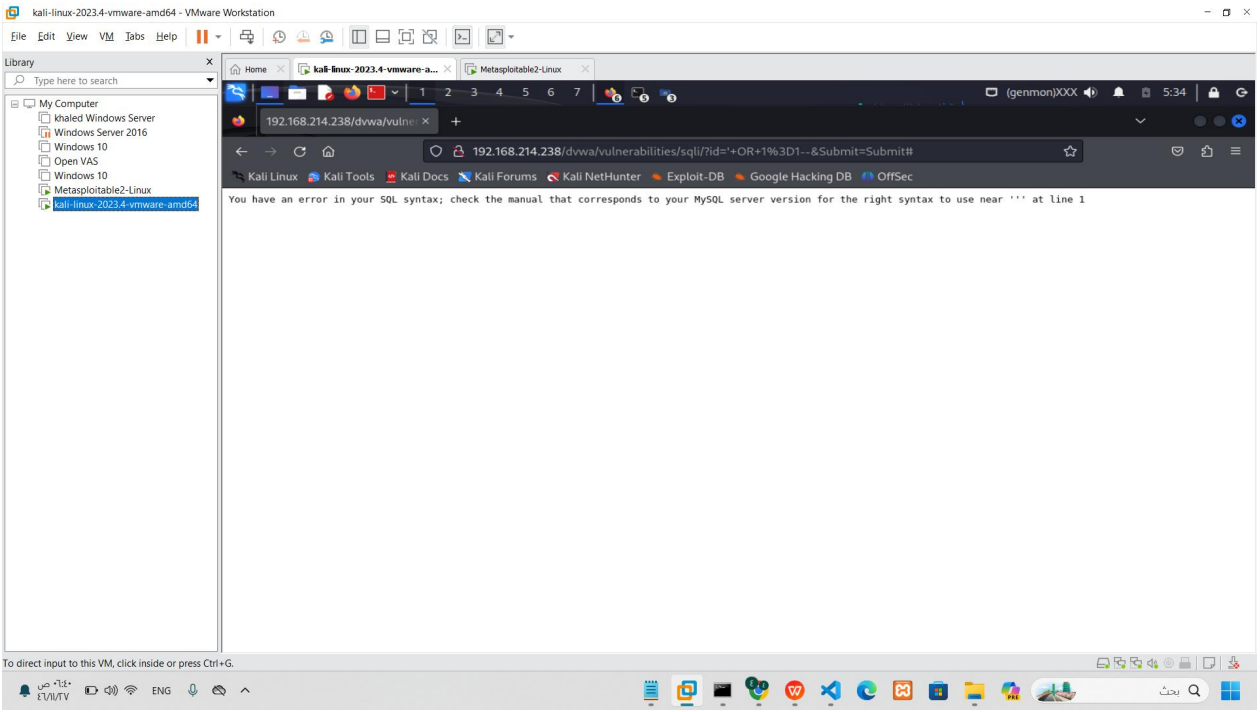

Figure 0- 21 Executing the attack

4. Monitoring Results: The sqlmap outputs were observed, confirming the detection of the vulnerability and its success in extracting database names.This baseline test establishes the reference point for comparing the effectiveness of the protection system in the subsequent test phase. By confirming the exploitability of the SQL injection vulnerability without protection, we can properly evaluate how well the protection system performs in preventing similar attacks.

5.4.2 Protection Effectiveness Test Implementation

1. Activating the Protection System: It was confirmed that the Flask inspection service was running and that Nginx was configured and activated to work as a reverse proxy directing requests to DVWA through the inspection service.

2. Updating the sqlmap Target: The previous sqlmap command was modified to now target the IP address and port of Nginx instead of the direct Metasploitable2 address. The same cookies and other parameters were retained.

"bash"

sqlmap -u "http://<NGINX_IP>:<NGINX_PORT>/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=<SESSION_ID>" --batch --dbs --level=5 --risk=3

- <METASPLOITABLE_IP>` was replaced with `<NGINX_IP>:<NGINX_PORT>`.

- --level=5` and `--risk=3` were added to increase the comprehensiveness of the tests conducted by
sqlmap.

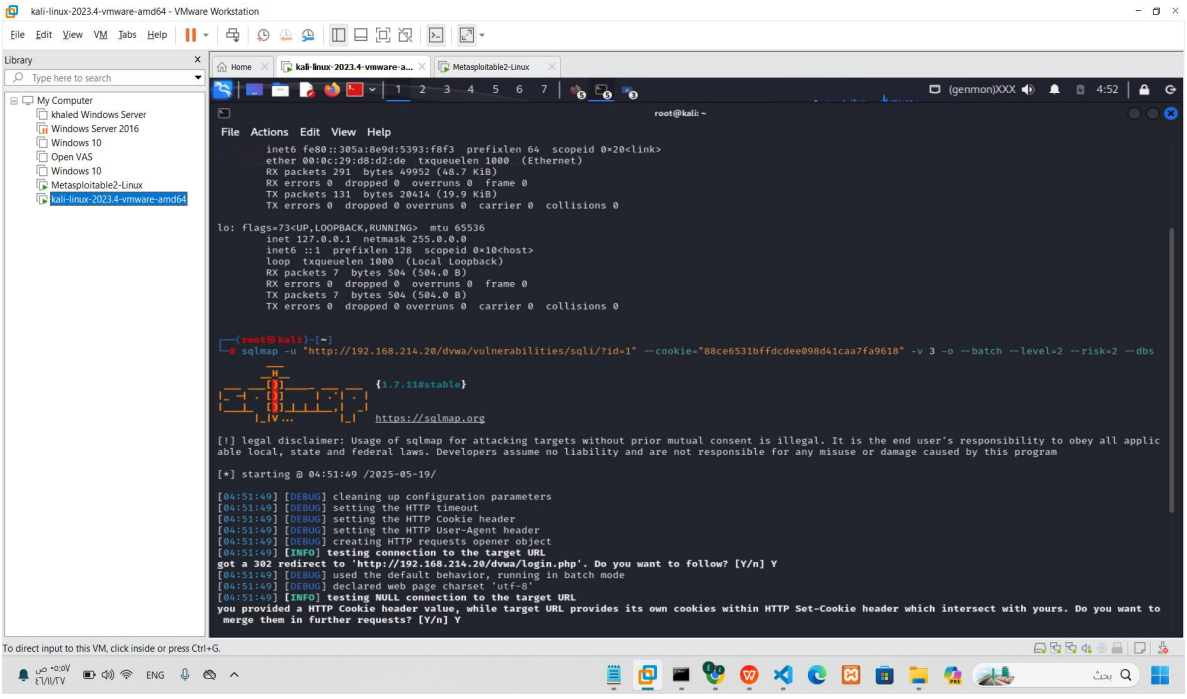

Figure 0- 22 Executing the attack

3. Monitoring sqlmap: The outputs of sqlmap were monitored. In this scenario, it was expected that sqlmap would fail to detect or exploit the vulnerability because the protection system should detect and block malicious requests.

4. Monitoring the Protection System: Concurrently with running sqlmap, the following were monitored:

- Flask Inspection Service Outputs: To look for any logs indicating reception of requests from Nginx/Lua and their analysis by libinjection.

- Detection Reports Interface: To verify the recording of new alerts corresponding to the attack launched by sqlmap, with details such as the attacker's IP and the detected payload.

- Nginx Logs: To search for HTTP status codes (such as 403 Forbidden) indicating the blocking of requests by the Lua script based on the response from the inspection service, as shown in Figure (5.9).

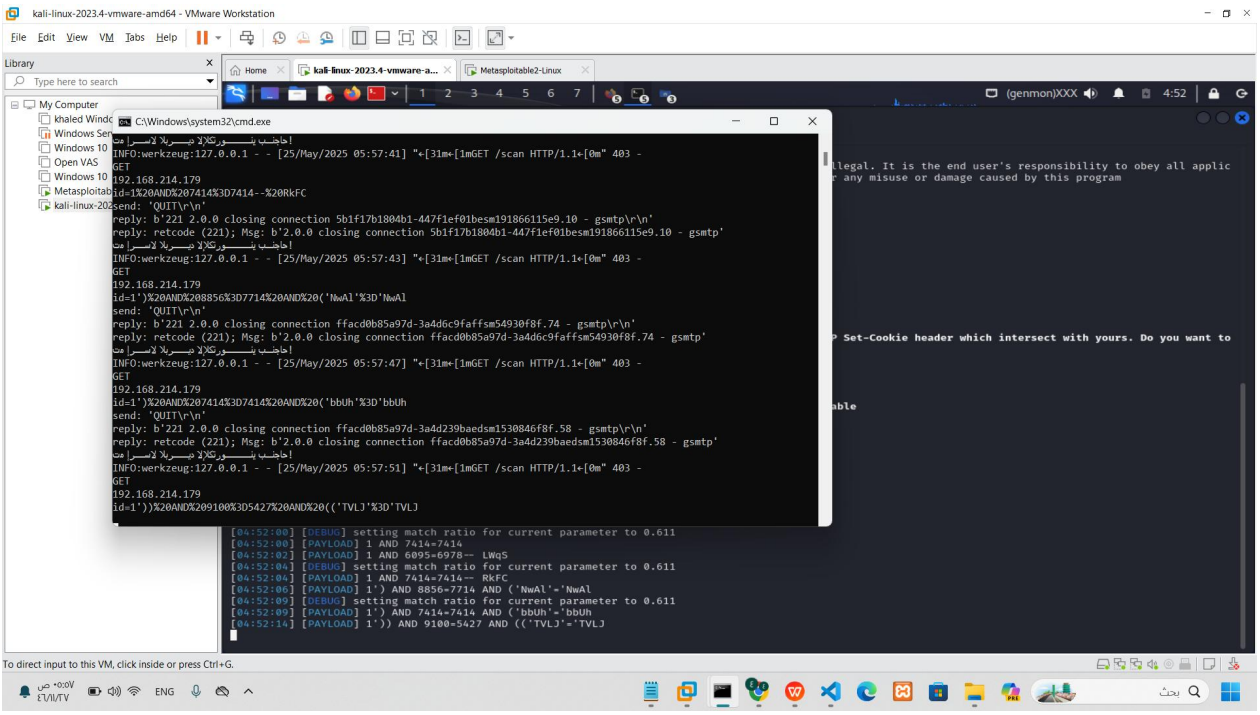

Figure 0- 23 Terminal scanning service

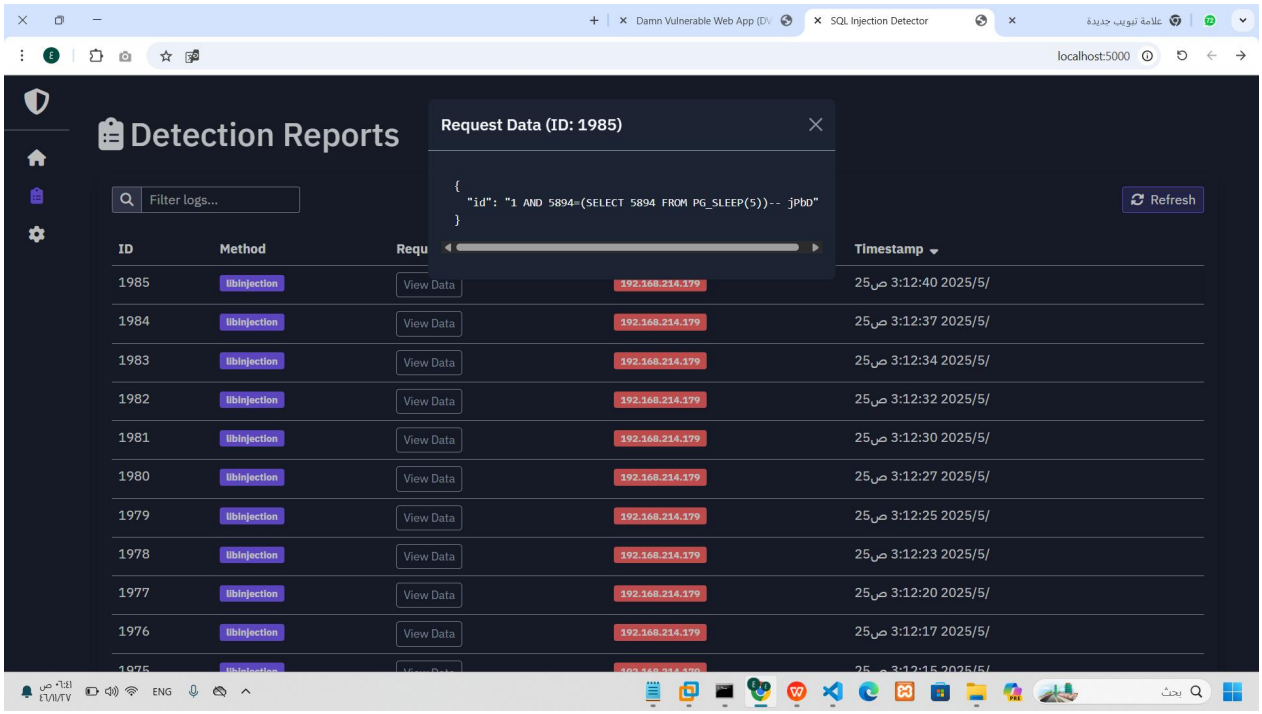

Figure 0- 24 Reports of attacks

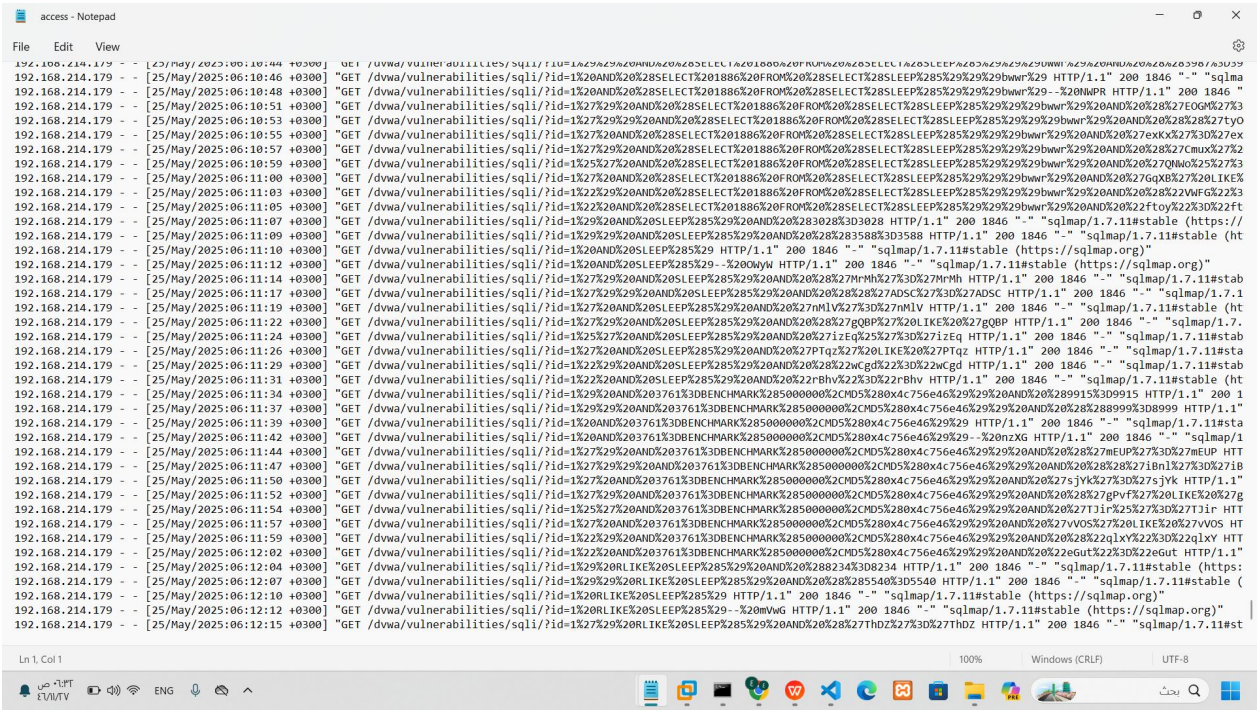

Figure 0- 25 Proxy logs

This test phase evaluates the effectiveness of the implemented protection system by attempting to exploit the same vulnerability that was confirmed exploitable in the baseline test. By monitoring both the attack tool and the protection system simultaneously, we can assess how well the system identifies and blocks malicious SQL injection attempts.

5.5 Results and Analysis

Based on the implementation of the test scenarios outlined above, the following results were obtained and analyzed:

Baseline Test Results:

The baseline test results confirmed the presence of an exploitable SQL injection vulnerability in the DVWA application (at a low security level). The sqlmap tool successfully detected the vulnerability and extracted sensitive information (database names) when targeting DVWA directly, providing a clear baseline for comparison.

Protection Effectiveness Test Results:

The results of the second test demonstrated high effectiveness of the proposed protection system. When directing the same sqlmap attack through Nginx and the inspection system:

Detection and Prevention:

The sqlmap tool failed to detect or exploit the vulnerability. The tool's outputs indicated either no injectable parameters or inability to confirm the existence of the vulnerability. In contrast, the protection system logs (reports interface, Flask outputs, and Nginx logs) showed simultaneous detection of a large number of injection attempts coming from the attack machine's IP address, and these requests were successfully blocked before reaching DVWA.

Evaluation Metrics:

Based on observations, the detection rate and blocking rate can be estimated at very close to 100% for the attacks tested using sqlmap with the specified settings. No false negatives were observed during this specific test. Evaluating false positives would require additional tests using legitimate and diverse traffic.

Performance:

No significant delay in response time was observed during normal browsing of DVWA through the protection system, indicating that the inspection process performed by the system did not add a noticeable burden on performance in this test environment. However, evaluating performance under high load may require additional tests using performance measurement tools.

Analysis of Results:

The results clearly indicate that the architecture based on the reverse proxy (Nginx/Lua) and the separate inspection service (Flask/libinjection) is capable of providing effective protection against automated SQL injection attacks launched by tools such as sqlmap. The system successfully intercepts incoming requests, analyzes their payloads for SQL injection patterns using libinjection, and makes blocking decisions in real-time before the malicious request reaches the target application. The system logs and reports demonstrate its ability to provide clear visibility into detected attack attempts.

This comprehensive testing approach validates the effectiveness of the designed protection system in identifying and blocking SQL injection attacks while maintaining acceptable performance levels in a controlled test environment.

5.6 Conclusion :

The testing and evaluation process conducted in this chapter has proven the effectiveness of the "Real-time SQL Injection Detection and Prevention System Based on Reverse Proxy Server." A realistic testing environment was successfully established, and a systematic testing plan was implemented that included baseline testing and protection effectiveness testing using the sqlmap tool.

The results demonstrated that the system is capable of detecting and preventing the tested SQL injection attacks with a high success rate, without significantly affecting access performance to the protected application in the test environment. The results were documented with expected visual evidence from the system logs and reports.These findings confirm the achievement of the project's main objectives of building an effective and responsive protection system against one of the most dangerous types of web application attacks. The system can be considered a promising solution that can be further developed and improved to provide a strong additional layer of protection for web applications.

Chapter 6 : Conclusions and future work

6.1 Introduction

This research reaches its conclusion, presenting an innovative system for the detection and prevention of SQL injection attacks in real-time, based on a reverse proxy architecture. In previous chapters, we have reviewed the increasing security challenges posed by these attacks, analyzed the requirements necessary for building an effective solution, and then designed, implemented, and tested the system. This chapter presents a concise summary of the main achievements of this work and offers insights and recommendations for its future development and expansion of research horizons in this field.

6.2 The Conclusions :

SQL injection attacks are among the most serious threats facing modern web applications, exploiting vulnerabilities in user input processing to gain unauthorized access to and manipulate sensitive databases. Recognizing these risks and the significant shortcomings of traditional security solutions, this project was launched with the aim of developing a proactive and intelligent protection system.

A comprehensive protection system based on the concept of a reverse proxy using Nginx and Lua (OpenResty) technologies was successfully designed and developed to act as an advanced front-end defense (WAF). This system is characterized by its ability to intercept and analyze incoming HTTP requests before they reach the target server. The essence of the innovation lies in linking the reverse proxy to a specialized inspection service, built using the Flask framework, which applies an effective hybrid approach to attack detection:

Signature-based analysis: to quickly identify known attack patterns.

Machine learning-based analysis: to detect anomalies and unknown or emerging attacks, providing the ability to adapt to evolving threats.

Practical tests conducted in a simulated environment, using standard attack tools such as SQLMap against known vulnerable web applications (DVWA on Metasploitable2), demonstrated the system's effectiveness and ability to achieve the desired objectives, The system demonstrated its ability to:

Accurate detection and immediate prevention: The system successfully identified and effectively blocked various SQL injection attempts, preventing them from reaching the targeted web application.

Comprehensive protection: The hybrid approach demonstrated a high ability to distinguish between legitimate and malicious requests, reducing the likelihood of false positives.

Practical application: The system is designed to operate as an external layer of protection without requiring any modifications to the source code of existing web applications, simplifying deployment and integration.

Manageability and monitoring: A graphical user interface (GUI) was designed to facilitate system configuration, performance monitoring, and reporting of detected attacks.

This project represents a valuable contribution to the field of web application security, providing a practical and scalable solution to combat one of the most serious cyber threats. Combining the power of a reverse proxy with machine learning intelligence provides a solid foundation for data protection and business continuity.

6.3 Recommendations

Based on the success of the current system, and to enhance its capabilities and expand its usefulness, we recommend working on the following development areas:

Expanding Detection Capabilities: Building on the current architecture to include detection mechanisms for other common web application attacks, such as cross-site scripting (XSS) and cross-site request forgery (CSRF), providing a more comprehensive security solution.

Developing Machine Learning Models: Exploring and applying more advanced machine learning algorithms, such as deep neural networks, and training them on larger, more representative datasets of real threats to increase detection accuracy and generalizability.

Improving Performance and Scalability: Conducting in-depth performance studies to measure the system's impact on application response time under varying loads, and exploring techniques to improve scanning efficiency and ensure the system's horizontal scalability.

Integrating with Security Systems: Developing log and alert export capabilities for seamless integration with Security Information and Event Management (SIEM) and Security Automation Systems (SOAR) platforms to achieve a unified security view and coordinated incident response. Validation in diverse environments: Conducting additional tests and piloting the system in diverse real-world operating environments to evaluate its performance and effectiveness against real-world scenarios.

Enriching the user interface: Adding more visualization tools and advanced user interface controls to facilitate analysis and decision-making for security administrators.

Investing in these development areas will enhance the system's value and consolidate its position as a leading and reliable security solution for protecting web applications against emerging cyber threats.

the reviewer

[1] Alazab, M., et al. (2021). "A systematic review of detection and prevention techniques of SQL injection attacks." *IET Information Security*, 15(6), pp. 518-537. URL:https://iitis.pl/sites/default/files/pubs/A%20systematic%20review%20of%20detection%20and%20prevention%20techniques%20of%20SQL%20injection%20attacks%20%281%29.pdf

[2] Appelbaum, A. (2020). "Creating a web API with Lua using Nginx OpenResty." *DEV Community*. URL:https://dev.to/forkbomb/creating-an-api-with-lua-using-openresty-42mc

[3] Bansal, A., et al. (2024). "SQL injection attack: Detection, prioritization & prevention." *Internet of Things and Cyber-Physical Systems*, 4, pp. 235-246. URL:https://www.sciencedirect.com/science/article/pii/S221421262400173X

[4] Bhattacharjee, B., et al. (2023). "Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks." *Applied Sciences*, 13(4), 2458.URL:https://pmc.ncbi.nlm.nih.gov/articles/PMC9965318/

[5] Bisht, P., et al. (2010). "Analysis of SQL Injection Detection Techniques." *arXiv preprint arXiv:1004.4883*.URL:https://www.researchgate.net/publication/302893095_Analysis_of_SQL_Injection_Detection_Techniques

[6] Fang, Y., et al. (2023). "A Semantic Learning-Based SQL Injection Attack Detection Method Using BERT." *Applied Sciences*, 12(6), 1344. URL:https://www.mdpi.com/2079-9292/12/6/1344

[7] Gaikwad, P., et al. (2021). "SQL Injection Attack Detection and Prevention Techniques Using Machine Learning." *2021 International Conference on Communication information and Computing Technology (ICCICT)*. URL:https://iopscience.iop.org/article/10.1088/1742-6596/1757/1/012055/pdf

[8] Gupta, S., & Kumar, P. (2022). "Machine Learning for Cybersecurity in Smart Grids: A Comprehensive Review." *Archives of Computational Methods in Engineering*, 29(7), pp. 4849-4879. URL:https://www.sciencedirect.com/science/article/pii/S1874548222000348

[9] Hindy, H., et al. (2022). "Uniting cyber security and machine learning: A survey." *Computers & Security*, 117, 102696. URL:https://www.sciencedirect.com/science/article/pii/S2405959522000637

[10] Kar, D., et al. (2021). "A Study of Web Application Firewall Solutions." *Information Systems Security. ICISS 2015. Lecture Notes in Computer Science*, vol 9478. Springer, Cham. URL:https://dl.acm.org/doi/10.1007/978-3-319-26961-0_29

[11] Li, J. H., et al. (2022). "The Role of Machine Learning in Cybersecurity." *ACM Computing Surveys*, 55(1), Article 18.URL: https://dl.acm.org/doi/10.1145/3545574

[12] Liu, Q., et al. (2024). “Overview of SQL Injection Attack Detection Techniques.” *Proceedings of the 2024 6th International Conference on Computer Science and Technologies in Education*. URL: https://dl.acm.org/doi/10.1145/3640912.3640956

[13] OpenResty Team. “OpenResty® - Open source.” URL: https://openresty.or/

[14] Prakash, A., et al. (2023). “Machine Learning in Cybersecurity: Techniques and Challenges.” *2023 International Conference on Disruptive Technologies (ICDT)*. URL: https://www.researchgate.net/publication/371247787_Machine_Learning_in_Cybersecurity_Techniques_and_Challenges

[15] R., Rohit, et al. (2023). “Web Application Security Testing Framework using Flask.” *2023 International Conference on Inventive Computation Technologies (ICICT)*. URL: https://ieeexplore.ieee.org/document/10140422/

16] Rawat, S., & Ramola, P. (2023). “A Review Study on SQL Injection Attacks, Prevention, and Detection.” *2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. URL: https://www.researchgate.net/publication/362751864_A_Review_Study_on_SQL_Injection_Attacks_Prevention_and_Detection

[17] Raza, M., et al. (2021). “Analysis of Web Application Firewalls, Challenges, and Research Opportunities.” *IEEE Access*, 9, pp. 154896-154916. URL:https://www.researchgate.net/publication/356066472_Analysis_of_Web_Application_Firewalls_Challenges_and_Research_Opportunities

[18] Sivakumar, V. (2024). “Nginx Reverse Proxy and Lua Scripting.” *vladsiv*. URL:https://www.vladsiv.com/nginx-reverse-proxy-lua-scripting/

[19] Spencer, R., et al. (1999). “The Flask Security Architecture: System Support for Diverse Security Policies.” *Proceedings of the 8th USENIX Security Symposium*. URL: https://www.cs.cmu.edu/~dga/papers/flask-usenixsec99.pdf

[20] Tam, K., et al. (2017). “iFlask: Isolate flask security system from dangerous execution environment.” *Journal of Systems Architecture*, 77, pp. 66 77.URL:https://www.sciencedirect.com/science/article/abs/pii/S0167739X17322239