SQL INJECTION DETECTION USING PATTERN MATCHING ALGORITHM FOR LIBRARY SYSTEM

MAR MAR THAN

M.I.Sc. **SEPTEMBER 2022**

SQL INJECTION DETECTION USING PATTERN MATCHING ALGORITHM FOR LIBRARY SYSTEM

BY

MAR MAR THAN

D.C.Sc.

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Information Science

(M.I.Sc.)

University of Computer Studies, Yangon

SEPTEMBER 2022

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 Mar Mar Than

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to everyone who assisted me in various ways when I was completing my research and writing this thesis. There are many things required, including my hard work and the assistance of many others, to finish this thesis.

First of all, I would like to express very special thanks to **Prof. Dr. Mie Mie Khin**, Rector, the University of Computer Studies, Yangon, for allowing me to develop this thesis and giving me general guidance during the period of my study.

My heartfelt thanks and respect go to **Prof. Dr. Thandar Win**, Principal and Pro-rector of the University of Computer Studies (Myeik), for her invaluable and administrative support.

I would like to express my deep gratitude to my supervisor, **Prof. Dr. Tin Thein Thwel**, Head of Faculty of Information Science, University of Computer Studies, Yangon, for providing me with incredibly valuable guidance and advice throughout my thesis. In addition, I would like to express my sincere gratitude to my supervisor for giving much of her time. I admire her courage, her positive outlook, her ability to offer advice. I am very lucky to be working under her supervision.

I would like to express my respectful gratitude to all my teachers for their encouragement and suggestion. To the reading committee teachers, especially **Daw Aye Aye Khine**, **Associate Professor** and Head of English Department, University of Computer Studies, Yangon, I would like to thank her for valuable supports and editing my thesis from the language point of view.

I also thank **Daw Nwe Zin Oo**, **Lecturer** and Head of my department, the Information Technology and Supporting Maintenance Department, University of Computer Studies (Myeik), for her co-operation and encouragement.

Moreover, I would like to extend my thanks to all my teachers who taught me throughout the master's degree course and my friends for their cooperation.

Last but not least, I am very much indebted to my parents, all of my colleagues, and friends for always believing in me, for their endless love and support. They are always supportive of me during my study period.

ABSTRACT

Security concerned vulnerabilities are frequently detected and exploited in modern library system. Intruders obtain unrestricted access to the information stored in the library system by exploiting security vulnerabilities. It becomes a greater challenge for a library due to network acceptance and security vulnerability. Traditional library system is unable to detect malicious users from SQL injection attacks. Pattern matching algorithm has grown in prominence alongside the emergence of security awareness. In this work, an effective library system is proposed to detect SQL injection attacks by using static pattern matching algorithm. The proposed system makes use of an effective pattern matching algorithm and validation with the static pattern lists whether the authenticated user or not for the library system. It can update a new anomaly pattern to the existing static pattern list whether any form of new anomaly occurs. Moreover, the matching percentage of the attacks can be calculated after detection. The matching algorithm is modified to check how many percentages based on the defined threshold and it is applied to evaluate the accuracy of the system when SQL Infections are attacked. The evaluation is performed using the Bayes Classifier. The proposed system provides the output result with the possible percentage of SQL injection attacks entering the library system.

TABLE OF CONTENTS

		Page
ACKNOWLEDGEMENTS		i
ABSTRACT		ii
TABLE OF CONTENTS		iii
LIST OF FIGURES		v
LIST OF TABLES		vii
LIST OF EQUATION		viii
CHAPTER 1	**INTRODUCTION**	
	1.1 Motivation of the Thesis	1
	1.2 Objectives of the System	2
	1.3 Organization of the Thesis	2
CHAPTER 2	**RELATED WORK**	
	2.1 SQL Injection Detection Using Machine Learning	4
	2.2 SQL Injection Detection Using Pattern Matching	5
	2.3 Chapter Summary	6
CHAPTER 3	**SQL INJECTION AND ATTACK TYPES**	
	3.1 Impacts of SQL Injection	7
	3.2 Types of SQL Injection Attack	8
CHAPTER 4	**PATTERN MATCHING BASED LIBRARY SYSTEM ARCHITECTURE**	
	4.1 Proposed Library System	11
	4.2 Naive Bayes Classifier	18
	4.3 Admin Login	18
	4.4 User Login	20
	4.5 Experiment and Result Discussion	32
	4.5.1 Alternate Encoding	32
	4.5.2 Inference Attack	34
	4.5.3 Logically Incorrect Attack	36

4.5.4 Piggy-Backed Attack 38
4.5.5 Stored Procedure Attack 39
4.5.6 Tautology Attack 41
4.5.7 Union Attack 42
4.5.8 Attack Categories Evaluation 44

CHAPTER 5 CONCLUSION

5.1 Advantages of the System 51
5.2 Limitation of the System 52
5.3 Further Extension 52

LIST OF PUBLICATIONS 53

REFERENCES 54

LIST OF FIGURES

FIGURE		PAGES
Figure 2.1	The Parsing Approach	3
Figure 2.2	The Machine Learning Approach	4
Figure 4.1	ER Diagram of the Proposed System	12
Figure 4.2	Flow Chart of Proposed Library System	17
Figure 4.3	Admin Login Page of the System	19
Figure 4.4	Correct Admin Login Page of the System	19
Figure 4.5	Incorrect Admin Login Page of the System	20
Figure 4.6	Dashboard Page of the System	20
Figure 4.7	Admin Profile Page of the System	21
Figure 4.8	Admin Password Change View of the System	21
Figure 4.9	User View from Admin Side of the System	22
Figure 4.10	Upload Book View from Admin Side of the System	22
Figure 4.11	Upload New Book to the System	23
Figure 4.12	Successful Upload New Book to the System	23
Figure 4.13	Update Number of Books in the System	24
Figure 4.14	Update Book Information in the System	24
Figure 4.15	Update Book Information by Admin	25
Figure 4.16	Alert Attack Information to Admin	25
Figure 4.17	Logout Page (Admin View)	25
Figure 4.18	Login Page of the System	26
Figure 4.19	Registration Page of the System	26
Figure 4.20	Successful Registration Page of the System	27
Figure 4.21	Find Book Page of the System	27
Figure 4.22	Result Page of the System	28
Figure 4.23	Finding Book Result of the System	29
Figure 4.24	Download Page of the System	29
Figure 4.25	User Profile Page	30
Figure 4.26	User Login Page	30
Figure 4.27	Incorrect User Login Page	31
Figure 4.28	User Validation Page	31
Figure 4.29	User Validation Alert Page	32

Figure 4.30	Alternate Encoding Attack from User Input	33
Figure 4.31	Alternate Encoding Attack from Search Input Box	33
Figure 4.32	Alternate Encoding Attack Detection	34
Figure 4.33	Inference Attack from User Input	35
Figure 4.34	Inference Attack from Search Input Box	35
Figure 4.35	Inference Attack Detection	36
Figure 4.36	Logical Incorrect Attack from User Input	37
Figure 4.37	Logical Incorrect Attack from Search Input Box	37
Figure 4.38	Logical Incorrect Attack Detection	38
Figure 4.39	Piggy-Backed Attack from User Input	38
Figure 4.40	Piggy-Backed Attack from Search Input Box	39
Figure 4.41	Piggy-Backed Attack Detection	39
Figure 4.42	Stored Procedure Attack from User Input	40
Figure 4.43	Stored Procedure Attack from Search Input Box	40
Figure 4.44	Stored Procedure Attack Detection	41
Figure 4.45	Tautology Attack from User Input	41
Figure 4.46	Tautology Attack from Search Input Box	42
Figure 4.47	Tautology Attack Detection	42
Figure 4.48	Union Attack from User Input	43
Figure 4.49	Union Attack from Search Input Box	43
Figure 4.50	Union Attack Detection	44

LIST OF TABLES

TABLES		PAGES
Table 4.1	Data Dictionary of the Proposed Library System	13
Table 4.2	Pattern Matching Procedure	14
Table 4.3	System Design Process	14
Table 4.4	Attack Types Evaluation	44
Table 4.5	Performance Evaluation	49

LIST OF EQUATION

EQUATION		PAGES
Eq 4.1	Naive Bayes Classifier	18

CHAPTER 1

INTRODUCTION

With growth of Internet and web-based system, the human dependency on websites and web applications has increased significantly in present days. Browsers and general web concepts are more familiar to most people to use than any other abstract computing interface. With the widespread use of web application, web-based e-library system can easily provide information on literature and academic areas. Users performing sensitive transactions online have paved a way for the attackers to spoof and tamper the transaction data. SQL Injection is a type of web application security vulnerability in which an attacker is able to submit a SQL command in order to extract or update information in the library database that they are not authorized to access. One of the most frequent web-based application vulnerabilities, SQL injection focuses on the form of incoming SQL queries and allows users to access restricted data, get beyond authentication, and execute unwanted data manipulation language. SQL Injection Attacks can be identified and avoided using a variety of methods, including encryption, extensible markup language (XML), pattern matching, parsing, and machine learning. These techniques can address login, URL, and search vulnerabilities processes by handling input type checking, pattern matching, and input encoding assaults.

In this proposed system, the user generated SQL Queries are checked whether they are SQL injected or not by applying static pattern matching algorithm. If any form of new anomaly occurs, then a new anomaly pattern will be updated to the existing static pattern list. On the basis of the corresponding scenario, this work serves as a pattern matching based e-library system for experimenting with different SQL injection attacks.

1.1 Motivation of the Thesis

The Databases of the library system often contain confidential and personal information. These databases and user personal information become target to the attacks. Injection attack is a method that can inject any kind of malicious string or anomaly string on the original string. There are numerous techniques to carry out SQL injection attacks, including data modification, query manipulation, data extraction, etc. Attackers can get unauthorized access to the application and steal sensitive

identification information by executing a modified SQL query. Pattern matching is a technique that can be used to identify or detect any anomaly packet from a sequential action. SQL Injection is a type of an injection attack that makes it possible to execute malicious SQL statements to control a database server. An effective library system against SQL injection attack is needed.

1.2 Objectives of the Thesis

The main objectives of the thesis are:

- To identify or detect malicious SQL queries against a database of library system which include the patterns from known SQL injection attacks.
- To update the existing static pattern database when any form of new anomaly query occurs by adding after detection.
- To describe the percentage matching after checking the query with static pattern database.

1.3 Organization of the Thesis

The thesis is organized as five chapters. They are as follows.

Chapter 1 outlines the study areas and defines the aims of the study. The thesis issues and motivations are presented.

Chapter 2 presents the related work of the study and recent literatures of the thesis.

Chapter 3 describes the overview of the SQL injection and different attack types.

Chapter 4 provides the proposed system design and process, detailed procedures of the algorithms and highlights the experiment results with the different types of SQL injection attacks.

Chapter 5 concludes the whole thesis work and the effectiveness of the thesis by the result discussion, the scope and limitations of the research and finally points out with future research directions.

CHAPTER 2

RELATED WORK

With the wide deployment of web application, a variety of emerging applications have been deployed at web-based system. To guarantee the safe and efficient operations of the web-based system, especially the extensive e-library system, it is important and challenging to detect SQL injection attacks, which can be expressed as a number of specific SQL query that may cause attacks. The target of a SQL injection attack is a web application that uses database services and is interactive. Such programs accept user input fields and then utilize that information in SQL queries, which are often used to query databases. In SQL injection, the malicious user delivers user input that causes a database request that is different from what the normal user intended. In other words, when user input is interpreted as a component of a larger SQL statement, the resulting SQL statement differs from what was initially intended. There are two main SQL injection detection approaches: parsing approach and machine learning approach.

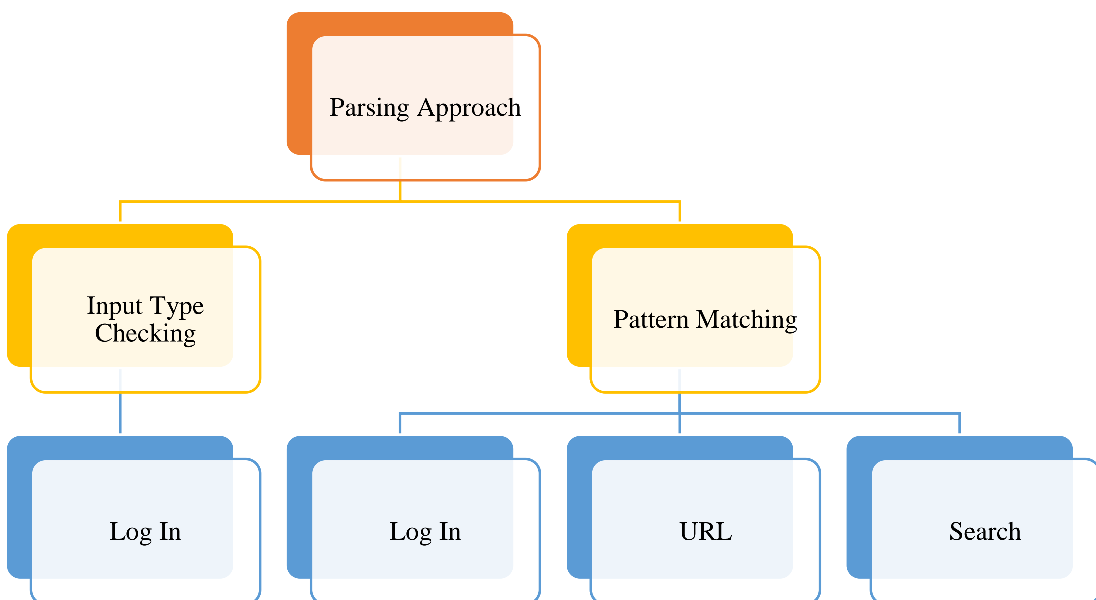


Figure 2.1 The Parsing Approach

Figure 2.1 shows the tree structure of the parsing approach. A data structure for the parsed representation of a statement is called a parse tree. The sentence construction

of a statement's language is necessary for parsing. The system can detect whether two queries are identical by parsing two statements and comparing their parse trees.

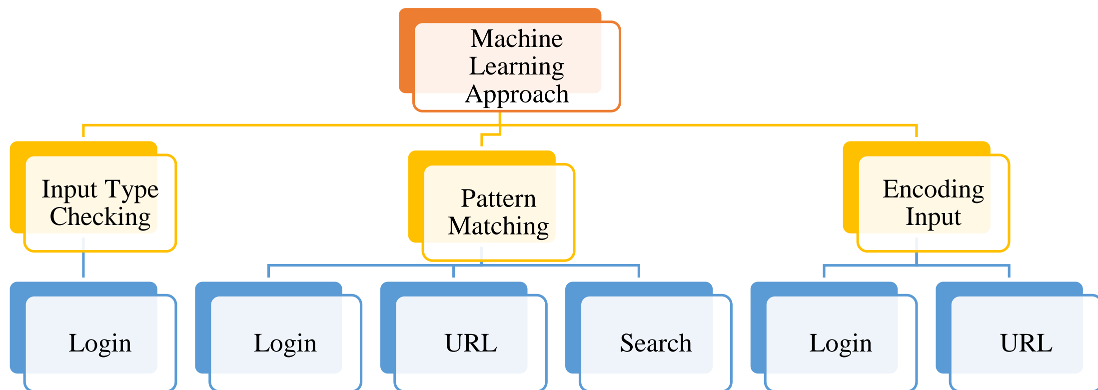


Figure 2.2 The Machine Learning Approach

Figure 2.2 shows the tree structure of the machine learning approach. By comparing the website access log file with the knowledge-based of malicious attributes, a machine learning technique is utilized to identify a SQL injection attack. Although some approaches have achieved remarkable progress, they are with limited applications since these approaches are dependent on attack types, e.g., signatures describing anomalies. Moreover, they might fail to detect the anomalies pattern that may have to inject malicious SQL statements into input fields at the system. To overcome these limitations and adaptively detect anomalies from SQL injection attacks, an effective library system is proposed to detect SQL injection attacks by using static pattern matching algorithm. This chapter reviews the current literature upon which the theoretical basis of this thesis is built.

2.1 SQL Injection Detection Using Machine Learning

For many years, SQL injection has been a problem, and numerous tools and methods have been created to address vulnerability. Some of the study perform SQL injection detection using machine learning algorithm. The study in [6] constructed and assessed the machine learning's Naïve Bayes classifier for detecting SQLIAs. The developed application accepts the training dataset from text files during the learning

phase and submits each piece of information to the learning algorithm of the classifier. The feature vectors are created from the input data by tokenizing and separating it into blanks, then the system learns those feature vectors using machine learning. In this study, the role of the user, which is used for categorization using a Role Based Access Control method, is also included in the feature vector. The dataset from text files is adopted and each piece of information is trained with the learning algorithm during the classification process. Classification is done using the generated feature vectors. They showed that the proposed classifier detected the malicious query and achieved the classification results with 93.3% accuracy.

The study in [1] provided a novel method to identify injection attacks by representing SQL queries as tokenized graphs and training a support vector machine (SVM) with the centrality measures of the node. They analyzed several token graph construction techniques and offered in different system designs that include both single and multiple SVMs. The system can defend numerous web applications in a shared hosting environment because it is made to operate at the database firewall layer. The results of the experiments show that this technique can identify fraudulent SQL queries with no performance impact.

2.2 SQL Injection Detection Using Pattern Matching

M. A. Prabakar et al. [8] proposed a detection and prevention technique for preventing SQL Injection Attack (SQLIA) using Aho–Corasick pattern matching algorithm. They introduced the efficient approach which consists of two modules: static and dynamic modules. In the static module, the incoming user query was examined with the existing static pattern lists. In the dynamic module, when a new abnormality of any type came to fruition, an alarm would be activated and a new anomaly pattern would indeed be formed. The Static Pattern List would be updated with the new abnormality pattern. The experimental results showed that the proposed algorithm worked well against the SQL Injection Attack based on some sample of standard attack patterns.

N. Patel and N. Shekokar [14] developed a detection and prevention technique for SQL Injection Attack using modified Aho–Corasick pattern matching algorithm. The system checked the user generated SQL queries by applying static pattern matching

algorithm whether these are SQL injected or not by using SQLMAP tool and AIIDA-SQL techniques.

The proper research had been done to pinpoint the flaws, exploits, and defenses against SQL injection attacks made use of these imperfections. The researchers presented a neural network-based solution for high accuracy SQL injection detection in [10]. The system acquired authentic user URL access log data from the Internet Service Provider (ISP). The statistical research was conducted on normal data and SQL injection data. Based on statistical findings, their experimental results showed that accuracy was over 99 percent.

A study by P. Javali and S. V. Chougule [4] applied Aho-Corasick pattern matching algorithm to detect and prevent SQL injections on Bank Application. To keep user information, they employed Apache Tomcat, and MySQL. The findings demonstrate that the pattern matching technique successfully identifies and secures websites from five different forms of attacks (tautologies, illegal or illogical requests, union, piggy-backed, and alternate encodings).

In order to distinguish between legitimate SQL queries and malicious SQL queries, fingerprinting technique and Pattern Matching are integrated in in the study [13] for a signature-based SQL injection attack detection framework. The system keeps track of all SQL requests made to the database and evaluates them against a database of signatures from previously reported SQL injection attacks. If the fingerprint approach is unable to validate a query on its own, the Aho Corasick algorithm is used to check for the presence of attack signatures in the requests. Their experimental results show that the proposed system can detect a variety of SQL injection attempts with little performance impact.

2.3 Chapter Summary

This chapter presents the recent literatures and related work of the thesis. The research work of SQL injection and detection using machine learning and pattern matching algorithm are described.

CHAPTER 3

SQL INJECTION AND ATTACK TYPES

A prominent attack method is SQL injection, which manipulates back-end databases to retrieve data that was not intended to be displayed. When harmful code is introduced as user input, it is processed by the system as a SQL query and then the malicious code is triggered to run. It has the ability to access data and either erase it or steal information (like user credentials) (to harm a business). An attacker who gains access to data and assumes the identity of a database administrator can then utilize the transferred credentials to get access to the entire system. SQL injection attack is divided into three main arrangements. They are as follow.

[1] Classic In-band SQLI,

[2] Inferential Blind SQLI and

[3] SQLI Based on Out-of-Band

The following is a thorough explanation of SQL injection attacks with accompanying examples. The most prevalent and convenient SQL injection attack is in-band SQL injection [2]. A specific kind of SQL Injection attack known as blind SQL (Structured Query Language) injection requests the system true or false questions and then determines the answer depending on the application's response [13]. Neither any data is actually communicated over the web - based application during an inferential SQLi attack, therefore the attacker cannot observe the attack's in-band results. When an attacker is unable to execute the attack and acquire data through the exact same channel, SQLI Based on Out-of-Band happens. The database server has the capacity to send information to an attacker who can send DNS or HTTP requests [3].

3.1 Impacts of SQL Injection

There are many distinct SQL injection vulnerabilities, attacks, and strategies that emerge in diverse situations.

[1] Leakage of sensitive information

Information leaks happen when private data is made available to unauthorized persons as a result of a security lapse or a cybercrime. For organizations, confidential information leaks are worrisome.

[2] Reputation decline

Likewise, a reputation that is eroding over time refers to one that is steadily getting less, poorer, or worse. It is a reputation that is deteriorating.

[3] Modification of sensitive information

Data that has to be protected is considered sensitive information. The loss of sensitive information, misuse, modification, or unauthorized access to those data has a detrimental impact on an organization's or an individual's welfare, privacy, assets, or security Loss of control of database server.

[4] Data Loss

An SQL injection allows intrusions on data-driven systems, typically to steal sensitive data, by using malicious SQL commands.

[5] Denial of service

Attacks that cause a denial of service drastically reduce the level of service that authorized users receive.

3.2 Types of SQL Injection Attack

[1] **Tautology**

It is a kind of attack in which condition becomes always true. Example of Tautology query attack:

```
SELECT * FROM employee WHERE name = ‘ ’ or 1=1 -- ’ AND password = ‘12345’;
```

[2] **Piggy-Backed Queries**

Additional malicious queries are inserted into an original injected query. Example of Piggy-backed query attack:

```
SELECT * FROM employee WHERE name = ‘guest’ and password = ‘1234’; DROP TABLE employee; --;
```

[3] **Union Query**

UNION keyword is used to get information by joining the injected query with safe query. Example of Union query attack:

```
SELECT emp_id FROM employee WHERE name = ‘’ UNION
SELECT cardNo   FROM    creditCard WHERE accNo = 10032 -- AND password
= ‘ ’ ;
```

[4] **Stored Procedure**

Built-in stored procedure is used with malicious SQL injection codes. Example of stored procedure query attack:

```
CREATE PROCEDURE DBO @userName varchar2, @pass varchar2,
AS EXEC (“SELECT * FROM user WHERE id= ‘ “+@userName+”’ and
password= ‘ “+@pass+’”); GO
```

[5] **Illegal/Logically incorrect query**

This attack lets an attacker to get information about the back-end database of a Web application using error message. Example of Illegal / Logical Incorrect query attack:

```
SELECT * FROM employee WHERE name = ‘ ’ UNION SELECT SUM(username)
from       users -- ’ and password= ‘ ’ ;
```

[6] **Alternate Encodings**

It is a kind of attack which is used to encode the attack strings to avoid the filtering from the programmer (e.g., by using hexadecimal, ASCII and Unicode character set). Example of alternate encoding query attack:

```
SELECT accounts FROM login WHERE username='' AND password=0;
exec ((char (0x73687574646j776e));
```

[7] **Inference**

In-Blind injection, hackers obtain database information by submitting a server’s true / false questions and the answers from this page gives leading information that will be exploited further.Example of inference (blind) SQL injection attack:

```
SELECT * FROM emp_name, emp_address, gender, from employee where 1=0;
drop     employee;
```

CHAPTER 4

PATTERN MATCHING BASED LIBRARY SYSTEM ARCHITECTURE

Lack of awareness and knowledge of security challenges and solutions often leads to ill-informed security decisions in traditional library system development. One of the most popular web attack techniques used by attackers to steal sensitive data from library systems is SQL Injection. It is a method of code injection that enables hackers to insert malicious SQL statements into input fields, which the underlying SQL database subsequently executes. Attackers can even insert malicious SQL queries using web-based library URLs. Even though there has been a great deal of research on SQLIA detection and prevention, SQL injection attacks are still frequent and cannot be totally eradicated. In this study, an effective library system is proposed to detect six common types of SQL injection attacks by using static pattern matching algorithm. The proposed system makes use of an effective token mapping and validation with the static pattern lists whether the authenticated user or not for the library system. It can update a new anomaly pattern to the existing static pattern list whether any form of new anomaly occurs. Moreover, the matching percentage of the attacks can be calculated after detection. Naïve Bayes algorithm is applied to check how many percentages based on the defined threshold and to evaluate the accuracy of the system when SQL Infections are attacked.

4.1 The Proposed Library System

In the proposed library system, static pattern matching algorithm is used to identify and detect any anomaly queries by using static pattern analysis. Figure. 4.1 illustrates the ER(Entity-Relationship) diagram of the proposed system. The relational database (DB) is used to keep the information of the admin, users, books and static patterns. In addition, the data dictionary for the proposed ER design is shown in Table 4.1.

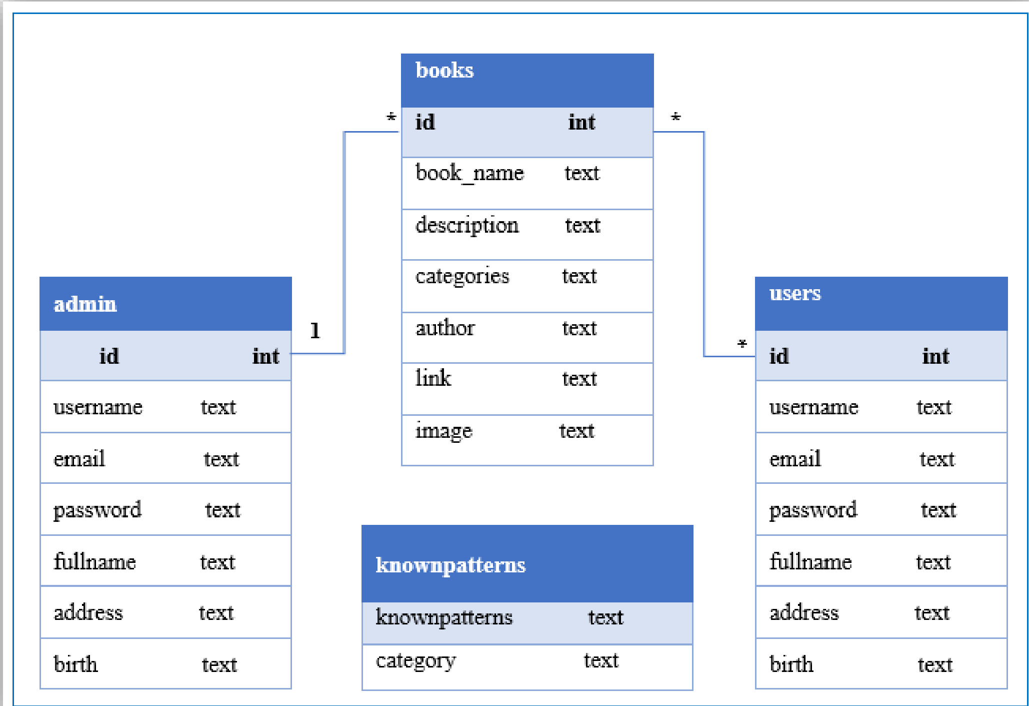


Figure 4.1. ER Diagram of the Proposed System

Table 4.1 Data Dictionary of the Proposed Library System

Table	Attribute	Type	Required	PK/FK	FK Reference Table
admin	id	int		PK	
	username	text	Yes		
	email	text	Yes		
	password	text	Yes		
	fullname	text			
	address	text			
	birth	text			
users	id	int		PK	
	username	text	Yes		
	email	text	Yes		
	password	text	Yes		
	fullname	text			
	address	text			
	birth	text			
books	id	int		PK	
	bookname	text	Yes		
	description	text	Yes		
	categories	text	Yes		
	author	text	Yes		
	link	text			
	image	text			
knownpatterns	knownpatterns	text	Yes		
	category	text			

The step-by-step procedure of pattern matching algorithm is presented in Table 4.2.

Table 4.2. Pattern Matching Procedure	
Input:	user input query
Output:	Pattern matched or not matched
Step 1:	User input SQL query is tokenized and sent to the Pattern Matching algorithm.
Step 2:	User query is compared with stored pattern in existing static database.
Step 3:	If it is equal to static pattern in back-end database, SQL injection attack is detected and exist from the system.
Step 4:	If it is not equal, then detected query is accepted. After mapping, it is added into the existing static database of library system to protect for the next SQL injections.

Table 4.3. System Design Process	
Step 1:	User generated SQL query is sent to the proposed system.
Step 2:	Then the queries are preprocessing by tokenized.
Step 3:	The procedure of pattern matching algorithm is shown in Table 4.2.
Step 4:	If the user generated query does not match each pattern in DB patterns, such a user will be validated with library users' data to identify authenticate user.
Step 5:	Otherwise, if the pattern is match with one of the stored patterns in the anomaly pattern list, this query is identified as entering the library system with a SQL injection attack.
Step 6:	The query is considered as malicious user and then reject the query.
Step 7:	Then, alert is sent to the admin about SQL injection attack, and then pattern mapping is performed.
Step 8:	The anomaly pattern that are not in the static pattern list are inserted and updated to the pattern list of the proposed library system to prevent further SQL injection attack.
Step 9:	The evaluation is performed using the Bayes Classifier.
Step 10:	The output result is showed, the percentage of what kind of SQL attack is injected to the library system.

As indicated in Table 4.3, the purpose of the proposed library system is to examine incoming SQL queries and to identify injection attacks. Algorithm 4.1 shows the detail procedure of the static pattern matching algorithm.

Algorithm 4.1. Static Pattern Matching Algorithm

1: Procedure SPMA (Query, SPL[ ])

INPUT: Query ← User Generated Query

SPL[ ] ← Static Pattern List with m Anomaly Pattern

2: For j=1 to m do

3: If (MA (Query, String Length (Query),

SPL[j][0]) == Ø) then

4: $Anomaly_{score} = \frac{Matching_{value} \text{ (Query, SPL[j])} \times 100}{\text{String.Length(SPL[j])}}$

5: If ($Anomaly_{score} \geq Threshold_{value}$)

6: then

7: Return Alarm →Admin

8: Else

9: Return Query→ Accepted

10: End If

11: Else

12: Return Query→ Rejected

13: End If

14: End For

15: End Procedure

Algorithm 4.2 shows the matching algorithm which is applied in step 3 of the static pattern matching algorithm.

```
Algorithm 4.2. Matching Algorithm
1: Procedure MA (Query, String Length (Query), SPL[j][0])
   INPUT: SPL[j][0] ← Known Pattern
                                  Query ← SQL Query Statement
                                n← Length of string, String Length (Query)
2: m← Length of pattern, prevpattern ← pattern,
    pattern← pattern. Split (" "),
    Query ← Query. Split (" ") matched ← 0,
    lenofmatched ← 0
3: For i=1 to n do
4:     For j=1 to m do
5:         If pattern[j] in Query[i] &&
              pattern[j] not in matched then
6:                 matched.append(pattern[j])
7:                 lenofmatched += len(pattern[j])
8:         End if
9:         return (lenofmatched/len(prevpattern)) * 100
10:    End for
11: End for
12: End Procedure
```

The flow chart of the proposed library system is show in Figure 4.2. The user generated SQL query is firstly validated with malicious pattern list in DB. If there is no match in the malicious pattern list, it will continue and test with the existing users from user registration lists. If there is a match, set it as a authenticate user and allow the library system to access data. If there is no match, identify that it is not the specified user and map the incoming query. The core of the proposed system consists of the following design process.

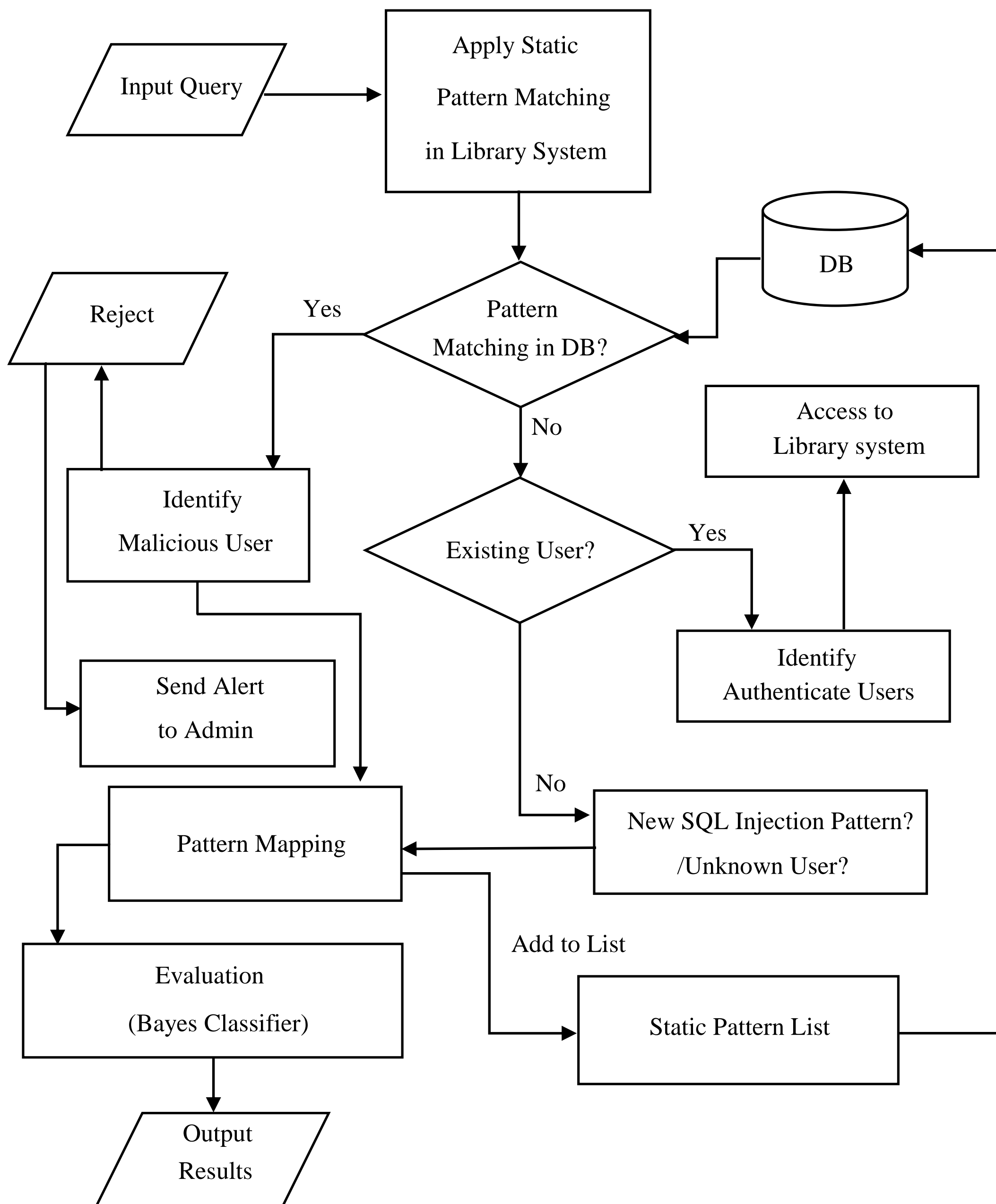


Figure 4.2. Flow Chart of Proposed Library System

If the matching percentage is greater than or equal to the threshold value, it is Yes.

```
Input is : OR 1=1
100 % matched pattern is ....OR 1=1
Detected Pattern is : OR 1=1
Pattern matched : 100.00 %
SQL Injection detected by 100 %
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Tautology
```

Otherwise,

```
Input is : 1 ORDER BY marmarthan -  -+
INFO: 127.0.0.1 - - [22/Sep/2022 18:19:15] "POST /search HTTP/1.1" 200 -
INFO: 127.0.0.1 - - [22/Sep/2022 18:19:15] "←[36mGET /static/js/app.js
HTTP/1.1←[0m" 304 -
INFO: 127.0.0.1 - - [22/Sep/2022 18:19:15] "←[36mGET /static/css/style.css
HTTP/1.1←[0m" 304 -
```

4.2 Naive Bayes Classifier

The Naive Bayes approach is employed in the proposed system to compute the probability of a SQL injection attack in which user generated queries are made against the pattern matching algorithm in a static database system. When calculating the probability of a distinct attack based on numerous occurrences, Naive Bayes outperforms the accuracy in circumstances when computing the probability of attack occurred. Let *A* represent the static database where the SQL injection attack was discovered. Let ***B*** be the emergence of the SQL injection attack.

$$P(A|B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{4.1}$$

Where is the P(A|B) probability of event *A* occurring given that event *B* has occurred. P (B|A) is the probability of event *B* occurring given that event a has occurred. *P(A*) and *P(B)* are the probabilities of observing *A* and *B* independently of each other.

4.3 Admin Login Page

Figure 4.3 depicts the admin login page of the proposed library system. An admin can access an application by providing their username and password on the login screen. This system will display a warning as shown in Figure 4.4 if the credentials of the admin do not match. During successful validation, the admin will see the secure portion of the application as shown in Figure 4.5.

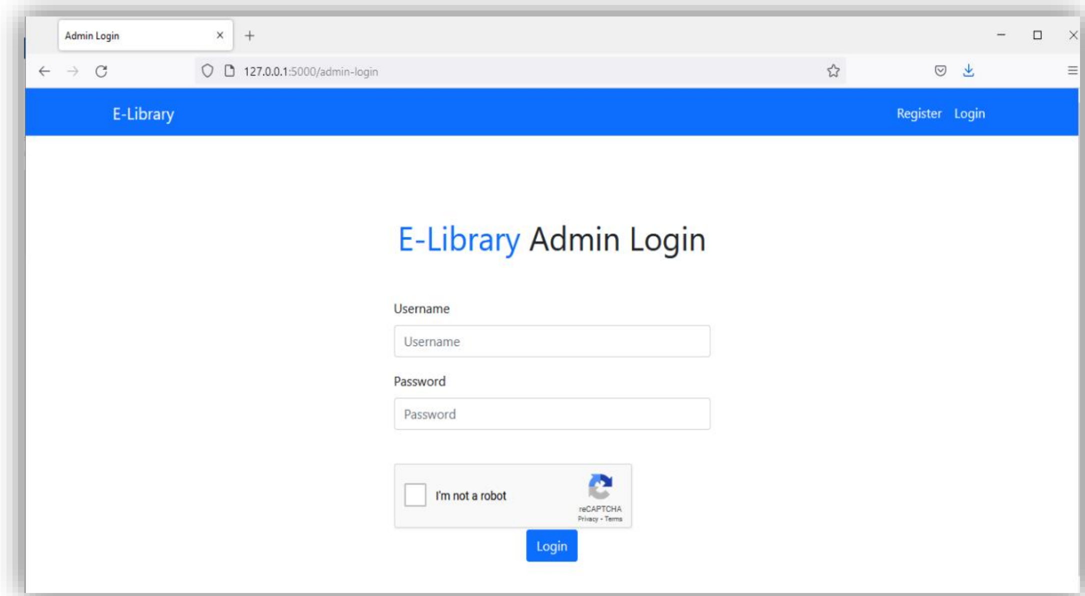


Figure 4.3 Admin Login Page of the System

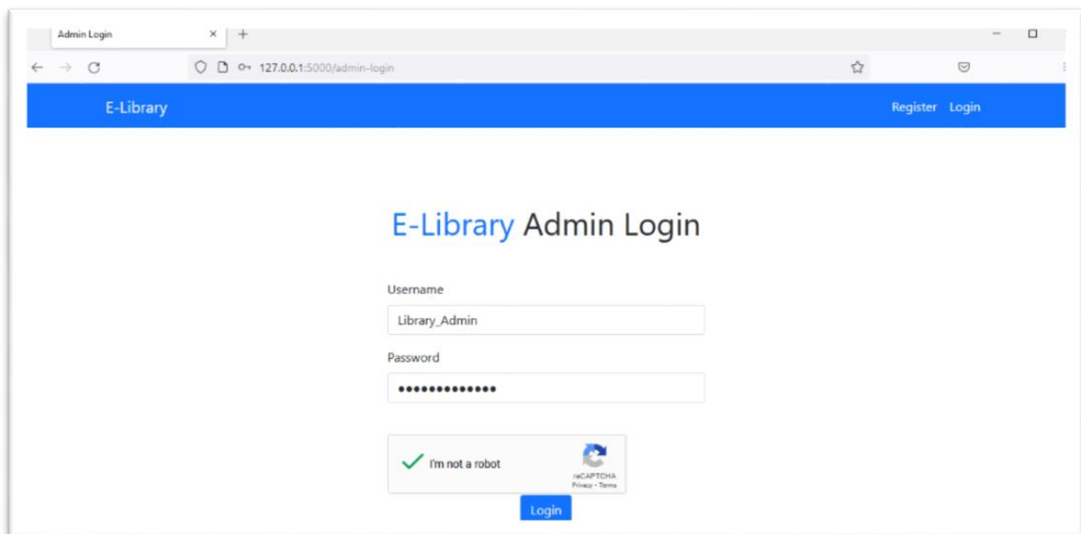


Figure 4.4 Correct Admin Login Page of the System

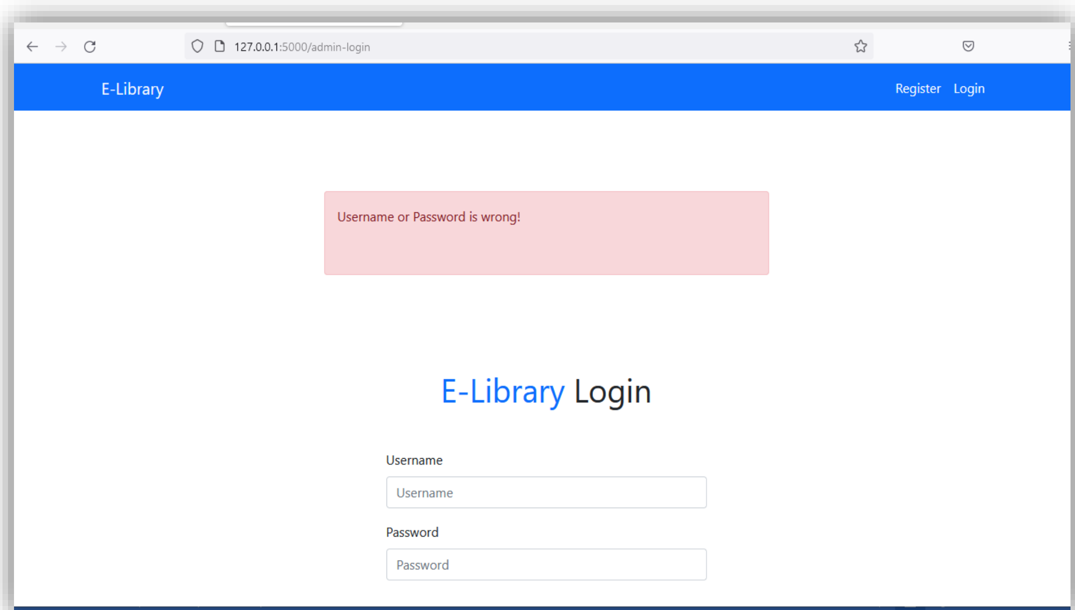


Figure 4.5 Incorrect Admin Login Page of the System

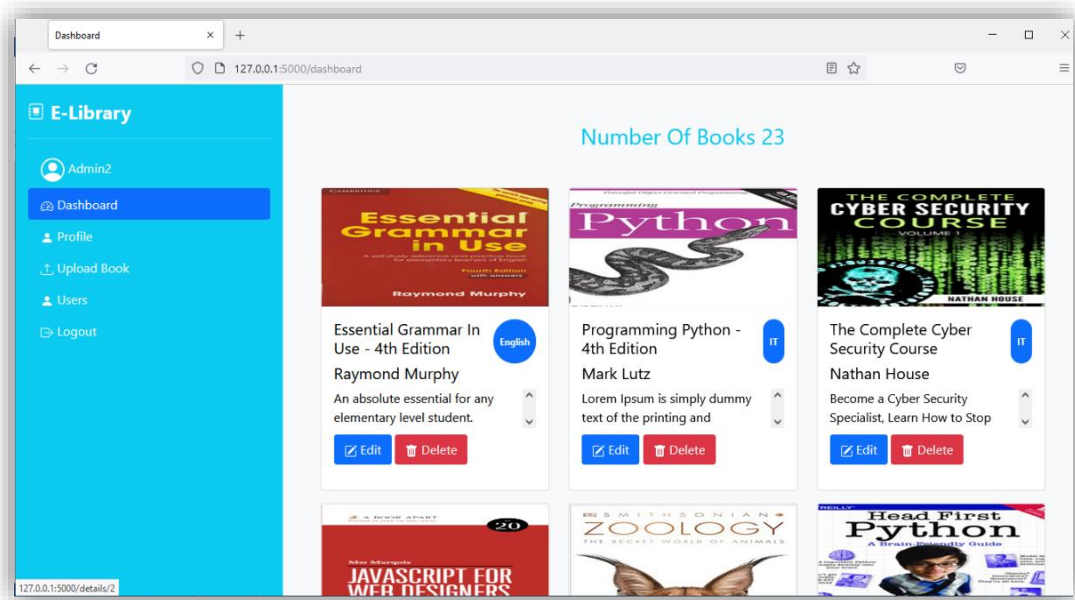


Figure 4.6 Dashboard Page of the System

Figure 4.6 shows the dashboard page of the proposed library system. This interface contains Admin profiles, upload Book Lists, number of users and Logout. This position comes duties and responsibilities which are related to the digital use of library items in all formats, as well as administration of the entire library system, including development, publishing, and content authorship. The profile view of the admin account is shown in Figure 4.7.

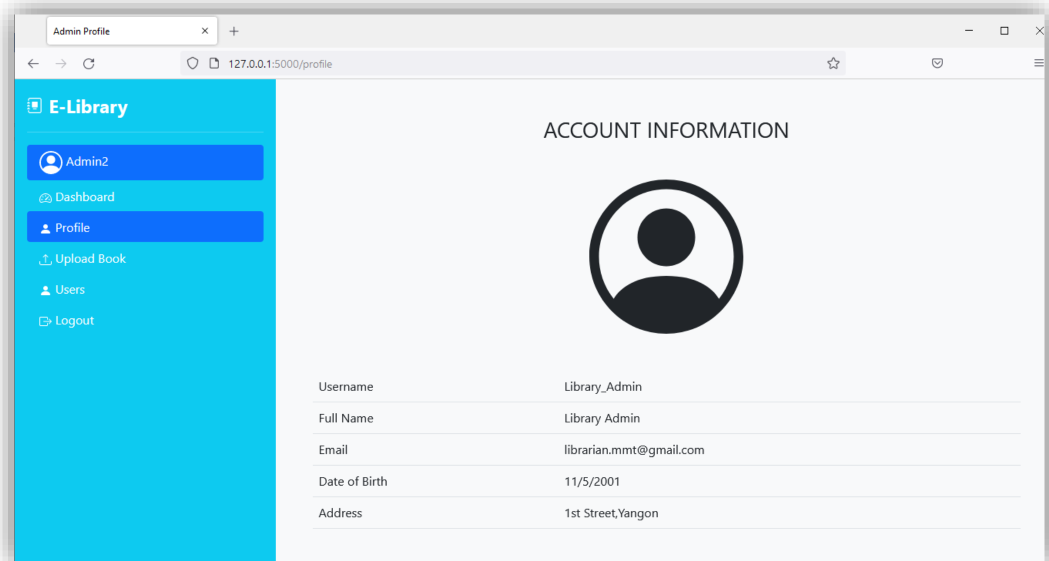


Figure 4.7 Admin Profile Page of the System

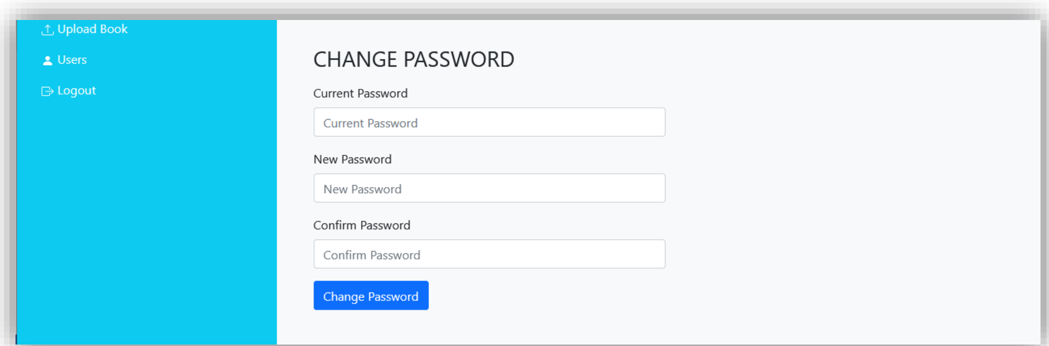


Figure 4.8 Admin Password Change View of the System

. If the admin wants to reset the password, he can change it in any time. The password reset page of admin is shown in Figure 4.8. Admin can view the number of users in the system. User view from admin side of the system is illustrated in Figure 4.9.

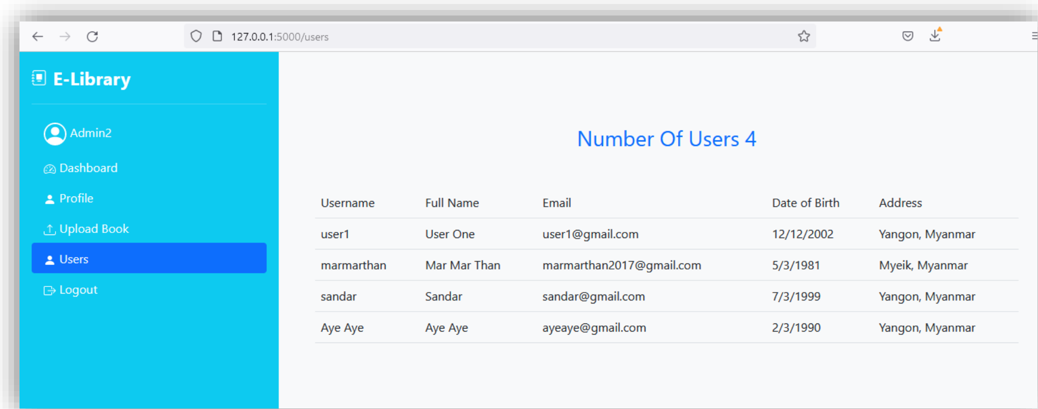


Figure 4.9 User View from Admin Side of the System

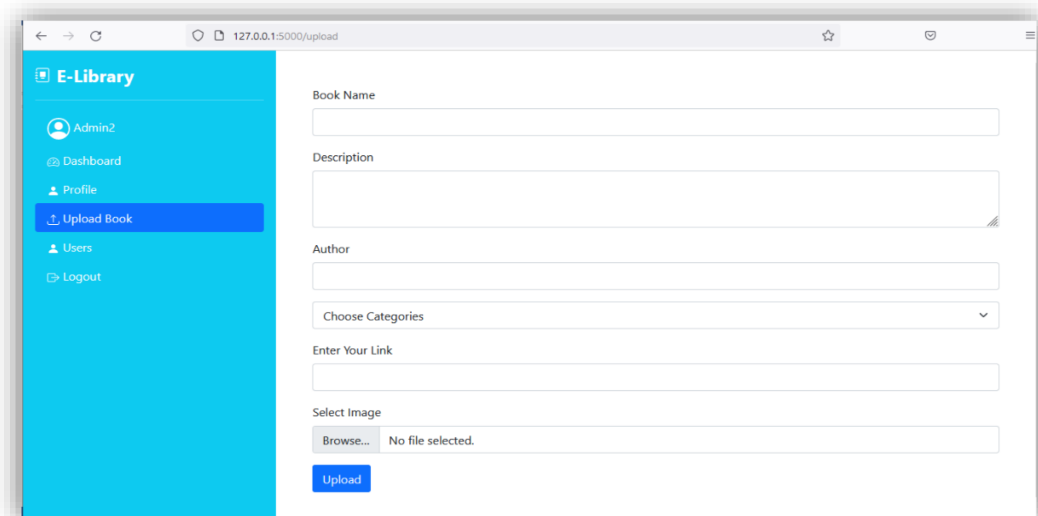


Figure 4.10 Upload Book View from Admin Side of the System

Figure 4.10 shows the upload book view from admin side of the system. The administrator can upload eBooks and add book details on the page for books. As show in Figure 4.11, the Elementary Information Security book is inserted to the library as an example. After upload the new book, the successful message will be displayed as depicted in Figure 4.12.

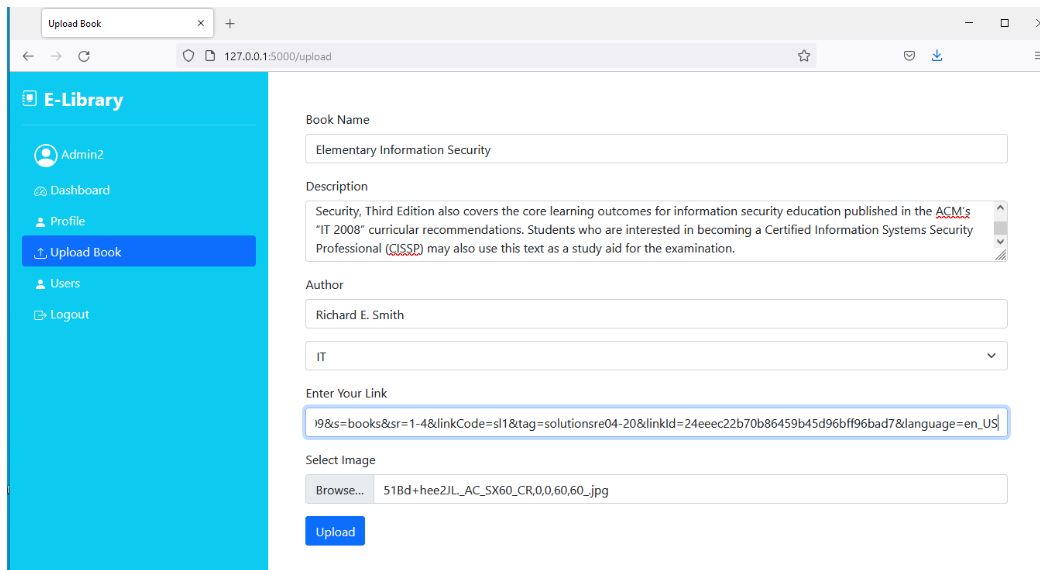


Figure 4.11 Upload New Book to the System

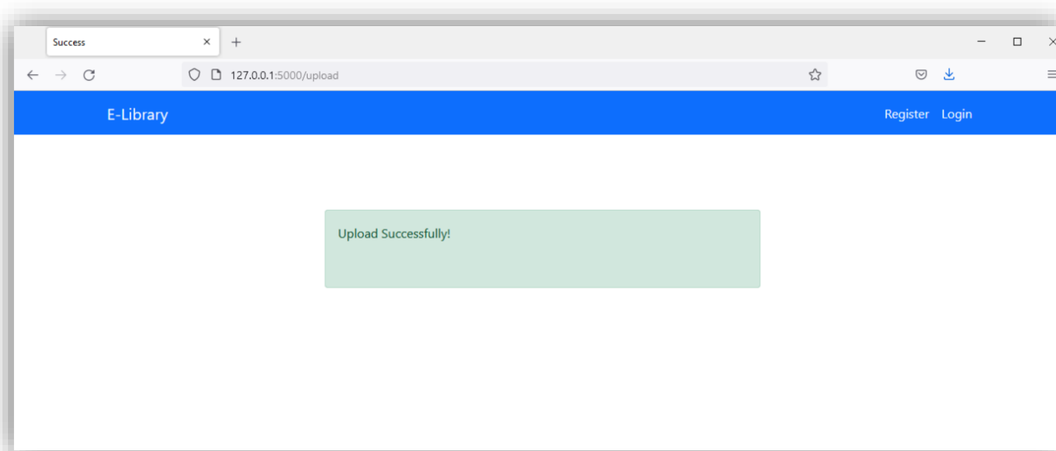


Figure 4.12 Successful Upload New Book to the System

The number of books in the library system would be updated after inserting the new book to the system. The update number of book in the system is shown in Figure 4.13.

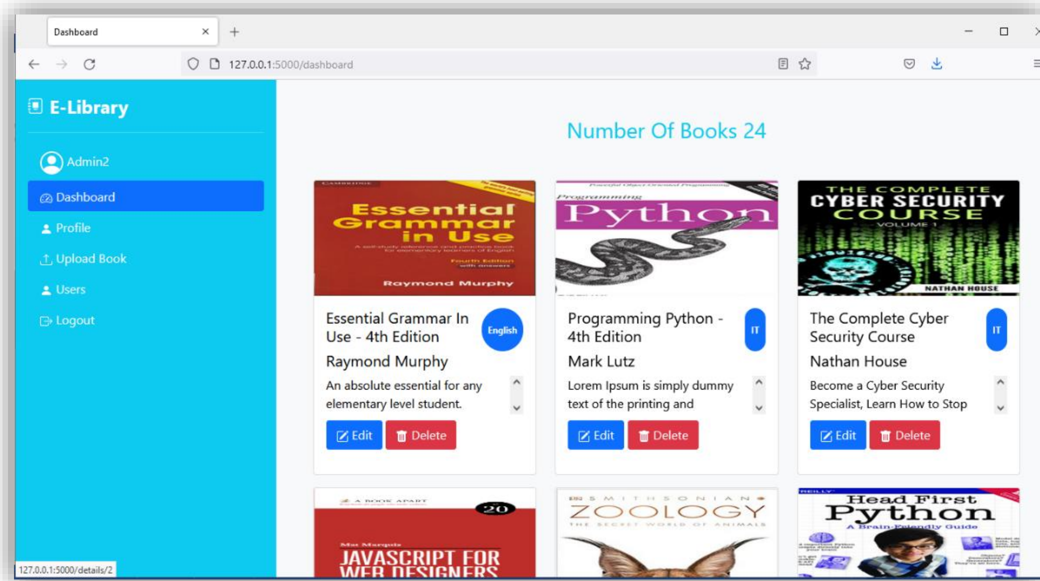


Figure 4.13 Update Number of Books in the System

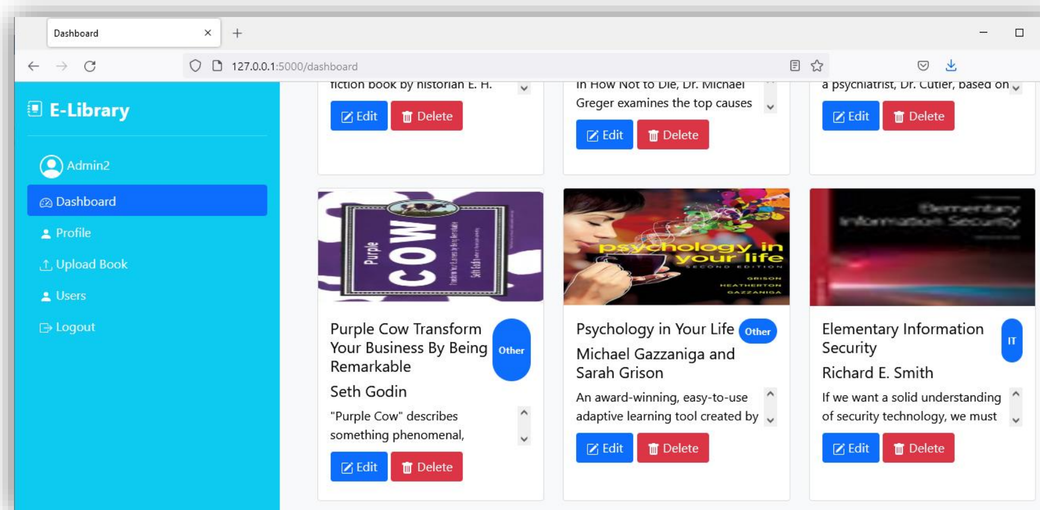


Figure 4.14 Update Book Information in the System

As shown in Figure 4.14, the inserted "Elementary Information Security" book can be seen in the total book lists. The admin can update and delete the information of books in the system. Figure 4.15 shows the book information edited by admin.

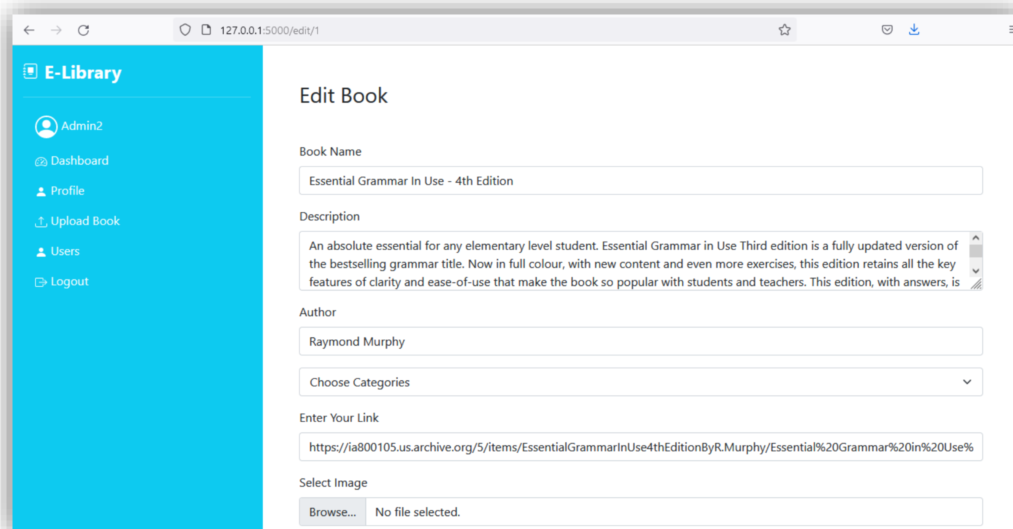
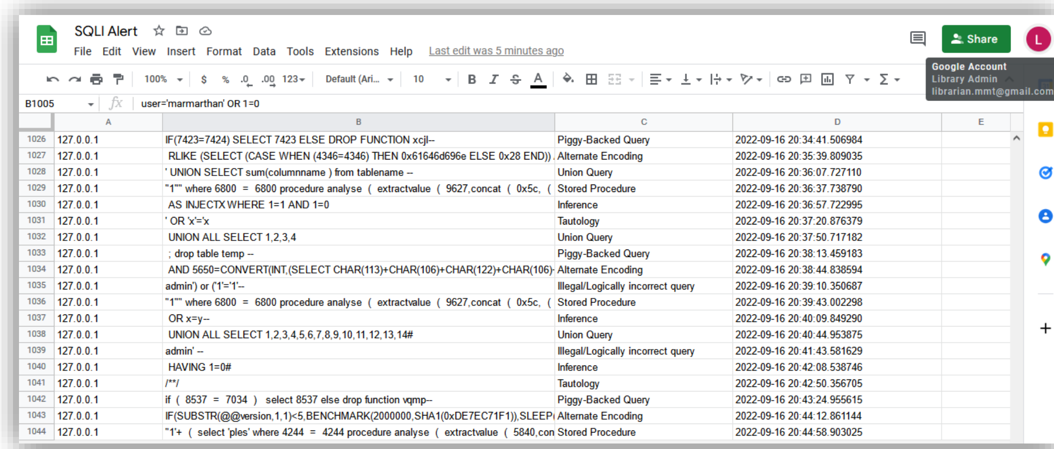


Figure 4.15 Update Book Information by Admin



	A	B	C	D	E
1026	127.0.0.1	IF(7423=7424) SELECT 7423 ELSE DROP FUNCTION xcjl--	Piggy-Backed Query	2022-09-16 20:34:41.506984	
1027	127.0.0.1	RLIKE (SELECT (CASE WHEN (4346=4346) THEN 0x61646d696e ELSE 0x28 END)) .	Alternate Encoding	2022-09-16 20:35:39.809035	
1028	127.0.0.1	' UNION SELECT sum(columnname) from tablename --	Union Query	2022-09-16 20:36:07.727110	
1029	127.0.0.1	"1"" where 6800 = 6800 procedure analyse (extractvalue (9627,concat (0x5c, (	Stored Procedure	2022-09-16 20:36:37.738790	
1030	127.0.0.1	AS INJECTX WHERE 1=1 AND 1=0	Inference	2022-09-16 20:36:57.722995	
1031	127.0.0.1	' OR 'x'='x	Tautology	2022-09-16 20:37:20.876379	
1032	127.0.0.1	UNION ALL SELECT 1,2,3,4	Union Query	2022-09-16 20:37:50.717182	
1033	127.0.0.1	; drop table temp --	Piggy-Backed Query	2022-09-16 20:38:13.459183	
1034	127.0.0.1	AND 5650=CONVERT(INT,(SELECT CHAR(113)+CHAR(106)+CHAR(122)+CHAR(106)-	Alternate Encoding	2022-09-16 20:38:44.838594	
1035	127.0.0.1	admin') or ('1'='1'--	Illegal/Logically incorrect query	2022-09-16 20:39:10.350687	
1036	127.0.0.1	"1"" where 6800 = 6800 procedure analyse (extractvalue (9627,concat (0x5c, (	Stored Procedure	2022-09-16 20:39:43.002298	
1037	127.0.0.1	OR x=y--	Inference	2022-09-16 20:40:09.849290	
1038	127.0.0.1	UNION ALL SELECT 1,2,3,4,5,6,7,8,9,10,11,12,13,14#	Union Query	2022-09-16 20:40:44.953875	
1039	127.0.0.1	admin' --	Illegal/Logically incorrect query	2022-09-16 20:41:43.581629	
1040	127.0.0.1	HAVING 1=0#	Inference	2022-09-16 20:42:08.538746	
1041	127.0.0.1	/**/	Tautology	2022-09-16 20:42:50.356705	
1042	127.0.0.1	if (8537 = 7034) select 8537 else drop function vqmp--	Piggy-Backed Query	2022-09-16 20:43:24.955615	
1043	127.0.0.1	IF(SUBSTR(@@version,1,1)<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(	Alternate Encoding	2022-09-16 20:44:12.861144	
1044	127.0.0.1	"1'+ (select 'ples' where 4244 = 4244 procedure analyse (extractvalue (5840,con	Stored Procedure	2022-09-16 20:44:58.903025	

Figure 4.16 Alert Attack Information to Admin

The system alert attack information to admin as show in Figure 4.16.

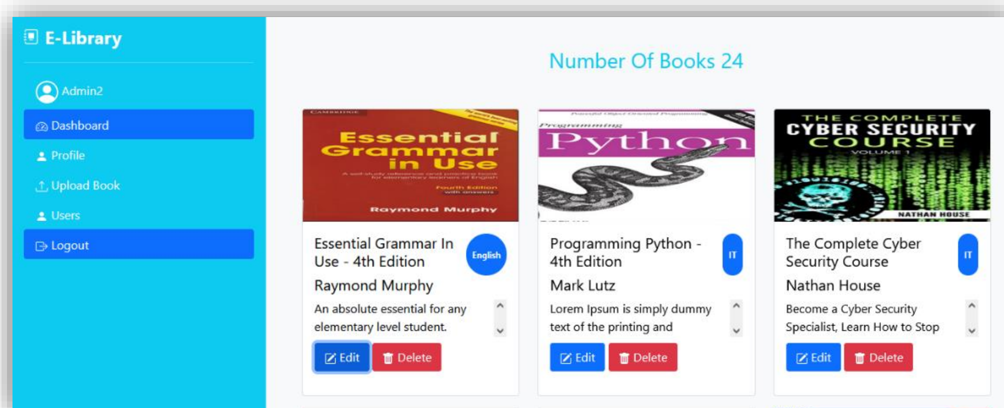


Figure 4.17 Logout Page (Admin View)

4.4. User Login Page

After admin has chosen to logout from the system, Logout page brings the Login page to enter the system as the user view. It is illustrated in Figure 4.17 and 4.18.

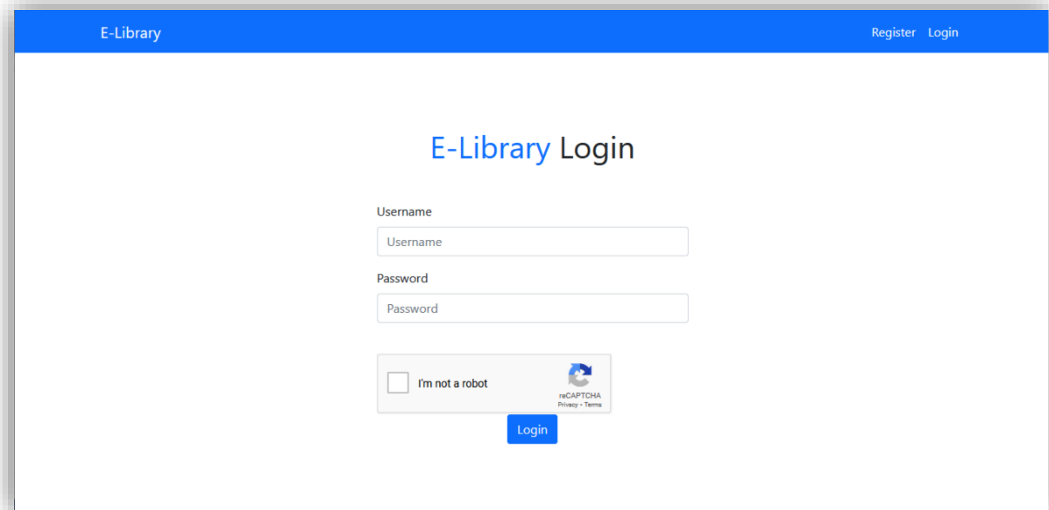


Figure 4.18 Login Page of the System

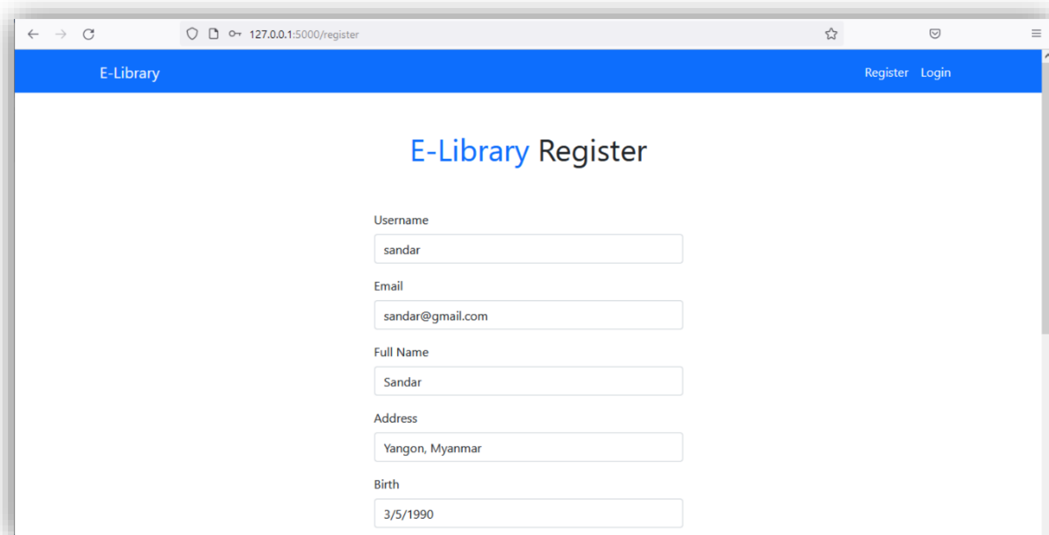


Figure 4.19 Registration Page of the System

User must first establish an account in order to begin the registration process. The new user needs to register to use the library system. All mandatory fields on the registration form must be completed properly. Asterisks (*) denote field, username and password that are required. The email address that the user enters on this form must be legitimate and their own. Figure 4.19 shows the registration page of the system.

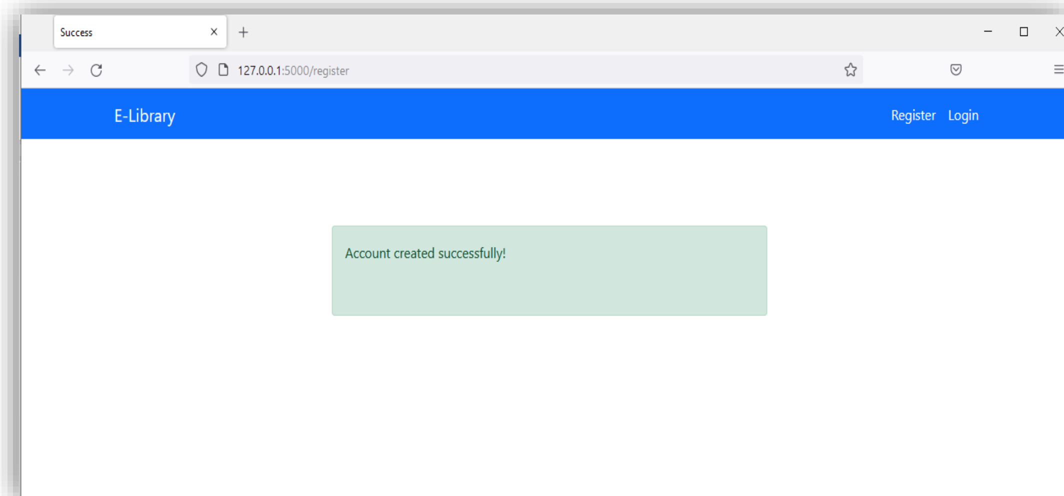


Figure 4.20 Successful Registration Page of the System

After registration process is performed completely, the successful registration message will be displayed as shown in Figure 4.20. When a user is ready to look for a book in the system, he/she can type the name of the book he/she wants to find into the search bar. The search bar has the ability to direct the user's search inquiry to a particular system activity. In this approach, the user can start a search from any activity that has a search bar, and the system will launch the proper activity to conduct the search and display the results. Figure 4.21 shows the find book page of the system.

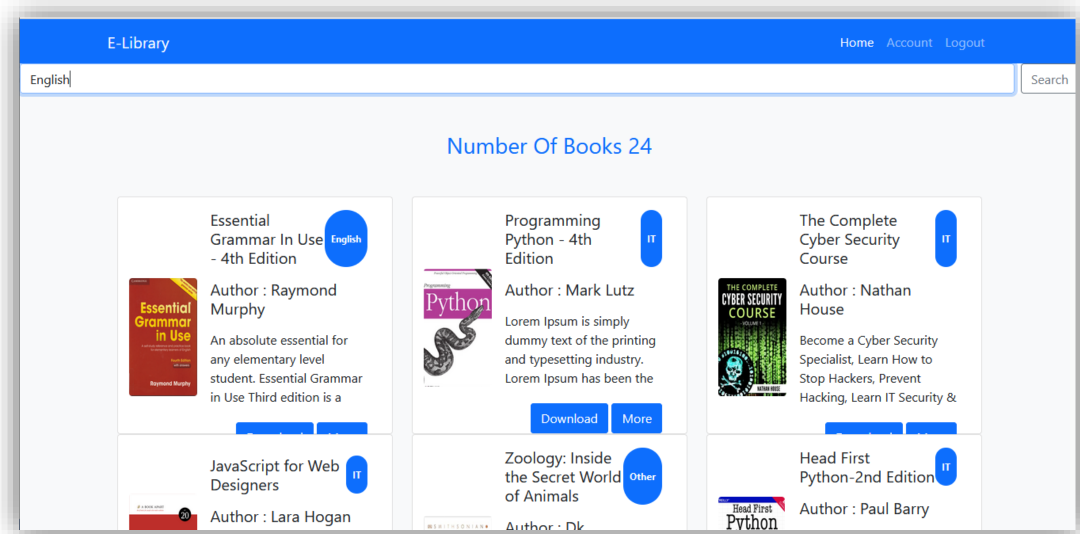


Figure 4.21 Find Book Page of the System

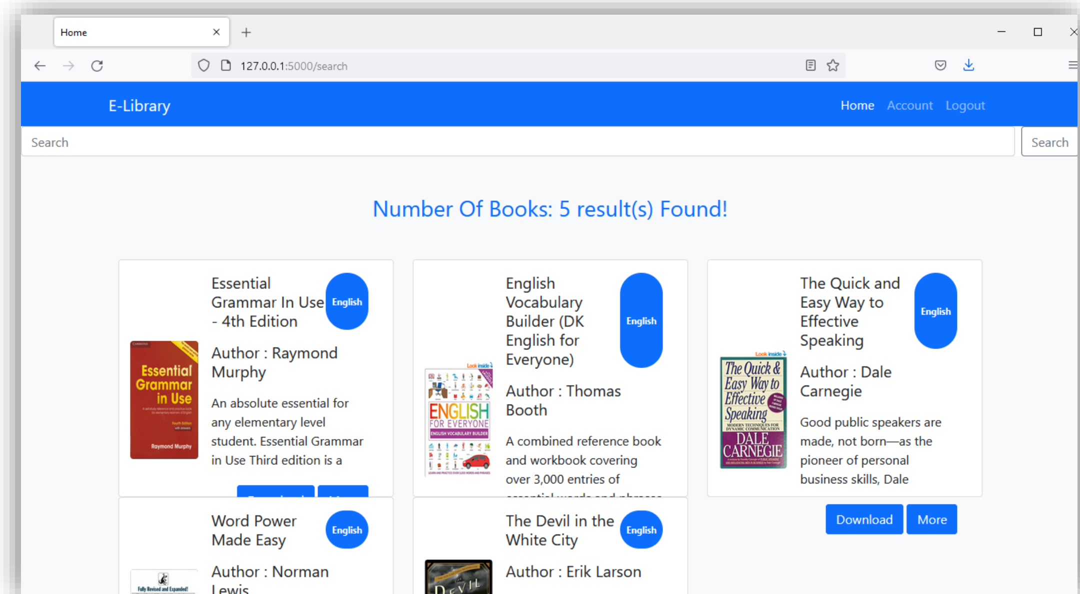


Figure 4.22 Result Page of the System

User can search by title, author, or keywords using the Search bar. One book at a time or the entire system's content can be searched by the user. Enter a word or phrase in the Search box at the top of the system homepage and press Search to conduct a system-wide search. The books in the database are explored, and a result page will be shown based on the number of items discovered. Only when the Facets feature is used will older versions or editions of content in the system—as well as content that its authors or publishers no longer deem to be current—be included in the search results. By choosing a book from the list of titles on the Browse Titles page, users can do one book at a time searches. User can simply click on the book’s cover image or hyperlink to view the Table of Contents page. There is a download and more button in this book located under the book’s title and details. User can perform one or more work to query the book. User’s search results will be displayed on a separate page and sorted by relevance. The results of the user finding book are displayed on Figure 4.22. When the user chooses the specific book in result lists, the desired book can be viewed in the following Figure 4.23.

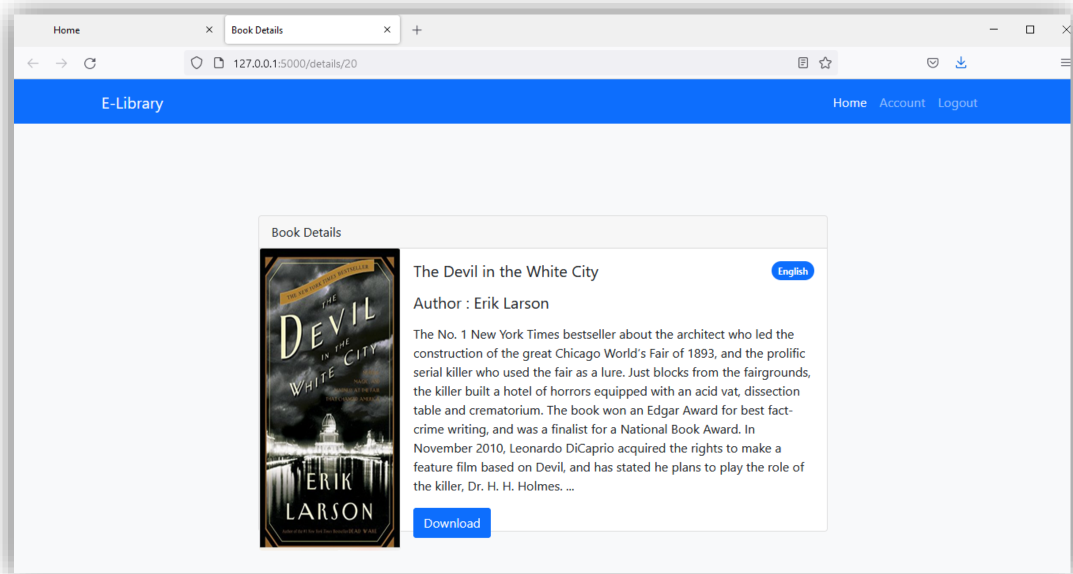


Figure 4.23 Finding Book Result of the System

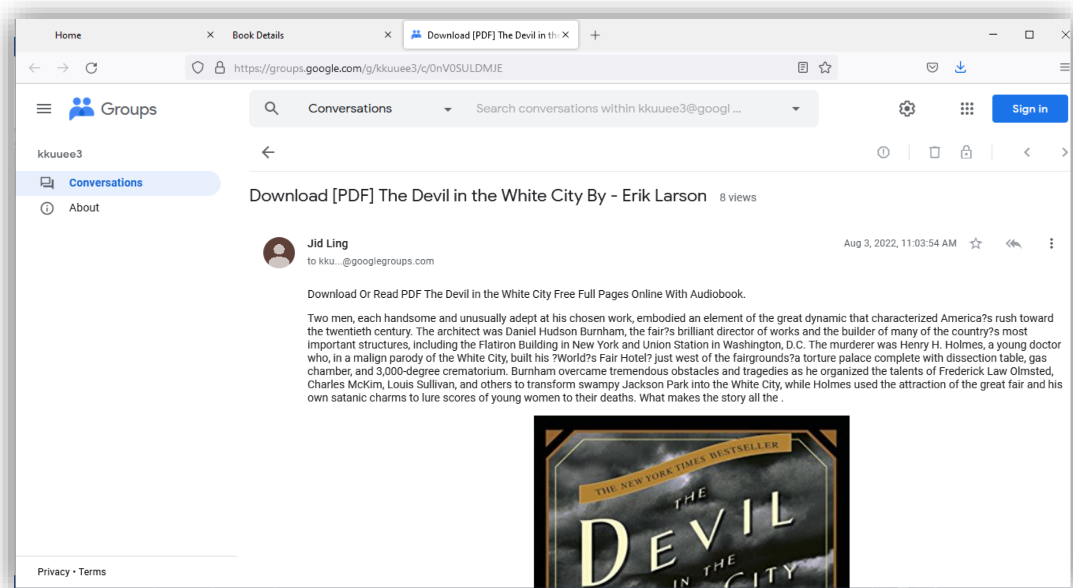


Figure 4.24 Download Page of the System

When the user clicks the download button in finding book, the download page of the system view can be seen as in Figure 4.24. Each user has a profile page, which can be accessed by selecting Profile from the user menu in the top right as shown in Figure 4.25. This page offers connections to additional pages where the user can examine their posts, modify their profile information and preferences.

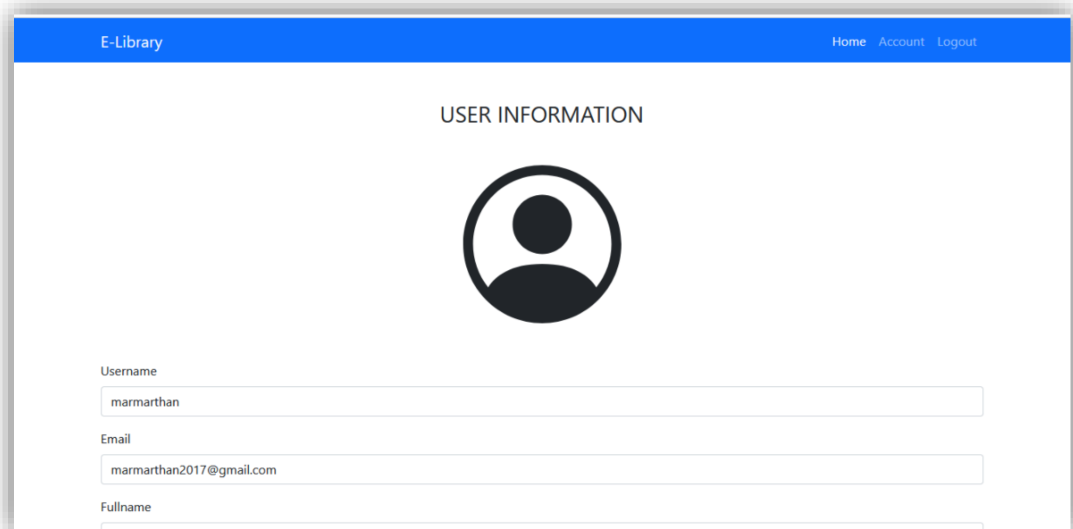


Figure 4.25 User Profile Page

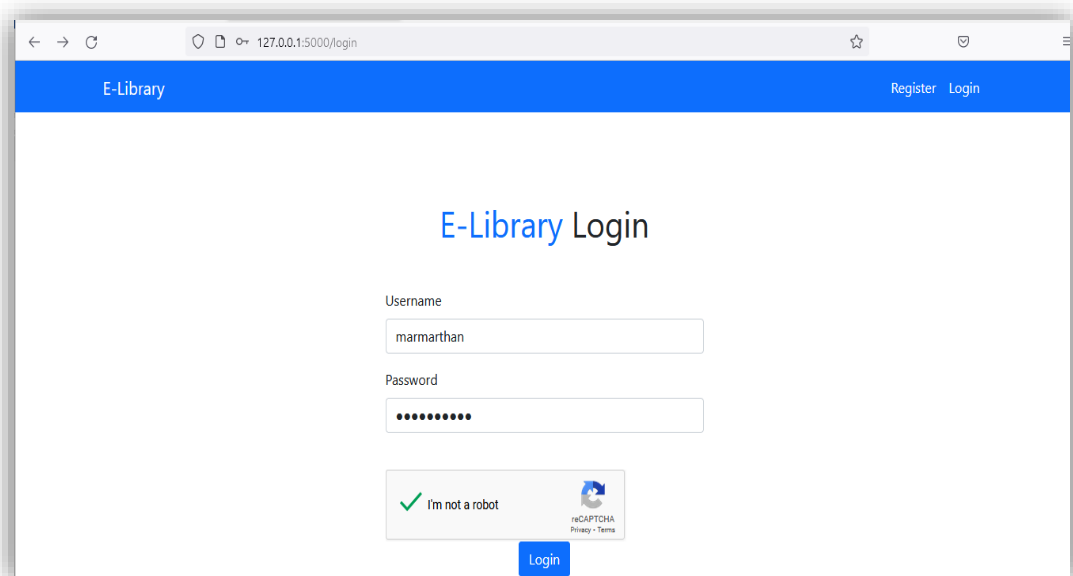


Figure 4.26 User Login Page

User Login Page is shown in Figure 4.26. Users must provide both a user name and a password to login into the system. Although logins are made for public, passwords need to be kept private. Only the user should be aware of their password. Users should frequently change their passwords because the malicious users can hack these passwords. When the user will type incorrect username or password, the alert "Username or Password is wrong!" is displayed on page as shown in Figure 4.27.

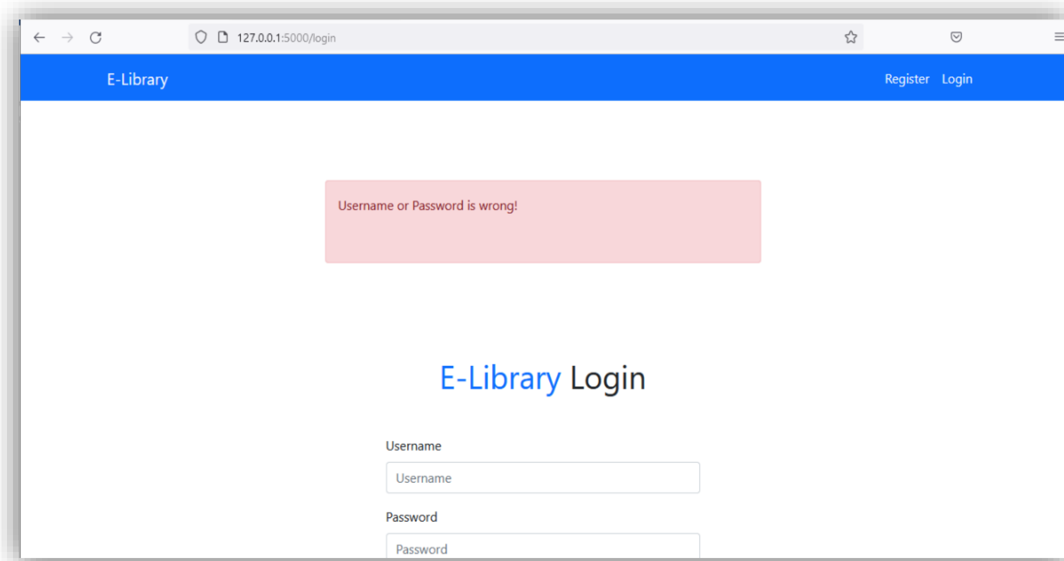


Figure 4.27 Incorrect User Login Page

Most probably, the network is spamming or the people are moving too quickly to keep up with the malware. To remove the "I'm not a robot" CAPTCHA message, consider about thoroughly inspecting the system network, slowing down user activity, and using public DNS. User must check the “I’m not a robot” button as shown in the Figure 4.28. If the user forgets or does not check this work, the verification message will be displayed from the system. The view of verifying Captcha message is shown in Figure 4.29.

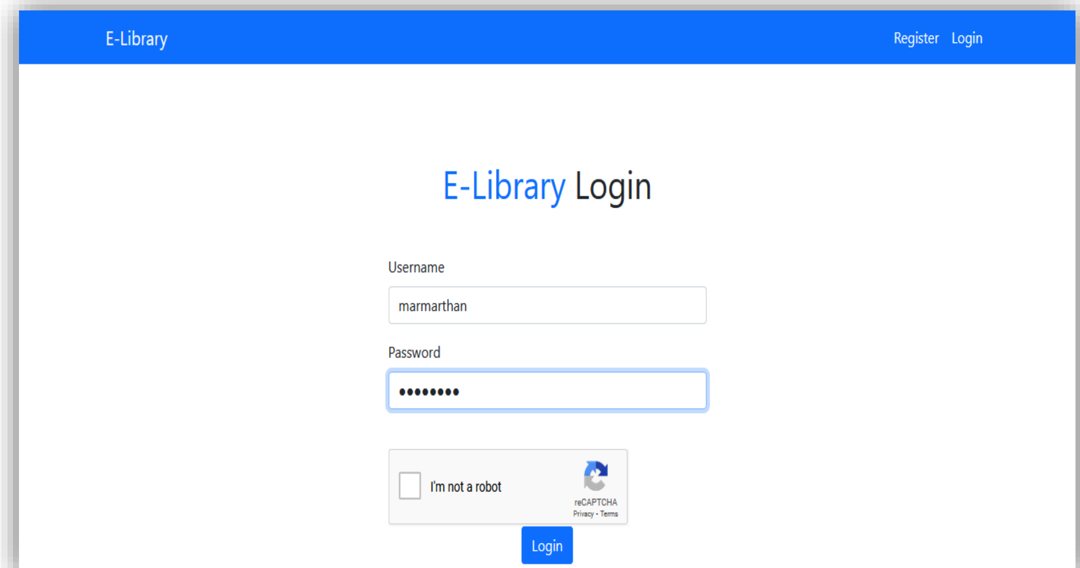


Figure 4.28 User Validation Page

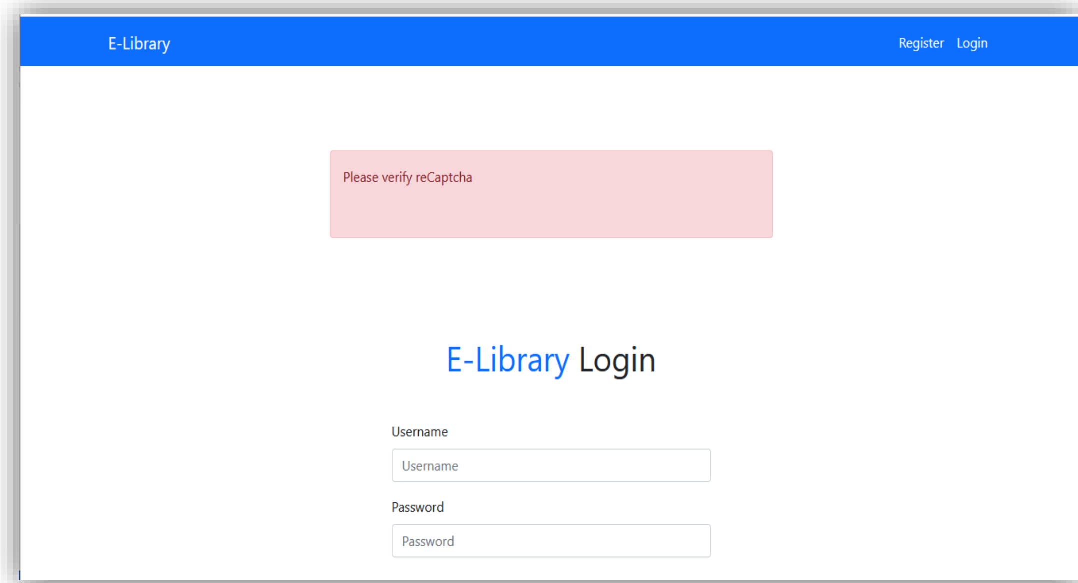


Figure 4.29 User Validation Alert Page

4.5 Experiment and Result Discussion

The experimental setup consisted of a standard desktop computer with Intel(R) Core (TM) i3-7100U CPU @ 2.40GHz and 2.4 GHz. Python 3.1 and VS code. This section presents the evaluation of the proposed system to detect SQL injection attack experiments, namely, Tautology, Union, Logically Incorrect, Piggy-Backed, Alternate Encodings, Stored Procedure and Inference. Testing threshold values is 80. The number of evaluation SQL patterns is 24648. These patterns are stored in static pattern list of the library system and evaluated with the incoming user generated query. This section explains how to inject SQL statements into vulnerable systems via user input.

An SQL query is generated from the user's information (user name and password) and submitted to the database for verification when a real user provides appropriate information. The user is given access if their username and password are both authentic. After verification, a legitimate user is given access and is permitted to display their information; otherwise, an error message is produced.

4.5.1 Alternate Encoding

This technique employed alternate methods of encoding attack strings in the following statement. By employing alternative encoding in the SQL commands, this

technique trickles the database of the system. In a SQL statement, for instance, an attacker may use hexadecimal, ASCII, or Unicode. Attackers will get beyond any fundamental validation carried out by the system in this manner.

RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='

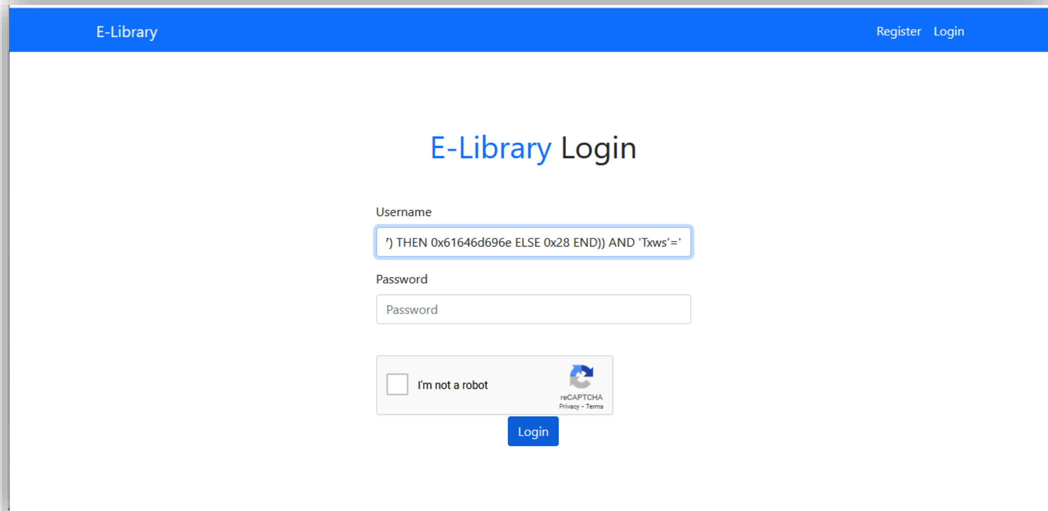


Figure 4.30 Alternate Encoding Attack from User Input

The system verifies this user generated query with static pattern list of the system. Firstly, the input username or password is incorrect message when it does not match. This has already shown in Figure 4.29. An attacker needs to locate an input that is vulnerable in the system in order to launch a SQL injection attack. When a system has a vulnerability for SQL injection, it directly uses user input in the form of a SQL query. Figure 4.30 shows the Alternate Encoding attack from user input and Figure 4.31 shows Alternate Encoding attack from search input box.

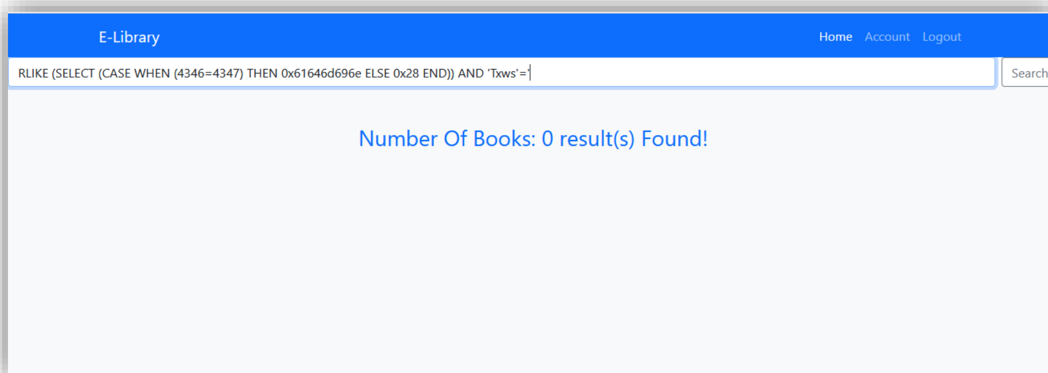


Figure 4.31 Alternate Encoding Attack from Search Input Box

The proposed system is a fully-automated and general technique for detecting and preventing all types of SQL injection. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Alternate Encoding". The detection process of Alternate Encoding attack is shown in Figure 4.32.

```
Input is :  RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='
Pattern is : RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='
Pattern matched : 100.00 %
SQL Injection detected!
100 % matched pattern is .... RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='
Detected Pattern is :  RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='
Pattern matched : 100.00 %
SQL Injection detected by 100 %
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Alternate Encoding
DEBUG: Resetting dropped connection: sheets.googleapis.com
DEBUG: https://sheets.googleapis.com:443 "POST /v4/spreadsheets/1n8DoWm2WRQtMrQ19uf6almcUpThy47cUpzHOBz3CJ94/values/%27Sh
eet1%27:append?valueInputOption=RAW&includeValuesInResponse=False HTTP/1.1" 200 None
INFO: 127.0.0.1 - - [28/Aug/2022 22:02:43] "POST /login HTTP/1.1" 200 -
INFO: 127.0.0.1 - - [28/Aug/2022 22:02:43] "←[36mGET /static/css/style.css HTTP/1.1←[0m" 304 -
INFO: 127.0.0.1 - - [28/Aug/2022 22:02:43] "←[36mGET /static/js/app.js HTTP/1.1←[0m" 304 -
```

Figure 4.32 Alternate Encoding Attack Detection

4.5.2 Inference Attack

This form of injection targets well-secured databases that do not provide any actionable feedback or informative error signals. Attacks are typically developed in the form of true false statements. After identifying the weak point, the attacker uses query to inject numerous conditions (that he wants to know whether they are true or not) and carefully monitor the environment. The page continues to operate normally if the statement is true. If false, the page behaves very differently from how it would usually. Blind Injection is the name given to this kind of injection. The term "Time Attack" refers to a different kind of inference attack. In this technique, an attacker creates a conditional statement, injects it through the parameter that is weak, and gathers data based on delays in the database's response.

AS INJECTX WHERE 1=1 AND 1=0#

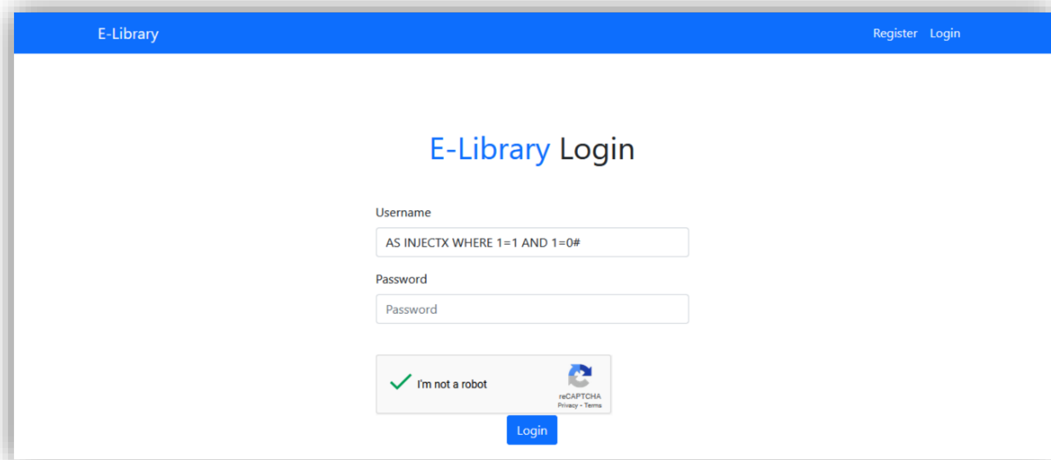


Figure 4.33 Inference Attack from User Input

. Figure 4.33 shows the Inference attack from user input and Figure 4.34 shows Inference attack from search input box.

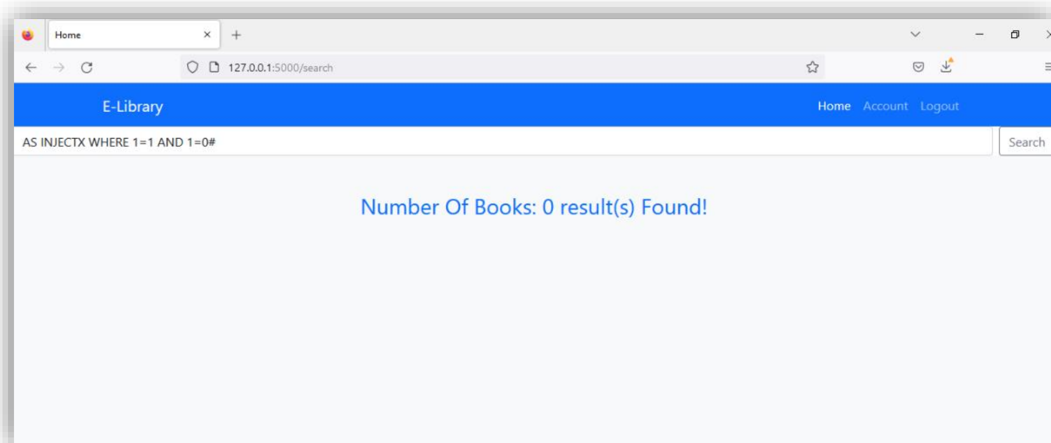


Figure 4.34 Inference Attack from Search Input Box

The system verifies this user generated query with static pattern list of the system. Firstly, the input username or password is incorrect message when it does not match. This has already shown in Figure 4.27.

```
Input is : AS INJECTX WHERE 1=1 AND 1=0#
Pattern is : AND 1=0
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : AND 1=0#
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : AND 1=1
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : AS INJECTX WHERE 1=1 AND 1=0
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : WHERE 1=1 AND 1=0
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : WHERE 1=1 AND 1=0#
Pattern matched : 100.00 %
SQL Injection detected!
100 % matched pattern is ....AS INJECTX WHERE 1=1 AND 1=0#
Detected Pattern is : AS INJECTX WHERE 1=1 AND 1=0#
Pattern matched : 100.00 %
SQL Injection detected by 100 %
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Inference
DEBUG: Making request: POST https://oauth2.googleapis.com/token
DEBUG: Resetting dropped connection: oauth2.googleapis.com
DEBUG: https://oauth2.googleapis.com:443 "POST /token HTTP/1.1" 200 None
DEBUG: Resetting dropped connection: sheets.googleapis.com
```

Figure 4.35 Inference Attack Detection

The proposed system is a fully-automated and general technique for detecting and preventing all types of SQL injection. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Inference". The detection process of Inference attack is shown in Figure 4.35.

4.5.3 Logically Incorrect Attack

By inserting illegal or illogical requests, such as injectable parameters, data types, database names, etc., an attacker may obtain knowledge. As an example, the following SQL injection query is used to enter the system illegally.

```
SELECT avg ( column_name ) FROM table_name WHERE condition ;
```

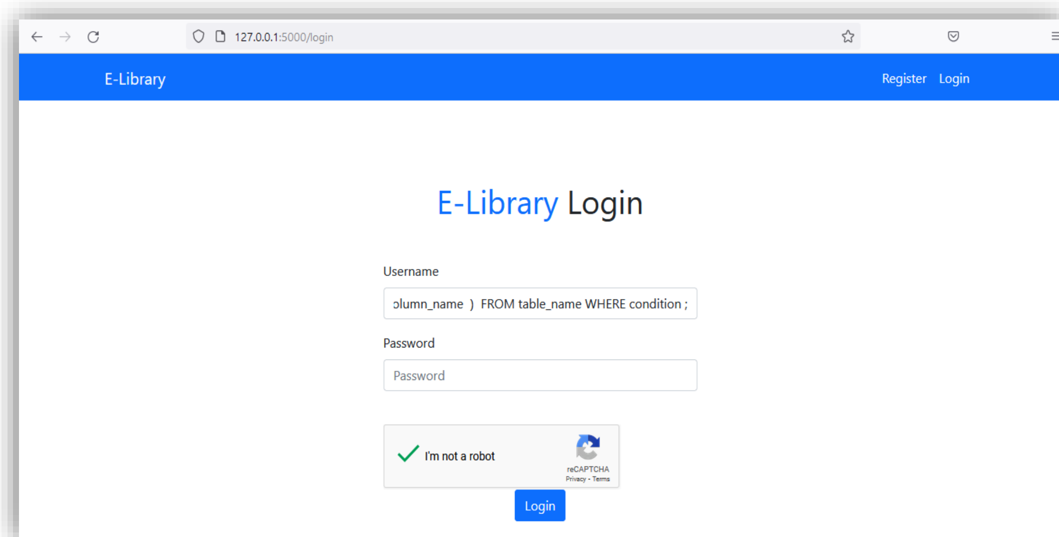
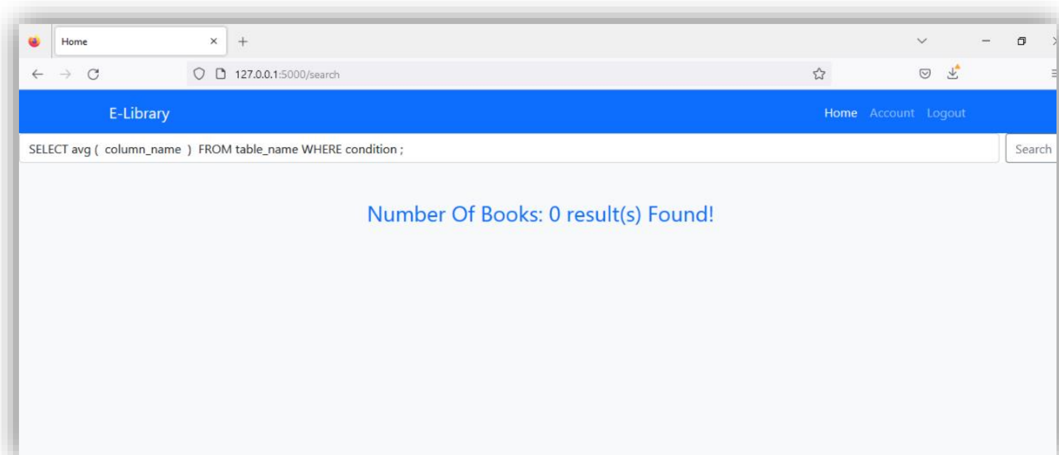


Figure 4.36 Logical Incorrect Attack from User Input



. **Figure 4.37 Logical Incorrect Attack from Search Input Box**

Figure 4.36 shows the Logically Incorrect attack from user input and Figure 4.37 shows Logically Incorrect attack from search input box. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Logically Incorrect". The detection process of Logically Incorrect attack is shown in Figure 4.38.

```
Input is : SELECT avg (  column_name  )  FROM table_name WHERE condition ;
Pattern is : SELECT * FROM table_name ;
Pattern matched : 96.15 %
SQL Injection detected!
Pattern is :  ( SELECT column_name FROM table_name WHERE condition  ) ;
Pattern matched : 100.00 %
SQL Injection detected!
Pattern is : SELECT ALL column_name FROM table_name WHERE condition ;
Pattern matched : 94.64 %
SQL Injection detected!
Pattern is : SELECT column_name ( s )  FROM table_name
Pattern matched : 97.56 %
SQL Injection detected!
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Illegal/Logically incorrect query
DEBUG: Resetting dropped connection: sheets.googleapis.com
DEBUG: https://sheets.googleapis.com:443 "POST /v4/spreadsheets/1n8DoWm2WRQtMrQ19uf6almcUpThy47cUpzHOBz3CJ94/values/%27Sheet1%27:append?valueInputOption=RAW&includeValuesInResponse=False HTTP/1.1" 200 None
```

Figure 4.38 Logical Incorrect Attack Detection

4.5.4 Piggy-Backed Attack

A form of attack known as "piggy-backed queries" attacks a system by inserting extra query statements into the original query using a query delimiter like ";". In this approach, the initial query is the original one, while the following queries are injections. This exploit is extremely serious since it allows the attacker to insert almost any kind of SQL statement. The piggy-backed query attack is demonstrated by the SQL statement in the following query.

IF (7423=7424) SELECT 7423 ELSE DROP FUNCTION xcjl--

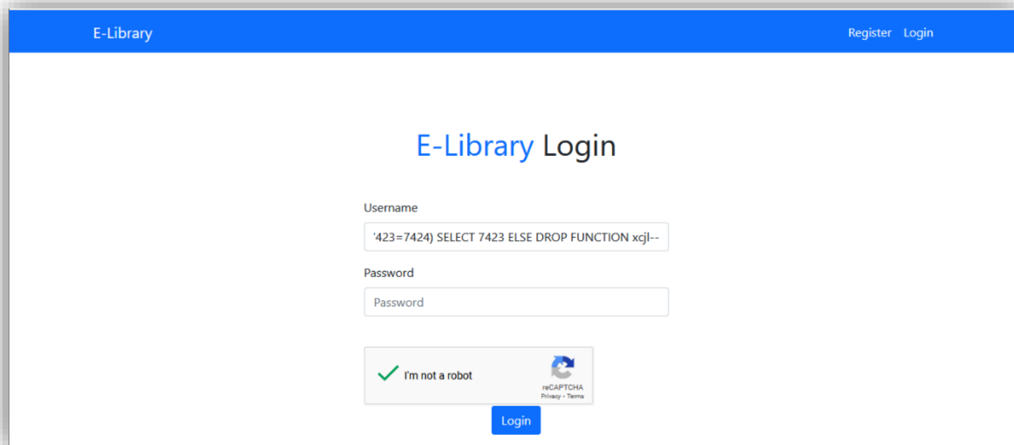


Figure 4.39 Piggy-Backed Attack from User Input

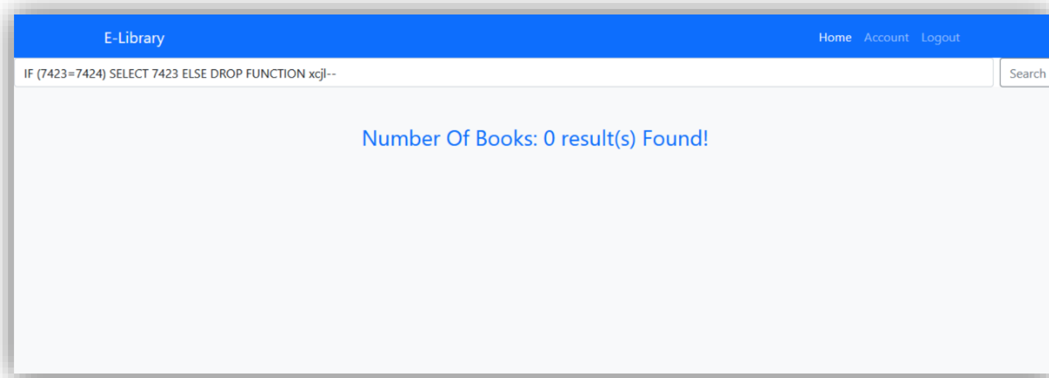


Figure 4.40 The Piggy-backed Attack from Search Input Box

Figure 4.39 shows the Piggy-backed attack from user input and Figure 4.40 shows Inference attack from search input box. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Piggy-Backed". The detection process of Piggy-Backed attack is shown in Figure 4.41

```
Input is : IF(7423=7424) SELECT 7423 ELSE DROP FUNCTION xcjl--
100 % matched pattern is ....IF(7423=7424) SELECT 7423 ELSE DROP FUNCTION xcjl--
Detected Pattern is : IF(7423=7424) SELECT 7423 ELSE DROP FUNCTION xcjl--
Pattern matched : 100.00 %
SQL Injection detected by 100 %
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Piggy-Backed Query
DEBUG: Making request: POST https://oauth2.googleapis.com/token
DEBUG: Resetting dropped connection: oauth2.googleapis.com
DEBUG: https://oauth2.googleapis.com:443 "POST /token HTTP/1.1" 200 None
DEBUG: Resetting dropped connection: sheets.googleapis.com
DEBUG: https://sheets.googleapis.com:443 "POST /v4/spreadsheets/1n8DoWm2WRQtMrQ19uf6almcUpThy47cUpzHOBz3CJ94/values/%27Sh
eet1%27:append?valueInputOption=RAW&includeValuesInResponse=False HTTP/1.1" 200 None
```

Figure 4.41 Piggy-Backed Attack Detection

4.5.5 Stored Procedure Attack

With this method, the attacker concentrates on the database system's stored procedures. Database engine can directly perform the execution of stored procedures. It is an exploitable section of code. For allowed or unauthorized clients, the stored process returns true or false results. For SQLIA, the attacker will include "; SHUTDOWN; --" with the secret key or login. The stored procedure attack will be generated by the SQL query mentioned in the following statement.

```
CREATE PROCEDURE DBO @userName varchar2, @pass varchar2, AS EXEC
("SELECT *
FROM user WHERE id= ' "+@userName+"' and password= ' "+@pass+"'"); GO
```

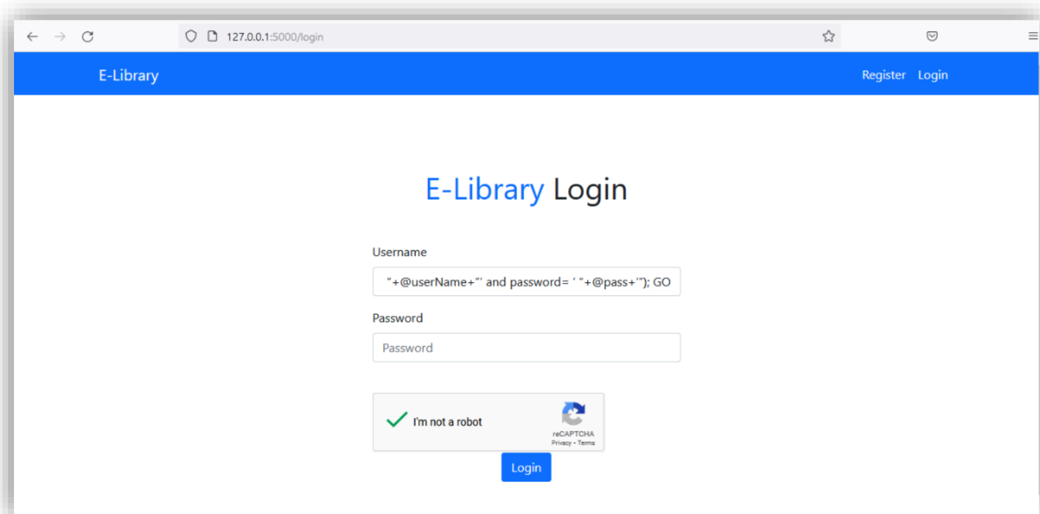


Figure 4.42 Stored Procedure Attack from User Input

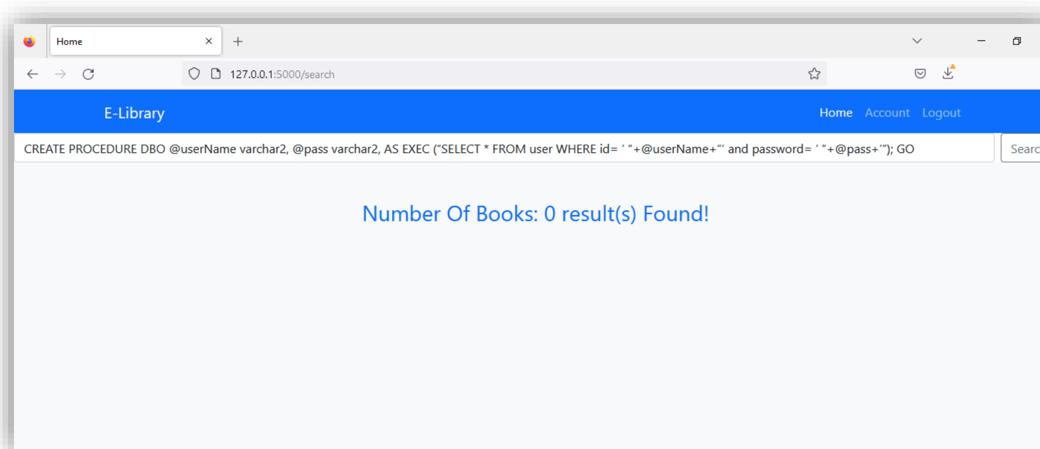


Figure 4.43 Stored Procedure Attack from Search Input Box

Figure 4.42 shows the Stored Procedure attack from user input and Figure 4.43 shows Stored Procedure attack from search input box. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Stored Procedure". The detection process of Stored Procedure attack is shown in Figure 4.44.

```
Input is : CREATE PROCEDURE DBO @userName varchar2, @pass varchar2, AS EXEC ("SELECT * FROM user WHERE id= ' "+@userName+
"' and password= ' "+@pass+'"); GO
Pattern is : SELECT *
Pattern matched : 100.00 %
SQL Injection detected!
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Stored Procedure
DEBUG: Resetting dropped connection: sheets.googleapis.com
DEBUG: https://sheets.googleapis.com:443 "POST /v4/spreadsheets/1n8DoWm2WRQtMrQ19uf6almcUpThy47cUpzHOBz3CJ94/values/%27Sh
eet1%27:append?valueInputOption=RAW&includeValuesInResponse=False HTTP/1.1" 200 None
```

Figure 4.44 Stored Procedure Attack Detection

4.5.6 Tautology

In order to make the SQL command evaluate as a true condition, such as (1=1) or (1=0), tautology-based attacks work by injecting code into one or more conditional SQL statement queries. This method is most frequently used to gain access to databases by avoiding authentication via user input. The tautology SQLIA is demonstrated by the SQL query in the following statement.

or 1=1--

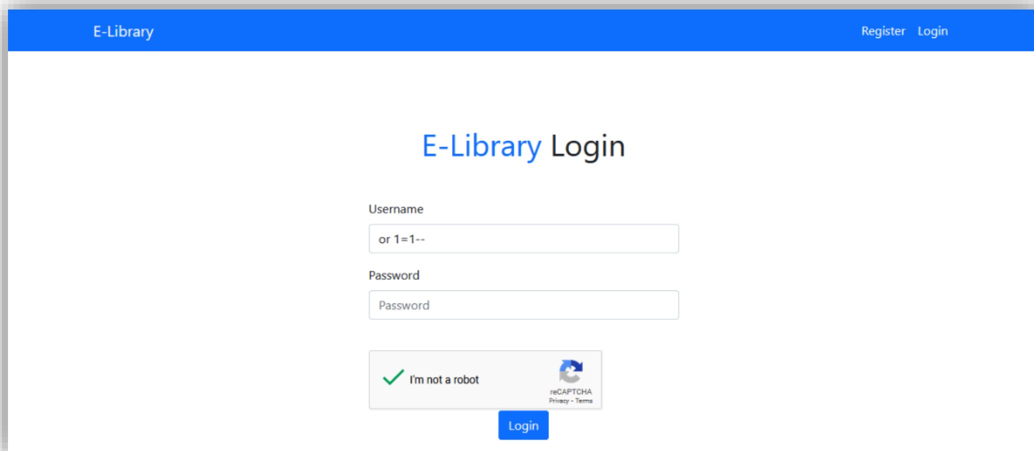


Figure 4.45 Tautology Attack from User Input

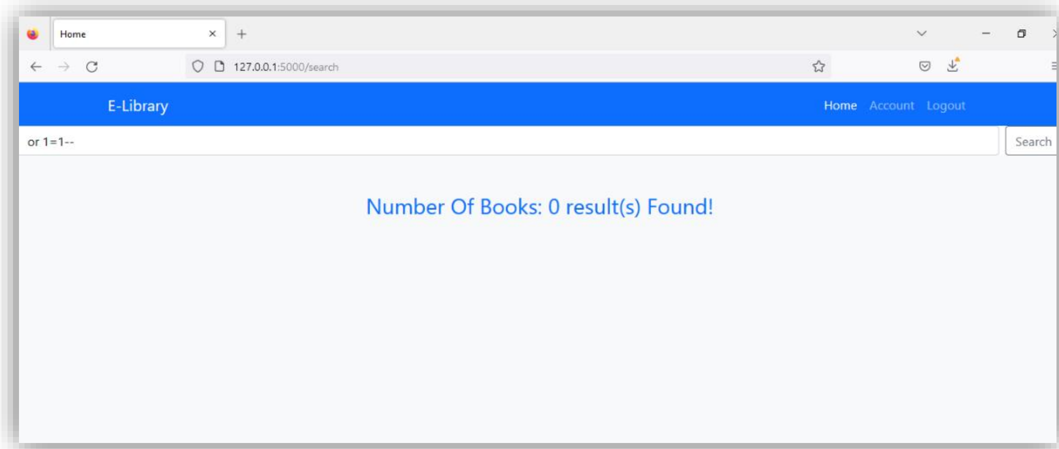


Figure 4.46 Tautology Attack from Search Input Box

Figure 4.45 shows the Tautology attack from user input and Figure 4.46 shows Tautology attack from search input box. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker's ip address and attack type is "Tautology". The detection process of Tautology attack is shown in Figure 4.47.

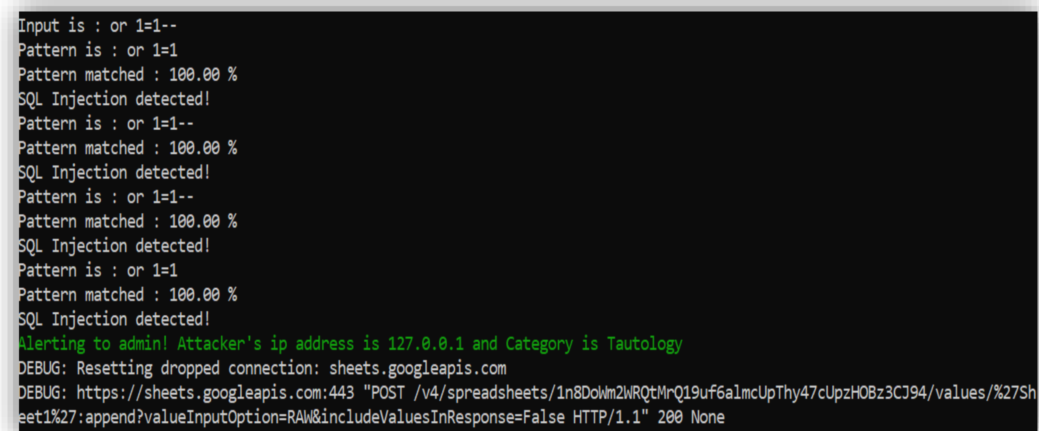


Figure 4.47 Tautology Attack Detection

4.5.7 Union Attack

Statement injection attack is also known as union query injection. In this attack, the attacker brings a new SQL statement to the previous one. As demonstrated in Figure 4.48, this attack can be carried out by entering a UNION query or a statement of the kind "; SQL statement >" into the weak factor. The system responds to this attack by

returning a record that combines the outcomes of the initial query with those of the injected query. The Union SQL injection is demonstrated by the following statement.

“Select * from users where id= “1’ union select \@@VERSION – 1”

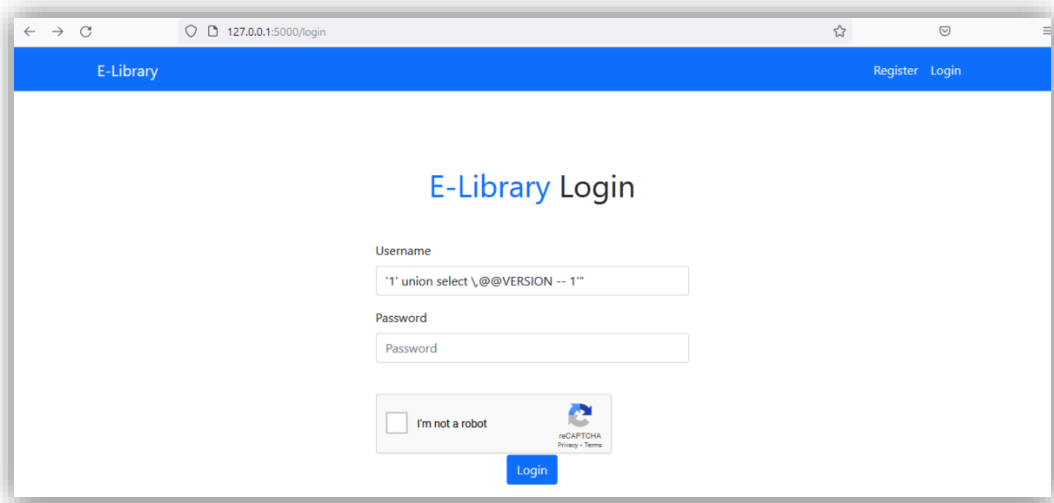


Figure 4.48 Union Attack from User Input

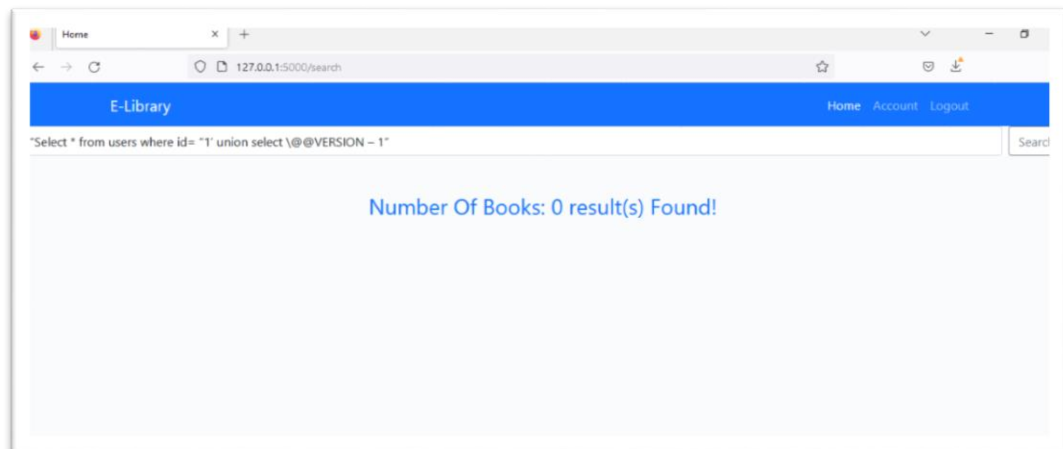


Figure 4.49 Union Attack from Search Input Box

Figure 4.48 shows the Union attack from user input and Figure 4.49 shows Union attack from search input box. If the user generated query exactly matches each pattern in DB patterns, the system can detect 100% to this pattern. Then, the system sends alert message to the admin with attacker’s ip address and attack type is “Union”. The detection process of Union attack is shown in Figure 4.50.

```
Input is : " select * from users where id  =  '1' union select \,@@VERSION -- 1'"
Pattern is : union select
Pattern matched : 91.67 %
SQL Injection detected!
Pattern is :  union select
Pattern matched : 84.62 %
SQL Injection detected!
100 % matched pattern is ...." select * from users where id  =  '1' union select \,@@VERSION -- 1'"
Detected Pattern is : " select * from users where id  =  '1' union select \,@@VERSION -- 1'"
Pattern matched : 100.00 %
SQL Injection detected by 100 %
Alerting to admin! Attacker's ip address is 127.0.0.1 and Category is Union Query
DEBUG: Resetting dropped connection: sheets.googleapis.com
DEBUG: https://sheets.googleapis.com:443 "POST /v4/spreadsheets/1n8DoWm2WRQtMrQ19uf6almcUpThy47cUpzHOBz3CJ94/values/%27Sheet1%27:append?valueInputOption=RAW&includeValuesInResponse=False HTTP/1.1" 200 None
```

Figure 4.50 Union Attack Detection

4.5.8 Attack Categories Evaluation

Table 4.4 illustrates the types of attacks and evaluation. It is assumed that if the matching percentage is greater than or equal to the threshold value, the probability of fully attacked is Yes. Otherwise, assuming is No.

Table 4.4. Attack Types Evaluation

Types of Attacks	Total	SQL Injection Attack	
		≥ threshold (Yes)	**< threshold (No)**
Tautology	90	81	9
Union	90	82	8
Logically incorrect	90	79	11
Piggy-Backed	90	80	10
Alternate Encodings	90	78	12
Inference	90	81	9
Stored Procedure	90	78	12
Total Tests	**630**	**559**	**71**

SQL Injection Attack			
Types of Attacks	**(Yes)**	**(No)**	**Probability**
Tautology	81	9	90/630
Union	82	8	90/630
Logically incorrect	79	11	90/630
Piggy-Backed	80	10	90/630
Alternate Encodings	81	9	90/630
Inference	78	12	90/630
Stored Procedure	5	1	90/630
	559/630	**71/630**	

Likelihood of *Yes* given Tautology attack is

P (*Tautology*|*Yes*) = 81/559 = 0.144
P(Tautology) = 90/630 = 0.143
P(Yes) = 559/630 = 0.887

$$P(Tautology|Yes) = \frac{P(Tautology|Yes) * P(Yes)}{P(Tautology)}$$

$$P(Yes|Tautology) = \frac{0.144 * 0.887}{0.143}$$

$$P(Yes|Tautology) = 0.893$$

Likelihood of *No* given Tautology attack is

P (*Tautology*|*No*) = 9/71 = 0.126
P(Tautology) = 90/630 = 0.143
P(No) = 71/630 = 0.113

$$P(No|Tautology) = \frac{P(Tautology|No) * P(No)}{P(Tautology)}$$

$$P(No|Tautology) = \frac{0.126 * 0.113}{0.143}$$

$$P(No|Tautology) = 0.099$$

Likelihood of *Yes* given Union attack is

P (*Union*|*Yes*) = 82/559 = 0.146
P(Union) = 90/630 = 0.143
P(Yes) = 559/630 = 0.887

$$P(Yes|Union) = \frac{P(\text{Union}|Yes) * P(Yes)}{P(Union)}$$

$$P(Yes|Union) = \frac{0.146 * 0.887}{0.143}$$

$$P(Yes|\text{Union}) = 0.905$$

Likelihood of *No* given Union attack is

P (*Union* |*No*) = 8/71 = 0.113
P(Union) = 90/630 = 0.143
P(No) = 71/630 = 0.113

$$P(No|\text{Union}) = \frac{P(Union|No) * P(No)}{P(Union)}$$

$$P(No|\text{Union}) = \frac{0.113 * 0.113}{0.143}$$

$$P(No|\text{Union}) = 0.089$$

Likelihood of *Yes* given Logically incorrect attack is

P (*Logically*|*Yes*) = 79/559= 0.141
P (Logically) = 90/630 = 0.143
P(Yes) = 559/630 = 0.887

$$P(Yes|\text{Logically }) = \frac{P(Logically|Yes) * P(Yes)}{P(Logically)}$$

$$P(Yes|Logically) = \frac{0.141 * 0.887}{0.143}$$

$$P(Yes|Logically) = 0.875$$

Likelihood of *No* Logically incorrect attack is

P (*Logically*|*No*) = 11/71 = 0.154
P (Logically) = 90/630 = 0.143
P(No) = 71/630 = 0.113

$$P(No|Logically) = \frac{P(Logically|No) * P(No)}{P(Logically)}$$

$$P(No|\text{Logically}) = \frac{0.154 * 0.113}{0.143}$$

$$P(No|\text{Logically}) = 0.121$$

Likelihood of *Yes* given Piggy-Backed attack is

P ($Piggy|Yes$) = 80/559 = 0.143
P(Piggy) = 90/559= 0.143
P(Yes) = 559/630 = 0.887

$$P(Yes|Piggy) = \frac{P(Piggy|Yes) * P(Yes)}{P(Piggy)}$$

$$P(Yes|Piggy) = \frac{0.143 * 0.887}{0.143}$$

$$P(Yes|Piggy) = 0.887$$

Likelihood of *No* given Piggy-Backed is

P ($Piggy|No$) = 10/71 = 0.140
P (Piggy) = 90/630 = 0.143
P (No) = 71/630= 0.113

$$P(No|Piggy) = \frac{P(Piggy|No) * P(No)}{P(Piggy)}$$

$$P(No|Piggy) = \frac{0.140 * 0.113}{0.143}$$

$$P(No|Piggy) = 0.110$$

Likelihood of *Yes* given Alternate Encoding attack is

P ($Alternate|Yes$) = 78/559= 0.139
P (Alternate) = 90/630 = 0.143
P(Yes) = 559/630 = 0.887

$$P(Yes|Alternate\ ) = \frac{P(Alternate|Yes) * P(Yes)}{P(Alternate\ )}$$

$$P(Yes|Alternate\ ) = \frac{0.139 * 0.887}{0.143}$$

$$P(Yes|Alternate\ ) = 0.862$$

Likelihood of *No* given Alternate Encoding attack is

P ($Alternate$ |*No*) = 12/71 = 0.169
P (Alternate) = 90/630 = 0.143
P(No) = 71/630= 0.113

$$P(No|Alternate\ ) = \frac{P(Alternate\ |No) * P(No)}{P(Alternate\ )}$$

$$P(No|Alternate\ ) = \frac{0.169 * 0.113}{0.143}$$

$$P(No|Alternate\ ) = 0.133$$

Likelihood of *Yes* given Inference attack is

P (*Inference*|*Yes*) = 81/559= 0.144
P(Inference) = 90/630= 0.143
P(Yes) =559/630 = 0.887

$$P(Yes|Inference) = \frac{P(Inference|Yes) * P(Yes)}{P(Inference)}$$

$$P(Yes|Inference) = \frac{0.144 * 0.887}{0.143}$$

$$P(Yes|Inference) = 0.893$$

Likelihood of *No* given Inference is

P (*Inference* |No) = 9/71 = 0.143
P (Inference) = 90/630 = 0.143
P (No) = 71/630 = 0.113

$$P(No|\text{Inference}) = \frac{P(Inference|No) * P(No)}{P(Inference)}$$

$$P(No|Inference) = \frac{0.126 * 0.113}{0.143}$$

$$P(No|Inference) = 0.099$$

Likelihood of *Yes* given Stored Procedure attack is

$P\ (Stored\ Procedure|Yes) = 78/559 = 0.139$
P(Stored Procedure) = 90/630 = 0.143
P(Yes) = 559/630 = 0.887

$$P(Yes|Stored\ Procedure\ ) = \frac{P(Stored\ Procedure\ |Yes) * P(Yes)}{P(Stored\ Procedure\ )}$$

$$P(Yes|Stored\ Procedure\ ) = \frac{0.139 * 0.887}{0.143}$$

$$P(Yes|Stored\ Procedure\ ) = 0.862$$

Likelihood of *No* given Inference is

$P\ (Stored\ Procedure\ |No) = 12/71 = 0.169$
P (Stored Procedure) = 90/630 = 0.143
P (No) = 71/630 = 0.113

$$P(No|Stored\ Procedure\ ) = \frac{P(Stored\ Procedure|No) * P(No)}{P(Stored\ Procedure\ )}$$

$$P(No|Stored\ Procedure) = \frac{0.169 * 0.113}{0.143}$$

$$P(No|Stored\ Procedure) = 0.133$$

Table 4.5. Performance Evaluation

Types of Attacks	Total Attacks	Probability of Yes	Probability of No
Tautology	90	0.893	0.099
Union	90	0.905	0.089
Logically incorrect	90	0.875	0.121
Piggy-Backed	90	0.887	0.110
Alternate Encodings	90	0.862	0.133
Inference	90	0.893	0.099
Stored Procedure	90	0.862	0.133
Total	630	6.177	0.784

Table 4.5 show the performance evaluation of the proposed library system in terms of accuracy.

$$P(Yes|Attacks\ ) = \frac{P(Attacks\ |Yes) * P(Yes)}{P(Attacks)}$$

$$= \frac{101.991 * 6.177}{630}$$

Accuracy = 0.999%

The experimental results show that the proposed system achieves above 100% detection rate in the input injected SQL statements for seven common types of SQL attack. The implementation of the proposed technique effectively detects and blocks all types of SQL Injection attacks. Therefore, the proposed library system can identify and detect SQL injections, according to the experimental results various injection attacks.

CHAPTER 5

CONCLUSION

SQL injection attacks and web-based attacks are major issues in the security of financial, health, and other critical data, and this problem only increases in importance to protect the malicious queries. This paper proposes a library system that can detect against 6 common types of SQL injection attacks when log-in authentication stage. In addition, it can detect and blocks code SQL injection vulnerabilities effectively using modified pattern matching technique. The experimental results provide that the proposed algorithm handles malicious queries effectively matching and prevents unauthenticated users for library system.

5.1 Advantages of the System

One of the biggest threats to web-based library system is SQL Injection. All user data stored in the database is exposed by SQL injection, making it possible for it to be misused or sold on the black market. The drawbacks of previously implemented SQL injection detection system is that they can only be able to detect those that they have seen before or have been trained on. In contrast, the proposed system can be able to tell whether the data being entered has been SQL injected or not by investigating patterns in the input.

The advantages of the proposed system are as follows.

(i) This library system allows effective and high detection of SQL injection attacks.

(ii) The present detection technique makes sure that alleviate in confidential data being deleted, lost or stolen.

(iii) The system can effectively identify unwanted access to systems or accounts, leading to eventual compromise of specific devices or file servers.

(iv) It has the ability to recognize and reject SQL injection attacks that use malicious code to trick the system database into revealing data.

(v) It reduces the probability of a high-risk compromise having an effect on the library system's features of authentication and authorization, as well as the confidentiality and integrity of the information.

5.2 Limitation of the System

The proposed system can detect and block code SQL injection vulnerabilities effectively. However, the system has some limitations. As attack types increase, new SQL injection methods and tools are continually being developed. Every potential SQL injection query cannot be covered by the system's specialized signatures. White box pen testing can be done with the proposed system. In contrast to black-box pen testing, which may not be viable, white box pen testing can be performed successfully by simulating a focused attack on a specific system using as many attack paths as possible. Therefore, the proposed system can only update the existing static pattern list if a new absolutely attack patterns has been attacked.

5.3 Further Extensions

In this thesis, the proposed system for the future work is considered to develop the automatic SQL injection detection system to identify potential vulnerabilities. Validating user inputs is a frequent initial step in mitigating SQL injection attacks. First, decide which SQL statements are absolutely necessary, then create a whitelist of all legitimate SQL statements, leaving invalid statements out of the query. This procedure is often referred to as query redesign or input validation. Therefore, input validation process will be considered in the future research work.

AUTHORS PUBLICATION

[1] Mar Mar Than, Nwe Zin Oo, Tin Thein Thwel, "SQL Injection Detection Using Pattern Matching Algorithm for Library System", Local Conference on Parallel and Soft Computing (PSC), UCSY, Yangon, Myanmar, 2022.

REFERENCES

[1] Appiah B., Opoku-Mensah E. and Qin Z., "SQL injection attack detection using fingerprints and pattern matching technique," In Proceeding of 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 583-587, doi: 10.1109/ICSESS.2017.8342983.

[2] Blind SQL Injection. https://owasp.org/www-community/attacks/Blind_SQL_Injection

[3] George, T. K., Jacob, K. P. and James, R. K., "Token based Detection and Neural Network based Reconstruction framework against code injection vulnerabilities", International Journal of Information and Application, vol.41, 2018. Doi: 10.1016/j.jisa.2018.05.005

[4] Hasan, M., Balbahaith, Z., Tarique, M., "Detection of SQL Injection Attacks: A Machine Learning Approach", In Proceeding of IEEE International Conference on Electrical and Computing Technologies and Applications (ICECTA),2019.

[5] Javali, P., Chougule, S.V., "SQL Injection Detection and Prevention using Pattern Matching Algorithm", International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 6, June 2016.

[6] Joshi A. and Geetha V., "SQL Injection detection using machine learning," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 1111-1115, doi: 10.1109/ICCICCT.2014.6993127.

[7] Kar, D., Panigrahi, S. and Sundararajan, S. "SQLiGoT: Detecting SQL Injection Attacks using Graph of Tokens and SVM, Computers & Security, 2016. Doi: 10.1016/j.cose.2016.04.005.

[8] Lee, I., Jeong, S., Yeo, S. and Moon, J., "A novel method for SQL injection attack detection based on removing SQL query attribute values". International Journal of Mathematical and Computer Modelling, Vol. 55, Issues 1–2, January 2012, pp. 58-68.

[9] Patel, N., Shekokar, N., "Implementation of pattern matching algorithm to defend SQLIA", In Proceeding of International Conference on Advanced Computing Technologies and Applications (ICACTA), 2015.

[10] Prabakar, M. A. Keyan, M. K. and Marimuthu, K., "An Efficient Technique for Preventing Sql Injection Attack Using Pattern Matching Algorithm". In Proceeding of IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN 2013), pp.503-506.

[11] SQLi.https://www.acunetix.com/blog/articles/sqli-part-4-in-band-sqliclassic-sqli/

[12] SQLi.https://www.acunetix.com/blog/articles/sqli-part-6-out-of-band-sqli/

[13] Tang, P., Qiu, W., Huang, Z., "“Detection of SQL Injection Based on Artificial Neural Network”, Journal Knowledge-Based Systems, vol.190,2020. Doi: 10.1016/j.knosys.2020.105528.

[14] Zhang, K., “A Machine Learning based Approach to Identify SQL Injection Vulnerabilities”. In Proceeding of 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1286-1288, pp.75-91,2019.