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

AITCS

Vol. 7 No. 1 (2026) p. 1290-1306
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Automated SQL Injection Scanning Tool for Vulnerable Website

Mohamad Mustaqim Mohd Shukri[1], Nordiana Rahim[1*]

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

*Corresponding Author: nordiana@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2026.07.01.069

Article Info

Received: 10 June 2026
Accepted: 23 June 2026
Available online: 30 June 2026

Keywords

SQL Injection Tool, Penetration Testing, Automated Tool, SQL Injection, Vulnerable Website

Abstract

SQL injection (SQLi) remains one of the most prevalent and dangerous security vulnerabilities in web applications, as it allows attackers to manipulate backend database queries and gain unauthorized access to sensitive data. This paper presents the development of an Automated SQL Injection Scanning Tool for vulnerable websites, which aims to improve the efficiency and accuracy of SQLi detection during web security assessments. The tool incorporates multiple detection techniques, including error-based, boolean-based blind, time-based blind, and union-based SQL injection, all executed through HTTP request analysis. Developed using Python and PyQt5, the tool features a modern graphical user interface, a comprehensive payload management system, SQL scanning capabilities, and a detailed reporting module. The development process followed the Agile methodology to support iterative enhancements and prioritize user-centered improvements throughout the project. The effectiveness of the tool was validated through testing on intentionally vulnerable web applications deployed in a controlled environment for security research and educational purposes. The testing process involved automated detection followed by manual verification to confirm identified SQL injection vulnerabilities. The results demonstrated that the tool accurately detected all targeted vulnerabilities without producing false positives during the evaluation phase. Future enhancements will focus on expanding the tool's detection capabilities to include other common web vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). Additional improvements will include the implementation of evasion techniques and multi-threaded scanning to further increase performance and coverage.

1. Introduction

In recent years, web application security has become a critical area of focus due to the increasing prevalence of cyber threats. Among these, Structured Query Language (SQL) injection remains one of the most severe and commonly exploited vulnerabilities. SQL injection (SQLi) attacks allow malicious users to manipulate backend database queries by injecting unauthorized SQL code into input fields, potentially leading to data breaches, unauthorized access, or complete system compromise [1].

Despite advancements in cybersecurity, many web applications continue to rely on traditional defenses such as firewalls and antivirus software. These solutions are often ineffective against injection-based attacks, as they do not inspect application-level logic or dynamic query behavior [2]. Furthermore, manual vulnerability

testing and code audits are resource-intensive, time-consuming, and subject to human error [3], creating the need for automated, reliable tools that can proactively detect SQLi vulnerabilities during development or deployment.

Automated SQL Injection Scanning Tool for Vulnerable Website, a security application that can assist web developers, security analysts, and penetration testers in identifying and mitigating SQLi flaws in websites. The objectives of this study are as follows:

- To design an Automated SQL Injection Scanning Tool for Vulnerable Website
- To develop an Automated SQL Injection Scanning Tool for Vulnerable Website
- To evaluate the performance and effectiveness of the Automated SQL Injection Scanning Tool on Vulnerable Website.

The scope of this project focuses on testing intentionally vulnerable websites within controlled and authorized environments for the purpose of evaluating SQL injection detection capabilities. The tool was developed using Agile methodology, ensuring iterative development, continuous testing, and user-centered design. The expected outcome is a functional, lightweight, and extensible vulnerability scanner that contributes to stronger web application security by identifying and reporting SQL injection risks early in the lifecycle.

This paper is organized as follows: Section 2 reviews related work and foundational concepts. Section 3 describes the methodology and design process. Section 4 presents the system architecture and key components. Section 5 discusses the implementation. Section 6 presents testing, and evaluation of the tool. Section 7 concludes the study and outlines recommendations for future enhancements.

2. Literature Review

Section 2 discusses sql injection, sql injection techniques, features of automated sql injection tool and existing sql injection tools.

2.1 Penetration Testing

Penetration testing, or pen testing, is a simulated cyberattack used to identify and exploit security weaknesses in web applications. It is a critical method for ensuring robust protection against various threats, particularly those involving input-based vulnerabilities. The process typically involves stages such as reconnaissance, scanning, exploitation, and reporting, enabling a comprehensive assessment of system security. One of the key focuses of penetration testing is identifying injection vulnerabilities, which occur when an attacker inputs malicious code into a system to manipulate its behaviour. Common examples include SQL injection, command injection, and cross-site scripting (XSS) [6]. As highlighted by Zhang et al. [7], regular penetration testing plays a vital role in maintaining the integrity and resilience of modern web systems. This project specifically addresses SQL injection vulnerabilities, one of the most impactful and widely exploited web application threats.

2.2 SQL Injection

SQL Injection (SQLi) is a critical web security vulnerability that allows attackers to manipulate database queries executed by a web application, potentially resulting in unauthorized data exposure, data loss, or full system compromise. SQLi attacks are becoming increasingly prevalent and sophisticated, targeting applications in sectors such as healthcare, finance, and government. According to research by [8], SQLi remains one of the most exploited vulnerabilities due to improper input validation and insecure coding practices. Different types of SQLi attacks, such as error-based, boolean-based blind, and union-based SQLi, exploit various aspects of database query construction, causing significant harm to affected systems. The consequences of a successful SQLi attack include downtime, loss of sensitive data, legal repercussions, and reputational damage. Detecting and mitigating SQLi vulnerabilities requires implementing robust countermeasures such as input validation, prepared statements, and web application firewalls. As highlighted by [9] and [10], advancements in machine learning and AI are proving instrumental in analyzing traffic patterns to identify anomalies and enhance security against SQLi. Given the potential for financial and reputational losses, addressing SQLi is vital for safeguarding modern web applications and ensuring data protection.

2.3 SQL Injection Detection Techniques

Various techniques are employed to detect and mitigate SQL Injection (SQLi) vulnerabilities, ensuring the security of web applications. A commonly used approach is error-based detection**,** which involves sending specifically crafted payloads to trigger database error messages. These error messages provide insight into the database structure by exposing SQL syntax errors, database exception signatures, and response patterns [9]. However, this method is less effective when error messages are suppressed by the system. Another widely used approach is boolean-based blind detection, which works without relying on error messages [8]. This technique

involves injecting logical TRUE/FALSE conditions into user inputs and observing changes in application behavior, such as variations in page content or responses. Additionally, time-based analysis, a subset of blind detection, measures response delays caused by injected time-delay commands to identify potential vulnerabilities. Both techniques play a significant role in SQLi detection, with error-based methods providing explicit insights and boolean-based detection offering an alternative when error messages are unavailable. Combining these methods ensures a robust and layered approach to securing web applications against SQLi threats.

2.4 Features of Automated SQL Injection Tool

The Automated SQL Injection Scanning Tool is a comprehensive PyQt5-based security application designed to detect and analyze SQL injection vulnerabilities across web applications through its sophisticated multi-threaded scanning architecture. Leveraging Python's Requests library, the tool implements robust request handling for both GET and POST methods with configurable timeouts, custom headers, and session management, while its intelligent response analysis system identifies vulnerabilities through pattern-based error detection, boolean-based testing, and time-based injection techniques across multiple database systems including MySQL, PostgreSQL, SQL Server, Oracle, and SQLite. The tool features an SQLite-based payload management system with pre-configured injection vectors that users can supplement with custom payloads, properly encoding each during testing to bypass input sanitization. With its intuitive user interface, real-time progress tracking, and detailed vulnerability reporting that captures affected parameters, successful payloads, response metrics, and extracted database error messages, the Automated SQL Injection Scanning Tool provides security professionals with a powerful yet user-friendly solution for comprehensive vulnerability assessment, automatically saving reports in a dedicated directory for easy reference and sharing.

2.5 Existing SQL Injection Tools

This section discusses various sql injection tools such as Burp Suite (Community Edition), Curl, and Wget.

2.5.1 Burp Suite (Community Edition)

Burp Suite (Community Edition), developed by PortSwigger, is a comprehensive web application security testing platform that includes tools for various manual testing activities, such as intercepting HTTP requests, analyzing web traffic, and identifying vulnerabilities. While the Community Edition lacks the automated scanner of the Professional version, which can slow down the testing process, it allows testers to manually inspect and manipulate requests to detect vulnerabilities, including SQL injection. The interactive nature of Burp Suite makes it highly versatile, providing features such as repeater, intruder, and proxy tools that facilitate thorough manual testing. However, its complex and feature-rich interface can present challenges for new users, as it requires significant time to learn and use effectively.

2.5.2 cURL

cURL is a command-line tool used for transferring data with URLs and supports a wide variety of protocols, including HTTP and HTTPS. It is a valuable asset for penetration testers conducting manual SQL injection attacks, allowing them to craft and send customized HTTP requests, modify headers, and analyze server responses to identify potential vulnerabilities. Its simplicity and flexibility make it popular for targeted tests and bypassing certain security measures. However, cURL relies heavily on the tester's knowledge of HTTP protocols and SQL injection techniques, which can be a barrier for beginners. Additionally, its lack of a graphical interface and built-in automation can limit efficiency, making it more suited for experienced testers

2.5.3 Wget

Wget, another command-line utility, is primarily used for non-interactive file downloads but can also be leveraged for manual SQL injection testing. With options to customize HTTP requests and capture server responses, Wget enables testers to probe for vulnerabilities by sending crafted payloads. Despite its straightforward usage and ability to chain commands for specific tasks, Wget is limited by its feature set and lacks advanced security testing capabilities compared to more specialized tools. This reliance on user expertise for effective usage can make it less practical for detailed vulnerability analysis, particularly for those unfamiliar with command-line operations.

2.6 Comparison of Existing Tools with Automated SQL Injection Scanning Tool

Table 1 shows a comparison between existing tools, such as Burp Suite (CE), cURL, Wget, and Automated SQL Injection Scanning Tool. Automated SQL Injection Scanning Tool shares some similarities with Burp Suite (CE), particularly in features like built-in payload management, error-based detection, and boolean-based

detection. However, unlike Burp Suite (CE), Automated SQL Injection Scanning Tool adds capabilities such as automated scanning and automated report generationWhile cURL and Wget rely on manual approaches for detection, the tool automates these processes, making it more efficient and user-friendly. Additionally, Automated SQL Injection Scanning Tool incorporates a GUI interface for easier interaction, unlike cURL and Wget, which require command-line expertise. The inclusion of automated scanning and report generation highlights the superiority of the tool in addressing gaps left by existing solutions.

Table 1 *Comparison table between existing system with Automated SQL Injection Scanning Tool*

Features \ Tools	Burp Suite (CE)	cURL	Wget	Automated SQL Injection Scanning Tool
Automated Scanning	No	No	No	Yes
GUI Interface	Yes	No	No	Yes
Built-in Payload Management	Yes	No	No	Yes
Error-Based Detection	Yes	Manual	Manual	Yes
Boolean-Based Detection	Yes	Manual	Manual	Yes
Time-Based Detection	Yes	Manual	Manual	Yes
Union-Based Detection	Yes	Manual	Manual	Yes
Automated Report Generation	No	No	No	Yes

3. Methodology

This project, Agile Model was use as the development methodology. The Agile Software Development Life Cycle (SDLC) is a dynamic and adaptive framework designed to address changing requirements and foster collaboration in software development. This approach is well-suited to the development of the proposed Automated SQL Injection Scanning Tool, ensuring it is efficient, user-friendly, and capable of adapting to evolving cybersecurity threats. The methodology includes several key phases that starts with Requirement Gathering and Analysis, Designing, Implementation, Testing, Deployment, and Feedback. These iterative phases are integral to ensuring the tool meets its objectives through effective detection of vulnerabilities like SQL injection. Through repeated iterations, feedback is continuously incorporated to refine features such as automated scanning, built-in payload management, and report generation. This iterative approach ensures the tool aligns with user needs and cybersecurity standards. Fig. 1 illustrates the six phases of the Agile SDLC, demonstrating how this methodology supports the development and enhancement of the proposed tool.

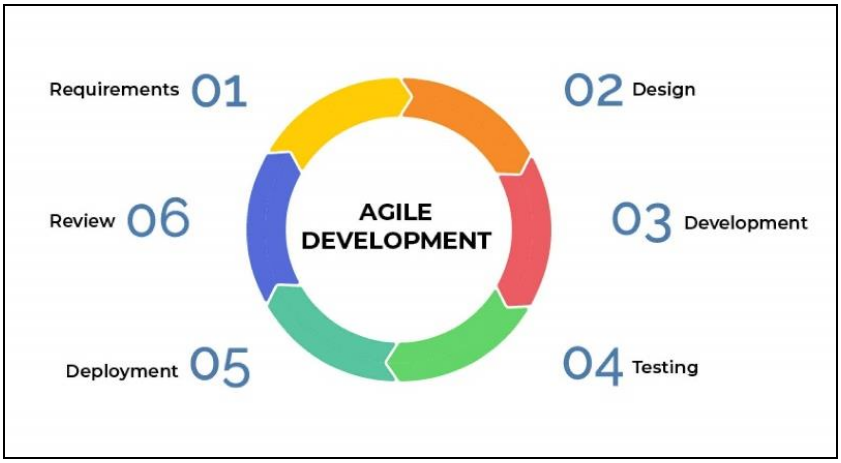


Fig. 1 *Phases in Agile Model*

3.1 Requirement Gathering Phase

In this phase, the focus is on identifying and understanding the requirements for the Automated SQL Injection Scanning Tool. These requirements encompass both functional features such as SQL injection detection, automated reporting, and a user-friendly interface and non-functional features, such as adherence to cybersecurity standards. A key aspect of this phase is defining the detection techniques to ensure the tool

accurately identifies vulnerabilities. This phase is critical in ensuring that the planned features and objectives are well-defined and meet the overall goals of the project, particularly in securing clinic management systems.

3.2 Design Phase

The design phase focused on developing features required by penetration testers, including automated scanning, customizable payload management, vulnerability classification, and report generation. User requirements were derived from common penetration testing workflows and existing industry tools such as Burp Suite Community Edition. This included outlining the tool's key modules, such as the vulnerability detection engine, reporting module, and payload management system. Both functional and non-functional requirements were carefully defined to ensure the tool met the project's objectives. Python was chosen as the programming language due to its extensive libraries and suitability for developing robust cybersecurity tools. Visual Studio Code was selected as the primary development environment, providing a flexible and efficient coding platform. Unified Modeling Language (UML) diagrams, including use case diagrams, activity diagrams, and sequence diagrams, were used to map out the tool's workflow and interactions. The outcome of this phase was a well-defined system design that aligns with the project's goals, ensuring that the Automated SQL Injection Scanning Tool would be effective, efficient, and easy to use while meeting cybersecurity standards.

3.3 Development Phase

In the development phase, Automated SQL Injection Scanning Tool was create based on the design specifications, incorporating the features and functionalities outlined during the requirement gathering and analysis phase. The coding was performed using Visual Studio Code, with Python as the programming language. The software requirements include a Windows operating system, while the hardware specification is an Intel 11th Gen i5 processor with 12.00GB of RAM. Key components such as the vulnerability detection engine and reporting module were developed using advanced algorithms. The user interface was designed to ensure simplicity and ease of use, utilizing wireframes created in Figma as a blueprint. The outcome of this phase was a fully functional tool capable of detecting SQL injection vulnerabilities, generating detailed reports, and providing a user-friendly experience.

3.4 Testing Phase

In the testing phase, the Automated SQL Injection Scanning Tool was evaluated using intentionally vulnerable websites deployed in controlled environments specifically designed for security research, training, and ethical penetration testing activities. To assess the tool's effectiveness from a practical and user-oriented perspective, a User Acceptance Testing (UAT) approach was implemented. A predefined UAT checklist was used to systematically evaluate the tool's core functionalities, including payload injection, parameter detection, response analysis, and report generation. The testing also measured the tool's usability, interface responsiveness, and overall scanning accuracy. Each feature was tested under typical usage conditions to ensure it functioned reliably and met the expected requirements. The tool successfully demonstrated its ability to identify SQL injection vulnerabilities on real-world test targets, while also providing a smooth and intuitive user experience.

3.5 Deployment Phase

In this phase, the Automated SQL Injection Scanning Tool was deployed in a local testing environment for controlled and safe execution. The deployment setup included a local machine configured with the required software dependencies, including Python, PyQt5, and supporting libraries. This environment allowed for complete control over system resources, testing conditions, and network interactions without exposing the tool to unintended live systems.

3.6 Review Phase

In the review phase, the performance and usability of the Automated SQL Injection Scanning Tool were evaluated based on feedback collected through User Acceptance Testing (UAT) forms. Testers who used the tool during the testing phase completed structured UAT forms that assessed various aspects of the tool, including functionality, user interface design, accuracy of vulnerability detection, and overall user experience. The responses were analyzed to determine whether the tool met user expectations and project objectives.

4. Analysis and Design

This section outlines the key components of the analysis and design phase. It encompasses the user requirements, functional and non-functional requirements, system architecture, use case diagram, activity diagram, sequence diagram, and design interface.

4.1 System Architecture

Fig. 2 shows the system architecture of the Automated SQL Injection Scanning Tool. The process begins at the User Interface (UI), where users enter the target URL, choose payloads, and start the scan. The input is first checked by the Input Validation module to ensure it is correct and complete. Next, the Scanner Thread Engine runs the scanning process in the background. It sends SQL injection payloads to the target website and analyzes the responses to detect any vulnerabilities. While scanning, the Progress Monitor updates the user with real-time status and progress information. Once the scan is completed, the results are shown in the Result View, which includes details such as vulnerable parameters, payloads used, and response information. Finally, the Report Generation module creates a summary report that can be saved or exported. This architecture provides a smooth, responsive, and user-friendly scanning experience.

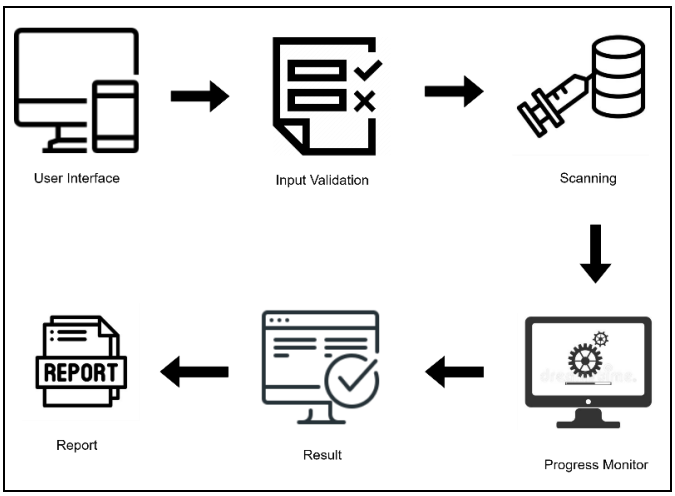


Fig. 2 *System Architecture of Automated SQL Injection Scanning Tool*

4.2 Functional Requirements

Table 2 outlines the functional requirements of the Automated SQL Injection Scanning Tool. The Scanning module is responsible for detecting SQL injection vulnerabilities using various techniques, including error-based, boolean-based, time-based, and union-based injection across both GET and POST methods. It supports multiple databases types and allows users to configure scanning behavior while maintaining a responsive user experience through threaded execution. The Payload Management module handles the storage, categorization, and customization of SQL injection payloads using an SQLite database, allowing users to add, edit, delete, import, or export payloads in multiple formats. The User Interface module provides a PyQt5-based graphical interface with dedicated sections for scanning, reporting, settings, and more, offering real-time progress updates, detailed results, and visual feedback. Finally, the Reporting module generates comprehensive reports summarizing detected vulnerabilities, affected parameters, and payloads used, with support for exporting reports in TXT and PDF formats and managing previously saved results. These modules collectively ensure the tool is functional, user-friendly, and effective for SQL injection vulnerability assessments.

Table 2 *Functional Requirements for Automated SQL Injection Tool*

Functional Requirements	Description
Scanning Module	Handles SQL injection scanning across various request types and databases.
Payload Management Module	Manages SQL injection payloads stored in a local database.
User Interface Module	Provides a responsive and intuitive graphical interface for user interaction.
Reporting Module	Generates detailed security reports, including information on vulnerabilities

4.3 Non-Functional Requirements

Table 3 outlines the non-functional requirements for the Automated SQL Injection Scanning Tool. The Operational requirement ensures that the tool can handle user configuration inputs, such as target URLs and scan parameters, smoothly and efficiently. The Performance requirement specifies that the tool should complete vulnerability scans within a reasonable and standard timeframe to maintain efficiency. Finally, the Security

requirement emphasizes that the tool must process and validate test results accurately, identifying vulnerabilities without compromising the integrity of the system.

Table 3 *Non-Functional Requirements for Automated SQL Injection Tool*

Non-Functional Requirements	Description
Operational	Handles user configuration inputs, such as target URLs and scan parameters.
Performance	Complete vulnerability scans within a standard timeframe.
Security	Processes and validates test results, identifying potential vulnerabilities.

4.4 User Requirements

User requirements focus on the features and functionalities that users expect from the tool. These requirements ensure that the tool is user-centric and meets their specific needs. Table 4 shows the user requirements of automated SQL Injection Scanning Tool.

Table 4 *User Requirements for Automated SQL Injection Tool*

No	User Requirements
1.	Users should be able to scan target URLs.
2.	Users should be able to initiate vulnerability scans with minimal effort.
3.	Users should receive detailed, actionable reports outlining vulnerabilities.
4.	Users should be able to export reports in multiple formats, such as PDF or txt.
5.	The tool should provide real-time feedback during scans, including progress updates.

4.5 Use Case Diagram

The use case diagram provides a visual representation of the interactions between the user and the tool. By mapping out these interactions, the diagram helps in understanding the tool's scope, user requirements, and key processes. Fig. 3 illustrates the interaction between a User and the Automated SQL Injection Scanning Tool, highlighting its core functionalities.

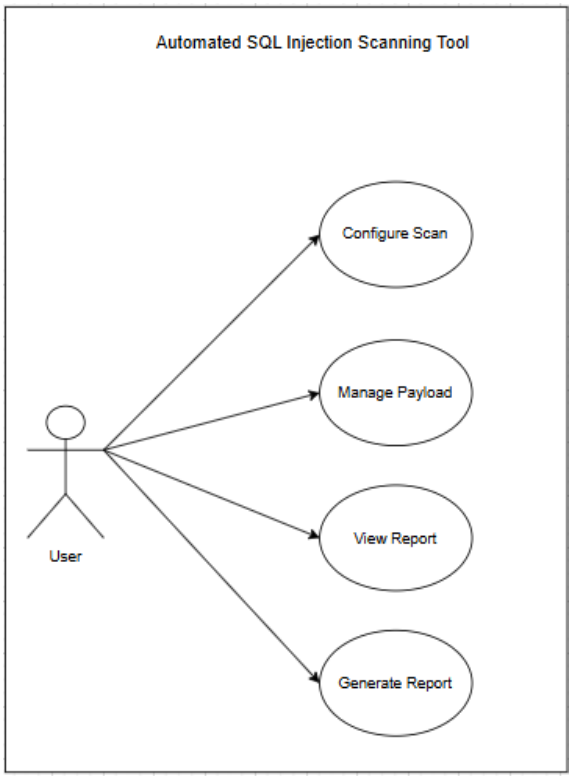


Fig. 3 *Use Case Diagram of Automated SQL Injection Scanning Tool*

4.6 Activity Diagram

Fig. 4 illustrates the activity diagram of the Automated SQL Injection Scanning Tool. This diagram represents the step-by-step flow of activities involved in using the tool. The process begins with the user inputting the target URL and adding or uploading SQL injection payloads. The system then performs a validation check on both the URL and payloads. If the input is invalid, an error message is displayed, and the user is prompted to make corrections. If the input is valid, the system proceeds to start the scanning process. During the scanning phase, the tool continuously monitors progress until the scan is complete. Once scanning finishes, the user is able to view the results, which include any vulnerabilities detected. The user then has the option to export the report. If selected, the system generates and saves a detailed report summarizing the scan findings. If the user chooses not to export, the process ends. This diagram provides a clear overview of how the user interacts with the tool, emphasizing key actions, validation steps, and decision points throughout the scanning workflow.

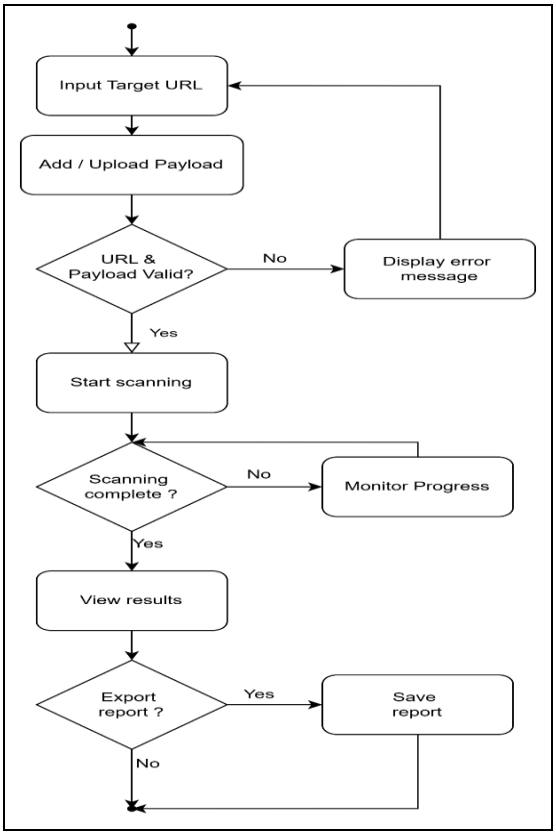


Fig. 4 *Activity Diagram of Automated SQL Injection Scanning Tool*

4.7 Sequence Diagram

The Sequence Diagram in UML provides a detailed representation of the interactions between the components of the Automated SQL Injection Scanning Tool. It illustrates the sequential flow of actions initiated by the user, starting with inputting the target URL and uploading payloads. The diagram highlights how the system validates the input, handles errors such as invalid URLs or payloads, and proceeds to the scanning process. Fig. 5 illustrates sequence diagram of Automated SQL Injection Scanning Tool.

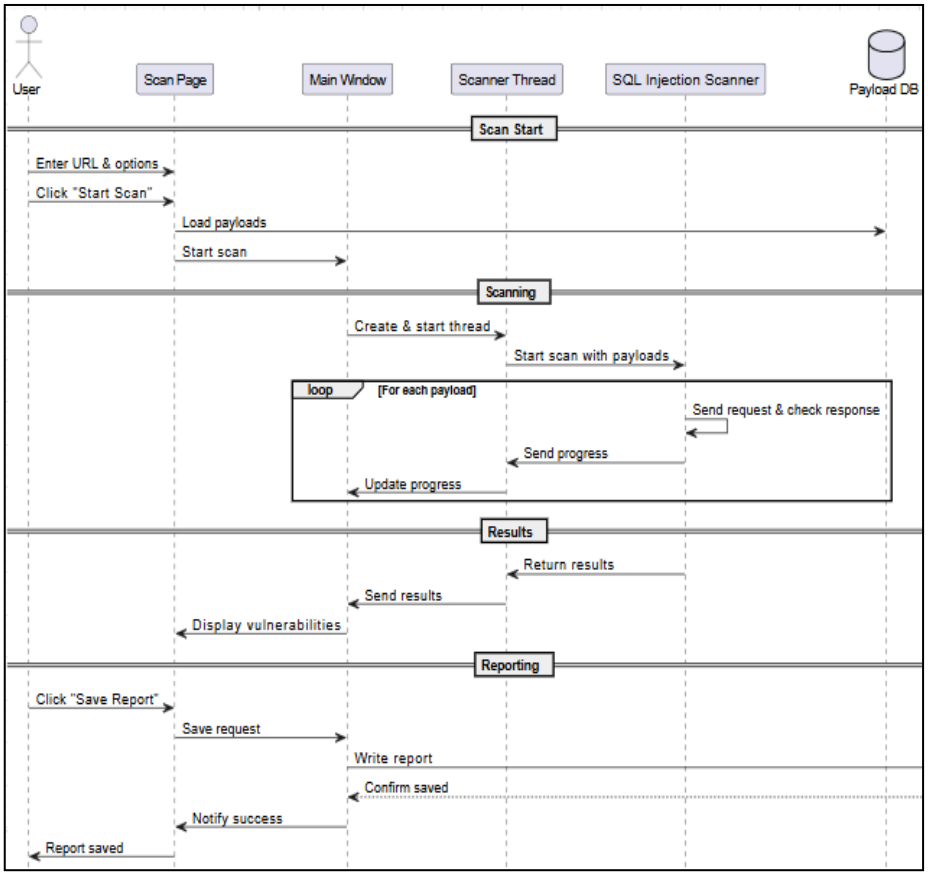


Fig. 5 *Sequence Diagram of Automated SQL Injection Scanning Tool*

4.8 Test Plan

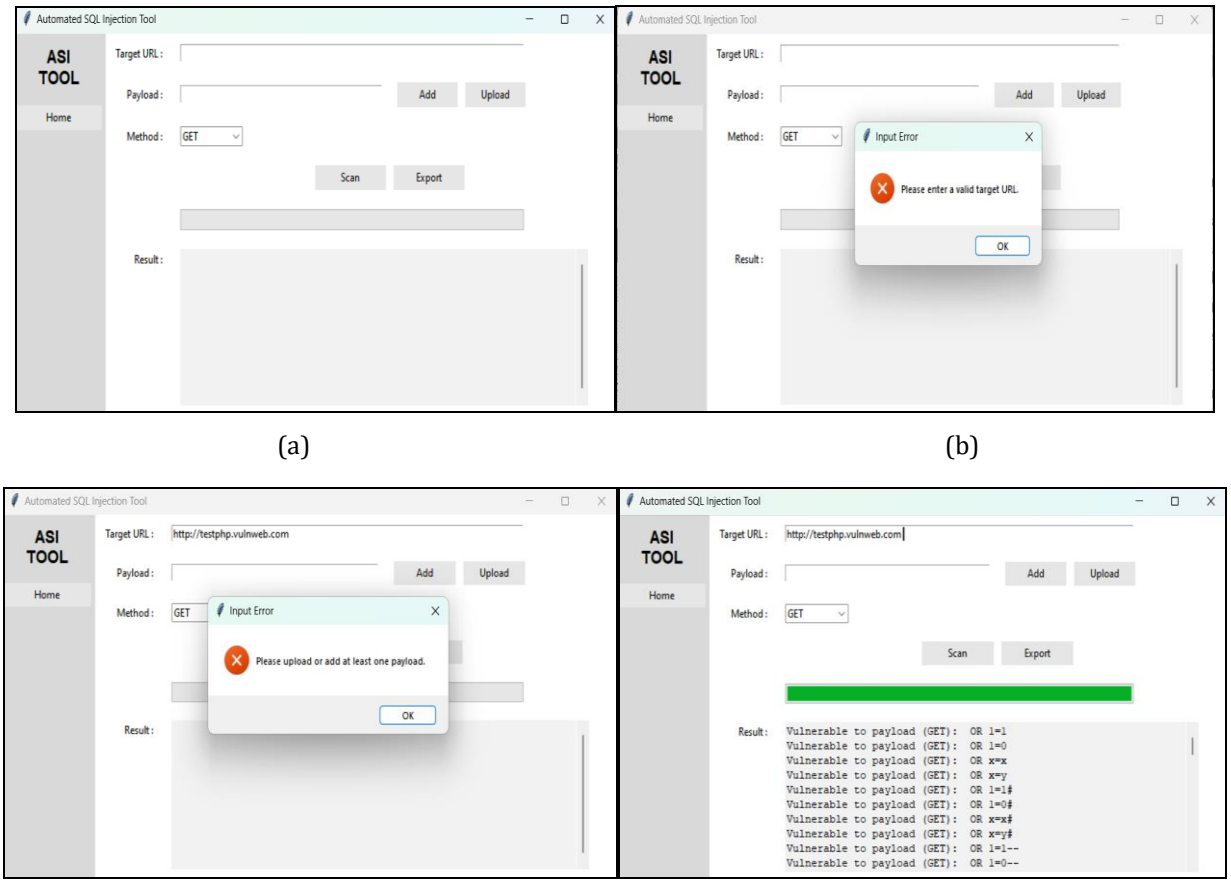
This test plan outlines the testing approach for the Automated SQL Injection Scanning Tool, focusing on evaluating its functionality, usability, and performance in a controlled environment. The intentionally vulnerable website serves as a controlled and authorized testing environment used to evaluate the tool's SQL injection detection capabilities. Table 5 shows the test plan.

Table 5 *Test Plan for Automated SQL Injection Scanning Tool*

No	Test Case	Expected Results	Actual Result
1.	Initiate Scan	Scanning starts and progress updates displayed.	Pass / Fail
2.	Payload Management	User can view, add, edit, delete, import, and export SQL injection payloads; changes are saved and reflected correctly in the payload database.	Pass / Fail
3.	Monitor Progress	Progress bar updates as scan progresses	Pass / Fail
4.	Detect SQL Vulnerabilities	System detects and classifies vulnerabilities.	Pass / Fail
5.	Generate Report	Detailed report with findings and insights.	Pass / Fail
6.	Export Report	Report successfully exported.	Pass / Fail

4.8 Design Interface

This section provides an overview of the interface layout and its functionalities, emphasizing its simplicity and efficiency in facilitating complex tasks. It is designed to be intuitive and user-friendly, enabling users to configure scans, input target details, upload payloads, monitor progress, and view results seamlessly. Fig. 6 shows the tool interface.



(a) (b)

(c) (d)

Fig. 6 *Design Interface for Automated SQL Injection Scanning Tool (a) Tool Interface (b) Input Error for URL Interface (c) Input Error for Payload Interface (d) Successful Scanning Interface*

Fig. 6 demonstrates the user interface design and error-handling features of the Automated SQL Injection Tool (ASI Tool), showcasing different scenarios during its operation. Fig. 6(a) represents the initial state of the tool's interface. At this stage, the user has not provided any inputs. The interface includes fields for entering the target URL, adding or uploading payloads, selecting the HTTP method (GET/POST), and initiating the scan or exporting results. The results section is empty, indicating no scans have been performed. Next. Fig. 6(b) highlights an error triggered when the user attempts to perform a scan without entering a valid target URL. The error message, "Please enter a valid target URL," is displayed, ensuring that the tool does not proceed with incomplete or invalid inputs. This validation is critical to avoid unnecessary errors during the scanning process.

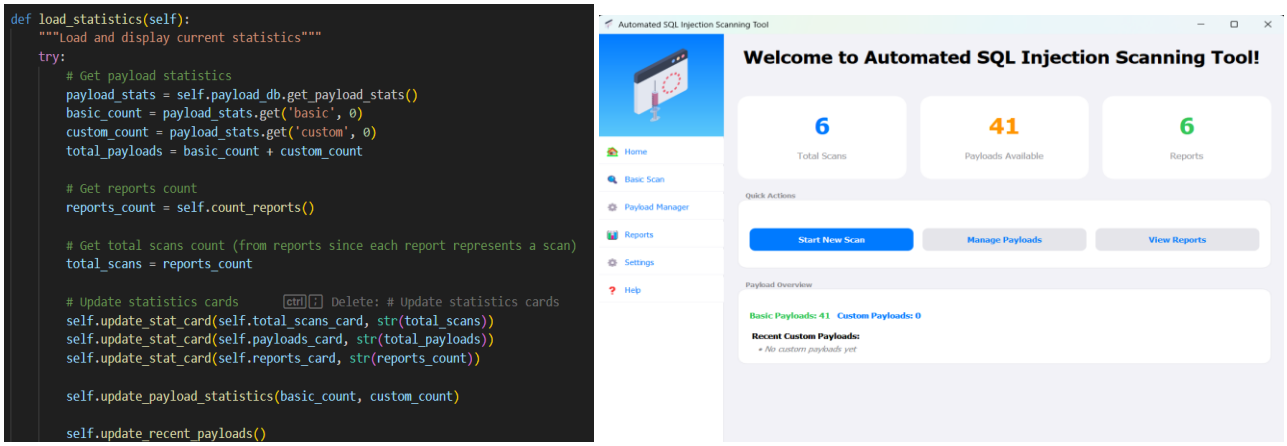
Fig. 6(c) shows another error scenario where the user tries to initiate a scan without adding or uploading any payloads. The error message, "Please upload or add at least one payload," is displayed to notify the user of the missing inputs. This step ensures the tool has the necessary payloads to conduct a meaningful SQL injection test. Finally, Fig. 6(d) presents the interface after a successful scanning process. The progress bar indicates that the scan is complete, and the results section displays vulnerabilities identified using various payloads. This output confirms that the tool successfully detected SQL injection vulnerabilities in the target URL.

5. Implementation and Testing

This section presents the implementation and testing of the Automated SQL Injection Scanning Tool for Vulnerable Website. Section 5.1 details the implementation of each module, including the scanning engine, payload management system, user interface, and reporting functionality. It also explains key algorithms and design decisions applied during development. Section 5.2 outlines the testing results, covering unit testing, integration testing, system testing, and user acceptance testing conducted using intentionally vulnerable websites to evaluate the tool's accuracy, performance, and usability.

5.1 Home Dashboard Module

The Home Dashboard serves as the entry point for users, providing statistics and quick access to key functionality. Fig. 7 illustrates the method used to retrieve and display system statistics on the dashboard. It gathers data from the payload database and the file system, including the total number of scans performed, the number of available payloads, and the count of saved reports. Fig. 7(b) shows the statistics displayed on the UI cards, providing users with a quick and informative overview of the tool's activity and usage.



(a) (b)

Fig. 7 Home Dashboard Algorithm (a) *Algorithm to retrieves statistics (b) Home Dashboard Overview*

5.2 Scanning Module

This module performs comprehensive SQL injection vulnerability scanning against web applications. Users can target specific URLs, select from various payload categories, and choose between GET and POST HTTP methods for testing. The scanner intelligently injects payloads into parameters and analyzes responses for database error patterns, timing anomalies, and other vulnerability indicators.

```
def test_url(self, url, payloads, method='GET', progress_callback=None):
    """Test URL with multiple payloads"""
    results = []
    total_payloads = len(payloads)

    print(f"DEBUG: Testing URL: {url} with {total_payloads} payloads using {method} method")

    for i, payload_data in enumerate(payloads):
        if progress_callback:
            progress = int(((i + 1) / total_payloads) * 100)
            progress_callback(progress, f"Testing payload {i+1}/{total_payloads}")

        payload = payload_data[1] if isinstance(payload_data, tuple) else payload_data

        print(f"DEBUG: Testing payload {i+1}: {payload[:50]}...")  # Show first 50 chars

        try:
            if method == 'GET':
                result = self.test_get_injection(url, payload)
            else:
                result = self.test_post_injection(url, payload)

            if result:
                print(f"DEBUG: Vulnerability found with payload: {payload}")
                results.append(result)
            else:
```

Fig. 8 Scanning Algorithm

Fig. 8 illustrates the main scanning method, which serves as the entry point for executing SQL injection tests. The method iterates through each selected payload and tests it against the target URL using either GET or POST HTTP methods. It employs advanced detection techniques, including error-based, boolean-based, time-based, and union-based SQL injection detection, across multiple database systems such as MySQL, PostgreSQL, SQL Server, Oracle, and SQLite. Throughout the scanning process, it analyzes server responses to identify potential vulnerabilities and compiles the results. Progress updates are communicated in real time through a callback function, allowing the user interface to dynamically display the scanning status.

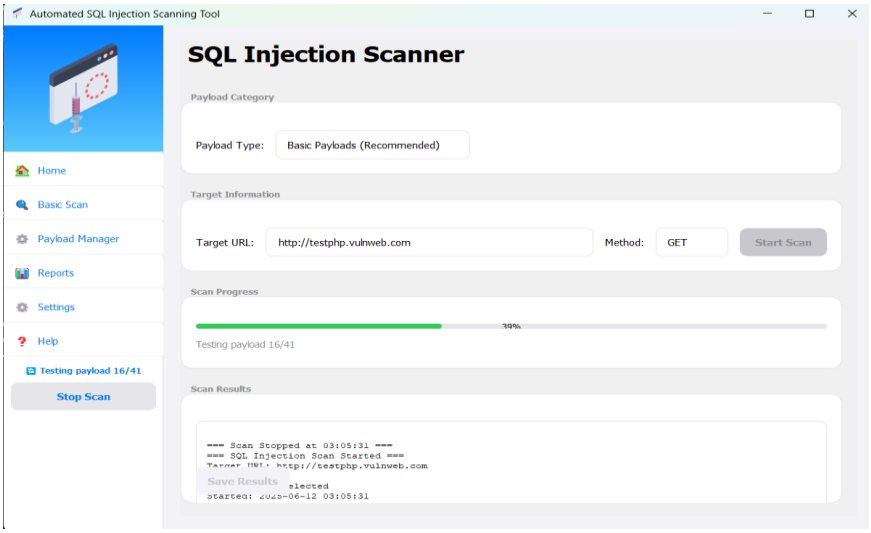


Fig. 9 Scanning Module Process

Fig. 8 presents the scanning interface of the Automated SQL Injection Scanning Tool. At the top, a target URL input field allows users to specify the website to be scanned, accompanied by method selection options (GET or POST). Below this, payload selection controls enable users to choose SQL injection payloads from different categories. The central section displays real-time scanning progress, featuring a progress bar and status messages that indicate which payloads are currently being tested. Detected vulnerabilities are shown in a structured table format, listing the vulnerable parameter, payload used, vulnerability type, and response details. Each entry is expandable to reveal additional technical information, such as database error messages and response times. The interface also includes action buttons for starting or stopping scans and exporting results for reporting purposes.

5.3 Payload Manager Module

This module manages SQL injection payloads used for vulnerability scanning. Users can view, create, edit, and delete custom payloads while maintaining a protected set of built-in basic payloads. The module integrates with the SQLite database backend for persistent storage and provides real-time statistics on payload counts by category.

```
def add_payload(self):
    """Add new custom payload"""
    try:
        dialog = PayloadDialog(self)
        if dialog.exec_() == dialog.Accepted:
            payload = dialog.get_payload()
            if payload:
                self.db.create_payload(payload, 'custom')
                self.refresh_payloads()
                QMessageBox.information(self, "Success", "Custom payload added successfully!")
            else:
                QMessageBox.warning(self, "Error", "Payload cannot be empty!")
    except Exception as e:
        QMessageBox.critical(self, "Add Error", f"Failed to add payload:\n{str(e)}")
```

(a)

```
def edit_payload(self):
    """Edit selected payload (only custom payloads)"""
    try:
        current_row = self.payload_table.currentRow()
        if current_row >= 0:
            payload_id_item = self.payload_table.item(current_row, 0)
            if not payload_id_item:
                QMessageBox.warning(self, "Selection Error", "Please select a valid payload to edit.")
                return

            payload_id = int(payload_id_item.text())

            # Get payload data
            payloads = self.db.read_payloads()
            payload_data = next((p for p in payloads if p[0] == payload_id), None)
```

(b)

```
def delete_payload(self):
    """Delete selected payload (only custom payloads)"""
    try:
        current_row = self.payload_table.currentRow()
        if current_row >= 0:
            payload_id_item = self.payload_table.item(current_row, 0)
            category_item = self.payload_table.item(current_row, 1)
            payload_preview_item = self.payload_table.item(current_row, 2)

            if not all([payload_id_item, category_item, payload_preview_item]):
                QMessageBox.warning(self, "Selection Error", "Please select a valid payload to delete.")
                return

            payload_id = int(payload_id_item.text())
            category = category_item.text().lower()
            payload_preview = payload_preview_item.text()

            if category == 'basic':
                QMessageBox.warning(self, "Cannot Delete", "Basic payloads cannot be deleted. Only custom payloads can be removed.")
                return

            reply = QMessageBox.question(
                self,
                "Confirm Delete",
                f"Are you sure you want to delete this custom payload?\n\n{payload_preview}",
                QMessageBox.Yes | QMessageBox.No
            )
```

(c)

Fig. 10 Payload Manager CRUD (a) *Algorithm to Add payloads (b) Algorithm to Edit payloads (c) Algorithm to Delete payloads*

Fig.10 illustrates the algorithms used in the Payload Management Module of the Automated SQL Injection Scanning Tool. This module allows users to manage SQL injection payloads used during scanning. Users can add new custom payloads in Fig. 10(a), edit existing ones in Fig. 10(b), or delete unwanted entries in Fig. 10(c), while a built-in set of basic payloads remains protected and read-only.

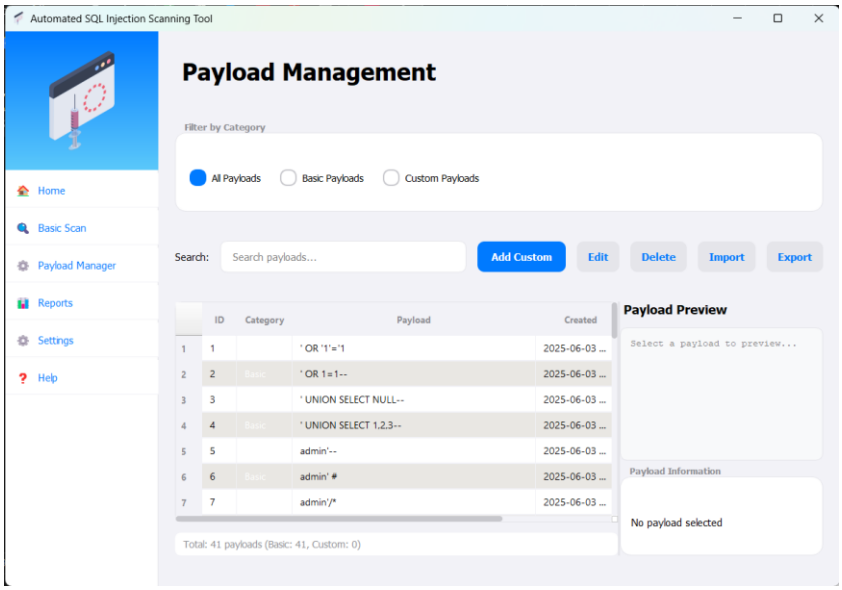


Fig. 11 Payload Manager Interface

The interface presents a searchable and filterable table that displays each payload's content, category, and creation date. Payloads can be imported from text files (one payload per line) or JSON format and exported for backup or sharing. A preview panel shows the full content and metadata of the selected payload. The system enforces proper validation and error handling for each operation to maintain data integrity. Custom payloads are stored in a persistent SQLite database, and real-time statistics display the number of payloads by category. The design ensures a smooth and user-friendly experience by providing helpful feedback messages throughout all payload operations.

5.4 Reporting Module

This module handles the generation, management, and export of SQL injection scan reports. Users can view a chronologically sorted list of scan reports with timestamps, access detailed report content, and export findings in multiple formats.

```
def from_text_file(cls, file_path: str) -> 'ScanReportData':
    """Parse scan report from existing text file"""
    report = cls()

    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            content = f.read()

        # Parse header information
        report.target_url = cls._extract_field(content, r'Target URL:\s*(.+)')
        report.method = cls._extract_field(content, r'Method:\s*(.+)')
        report.scan_date = cls._extract_field(content, r'Scan Date:\s*(.+)')

        payloads_str = cls._extract_field(content, r'Payloads Tested:\s*(\d+)')
        report.payloads_tested = int(payloads_str) if payloads_str else 0

        vulns_str = cls._extract_field(content, r'Vulnerabilities Found:\s*(\d+)')
        report.vulnerabilities_found = int(vulns_str) if vulns_str else 0

        # Parse vulnerabilities
        report.vulnerabilities = cls._parse_vulnerabilities(content)

    except Exception as e:
        print(f"Error parsing report file {file_path}: {e}")

    return report
```

(a)

```
def export_to_pdf(self, report_data: ScanReportData, output_path: str) -> bool:
    """Export scan report to PDF format"""
    if not REPORTLAB_AVAILABLE:
        print("Reportlab not available. Cannot export to PDF.")
        return False

    try:
        doc = SimpleDocTemplate(
            output_path,
            pagesize=A4,
            rightMargin=72,
            leftMargin=72,
            topMargin=72,
            bottomMargin=18
        )

        story = []

        # Add title page
        story.extend(self._create_title_page(report_data))
        story.append(PageBreak())

        # Add executive summary
        story.extend(self._create_executive_summary(report_data))
        story.append(PageBreak())

        # Add vulnerability details
        story.extend(self._create_vulnerability_details(report_data))
```

(b)

Fig. 12 Reporting Module Algorithm (a) *Algorithm to export as txt file (b) Algorithm to export as pdf*

Fig. 12 presents the implementation of the report export functionality within the Automated SQL Injection Scanning Tool. Fig. 12(a) shows the code responsible for exporting scan results to a structured TXT file, while Fig. 12(b) displays the implementation for generating a professional PDF report. The TXT export generates a well-organized plain text report with clear section divisions using ASCII borders, consistent

formatting of vulnerability details, and a logical structure that includes an executive summary, severity statistics, and technical findings. The PDF export uses the ReportLab library to create a visually polished document featuring a title page, executive summary, and detailed tables with color-coded severity levels and proper pagination for improved readability. Both export methods process data through the ScanReportData class, which formats vulnerability details, calculates severity distributions, and appends a timestamp for traceability. The implementation includes error handling mechanisms to manage export failures gracefully and ensures fallback behavior when necessary, components are unavailable. Together, these export options provide users with flexible, professional reporting capabilities suitable for documentation or audit purposes.

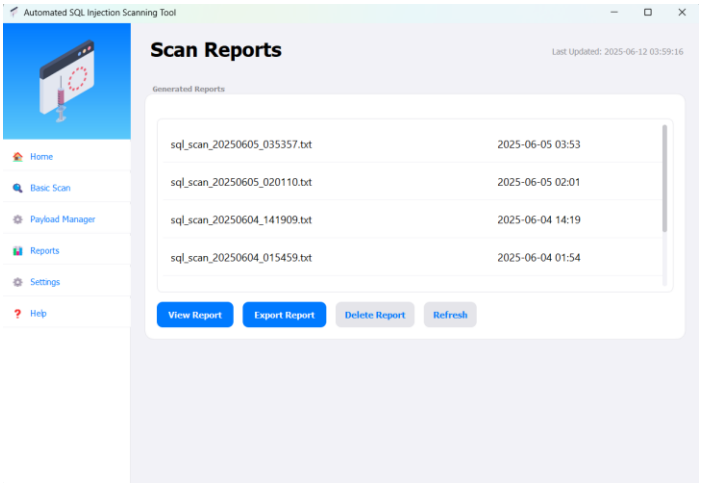


Fig. 13 Reporting Module Interface

Fig. 13 shows the report management interface of the Automated SQL Injection Scanning Tool. The reports are listed in chronological order, with the most recent scans, such as those conducted on June 4th and 5th, 2025, displayed at the top. Each report entry includes relevant details to help users quickly identify the scan. At the bottom of the interface, there are four action buttons. The "View Report" button allows users to open and read the selected report. The "Export Report" button opens a dialog for exporting the report in PDF, enhanced TXT, or original format. The "Delete Report" button removes the selected report from the system. The "Refresh" button updates the report list to reflect any newly generated or modified entries. This interface provides a clear and efficient way for users to manage and access their scan reports.

5.5 Functional Testing

Functional testing is crucial for ensuring the tool's reliability and quality. It helps identify problems, enhance user experience, and verify that the tool meets its functional requirements.

Table 6 *Functional Testing results for Automated SQL Injection Scanning Tool*

No	Descriptions	Expected Results	Actual Result
1.	The system must be executed from start to end.	System launches successfully, completes full execution cycle, and terminates gracefully without errors.	Pass
2.	User able to initiate the scan.	Scan starts when user clicks start button	Pass
3.	User able to monitor the scan progress in real-time	Progress indicator updates continuously showing current scan status and completion percentage	Pass
4.	User able to generate detailed reports	System produces comprehensive reports containing scan results and vulnerability findings.	Pass
5.	User able to export reports in PDF or TXT format.	Reports are successfully exported in both PDF and TXT formats without data loss.	Pass
6.	System accurately identifies vulnerabilities	Tool correctly detects SQL injection vulnerabilities with proper classification and severity ratings.	Pass
7.	The system is user-friendly and intuitive.	Interface is easy to navigate with clear menus and logical workflow for all user operations.	Pass

Table 6 presents the functional testing results for the Automated SQL Injection Scanning Tool, confirming successful validation of all core system functionalities with a 100% pass rate. The testing covered seven critical areas, including system execution, scan configuration, real-time progress monitoring, payload handling, and report generation. Each test case passed, demonstrating that the tool performs reliably from start to finish and enables users to initiate scans intuitively. Real-time scan progress was accurately displayed through status updates and a responsive UI. The reporting module was thoroughly tested, successfully generating detailed vulnerability reports and exporting them in both PDF and TXT formats without data loss or formatting issues. The tool's primary objective, accurately detecting and classifying SQL injection vulnerabilities, was also validated. Additionally, usability testing confirmed the system's user-friendly interface, with intuitive navigation and clearly labeled features. These results confirm that the tool is fully functional, user-ready, and reliable for identifying SQL injection vulnerabilities in intentionally vulnerable websites used for security assessment and educational purposes.

5.6 Test Plan Results

The test plan results demonstrate comprehensive validation of the automated SQL injection scanning tool's core functionalities, with all six test cases achieving successful outcomes. The test covered the complete workflow from scan initiation through report generation, validating that users can effectively start scans with proper progress tracking, manage SQL injection payloads through full CRUD operations with persistent database storage, monitor real-time scan progress through dynamic progress indicators, and rely on the system's vulnerability detection capabilities to accurately identify and classify SQL injection vulnerabilities. Additionally, the results confirm the tool's reporting functionality generates detailed findings with actionable insights and successfully exports reports in the required formats. This 100% pass rate across all critical features indicates the tool meets user requirements for usability, functionality, and reliability, providing security professionals with a robust platform for identifying SQL injection vulnerabilities in their target applications. The successful completion of these tests validates the tool's readiness for deployment and demonstrates its capability to serve as an effective security assessment instrument in professional environments. Table 7 shows the test plan results for Automated SQL Injection Scanning Tool.

Table 7 *Test Plan results for Automated SQL Injection Scanning Tool*

No	Test Case	Expected Results	Actual Result
1.	Initiate Scan	Scanning starts and progress updates displayed.	Pass
2.	Payload Management	User can view, add, edit, delete, import, and export SQL injection payloads; changes are saved and reflected correctly in the payload database.	Pass
3.	Monitor Progress	Progress bar updates as scan progresses	Pass
4.	Detect SQL Vulnerabilities	System detects and classifies vulnerabilities.	Pass
5.	Generate Report	Detailed report with findings and insights.	Pass
6.	Export Report	Report successfully exported.	Pass

Table 8 shows the security test plan results for the Automated SQL Injection Scanning Tool reveal a mixed security posture across three critical areas. The tool successfully passes two requirements while failing one important security measure. For secure HTTP methods and headers, the tool earns a passing grade by properly implementing HTTPS protocol with appropriate security headers, which protects data in transit and prevents man-in-the-middle attacks during server communications. Similarly, the error message handling requirement passes inspection, with the system correctly displaying appropriate error messages when users enter malformed URLs or invalid input parameters, demonstrating good input validation practices without exposing sensitive information. However, the tool fails the crucial requirement of encrypting scan results and reports, as it does not apply any encryption algorithms before saving data to the database or exporting files. This represents a significant security vulnerability since sensitive information about discovered SQL injection vulnerabilities could be exposed if the database or exported files are compromised. The development team should prioritize implementing strong encryption for all scan results and reports to address this security gap.

Table 8 *Test Plan Security for Automated SQL Injection Scanning Tool*

No	Checklist	Expected Results	Actual Result
1.	Ensure the tool uses secure HTTP methods and headers.	All HTTP requests use HTTPS protocol with proper security headers implemented.	Pass
2.	Print out error message for invalid inputs.	System displays appropriate error messages when users enter malformed URLs or invalid input parameters.	Pass
3.	Results are encrypted before storage or export	Scan results and reports are encrypted using strong encryption algorithms before being saved to database or exported to files.	Fail

6. Conclusion

In conclusion, this project successfully developed an Automated SQL Injection Scanning Tool for Vulnerable Websites. The tool was evaluated using intentionally vulnerable web applications in controlled and authorized environments, demonstrating its effectiveness in detecting and reporting SQL injection vulnerabilities while supporting ethical security assessment practices. The tool utilizes multiple detection techniques, including error-based, boolean-based, time-based, and union-based methods, to thoroughly assess the target system. Key features such as automated scanning, a modern and user-friendly interface, real-time progress tracking, and detailed report generation contribute to its effectiveness and accessibility compared to traditional or manual testing approaches.

The results of this study highlight the critical need for proactive vulnerability detection tools in securing web applications. The tool proved capable of detecting various SQL injection threats, providing actionable insights that can help developers and security professionals reduce exposure to attacks and improve overall system resilience.

For future work, the tool can be expanded to support detection of additional web vulnerabilities, such as Cross-Site Scripting (XSS), Session Hijacking, and Cross-Site Request Forgery (CSRF). Integrating support for multi-threaded scanning, advanced evasion techniques, and customizable scan profiles would further improve performance and adaptability. Since the evaluation was conducted only on DVWA, future work should include testing on additional vulnerable web applications such as OWASP Juice Shop, WebGoat, and Mutillidae to further validate the effectiveness and generalizability of the tool. Additionally, incorporating machine learning-based anomaly detection and real-time collaboration features could transform the tool into a more comprehensive web security assessment platform. Regular updates and user feedback will be essential to ensure the tool remains effective in addressing evolving cybersecurity threats.

Acknowledgement

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

Conflict of Interest

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

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *M. M. Mohd Shukri, N. Rahim;* ***data collection*** *M. M. Mohd Shukri, N. Rahim;* ***analysis and interpretation of results:*** *M. M. Mohd Shukri, N. Rahim;* ***draft manuscript preparation:*** *M. M. Mohd Shukri, N. Rahim. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Okesola, J. O., Ogunbanwo, A. S., Owoade, A., Olorunnisola, E. O., & Okokpuji, K. (2023). Securing web applications against SQL injecti on attacks - A Parameterised Query perspective). 2023 International Conference on Science, Engineering and Business for Sustainable Development Goals, SEB-SDG 2023. https://doi.org/10.1109/SEB-SDG57117.2023.10124613

[2] Anwar, R. W., Abdullah, T., & Pastore, F. (2021). Firewall best practices for securing smart healthcare environment: A review. In Applied Sciences (Switzerland) (Vol. 11, Issue 19). MDPI. https://doi.org/10.3390/app11199183

[3] Barth, A., Rubinstein, B. I. P., Sundararajan, M., Mitchell, J. C., Song, D., & Bartlett, P. L. (2012). A Learning-Based Approach to Reactive Security. https://doi.org/10.1109/2011.42

[4] Hidayanto, B. C., Akbar, I. A., & Putra, A. Z. D. (2024). Automated Web Security Testing Guide Mapping to Accelerate Process on Penetration Testing. Procedia Computer Science, 234, 1412–1419. https://doi.org/10.1016/J.PROCS.2024.03.140

[5] Kaur, P., Sharma, M., & Mittal, M. (2018). Big Data and Machine Learning Based Secure Healthcare Framework. Procedia Computer Science, 132, 1049–1059. https://doi.org/10.1016/j.procs.2018.05.020

[6] Farea, A. A. R., Amran, G. A., Farea, E., Alabrah, A., Abdulraheem, A. A., Mursil, M., & Al-Qaness, M. A. A. (2023). Injections Attacks Efficient and Secure Techniques Based on Bidirectional Long Short Time Memory Model. Computers, Materials and Continua, 76(3), 3605–3662. https://doi.org/10.32604/cmc.2023.040121

[7] Zhang, F., Yang, J., Li, J., Zhang, J., Li, J., Chen, L., Diao, X., Wang, Q., & Dou, Z. (2024). Integrated physical safety–cyber security risk assessment based on layers of protection analysis. Chemical Engineering Research and Design, 212, 405–420. https://doi.org/10.1016/j.cherd.2024.10.036

[8] Paul, A., Sharma, V., & Olukoya, O. (2024). SQL injection attack: Detection, prioritization & prevention. Journal of Information Security and Applications, 85. https://doi.org/10.1016/j.jisa.2024.103871

[9] Marandi, M., Bertia, A., & Silas, S. (2023). Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline. 2023 World Conference on Communication and Computing, WCONF 2023. https://doi.org/10.1109/WCONF58270.2023.10235015

[10] Zhang, S., Li, Y., & Jiang, Q. (2023). Feature Ratio Method: A Payload Feature Extraction and Detection Approach for SQL Injection Attacks. Proceedings - 2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science, ACCTCS 2023, 172–175. https://doi.org/10.1109/ACCTCS58815.2023.00019

[11] Su, Z. C., Hlaing, S., & Khaing, M. (2020). A Detection and Prevention Technique on SQL Injection Attacks. https://ieeexplore.ieee.org/document/9022833

[12] Alhogail, A., & Alkahtani, M. (2024). Automated Extension-Based Penetration Testing for Web Vulnerabilities. Procedia Computer Science, 238, 15–23. https://doi.org/10.1016/J.PROCS.2024.05.191