DETECTION OF SQL INJECTION ATTACKS IN ONLINE LEARNING SYSTEM USING RABIN-KARP PATTERN MATCHING ALGORITHM

SAN SAN WAI

M.C.Sc. **DECEMBER 2022**

DETECTION OF SQL INJECTION ATTACKS IN ONLINE LEARNING SYSTEM USING RABIN-KARP PATTER MATCHING ALGORITHM

BY

San San Wai

B.C.Sc(Q)

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

(M.C.Sc.)

University of Computer Studies, Yangon

DECEMBER 2022

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to the following persons who have contributed directly or indirectly towards the completion of this thesis and helped me make this dissertation possible.

Secondly, I would like to express my special appreciation and thanks to my principal **Dr. Mie Mie Khin,** the Rector of the University of Computer Studies, Yangon who gave me the opportunity to develop this thesis for her general guidance during the period of study.

I would also like to extend my deepest gratitude to **Dr. Mie Mie Thet Thwin,** the former Rector of the University of Computer Studies, Yangon who gave me the opportunity to develop this thesis for her general guidance during the period of study.

I would like to thank and my sincere gratitude to Course Coordinator, **Dr. Si Si Mar Win,** Professor and **Dr. Tin Zar Thaw**, Professor, Deans of the Master 26th batch, University of Computer Studies, Yangon, for her excellent guidance.

I would like to special thanks and my sincere gratitude to my supervisor, **Yi Mon Thet,** Associate Professor of Faculty of Information Science, University of Computer Studies, Yangon, gave me the opportunity to do this thesis, and also gave invaluable recommendations regarding to this thesis.

Finally, I would like to thank the technical and support staff, teachers, from University of Computer Studies, Yangon. Also, I would like to thank my superiors for all their hard work guiding me to the completion of this thesis. In addition, I would like to thank all of my thesis's board examiners who gave the precious comments and corrections to my work

for getting good end result. I would like to thank **Daw Aye Aye Khin**, Lecturer, Department of English, University of Computer Studies, Yangon, for her valuable supports and editing my thesis from the language point of view.

The completion of this master's course. would not have been possible without the guidance and support of my parents, so I would like to thank them. This would not have been possible without the help of my sisters, so I'd like to thank them, too. Therefore, I also thank my family who encouraged me and prayed for me throughout the time of my research. Last but not least, I am extremely grateful to all of my teachers, my colleagues and all of my friends for their invaluable and precious help and general guidance.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

---------- ----------

Date San San Wai

ABSTRACT

SQL injection is one of the most threatening web application attacks used against SQL database servers and web applications such as online learning, online banking, and online shopping, etc. Due to the pandemic of COVID-19, a variety of web application activities such as learning, banking, and shopping are available. Online learning is also an important role in universities, colleges, institutions and schools for continuous learning from anywhere and anytime. Attackers mainly target online learning web application with these opportunities by using SQL injections to get unauthorized access and perform unauthorized data modification. SQL Injection is also a type of web application security vulnerability in which an attacker is able to submit a database SQL command which is executed by a web application, exposing the back-end database. To overcome this problem from attacking with SQL injection in web applications, there are many methods to detect SQLIAs. Among them, the pattern matching approach is one of the most popular approaches in SQL injection detection. Pattern matching is a technique that can be used to identify or detect any anomaly pattern in SQL query sequence. The proposed system uses Rabin-Karp Pattern Matching Algorithm that matches the hash value of the pattern with the hash value of the substring text. The individual characters matching will start if the hash values equal. The hash values calculation step is required as the first step. The proposed system will use SQL injection dataset from Kaggle. The total number of SQL injection patterns is 1224 inject patterns in this dataset. The experimented results show that the detection of SQL injection attack types and attackers' information (such as MAC address, IP address, etc.) and the evaluate the performance in SQL injection detection in terms of Accuracy (ACC). Therefore, this thesis proposes how to detect SQL injection attacks in online learning system web application. The proposed system uses Rabin-Karp Pattern Matching Algorithm to detect the SQL injection attacks and will be implemented with PHP and MySQL database.

Keywords: SQL injection, information security, attack detection, Rabin-Karp Pattern matching Algorithm

CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

		Page
SQL	Structure Query Language	1
SQLIAs	Structure Query Language Injection Attacks	1
MAC	Media Access Control	2
IP	Internet Protocol Address	2
HTTP	Hypertext Transfer Protocol	2
OWASP	Open Web Application Security Project	5
XSS	Cross-Site Scripting	6
XML	External Entities (XXE)	6
DBMS	Database Management System	16
CSV	Comma Separated Value	33
URL	Uniform Resource Locator	47

CHAPTER 1

INTRODUCTION

A variety of web applications are available for day-to-day activities such as online learning, online banking, online shopping, etc. They are attracting the malicious attackers and inevitably facing the vulnerabilities. According to the Web Application Attack Statistics: 2021, a web application on average 500 - 700 attacks per day. SQL Injection Attacks (SQLIAs) are widely used by attackers to obtain unauthorized access to sensitive information as one of the most serious threats to web applications. Therefore, this proposed system aims to detect SQLIAs for online learning web application by using Rabin-Karp pattern matching algorithm.

SQL Injection Attack (SQLIA) is the major and common attacks performed by the attacker. It has turned out to be one of the serious threats. It has also been placed in top ten vulnerabilities of web applications. A malicious SQL query is inserted by the attacker into the web application appending it to the input parameter. Due to lack of strong input validation, SQL injection gets easily appended to the web application and it is executed on the database. It is accessed by the hacker to manipulate the sensitive information present in the database. The current study summarizes various types of attacks like Boolean-based, Union-based, Like-based, Batch Query, Comment-based and Time-based SQL injection attacks, its methods and mechanisms, and detection techniques.

Motivation

There are many web application vulnerabilities so that they are a big area of research. One of the most common and dangerous vulnerabilities is SQL (Structure Query Language) injection that allows the attacker to damage and steal the data from web application backend database. By using various techniques, SQL injection attacks can be done. Some of the attackers use manually by executing SQL commands and others use the existing SQL injection tools (eg. Sqlmap). The variety of the day-to-day activities go to online in the 21st century and also due to the pandemic of COVID-19. It is still necessary to improve the detection of SQL injection attacks in web applications

such as online learning, online banking and online shopping, etc. Therefore, the detection of SQLIAs in online learning system is proposed in this system.

1.1.Related Work

There are a variety of many techniques for detection and prevention of SQL Injection Attack (SQLIA). SQLIA has the top most priority in web- based security problems,

The author's in [1] proposed a novel technique in SQL injection attacks detection and prevention using Bitap string matching algorithm. The algorithm checks whether a given text contains a substring which is "equal" to a given pattern. The system begins by precomputing a set of bitmasks containing one bit for each element of the pattern.

By using Knuth-Morris-Pratt string match algorithm, a novel technique to prevent SQL injection and cross-site scripting attacks was also proposed in [2]. In this work, the filter() function is used to pass every input and this function will block the user, reset the HTTP request, and display a corresponding warning message if at least one function returns True.

Before inclusion of user input with that resulting after inclusion of input, the approach in [3] based on comparing, at run time, the parse tree of the SQL statement. Therefore, not only the database size will also increase but also code conversion to each and every user input is more time consuming.

In [4], a hybrid technique for SQL Injection Attacks detection and prevention was proposed and regardless of the system development language or the database engine the system detects and prevents all types of SQLIAs in different system categories. However, when the database recovery operation is performing after the SQLIA is detected that it takes lot of time delay.

By using Aho–Corasick pattern matching algorithm, the author's in [5] proposed a scheme for detection and prevention of SQL Injection Attack. By using sample of well-known attack patterns, the proposed scheme was evaluated and the initial stage

evaluation showed that the proposed scheme produced not false positive and false negative.

1.2.Objectives of the Thesis

The main objectives of the thesis are as follows:

- To study the SQL Injection Attacks (SQLIAs).
- To apply the Rabin-Karp Pattern Matching Algorithm for detecting SQLIAs (such as Boolean Based, Like Based and Union Based SQL Injection Attacks) in Online Learning System.
- To analyze the detection of SQL injection attack types and attackers' information (such as MAC address, IP address, etc.)

1.3.Organization of the Thesis

This thesis consists of five chapters.

Chapter 1 is the introductory section where the introduction to SQL Injection in web applications. And the related works, the objectives, and the organization of the thesis are presented.

Chapter 2 describes the background theory related to this thesis. It will include web application security, sources, goals and attack types of SQL injection and detection techniques. As one of the detection techniques, how to detect by using Rabin-Karp pattern matching algorithm will be described in detail.

Chapter 3 presents the design of the proposed system by describing system flow, the detail explanation with algorithms.

Chapter 4 describes the implementation of the proposed system in detail and the experimental results.

Finally, Chapter 5 includes conclusion, limitation and further extension of the proposed system.

CHAPTER 2

BACKGROUND THEORY

The online services have been increasing due to rapid development of software and the Internet communications. By the improvement of the Internet, there are many institutions that have been made their online services accessible. Those institutions are looking to attract the users to access their website and services to achieve the best return of their availability on the Internet depending on their activity based on their various aims and purposes. Consequently, the data and the services are normally placed in a web application. Therefore, the users can access the web application over the Internet. They provide web application's features, such as accessibility, availability, and scalability [14].

2.1. Web Application Architecture

A web application [1] generally has a three-tier construction as shown in Figure 2.1 although a web application is simply recognized as a program running on a web browser,. In this figure, a presentation tier is sent to a web browser by request of the browser.

(1) **Presentation Tier:** This tier receives the user input and shows the result of the processing to the user. It can be thought of as the Graphical User Interface (GUI). Flash, HTML, Java script, etc. are all part of the presentation tier, which directly interacts with the user. This tier is analyzed by a web browser.

(2) **CGI Tier:** Also known as the Server Script Process, this is located in between the presentation and database tiers. The data inputted by the user is processed and the result is sent to the database tier. The database tier sends the stored data back to the CGI tier, and it is finally sent to the presentation tier to be viewed by the user. Therefore, data processing within the web application is performed at the CGI Tier and can be programmed in various server script languages such as JSP, PHP, ASP, etc.

(3) **Database Tier:** This tier only stores and retrieves all of the data. All sensitive web application data are stored and managed within the database. Since this tier

is directly connected to the CGI tier without any security check, data in the database can be revealed and modified if an attack on the CGI tier succeeds.

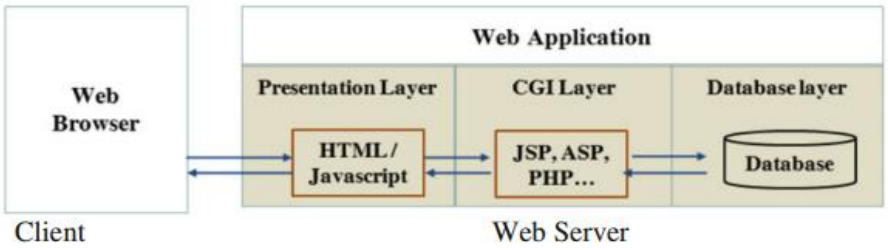


Figure 2.1 Web Application Architecture

2.2. Web Application Security

To access the obtainable services, web applications allow various types of users. The permanent availability of web applications will increase the opportunity for everyone who is looking to exploit and damage these applications for illegal purposes. Hackers also known as people who are damaging a web application and the technique is called hacking. Sometimes, the developers neglect to consider the security side and they are working to implement a functional web application. Consequently, many approaches have been developed to secure the web application harmful attacks. Each approach is looking for the solution from a special perspective; some approaches are to secure the application or the application server and others approaches are looking for to secure the network. Thus, to secure the web application, one needs to start finding the problem that requires a solution.

2.2.1 Vulnerabilities in Web Application

The widespread occurrence of different types of web application vulnerability is defined as the common threat against the security of web application. A vulnerability is a weak point or gap in the application, which allows the malicious attacker to endanger the application stakeholders. The stakeholders can be considered as the user, the owner and other objects that are depending on the application.

There are several types of web application vulnerability; each one has special properties, such as the detection and prevention techniques and the vulnerability style. Figure 2.2 shows the statistics of OWASP (open web application project) top ten vulnerabilities which have classified the percentage of the vulnerability that is used in the hacking of web application in 2022 [19].

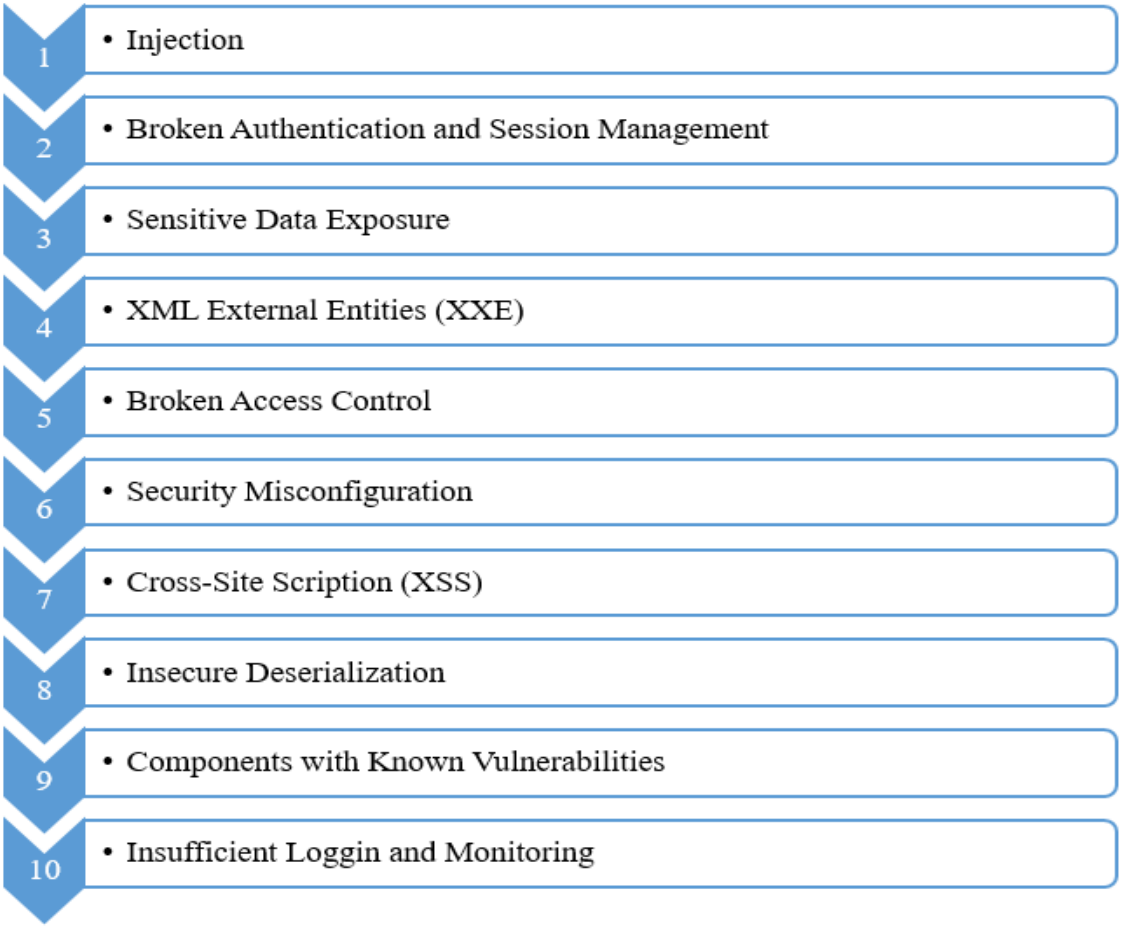


Figure 2.2. OWASP Top 10 2022

The statistics have been conducted according to the number of exploiting the same vulnerability. Accordingly, the OWASP top ten 2022 SQL Injection vulnerability is as follow:

Injection: This type occurs when the attacker injects the application command or queries by untrusted data. The application interpreter will execute the injected command together with the normal command of the application. In this way, the application data will be affected by unauthorized accesses, as well as the execution of unintended commands. The common example of this type is SQL (structured query language).

SQL Injection attacks are one of the major attacks targeting web applications as reported by Open Web Application Security Project (OWASP) [11]. SQL injection, frequently referred to as SQLI, is an arising attack vector that uses malicious SQL code

for unauthorized access to data. This can leave the system vulnerable and can result in severe loss of data [6].

A SQLI attack is one of the deadliest attacks because it compromises authentication, integrity, authorization and confidentiality [7]. This is done by injecting malicious query into forms and getting access to database and manipulate its data. One of the main reasons for high success rates of SQLI attacks is improper form validation which can lead to collecting data from databases and publishing sensitive content for monetary gains. Due to its high frequency and vast scope of research a lot of work has been done in the field, but it was until late when machine learning algorithms started to give promising results.

Because the Structured Query Language Injection (SQLI) attack compromises the main security services: confidentiality, authentication, authorization and integrity [8] it can be considered as the most dangerous attacks of the injection category. Roughly speaking, to get access to a database or manipulate its data (e.g. send the database contents to the attacker, modify or delete the database content, etc.), SQLI attack consists in injecting (inserting) malicious SQL commands into input forms or queries [9], [10].

2.3. SQLI Attack Overview

In this section, a general overview of the SQLI attack will be presented, and it also discusses the SQLI sources and classify their goals and types. The classification and summarization of the main ideas and points are described in Table 2.1.

Table 2.1. SQLI Attack Sources, Types and Goals Classification

Parameter For Classification	**Categories**
Attack Sources	User input Cookies Server variables Second order injection

Attack Goals	Database finger printing Analysing schema Extracting data Amending data Executing dos Equivocating detection Bypassing authentication Remote control Privilege intensification
Attack Types	Tautology Illegal/logically incorrect queries Union query Piggyback query Stored procedure Inference Alternate encoding

2.3.1. SQLI Attack Sources

SQL injection vulnerabilities can be used in a database query and may be found in any application parameter. The authors in [13] cited four sources, user input, cookies, server variables and stored injection through which the SQL Injection Attack (SQLIA) can start.

Injection through user input: To collect data from users (such as registration, login, etc.) or to permit users to specify the data to be retrieved (such as search, adapted view, etc.) web applications generally use forms. Attackers exploited these forms to inject

malicious code, which results in gaining an indented data (retrieve secret data, etc.) or making an indented action (manipulate database, etc.). Login Name, Password, Address, Phone Number, Credit Card Number, and Search are the common form fields.

Injection through cookies: to store users' preferences, web applications use cookies that are files stored on the client machine, which contain state information generated by the web applications. An attacker could embed malicious code into the cookies contents stored in his computer, and therefore, putting web application using the cookies contents to build SQL queries vulnerable to attacks [12].

Injection through server variables: Server variables are a set of parameters that contain network headers, HTTP metadata, and environmental variables. Generally, for auditing usage statistics and identifying browsing trends, web applications use these server variables. The attackers can exploit this vulnerability by placing an SQLIA directly into the server variables if these variables are stored to a database without validation.

Stored injection: The attackers embed malicious inputs into a database to indirectly launch an SQLIA each time that input is used. The example of second-order SQL injection is shown in the following code. In this example, the attacker as a normal user of the website, firstly registers to the application with a seeded username like "admin'--". Then, the attacker will try to change his password. The SQL query to change a user password has generally the following form:

```
queryString="UPDATE users SET password='" + newPassword +"' WHERE
userName='" +userName + "' AND password='" +oldPassword+ ""
```

Assume that newPassword and oldPassword are "newpwd" and "oldpwd", which are chosen by the attacker, the query that will be sent to the database is the following:

```
UPDATE users SET password="newpwd" WHERE userName= "admin" - - " AND
password="oldpwd"
```

Because "- -" is the SQL comment operator, everything after it is ignored, the result of this query is that the database changes the password of the administrator ("admin") to an attacker-specified value.

2.3.2. SQLI Attack Goals

For launching the SQLI attack, the hackers can have different intentions and goals. The main SQLI attack goals are:

Identifying injectable parameters: To inject malicious code the hackers try to identify the parameters. These parameter could be a ”username” field in a form, a ”card number” in a cookie, etc. An attacker can modify the logic of the statement by injecting SQL code, so that when it performs another action. For example, injecting a single quote that is used in SQL to delimit the start or end of a string value could disrupt the pairing of string delimiters and generate an application error, indicating a potential vulnerability to SQL injection.

Performing database fingerprinting: The attacker needs to know the database finger-print to construct a query format supported by the target database engine. The information that identifies a precise type and edition of a database system is known as database finger-print. A different proprietary SQL language syntax is used for each database system. For example, Oracle SQL server uses PL/SQL but Microsoft SQL server uses T-SQL. The attacker must first find out the type and version of the database is used in a web application, and then craft malicious SQL is inputted for that database. Moreover, attackers exploit default vulnerability associated with that version of the database.

Determining database schema: The attacker needs to know the database schema information, such as table names, column number and names, and column data types to successfully extract data from a database. The hackers use the database schema to create an accurate consequent attack with the purpose of extract or modify data from database. Figure 2.3 presents an error message returned by the database system that shows different information related to the database schema (such as number and name of columns) and system (such as ODBC). The hacker uses these pieces of information to construct a successful SQLI attack.

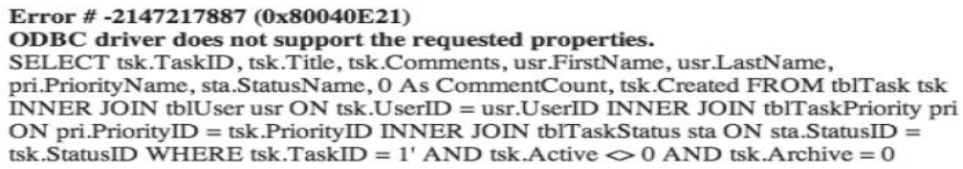


Figure 2.3. Verbose Error Message

Extracting data: In order to extract data values from the database, these types of attacks employ techniques. This attack presents critical risk to web application as extracted information could be sensitive and highly top secret to the web application (example getting customer bank information). Attacks with this intentions are the most common type of SQLIA.

Database alteration: These attacks aim to alter or change information in a database. A hacker can pay much less for an online product by modifying its price, which is generally stored in a database. Another possible attack, consists of adding a malicious link in an online discussion database to commence succeeding Cross-Site-Scripting attacks.

Performing denial of service: This attack intention is to deny service to other users and can have different form, such as shutdown the database of a web application, locking or dropping database tables, etc.

Bypassing authentication: The goal of this attack is to bypass the authentication mechanisms of the web application. It could take the rights and privileges of another user, generally with high rights and privileges if the intruder succeeds to launch such attack [13].

Executing remote commands: Remote commands can store procedures or functions available to database users that are executable code resident on the compromised database server. The hackers attempt to execute arbitrary commands on the database, which can lead to denial of service by executing the shutdown command or database disruption in this type of attack.

Performing privilege escalation: These attacks try to escalate the privileges of the attacker taking advantage of some implementation errors or logical flaws in the database. These attacks focus on exploiting the database user privileges as opposed to

bypassing authentication attacks. When the attacker gains the root privilege, this attack can have a critical consequence especially.

2.3.3. SQLI Attack Types

SQL injections typically fall under three categories [20]: In-band SQLi (Classic), Inferential SQLi (Blind) and Out-of-band SQLi. SQL injections types can be classified based on the methods they use to access backend data and their damage potential.

In-band SQLi

The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQLi's simplicity and efficiency make it one of the most common types of SQLi attack. There are two sub-variations of this method:

1. **Error-based SQLi:** the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.
2. **Union-based SQLi:** this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

Inferential (Blind) SQLi

The attacker sends data payloads to the server and observes the response and behavior of the server to learn more about its structure. This method is called blind SQLi because the data is not transferred from the website database to the attacker, thus the attacker cannot see information about the attack in-band.

Blind SQL injections rely on the response and behavioral patterns of the server so they are typically slower to execute but may be just as harmful. Blind SQL injections can be classified as follows:

1. **Boolean:** that attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the

query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result.

2. **Time-based:** attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database.

Out-of-band SQLi

The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

Out-of-band SQLi is performed when the attacker can't use the same channel to launch the attack and gather information, or when a server is too slow or unstable for these actions to be performed. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker.

SQLI attack can have other several types and forms [13]. In this section, the main SQLI attack types are described as follows:

Tautologies: The general aim of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages are to bypass authentication pages and extract data. In the following example, an attacker submits " ' or 1=1 - -" for the login input field (the values submitted for the other fields are irrelevant). The resulting query is:

```
SELECT accounts FROM users WHERE login=" or 1=1 - - AND pass=" AND pin="
```

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology, which results in a successful authentication of the attacker and the attacker can show all the accounts saved in the database.

Blind SQL injection: Blind SQL injection is a type of SQLI attack that asks the database true or false questions and determines the answer based on the application's response. When the web application is configured to show generic error messages, this attack is often used, but has not mitigated the code that is vulnerable to SQL injection.

Union query: The attacker uses the UNION operator to join a malicious query to the original query in union query attack. The result of the malicious query will be joined to the result of the original query, allowing the attacker to obtain the values of columns of other tables. An example of a union query SQL injection attack is described as follows:

```
SELECT accounts FROM users WHERE login=" UNION SELECT cardNo from
CreditCards where acctNo=10032 - - AND pass=" AND pin=
```

Although, the original first query returns the null set, whereas the second query returns data from the "CreditCards" table.

Piggy-backed query: In this attack, the attacker intends to inject additional queries to extract data, modify or add data. As a result, the DBMS receives multiple SQL queries and the attackers inject additional queries to the original query. An example of a piggy-backed query SQL injection attack is described below:

```
SELECT * FROM userDetails WHERE userid = '12' and password = 'cle';
drop table userDetails ;
```

Stored procedures: The attacker aims to run stored procedures already saved in the database. A standard set of implemented functions called stored procedures that allow even the interaction with the operating system is extended in most existing databases. The developers invoke these stored procedures in their codes to avoid re-writing standard functions. Therefore this property is exploited by an attacker and once determines the web application backend database, SQLIAs can be crafted to execute stored procedures existing in the determined database, including even procedures that interact with the operating system [13].

Alternate encoding: In this type of attack, the attacker tries to conceal the injected text in order to avoid detection by defensive coding practices and automated prevention techniques. These types of attacks allow attackers to escape detection countermeasures. Because they know that most IDS scan the query for certain known "bad characters",

such as single quotes and comment operators, the intruder used these evasion techniques. A malicious code can evade detection mechanism by changing the character

```
SELECT accounts FROM users WHERE login=”legalUser”;
exec(char(0x73687574646f776e)) - - AND pass=”” AND pin=
```

encoding. An example of alternate encoding SQLIA:

The char () function, cited in this example, takes an integer or hexadecimal encoding of a character as input and returns the character spelling. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string “SHUTDOWN”. This result in database shutdown and might lead to denial-of-service attack.

Illegal/logically incorrect queries: The attackers intend to input a manipulated query into the database to generate an error message which contains some information about the cause of the error, in general error message contains give an idea what’s look like of what the database schema looks like. Table 2.1 summarizes the different SQLI attack sources, goals and types [7] and the most common types of SQLIAs are described in Table 2.2.

Table 2.2. The Most Common Types of SQLIAs

No.	Attack Types	Sample Injection Code
1.	Boolean-Based SQLi	anything' or 'x' = 'x
		123 or 1 = 1; --
2.	Like-Based SQLi	username LIKE a%
		or uname like %s

3.	Union-Based SQLi	'UNION select * from users
		union select * from uid;
4.	Batch Query	drop table users;
		; drop table temp --
5.	Comment-based SQLi	--
		"-"
6.	Time-based SQLi	'sleep 50'
		1 waitfor delay '0:0:10'--

2.3.4. SQL Injection Attacks (SQLIAs) Process

SQLIA is a hacking technique which the attacker adds SQL statements through a web application's input fields or hidden parameters to access to resources. Lack of input validation in web applications causes hacker to be successful. For the following examples a web application receives a HTTP request from a client as input and generates a SQL statement as output for the back-end database server [16].

For example, an administrator will be authenticated after typing: employee id=112 and password=admin. Figure 2.2 describes a login by a malicious user exploiting SQL Injection vulnerability. Basically, it is structured in three phases:

1. an attacker sends the malicious HTTP request to the web application

2. creates the SQL statement

3. submits the SQL statement to the back-end database

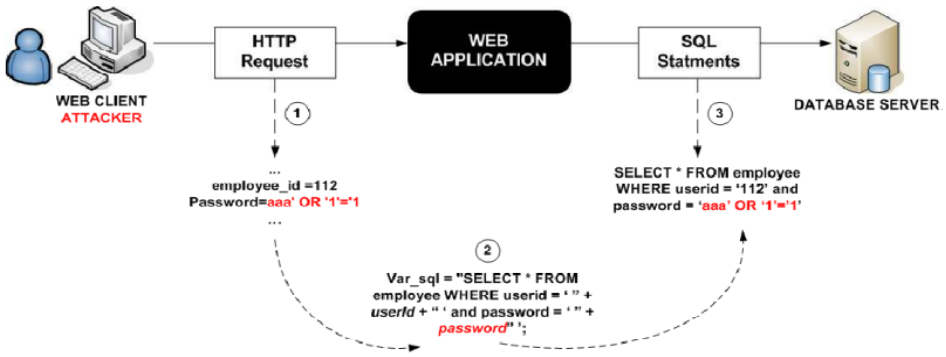


Figure 2.4. SQL Injection Attack Example

2.3.5. Consequence of SQLIA

The results of SQLIA can be disastrous because a successful SQL injection can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administrative operations on the DB (such as shutdown the DBMS), recover the contents on the DBMS file system and execute commands (xp cmdshell) to the operating system. The main consequences of these vulnerabilities are attacks on [16]:

Authorization: Critical data that are stored in a vulnerable SQL database may be altered by a successful SQLIA, as authorization privilege.

Authentication: If there is no any proper control on username and password inside the authentication page, it may be possible to login to a system as a normal user without knowing the right username and/or password.

Confidentiality: Usually databases are consisting of sensitive data such as personal information, credit card numbers and/or social numbers. Therefore, loss of confidentially is a big problem with SQL Injection vulnerability. Actually, theft of sensitive data is one of the most common intentions of attackers.

Integrity: By a successful SQLIA not only an attacker reads sensitive information, but also, it is possible to change or delete this private information.

2.4. SQLIA Detection Techniques

Many studies have been conducted for the detection and prevention against web application vulnerabilities in general and SQL injection vulnerabilities in particular; these studies have discussed the detection and prevention techniques from different point of views and using different techniques. Some of them used static techniques which are used during development time by analysing the web application code to detect the injectable point in the application. There are other techniques that use both dynamic and static techniques by monitoring the user input at runtime.

To address the SQL-Injection problem, the researchers have proposed various methods in SQL injection detection. The occurrence of SQL injection attack (SQLIA) has the top most priority of web-based security problems. There are two broad categories in in SQL injection detection and prevention techniques. First is to detect SQLIA through checking anomalous SQL Query structure using string matching, pattern matching and query processing. Other approach uses data dependencies among data items which are less likely to change for identifying malicious database activities [15]. The different detection techniques are described in [17].

Tautology Checker: Static analysis is used to prevent tautology attack. Arithmetic and logical loops are used to check for possible SQL injection. Abstract model of a source program is used in the framework that takes inputs from user and constructs SQL queries. Particularly, the set of SQL queries which a program could generate as a finite state automaton is approximated. The framework then applies some novel checking algorithms on this automaton. It indicates or verifies lack of security violations in the program of application. This method is not suitable for finding out other SQL injection attacks.

Predictive Errors by the Feature of the Single Character: Probabilistic model to the SQL Injection attack is designed. It tries to minimize the predictive error in the SQL Injection attack detection. Twenty single characters are taken as candidates. A function similar to a sigmoid function is designed to evaluate the features.

Validation using Parse Tree: A parse tree is a widely used as data structure, it is used to represent a statement in the parsed representation. Grammar of the statement's language is needed to Parse a statement. Parsing two statements then using the parsed

tree of both the statements for comparison, it can be determined whether the two queries are equal. This method is easy to integrate with even existing software. This implementation reduces the effort of the programmer, as it captures both the intended query and actual query with least changes required by the coder.

Analysis and Monitoring for Neutralizing SQL Injection Attacks (AMNESIA): AMNESIA is a technique that detects and prevents SQL injection attacks. It combines static analysis and runtime monitoring. This method is very effective and efficient against SQL injection attacks. Initially hotspots are identified in the application code, then SQL query model is built. Each identified hotspot to runtime monitor call is added. Dynamically generated queries are checked against the SQL query model at the runtime. Queries that violate the model are rejected in the process.

SQL Check: The approach is validated with SQLCHECK, it is an implementation for the setting of SQL command injection attacks. SQLCHECK is evaluated on real-world web applications with real-world attack data as input which are systematically compiled [13]. SQLCHECK produced least false positives and false negatives so F1 score is very high. Runtime overhead is very low and it can be applied straightforwardly to web applications written using different programming languages.

Preventing SQL Injection Attacks using SQLrand: Practical protection method to prevent SQL injection attacks is presented. The concept of randomizing instruction-set to SQL is applied, to generate instances of the language which cannot be predicted by the attacker. The queries injected by the attacker to the application will be detected and terminated by the database parser. This technique imposes almost negligible performance overhead in query processing and it can be easily integrated with existing systems that are using databases.

Preventing SQL Injection Attacks with Stored Procedures: This method eliminates the occurrence of SQL injection attacks by combining static application code analysis with runtime validation. Stored procedure parser is designed in the static part, this parser is used to instrument the necessary statements for any SQL statement that uses user inputs. Main purpose of this is to compare the original SQL statement structure with the statement that includes user inputs. The deployment of this technique can be automated and can be only when necessary.

Combinatorial Approach: The approach against SQL injection attacks is based on Signature based approach, that is based on validation of input to solve security vulnerability. Three modules are designed to detect security issues.

1. Monitoring module: Every SQL query is being checked before the execution, to prevent possible attack. This unit will decide whether to send SQL statement to the database for execution.
2. Analysis module: Hirschberg algorithm is used to compare SQL statement with predefined keywords.
3. Auditing module: If there are any suspicious statements found then it stops the transaction and audits the report of attack.

Pattern Matching: Pattern matching is a technique that can be used to identify or detect any anomaly packet from a sequential action. It formulates the different SQL injection string pattern and compares the input string with the patterns that have been formulated for detecting and preventing SQL injection and XSS attacks. For the prevention of SQL injection attack, a filter function is formulated. If the function returns true, an injected string is found. It then provides some warning messages and blocks the user. Otherwise, the permission for access is granted [18].

CHAPTER 3

DESIGN OF THE PROPOSED SYSTEM

A variety of web applications are available for day-to-day activities such as online learning, online banking, online shopping, etc. Along with the prosperity of web applications, inevitably they are becoming the main targets of malicious attackers. SQL injection is one of the threatening attacks in web applications and has the top priority. The main purpose of this proposed system is to detect SQL injection attacks in online learning system web application. The proposed system uses the SQL injection attack dataset that is downloaded from Kaggle and Rabin-Karp pattern matching algorithm in SQL injection detection.

3.1. Overview of the Proposed System

There are two main phases in the proposed system: data collection phase and detection phase. In data collection phase, the SQL injection attack dataset is downloaded from Kaggle and import the downloaded dataset into the SQL injection patterns database. When the attacker enters the SQL injection patterns into the system, the proposed system will compare these patterns with the injection patterns from the database by using Rabin-Karp pattern matching algorithm. The detailed description of Rabin-Karp pattern matching algorithm is presented in Section 3.2. The overview system design of the proposed system and the system flow diagram are described in Figure 3.1 and Figure 3.2.

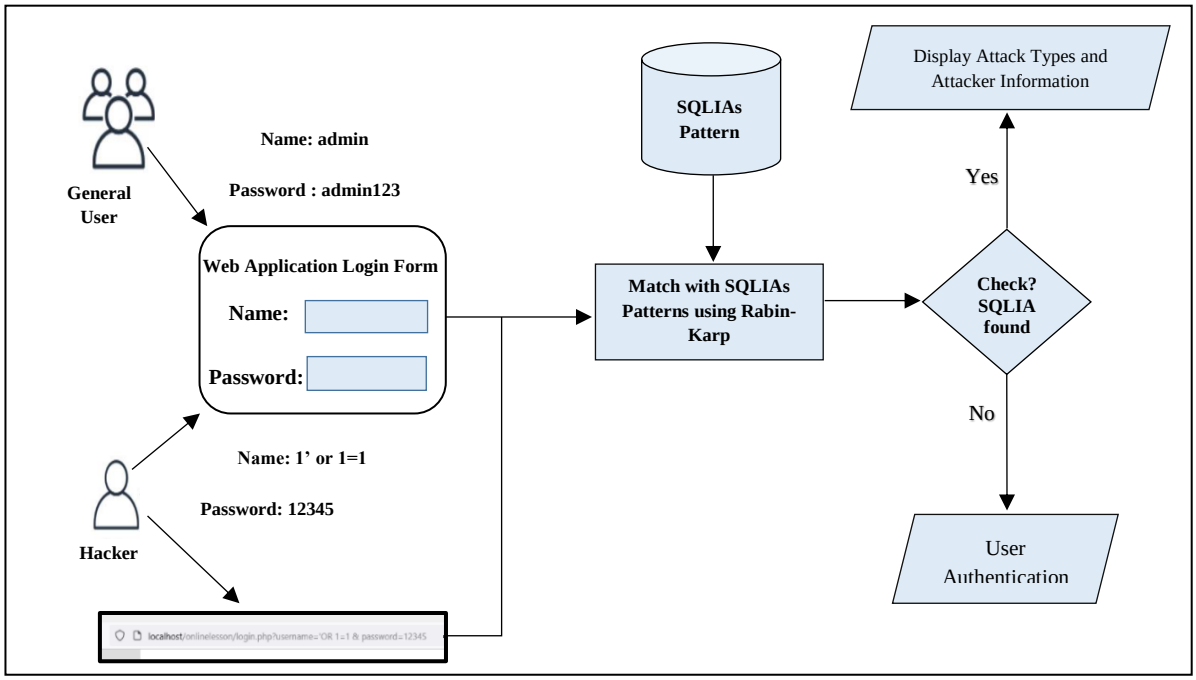


Figure 3.1. Overview System Design of the Proposed System

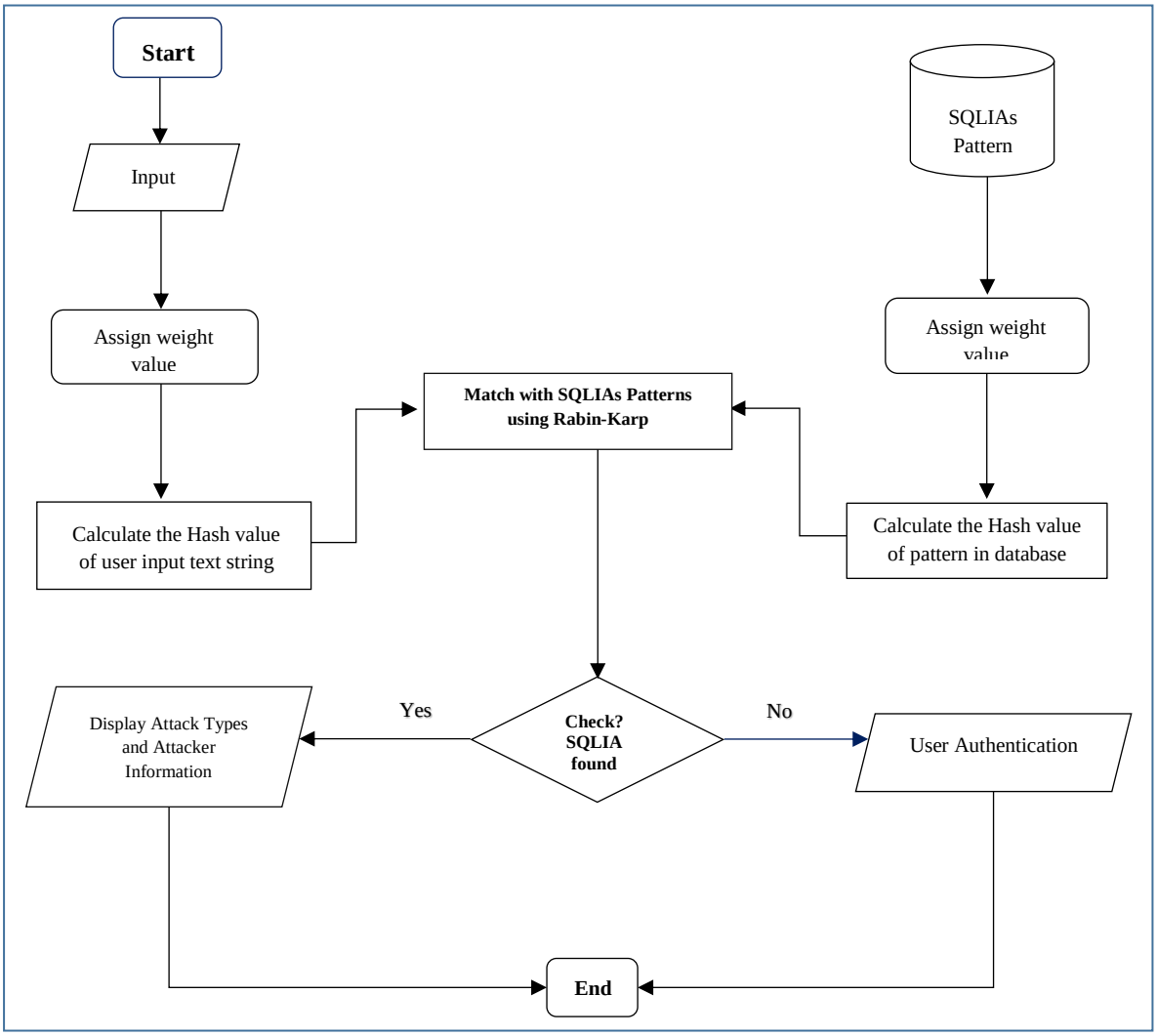


Figure 3.2. System Flow Diagram

The step-by-step procedure of the proposed system is described as follows. As shown in the pattern matching algorithm, the attacker will input the injection query to login into the system or modify the credentials of users in database. The system will detect by comparing the attacker input query (substring) with all of retrieved patterns with Rabin-Karp pattern matching algorithm. The system will check whether the pattern matches or not. If the patterns are equal, SQL injection attack is detected and SQL injection attack type and attacker's information are displayed by the system. If the input pattern is not equal with any SQL injection pattern, the system will allow the user to login as an authenticated user or display the query's results.

Algorithm - Pattern Matching

Input - User Input SQL query.

Output - Pattern found or not found.

Step 1 - User input query is made as substring according to pattern size.

- Assign weight value.

- Calculate the hash value.

Step 2 - Compare user input query (substring) with all of retrieved patterns from SQL injection pattern database until end of string query by using Rabin Karp pattern matching algorithm.

Step 3 - Check whether the pattern matches or not?

- If it is equal to one of retrieved patterns, SQL injection attack is detected and SQLI attack type is displayed by the system.

- If it is not equal with any pattern, then the system will allow login as authenticated user or display query's results.

3.2. Data Collection

In this proposed system, the SQL injection patterns are needed to collect SQL injection patterns database. Therefore, the SQL injection dataset in csv file is downloaded from Kaggle, the world's largest data science community with powerful tools and resources help us achieve data science goals. Then, this dataset file is converted into excel file because the csv file format can be opened in excel format. The total numbers of SQL Injection Queries is 1124 samples in this dataset which will be used to build SQL injection patterns database.

3.3. SQL Injection Detection

The proposed system will detect SQL injection attacks into different attack types: Boolean-based, Union-based, Like-based, Batch Query, Comment-based and Time-based. This system uses Rabin-Karp pattern matching algorithm to SQL injection attack. The procedures and calculation steps of the proposed algorithm will be described in the following sub-sections.

Tautology or Boolean based attack: Tautology is a formula which return True or False values and the malicious SQL query forces the web application to return a different result depending on these returning values. Tautology-based SQL injection attacks are usually bypass user authentication and extract data by inserting a tautology in the WHERE clause of a SQL query. The query transforms the original condition into a tautology, causes all the rows in the database table are open to an unauthorized user. A typical SQL tautology has the form "or <comparison expression>", where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. If an unauthorized user input user id as 'admin' and password as 'anything' or 'x'='x' then the resulting query will be:

*SELECT * FROM login WHERE username = 'admin' AND password = 'anything' or 'x'='x'*

Union based attack: By inserting a UNION query into a vulnerable parameter which returns a dataset. This type of attack is the union of the result of the original first query and the results of the injected query. The SQL UNION operator combines the results of two or more queries and makes a result set which includes fetched rows from the

participating queries in the UNION. The attacker who tries to use this method must have solid knowledge of DB schema. For example, in the SQL query *"SELECT * FROM customers WHERE password = 123 UNION SELECT creditCardNo, pin FROM customers"* the attacker injects the SQL statement "123 UNION SELECT creditCardNo, pin FROM customers" instead of the required password. The query therefore exposes all the credit card numbers with their PINs from the customer's table.

Like-based attack: The attackers uses this type of attack to impersonate a particular user using the SQL keyword LIKE with a wildcard operator (%). For example, an attacker can inject input*: 'anything' OR username LIKE 'S%';* # instead of a username to have SQL query: *SELECT * FROM login WHERE username ='anything OR username LIKE 'S%'; #".* The LIKE operator implements a pattern match comparison, that is, it matches a string value against a pattern string containing wildcard character. The query searches the user's table and returns the records of the users whose username starts with letter S. The wildcard operator (%) means zero or more characters (S…), and it can be used before or after the pattern.

Batch Query-based: This injection type collects all the data from the table and delete the entire table or database. For example, drop table users;

Comment-based: Comments injected into an application through input can be used to compromise a system. As data is parsed, an injected/malformed comment may cause the process to take unexpected actions that result in an attack.

Time-based: In a time-based attack, someone could inject a SQL command to the server with code to force a delay in the execution of the queries or with a heavy query that generates this time delay. Depending on the time response, it is possible to deduct some information and determine if a vulnerability is present to exploit it.

Table 3.1. Different Forms of Injection Code with their Common Patterns

No.	Injection Type	Common Pattern	Example
1.	Boolean-based	‘ OR ‘…’ = > | < <> ! =’…’;	‘ OR ‘1’ = ‘1’; 123’ OR ‘a’ <> ‘b’ ; ‘ OR ‘2 + 3’ < = ‘10’ ;
2.	Union-based	‘ union select … from …;	‘ union select * from users; ‘ union select name from a;
3.	Like-based	‘ OR … LIKE ‘…%’;	‘OR username LIKE ‘S%’;
4.	Batch Query	drop…..;	drop table users;
5.	Comment-based	/*….	/*
6.	Time-based	a' wait for delay……	a' waitfor delay '0:0:10'--

3.3.1. Rabin-Karp Pattern Matching Algorithm

Rabin–Karp algorithm or Karp–Rabin algorithm is a string-searching algorithm created by Richard M. Karp and Michael O. Rabin (1987) that uses hashing to find an exact match of a pattern string in a text. It uses a rolling hash to quickly filter out positions of the text that cannot match the pattern, and then checks for a match at the remaining positions. Generalizations of the same idea can be used to find more than one match of a single pattern, or to find matches for more than one pattern.

To find a single match of a single pattern, the expected time of the algorithm is linear in the combined length of the pattern and text, although its worst-case time complexity is the product of the two lengths. To find multiple matches, the expected

time is linear in the input lengths, plus the combined length of all the matches, which could be greater than linear.

A practical application of the algorithm is detecting plagiarism. Given source material, the algorithm can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Because of the abundance of the sought strings, single-string searching algorithms are impractical.

Unlike Naïve string-matching algorithm, this algorithm does not travel through every character in initial phase. It filters the characters that do not match the hash values and performs comparison. In this proposed system, Rabin-Karp pattern matching algorithm is used to detect SQL injection attacks in online learning system. The step-by-step procedure how the Rabin-Karp algorithm works is described as follows:

1. Takes a sequence of characters.
2. Checks for possibility of the presence of the required string.
3. If the possibility is found then, character matching is performed.

3.3.2. Procedure of Rabin-Karp

The procedure Rabin-Karp and nested procedure calls are described as follows. The length of pattern is defined as m. Firstly, the procecure will calculate the hash value of pattern and text string by calling procedure Calculate-Hash(). Then, the two hash values are compared. If the two hash values, the characters in pattern and text string will be performed the string matching. When the characters are exactly equal, the pattern can be regarded as found. If not, the pattern can be regarded as not found.

```
Procedure Rabin-Karp (Text, Pattern, Prime):
m := Pattern.Length
HashValue := Calculate-Hash(Pattern, Prime, m)        //Compute Hash Value of Pattern
CurrValue := Calculate-Hash(Text, Prime, m)           //Compute Hash Value of Text with size m
for i from 1 to Text.length – m
    if HashValue == CurrValue and String-Match(Text, Pattern, i) is true          //Found
Return i
   end if
CurrValue := Recalculate-Hash(String, i+1, Prime, CurrValue)          //For Next Text Window
   end for
Return -1          //Not Found
```

Figure 3.3. Procedure of Rabin-Karp Algorithm

```
Procedure Calculate-Hash(String, Prime, x):
hash := 0
for m from 1 to x                                  // Here x denotes the length to be considered
    hash := hash + (Value of String[m])        // to find the hash value
  end for
Return hash
```

Figure 3.4. Procedure of Hash Value Calculation

```
Procedure String-Match(Text, Pattern, m):
for i from m to Pattern. Length + m - 1
   if Text[i] is not equal to Pattern[i]
      Return false
   end if
end for
      Return true
```

Figure 3.5. Procedure of String Matching

```
Procedure Recalculate-Hash(String, Curr, Prime, Hash):
   Hash := Hash - Value of String[Curr-1]    //here Curr denotes First Letter of Previous String
   Hash := Hash % Prime
    m := Pattern.length
  New := Curr + m - 1
  Hash := Hash + (Value of String[New])
Return Hash
```

Figure 3.6. Procedure of Hash Value Recalculation

3.3.3. Calculation Steps of Rabin-Karp

The calculation steps of Rabin-Karp pattern matching algorithm that is used in the proposed system are described as follows:

Example - 1

Attacker / User **Input (Text string / SQL Query)** - **‘ OR ‘ 1 ’ = ‘ 1 ’**

‘		O	R		‘	1	’		=		‘	1	’

Assign **Input Text** Weight Value

‘		O	R		‘	1	’		=		‘	1	’
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Calculate the Hash Value of Pattern (‘) in Database

- For **Pattern (‘)** in Database
- n = Length of **pattern** in database
- **v** = Assign text weight values (v = 1)

Pattern in Collection of Database Text Sequence → n = 1 | ‘ | V = 1

- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash Value for **Pattern** in database **(p = ‘)** = $\sum\left(v * d^{(n-1)}\right)$) mod 13 ***Equ -1***

= $(1*10^{1-1})$ mod 13

= $(1*10^{0})$ mod 13

= 1 mod 13

= 1

Calculate the Hash Value of Input Text window size

Attacker / User ***Input Text*** → m = 1 | ‘ | v = 1

- For the first window (**‘)**
- m = Length of **input text** window size
- **v** = Assign text weight values (v = 1)
- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of **Input Text (t = ‘)** = $\sum\left(v * d^{(m-1)}\right)$) mod 13 ***Equ -2***

= $(1*10^{1-1})$ mod 13

= $(1*10^{0})$ mod 13

= 1 mod 13

= 1

Because the hash value of the pattern in database is equal to the hash value of the input text, character-matching is performed. Both the characters match, and SQL Injection Attack are found.

Example - 2

Attacker / User **Input (Text string / SQL Query)** - **123 OR a < > b**

1	2	3		O	R		a	<	>	b

Assign **Input Text** Weight Value →

1	2	3		O	R		a	<	>	b
1	2	3	4	5	6	7	8	9	10	11

Calculate the Hash Value of Pattern in Database (OR)

- For Pattern in Database (OR)
- n = Length of pattern in database
- v = Assign text weight values (v = 5, v = 6)
- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= }= 10
- Hash Value for Pattern in database (p = OR) = $\sum\left(v * d^{(n-1)}\right) mod\ 13$ ***Equ - 3***

Pattern *in Collection of Database Text Sequence* →

n = 2	
O	R
v = 5	v = 6

$= ((5*10^{2-1})+(6*10^{1-1}))$ mod 13

$= ((5*10^{1})+(6*10^{0}))$ mod 13

= (50 + 6) mod 13

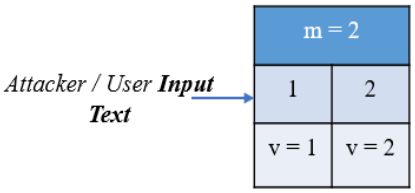
= 56 mod 13

= 4

Because the hash value of the **pattern** in database is not equal to the hash value of the **input text**, go for the Next window.

Calculate the Hash Value of Input Text first window size

- For the first window (1, 2)
- m = Length of input text window size
- v = Assign text weight values (v = 1, v = 2)



- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of Input Text (t = 12) $= \sum\left(v * d^{(m-1)}\right) mod\ 13$ ***Equ - 4***

$= ((1*10^{2-1})+(2*10^{1-1}))$ mod 13

$= ((1*10^{1})+(2*10^{0}))$ mod 13

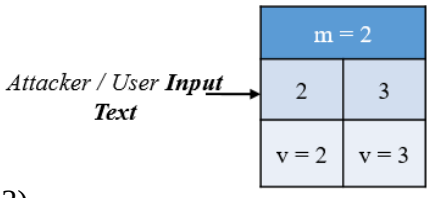
= 12 mod 13

= 12

Because the hash value of the **pattern** in database is not equal to the hash value of the **input text**, go for the Next window.

Calculate the Hash Value of Input Text next window size

- For the Next window (2, 3)
- m = Length of input text window size
- v = Assign text weight values (v = 2, v = 3)



- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of Input Text (t = 2 3) $= \sum\left(v * d^{(m-1)}\right) mod\ 13$ ***Equ - 5***

$= ((2*10^{2\ -\ 1})+(3*10^{1\ -\ 1}))$ mod 13

$= ((2*10^{1})+(3*10^{0}))$ mod 13

= 23 mod 13

= 10

Because the hash value of the **pattern** in database is not equal to the hash value of the **input text**, go for the Next window.

Calculate the Hash Value of Input Text next window size

- For the Next window (3, (space))
- m = Length of input text window size
- **v** = Assign text weight values (v = 3, v = 4)

Attacker / User ***Input Text*** →

m = 2	
3	(space)
v = 3	v = 4

- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of **Input Text** (t = 3 (space)) $= \sum\left(v * d^{(m-1)}\right) mod\ 13$ ***Equ - 6***

$= ((3*10^{2-1})+(4*10^{1-1}))$ mod 13

$= ((3*10^{1})+(4*10^{0}))$ mod

= 34 mod 13

= 8

Because the hash value of the **pattern** in database is not equal to the hash value of the **input text**, go for the Next window.

Calculate the Hash Value of Input Text next window size

- For the Next window ((space) , O)
- m = Length of input text window size
- **v** = Assign text weight values (v = 4, v = 5)

Attacker / User ***Input Text*** →

m = 2	
(space)	O
v = 4	v = 5

- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of **Input Text** (t = (space) O) $= \sum\left(v * d^{(m-1)}\right) mod\ 13$ ***Equ - 7***

$= ((4*10^{2-1}) + (5*10^{1-1})) \bmod 13$

$= ((4*10^{1}) + (5*10^{0})) \bmod 13$

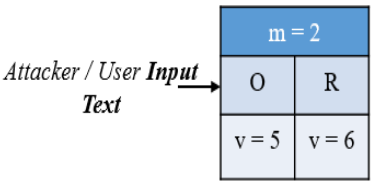
$= 45 \bmod 13$

$= 6$

Because the hash value of the **pattern** in database is not equal to the hash value of the **input text**, go for the Next window.

Calculate the Hash Value of Input Text next window size

- For the Next window (O R)
- m = Length of input text window size
- v = Assign text weight values (v = 5, v = 6)



- d = number of chars in input dataset {‘ , ; , # , = , OR , || , > , >= , < , <= } =10
- Hash value of Input Text (t = O R) $= \sum\left(v * d^{(m-1)}\right) \, mod \, 13$ ***Equ - 8***

$= ((5*10^{2-1})+(6*10^{1-1})) \bmod 13$

$= ((5*10^{1})+(6*10^{0})) \bmod 13$

$= 56 \bmod 13$

$= 4$

Because the hash value of the pattern in database is equal to the hash value of the input text, character-matching is performed. Both the characters match, and SQL Injection Attack are found.

CHAPTER 4

IMPLEMENTATION OF THE PROPOSED SYSTEM

The purpose of this chapter is to present the implementation of the proposed system. The proposed system will use SQL injection dataset from Kaggle to detect the SQL injection attack. (https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset). The downloaded dataset file is CSV (Comma Separated Value) file format which can be opened with as Excel file. Kaggle is the world's largest data science community with powerful tools and resources to help us achieve data science goals. The total number of SQL injection patterns is 1124 inject patterns that are included in this dataset. The proposed system is implemented with PHP programming language and MySQL Database.

4.1. Experimental Setup

In this proposed system, there are two main users: administrator and student. Firstly, the administrator needs to import SQL injection patterns by using the downloaded dataset (.csv) file. Figure 4.1 shows the administrator main login form to login into the system. When the administrator enters the correct credentials (username and password), the administrator's dashboard can be viewed as shown in Figure 4.2.

Figure 4.1. Administrator Login Form

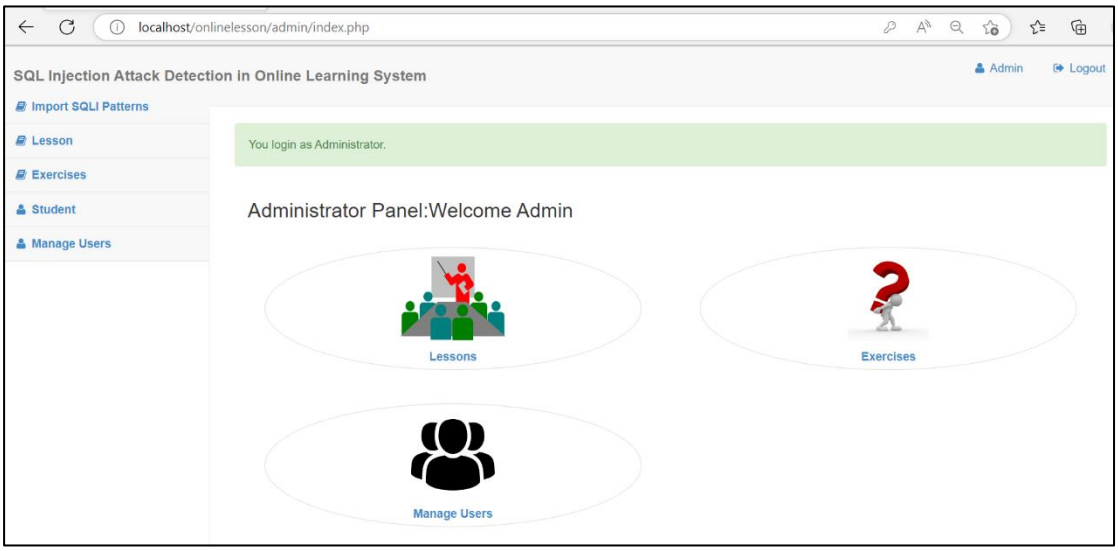


Figure 4.2. Administrator Dashboard

In administrator dashboard, there are five functions 1) Import SQL injection patterns, 2) Add lessons 3) Add exercises 4) View registered students and 5) Manage students or users. When the administrator click “Import SQLI Patterns”, the import form will be displayed to import SQL injection patterns into database as shown in

Figure 4.3. Then, the administrator needs to browse the dataset (csv) file and click import button. If the importing process is successful, the import completed message will be displayed.

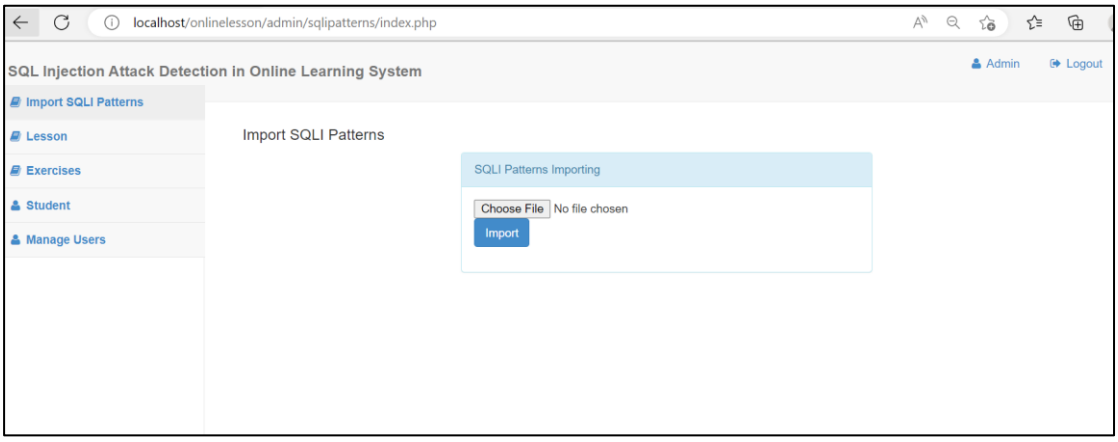


Figure 4.3. Import SQL Injection Patterns

To add the lessons for the students to learn, the administrator needs to click "Lesson". Then, the administrator can see the all of the lessons that have already added into this online learning system as shown in Figure 4.4. The added lessons can be updated and deleted by the administrator. Then, the administrator can add new lessons by clicking +new button. The adding new lesson form will be displayed as shown in Figure 4.5. If the required information is filled and the lesson file is uploaded, the adding new lesson process will be successful.

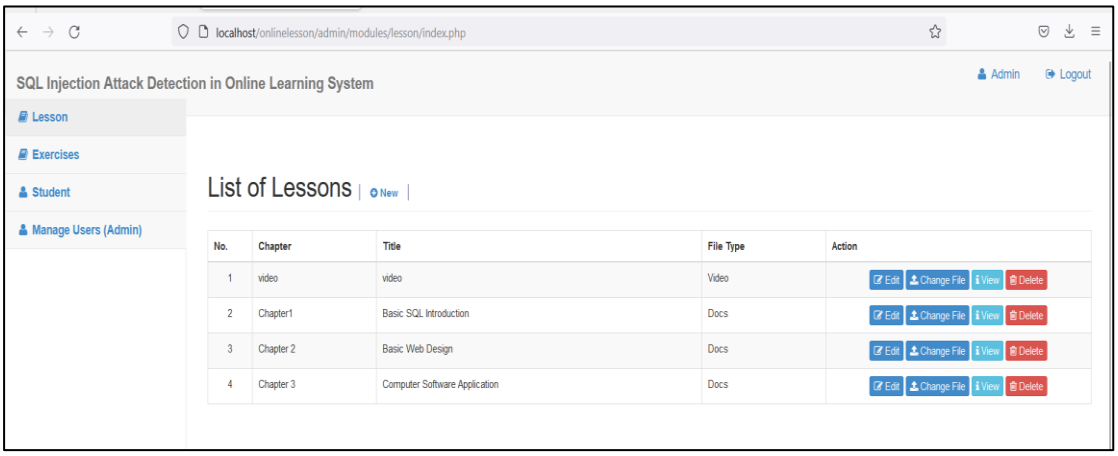


Figure 4.4. List of Lessons Form

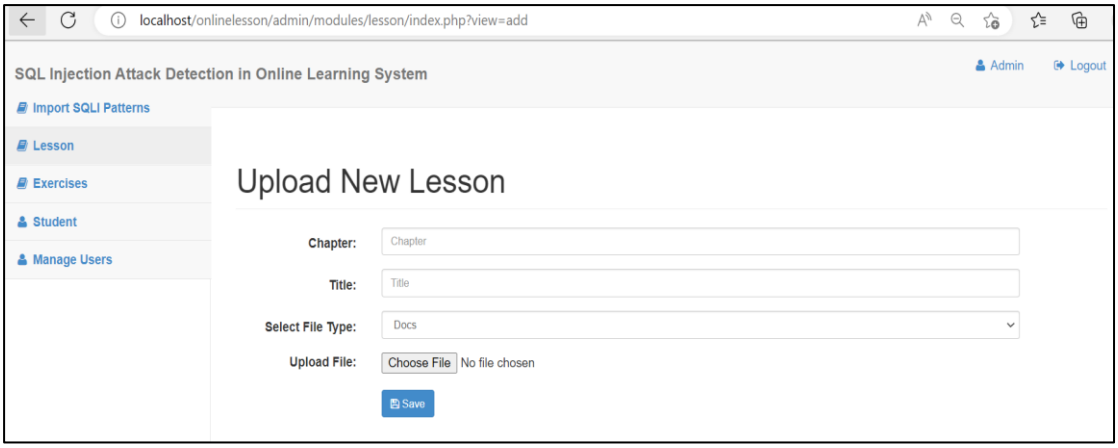


Figure 4.5. Upload New Lesson Form

To add the exercises for the students to learn, the administrator needs to click "Exercises". Then, the administrator can see the all of the exercises that have already added into this online learning system as shown in Figure 4.6. The added exercises can be updated or deleted. Then, the administrator can add new exercises by clicking +new button. The adding new exercise form will be displayed as shown in Figure 4.7. If the required information (question and answers) is filled and the exercise file is uploaded, the adding new exercise process will be successful.

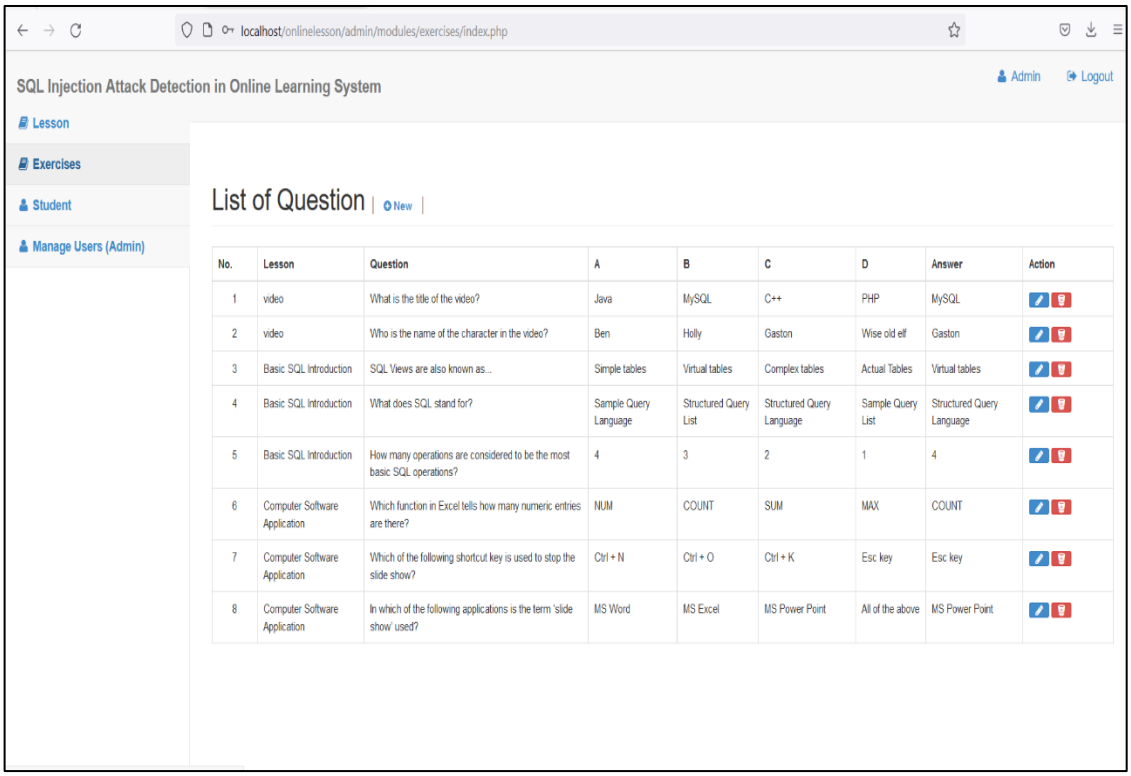


Figure 4.6. List of Exercises Question Form

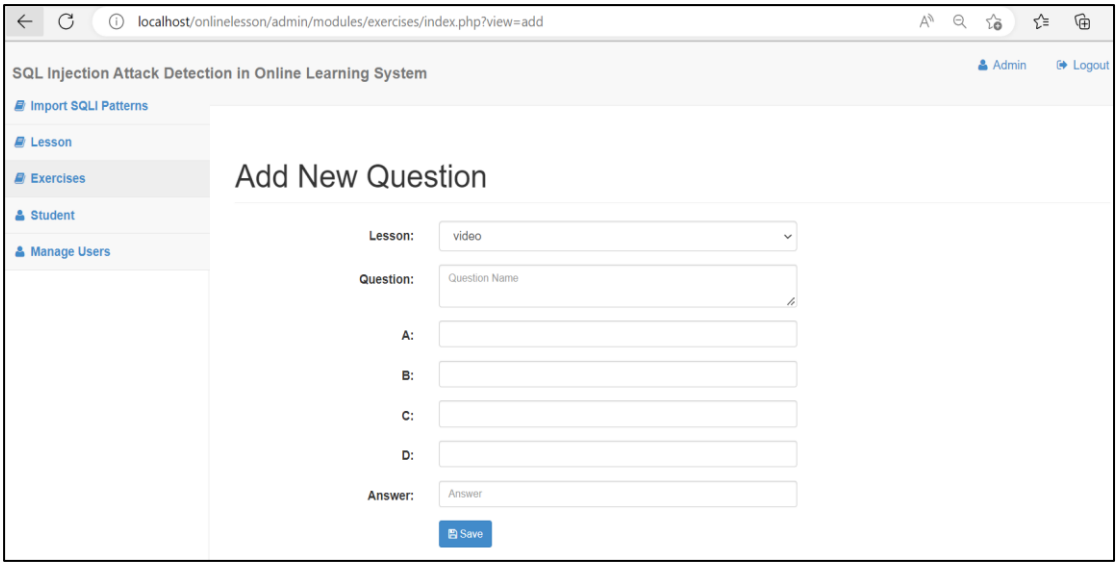


Figure 4.7. Add New Exercise Question Form

The administrator can see the already students who have registered of online learning system as shown in Figure 4.8. He/she can also manage the users by inserting, updating and deleting. To insert the new user, he/she needs to click + New button and fill the required information as shown in Figure 4.9.

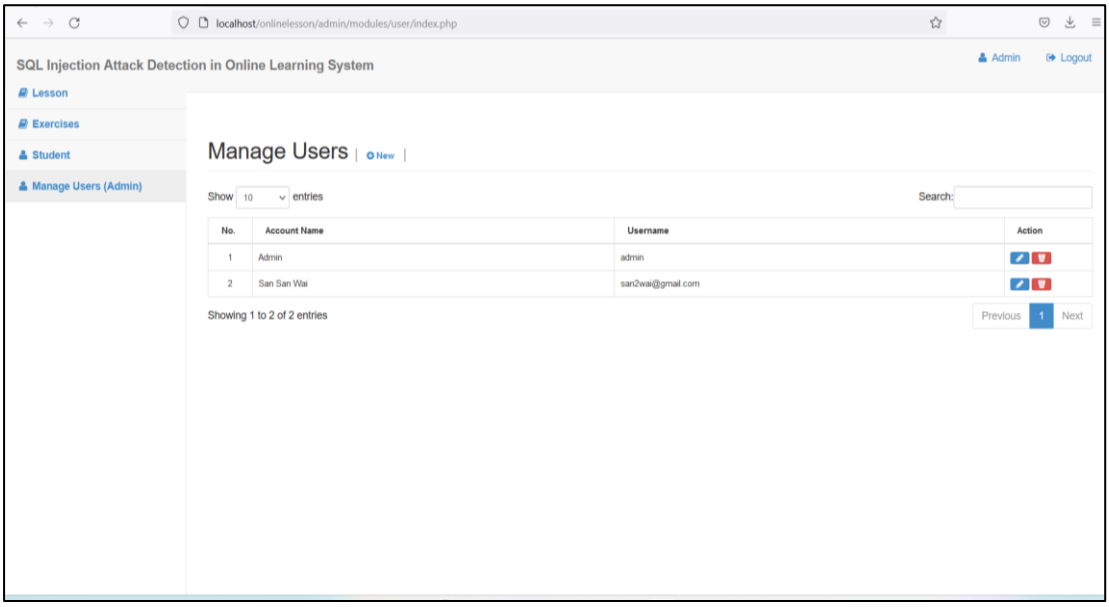


Figure 4.8. Manage Users Form

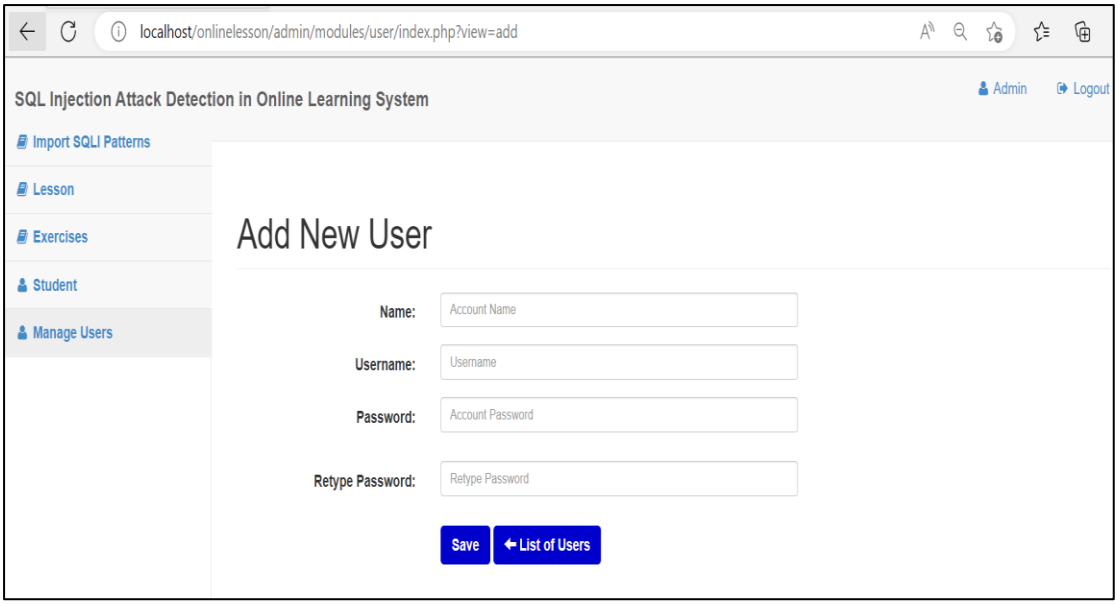


Figure 4.9. Add New User Form

When the attacker attacks to the proposed system by logging in as an administrator with SQL injection attack, this system will detect these attacks and classify the different types of attacks by using pattern matching algorithm. The following figures will display the SQL injection detection forms.

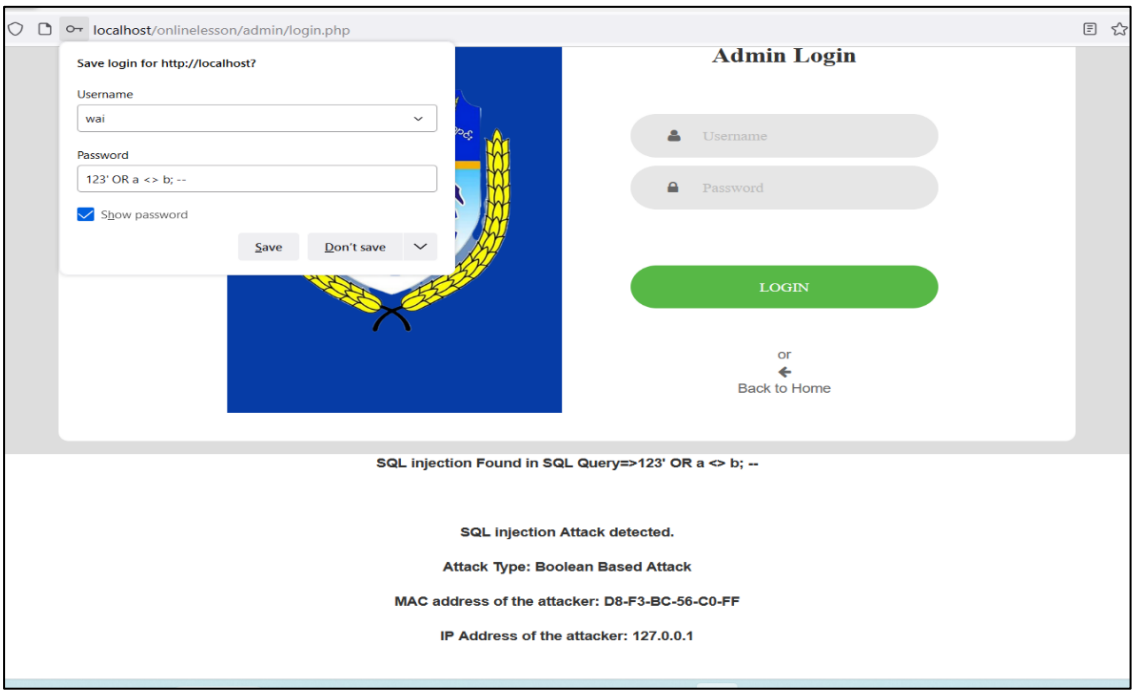


Figure 4.10. SQL Injection Detection

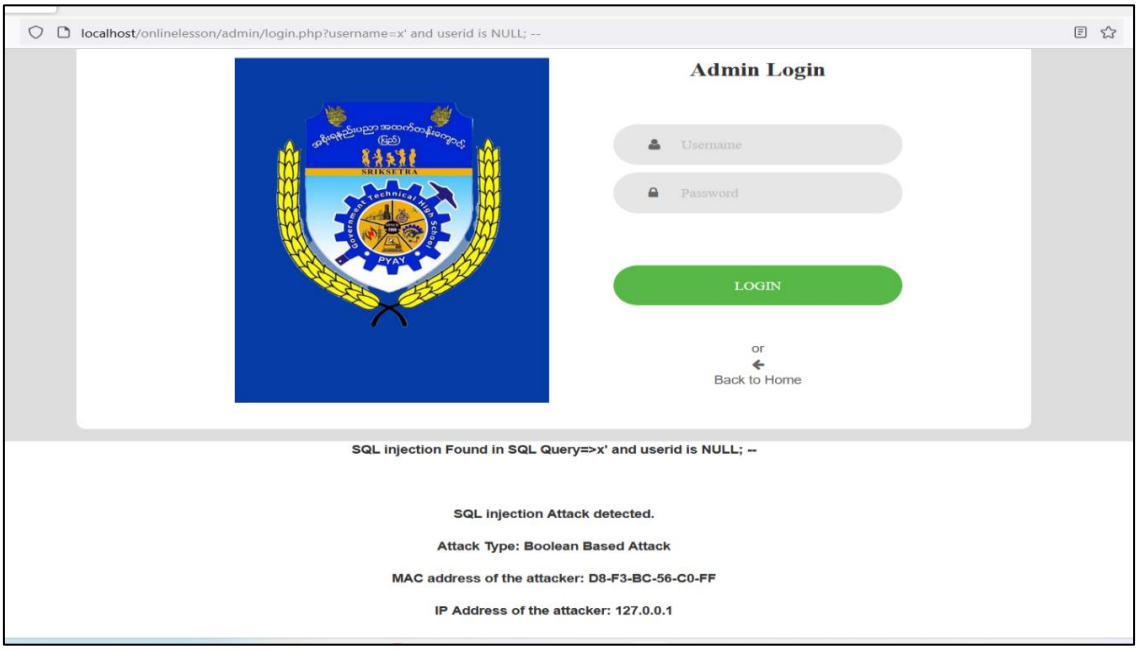


Figure 4.11. URL Filtering for SQL Injection Attack

The proposed system also detect URL filtering SQL injection as shown in Figure 4.11.

From the student side of this online learning system, the students can login into this system by inputting their username and password. If the credentials of the student is valid, the system will allow to search, see and download the lessons and exercises from the dashboard are shown in Figure 4.12.

Figure 4.12. Student Dashboard Form

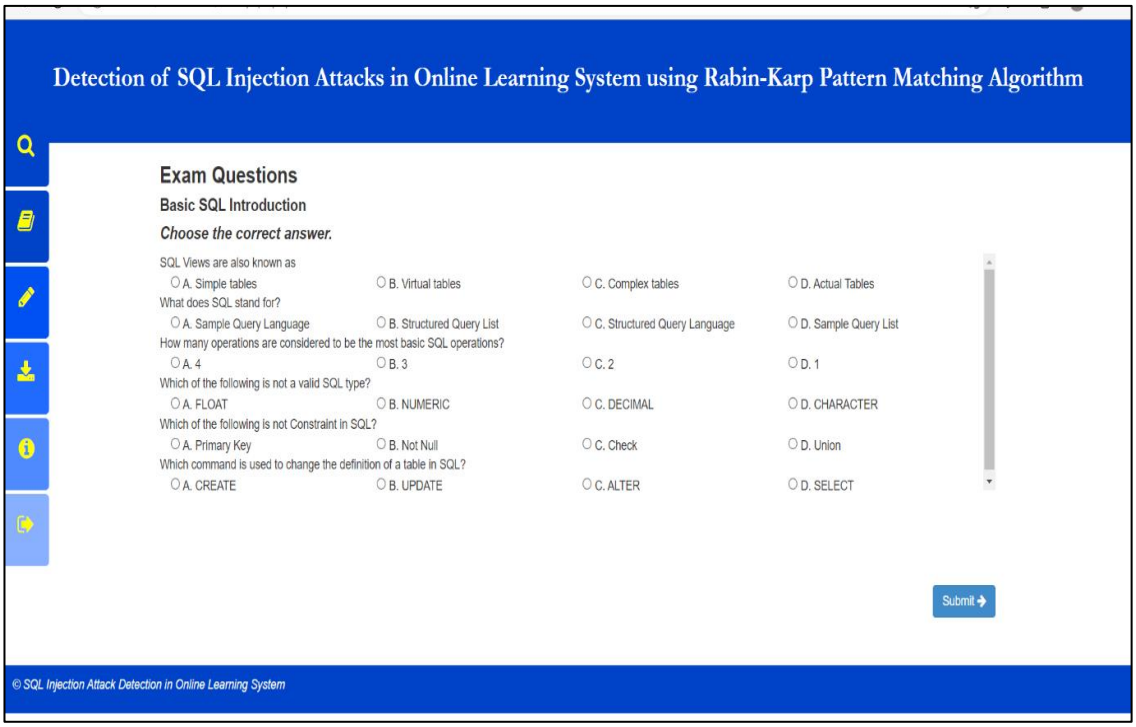


Figure 4.13. Lesson and Exercises Search Form

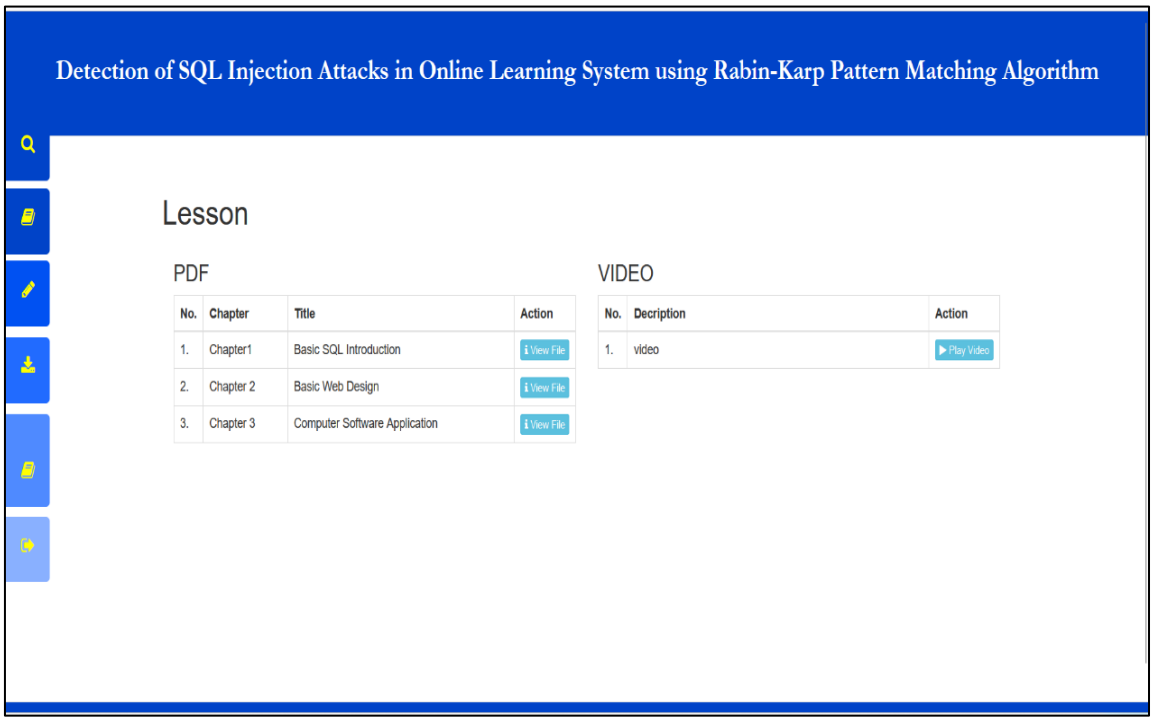


Figure 4.14. View Lessons Form

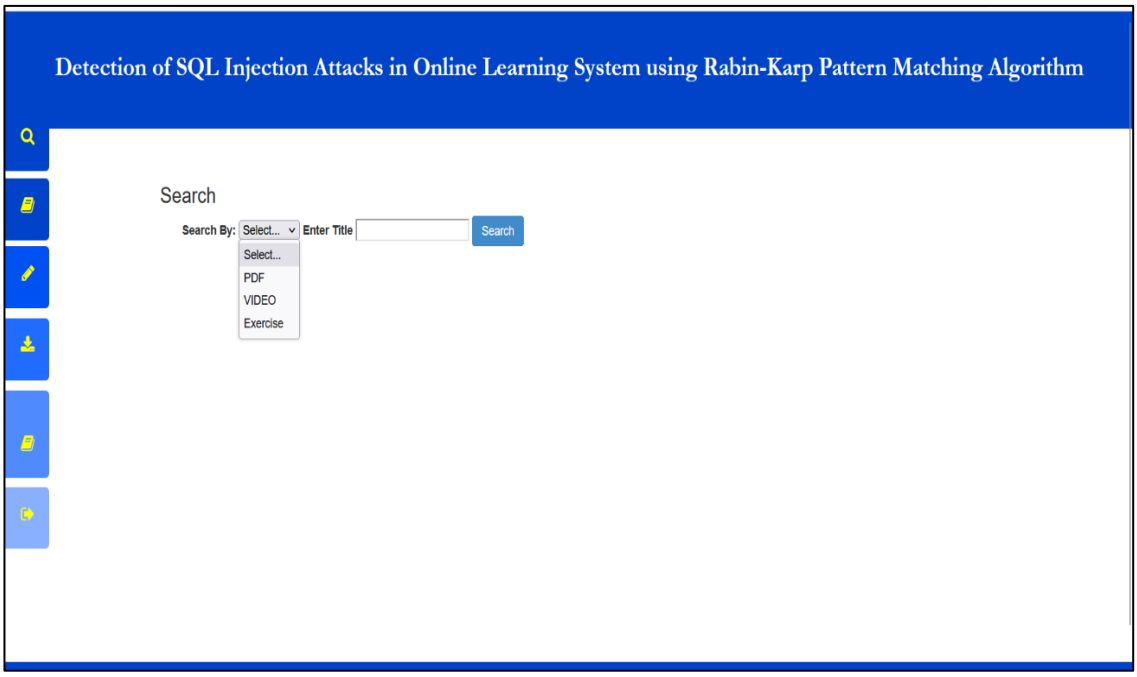


Figure 4.15. View Exercises Form

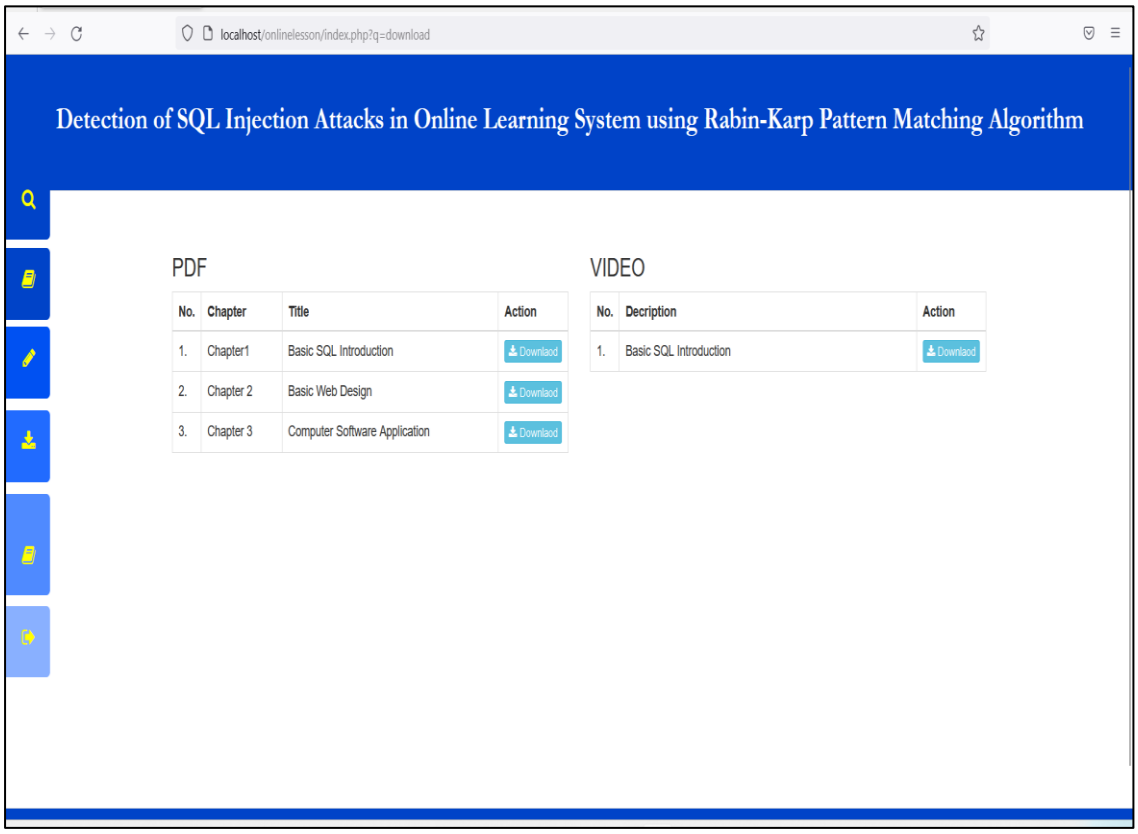


Figure 4.16. Download Lessons and Exercises Form

Table 4.1 The Test Plan

1	anything' or 'x' = 'x
2	a' or 1 = 1; --
3	123 or a = a; --
4	123 or 1 = 1; --
5	union select
6	1 waitfor delay '1:10:10'--
7	select * from users where id = 1 *1 or 1 = 1 -- 1
8	" or "a" = "a
9	or true--
10	or 1 = 1#
11	or 1 = 1/*
12	" or "1" = "1"--
13	' or 2 + 3 <= 10; --
14	or 123 = 123 or '' = '
15	a' waitfor delay '1:5:10'--
16	UNION ALL SELECT
17	union select * from users where login = char ...
18	x' or 1 = 1 or 'x' = 'y
19	select * from users where id = 1 +$ 1 or 1 = 1 -- 1
20	or uname like %a

21	or pwd like 1%
22	or username like char (11) ;
23	or 2 between 1 and 3
24	1 waitfor delay '0:0:10'--
25	hi or 1 = 1 --"
26	a' or 1 = 1; --
27	123 or 1 = 1; --
28	aaa' or 1 = 1; --
29	1 or 1 = 1
30	1' or '1' = '1
31	x' and email is NULL; --
32	x' and userid is NULL; --
33	123' or 'x' = 'x
34	admin' or 'x' = 'x
35	select * from users where id = 1 +1 or 1 = 1 -- 1
36	or 1 = 1--
37	1 and 1 = 1
38	or 0 = 0 --
39	123 or 1 = 1; --
40	a' or 7 = 7; --
41	or username is not NULL or username = '

42	union select * from users where login = char ...
43	x' or 1 = 1 or 'x' = 'y
44	or uname like a%
45	or uname like %s
46	or pwd like 1%
47	123 or a = a; --
48	or username like char (11) ;
49	or 2 between 1 and 3
50	or '1' = '1

4.2. Experimental Results

The proposed system evaluates the performance in SQL injection detection in terms of Accuracy (ACC) with the following formula.

		Actual Class	
		Attacked	*Normal*
Predicted Class	*Attacked*	TP (True Positives)	FP (False Positives)
	Normal	FN (False Negatives)	TN (True Negatives)

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \qquad \textbf{\textit{Equ - 9}}$$

Where TP is the True Positive rate. It presents the number of correctly predicted injection attacks.

FP is the False Positive rate. It presents the number of incorrectly predicted injection attacks.

FN is the False Negative rate. It presents the number of incorrectly predicted normal requests.

TN is the True Negative rate. It presents the number of correctly predicted normal requests.

- Total numbers of SQL injection patterns: 100 inject patterns (Boolean-based SQLi) in this dataset.
  - Total Numbers of SQLi queries = 50
  - SQL Injection query = 27
- Normal attempt to Login or Search = 23
- No of correctly predicted malicious requests, TP = 22
- No of incorrectly predicted malicious requests, FP = 5
- No of incorrectly predicted normal requests, FN = 3
- No of correctly predicted normal requests, TN = 20

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

$$ACC = \frac{22 + 20}{22 + 5 + 3 + 20} = \frac{42}{50} = 0.84 = 84\%$$

Figure 4.17. Experimental Results I

- Total numbers of SQL injection patterns: 100 inject patterns (Boolean-based SQLi, Like-based SQLi, Union-based SQLi, Comment-based SQLi) in this dataset.
- Total Numbers of SQLi queries = 75
  - SQL Injection query = 58
  - Normal attempt to Login or Search = 17
- No of correctly predicted malicious requests, TP = 50
- No of incorrectly predicted malicious requests, FP = 8
- No of incorrectly predicted normal requests, FN = 0
- No of correctly predicted normal requests, TN = 17

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

$$ACC = \frac{50 + 17}{50 + 8 + 0 + 17} = \frac{67}{75} = 0.89 = 89\%$$

Figure 4.18. Experimental Results II

- Total numbers of SQL injection patterns: 1224 inject patterns (Boolean-based SQLi, Like-based SQLi, Union-based SQLi, Comment-based SQLi, Batch Query, and Time-based SQLi) in this dataset.
- Total Numbers of SQLi queries = 100
  - SQL Injection query = 63
  - Normal attempt to Login or Search = 37
- No of correctly predicted malicious requests, TP = 56
- No of incorrectly predicted malicious requests, FP = 7
- No of incorrectly predicted normal requests, FN = 3
- No of correctly predicted normal requests, TN = 34

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

$$ACC = \frac{56 + 34}{56 + 7 + 3 + 34} = \frac{90}{100} = 0.9 = 90\%$$

Figure 4.19. Experimental Results III

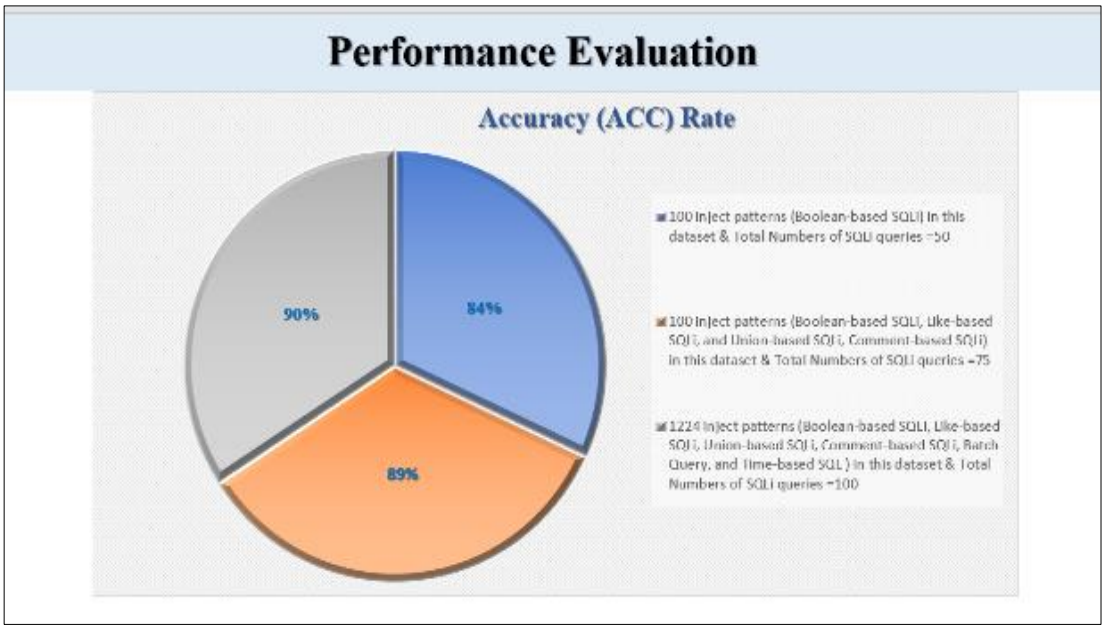


Figure 4.20. Performance Evaluation of the of the Proposed System

In the proposed system, the performance evaluation of Rabin-Karp Pattern Matching algorithm is calculated using the accuracy. In the database, a malicious query which consists of attacks like Boolean-based SQL, Like-based SQL, Union-based SQL, Comment-based SQL, Batch Query, and Time-based SQL are injected into the database. The efficiency of the proposed technique is to identify and detect the database from SQLIA is presented in the above chart and it shows the accuracy (ACC).

Calculating 100 (Boolean-based) SQL injection patterns in the dataset and the total numbers of 50 SQLi queries, an accuracy value is 84 %. Calculating 100 (Boolean-based, Like-based, Union-based, and Comment-based) SQL injection patterns in the dataset and the total numbers of 75 SQLi queries, an accuracy value is 89 %. Calculating 1224 (Boolean-based, Like-based, Union-based, and Comment-based, Batch Query and Time-based) SQL injection patterns in the dataset and the total numbers of 100 SQLi queries, an accuracy value is 90%.

CHAPTER 5

CONCLUSION

SQL injection attack is a common type of attack over the web application. SQL Injection Attacks (SQLIAs) occurs when an attacker is able to insert a series of malicious SQL statements into a query through manipulating user input data and URL for execution the back-end database. In this proposed system, Rabin-Karp Pattern Matching Algorithm is used to detect SQL injection attacks on the Online Learning System and defining SQLI attack types (such as Boolean-based, Like-based and Union-based, etc.). The performance of the proposed system will be measured by correctly classifying user input queries entering the online learning system as normal queries or malicious queries.

5.1. **Limitation and Further Extension**

In this thesis, the proposed system will not detect unknown SQL injection attacks that are not in the dataset. This system detect only the different types of attacks: Boolean-based, Union-based, Like-based, Batch Query, Comment-based and Time-based SQL injection attacks. As further extension, the system needs to develop to detect other types of SQL injection attacks and build more sufficient SQL injection patterns dataset.

AUTHOR'S PUBLICATION

[1] San San Wai, Yi Mon Thet, University of Computer Studies, Yangon, Myanmar, "*Detection of SQL Injection Attacks in Online Learnig System using Rabin-Karp Pattern Matching Algorithm*" to be published in the Proceedings Journal Organizing Committee PSC 2022, Yangon, Myanmar, 2022.

REFERENCES

[1] N. Karthikeyan, R. Vivekanandan, M. Sakthivel, N. Dinesh, "A Novel Technique to Detect and Prevent SQL Injection Attacks using Bitap String Matching Algorithm", Volume 27, Issue 4, 2021

[2] Abikoye et al. "A Novel Technique to Prevent SQL Injection and Cross-Site Scription Attacks using Knuth-Morris-Pratt String Match Algorithm", EURASIP Journal on Information Security (2020) 2020:14

[3] A. John, A. Agarwal, M. Bhardwaj, "An Adaptive Algorithm to Prevent SQL Injection", American Journal of Networks and Communications 2015; 4(3-1)

[4] J. O. Atoum and A. J. Qaralleh, "A Hybrid Technique for SQL Injection Attacks Detection and Prevention", International Journal of Database Management Systems (IJDMS) Vol.6, No.1, February 2014

[5] S. Kharche, J. Patil, K. Gohad, B. Ambetkar, "Implementation of Pattern Matching Algorithm to Prevent SQL Injection Attack", IJSRST, Vol. 4, Issue 5, 2018

[6] M. Hirani, A. Falor, H. Vedant, "A Deep Learning Approach for Detection of SQL Injection Attacks using Convolutional Neural Networks", 2020

[7] I. Jemal, O. Cheikhrouhou, H. Hamam, A. Mahfoudhi, "SQL Injection Attack Detection and Prevention Techniques Using Machine Learning", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 15, Number 6 (2020) pp. 569-580 ©Research India Publications. http://www.ripublication.com

[8] G. Deepa, P. S. Thilagam, F. A. Khan, A. Praseed, A. R. Pais, and N. Palsetia, "Black-box detection of xquery injection and parameter tampering vulnerabilities in web applications," International Journal of Information Security, vol. 17, no. 1, pp. 105–120, 2018.

[9] Y. Fang, J. Peng, L. Liu, and C. Huang, "Wovsqli:Detection of sql injection behaviors using word vector and lstm," in Proceedings of the 2nd International Conference on Cryptography, Security and Privacy. ACM, 2018, pp. 170–174.

[10] Q. Li, F. Wang, J. Wang, and W. Li, “Lstm-based sql injection detection method for intelligent transportation system,” IEEE Transactions on Vehicular Technology, 2019.

[11] OWASP, “Owasp top ten project,” https://www.owasp.org/index.php/Category: OWASP Top Ten Project, 2019, accessed on April 2019

[12] M. S. Aliero, I. Ghani, S. Zainudden, M. M. Khan, and M. Bello, “Review on sql injection protection methods and tools,” Jurnal Teknologi, vol. 77, no. 13, 2015.

[13] W. G. Halfond, J. Viegas, and A. Orso, “A classification of sql-injection attacks and countermeasures,” in Proceedings of the IEEE International Symposium on Secure Software Engineering, vol. 1. IEEE, 2006, pp. 13–15.

[14] SQL Injection Detection and Exploitation Framework for Penetration Testing, Thesis Book

[15] N. Patela, N. Shekokarb, “Implementation of pattern matching algorithm to defend SQLIA”, International Conference on Advanced Computing Technologies and Applications (ICACTA, 2015)

[16] A. Tajpour, S. Ibrahim, M. Masrom, “SQL Injection Detection and Prevention Techniques”, International Journal of Advancements in Computing Technology · August 2011

[17] A. K. Hegde1, P. N. Jayanthi, “A Survey on SQL Injection Attacks and Prevention Methods”, International Research Journal of Engineering and Technology (IRJET) Volume: 07 Issue: 06, June 2020

[18] A. Joy, R. M. Daniel, “A Survey on SQL Injection Attack: Detection and Prevention”, IJRTI , 2022, Volume 7, Issue 5, ISSN: 2456-3315

[19] https://www.sonarqube.org/features/security/owasp/?gads_campaign=ROW-2-Generic&gads_ad_group=OWASP&gads_keyword=owasp%20top%2010&gclid=EAIaIQobChMIqJjEwN2i-wIVtZJmAh0qQg-hEAAYASABEgLBd_D_BwE

[20] https://www.imperva.com/learn/application-security/sql-injection-sqli/