A Novel Approach to Foil SQL Injection Attack at Login Phase

Shivraj Sharma
PG Scholar, Department of Computer Science & Engineering, Shekhawati Institute of Engineering & Technology, Sikar, Rajasthan, India

Abstract- **SQL injection techniques have been recognized as the most exhaustively used security threats for the web applications. According to a survey performed by the Open Web Application Security Project (OWASP) in 2017 SQL Injection is at the first place of the most crucial 10 critical web based application security attacks or risks. A wrongdoer can get unwarranted retrieval of data from or to the database beneath the web application. These attacks have various kinds and the intention to launch a particular attack also differs from the other attack. In order to shield the database from any SQL injection assault the close examination of these SQL queries is necessary. SQL Injection Attacks have been explored at various types of web based applications. However, one thing has been common in all these applications is that the malicious user requires some sort of input method by which he can send the crafted SQL query to the database. In this paper a special case of SQL Injection Attack has been explored in great detail. SQL Injection Attack at login phase is a very general and mostly explored SQL Injection Attack forever. A novel approach to detect and prevent SQL Injection Attack at login phase has been proposed. This approach is a mixture of defensive coding and Secure Hash Algorithm (SHA) based OTP generation process.**

Index Terms- **Attack, Injection, SQL, Vulnerability, Tautology, Hash Function.**

I. INTRODUCTION

With the upgradation to fourth generation data services the internet usage has augmented drastically. Now a days web based applications has evolved in such a way that they are playing a vital role in everyone's day to day life. These web applications are providing the facilities of a bank, newspaper, shopping mall, railway's reservation counter and many other day to day services that a person may require. The usage of these web applications is increasing exponentially because they are easy to use, available at the user's end and most importantly they save enormous time of the users. All the information auxiliary to these web applications is stored at the backed database beneath the applications.

The backend database is connected to the web application. The information can be retrieved at the user's demand or added, edited or deleted. SQL is the only language that is used to set the communication with the underneath database. A vulnerable web application gives way to illicit access to the underneath database by the application of SQLIA. SQLIAs are launched by the users through specially managed inputs that are supplied to the web application. Therefore it is always required to detect the queries that can launch SQLIA. Work presented here aims for the same.

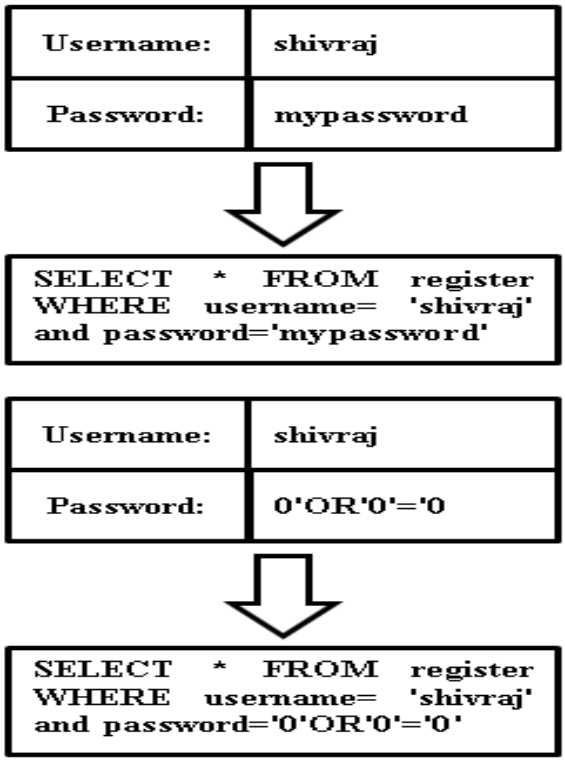

Fig. 1 Demonstration of SQL Injection

SQL Injection is a one of the type of injections or attacks in the web based applications, in which the attacker provides Structured Query Language code to a user input box or the entry box of a web form to gain the unauthorized and also the unlimited access. The attacker's input is also transmitted into the SQL based query in such a way that it will form an SQL code [2]. Figure 1 shows an example of SQLIA[1] at login phase. In this example it is shown that how username and password inputs are converted in a dynamic SQL query that is sent to the underneath database. When the user provides specially crafted password to the login form the dynamic query can be managed to get unauthorized access to the database.

According to a survey of performed by Open Web Application Security Project (OWASP) in the year 2017 the SQL Injection is almost at the first place of the top of the 10 most critical of the web based application security risks [11].

Key concepts of SQL Injection:-

1. SQL injection is a product powerlessness that happens when data entered by users is sent to the SQL translator as a piece of a SQL query [3].
2. Attackers give uncommonly created input data to the SQL translator and trap the mediator to execute unintended charges [3].
3. Attackers use this powerlessness by giving extraordinarily created input data to the SQL translator in such a way, to the point that the mediator can't recognize the planned orders and the assailant's exceptionally made data. The translator is deceived into executing unintended orders [3].
4. SQL injection abuses security vulnerabilities at the database layer. By abusing the SQL injection defect, aggressors can make, read, change, or erase delicate data [3].

The rest of the paper is organized as follows; Section II defines SQL injection types that are pertinent to our present discussion. Section III discussed the related work in the detection and prevention of SQLIA at login phase. Section IV is a case study of a particular type of SQLIA i.e. SQLIA at login phase. Section IV goes deep into demonstration of SQLIA at login phase, then it further presents the proposed work to foil SQLIA at login phase. Section V concludes the discussion with a future scope.

II. SQL INJECTION TYPES

2.1 Tautologies

The main objectives of tautology- based attacks is to inject code in the conditional based statements so that they are always evaluated as true [6]. It is done, by simply making the where clause always true for every query, which results in bypassing the condition inside the SQL statement, for instance

SELECT * FROM User_info WHERE Username='RAHUL' and Password='12500'

In this query attacker can inject OR'1'='1' The resulting query will be:

SELECT * FROM User_Info WHERE Username=''OR'1'='1'-- and Password='Idontcare'

This enables the attacker to get all the records from the table. So in this way username and password of all the stored users in the database can be extracted.

Tautology- based SQL injection techniques are used by maximum hackers to bypass the authentication phase by just adding "--" inline comment, which makes the rest of the SQL command as comment [7].

2.2 Logically Incorrect Queries

The aim of the attacker is to gather all possible information about the structure and the schema of the tables and their fields inside the database. This belongs to the SQL manipulation attack where the error message generated by the database provides the attacker with an advantage. The working is quite simple. Here, injected wrong or incorrect SQL statement will generate some error message from databases that will provide the attacker the necessary information. Due to incongruous error handling, some internal database error message, specific to that particular database version, can be shown to the attacker because of which vital information about the database structure is revealed to the attackers which help him to conduct more exact attack that will have more impact on its target website [7].

We can limit the output errors or use customized error messages to avoid information leakage. The error that is shown will be devoid of any specific useful information.

Some specific examples that falls under this category are:-

SELECT * FROM User_unit1 WHERE Username=abc HAVING '1'='1'—and Password='123456'

Error generated:

"Column 'User_unit1.UserID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause"
This error message displays the table with column name 'UserInfo.Username'
In this way all column names of the table can be extracted. We can have two types of error returns logical and syntactical.
The name of the columns are revealed by the logical errors which fetches the columns or table names while syntactical errors informs which parameters are vulnerable for an injection attack.

2.3 Union Query
In this type of SQLIA, which is injected based query which once joined with the safe query using the keyword UNION in order to get information one which related to the other tables from the concepts or the application [4] And also this type of attack is surely used in order to bypass the authentication process and also to fetch or extract data by inserting the UNION operator to the normal query. In the following example the second query is malicious because the text "--" is disregarded as it becomes comment for the SQL parser. However, if the query is executed the attacker receives the credit card information [10].
SELECT * FROM accounts WHERE id='212' UNION SELECT * FROM credit_card WHERE user='admin'—and pass='pass' [10].
2.4 Piggy- Backed Queries
This attack also inserts the malicious SQL queries into the normal SQL query. It is possible because many SQL queries can be processed if the operator ";" is added after each query. Following query is an example. Note that the operator ";" is inserted at the end of query.
SELECT * FROM user WHERE id='admin' AND password='1234';DROP TABLE user;--;
The result of query 3 is the deletion of the user table [9].

III.RELATED WORK

A. Preventing SQL Injection Attack in Web Application[3]
Three components of this technique are: User Login Interface, SQL Query Component and User Account Table. This technique can only protect authentication mechanism but other types of SQLIAs cannot be handled using this technique [3].

B. Blend of Parse Tree Validation Method and Code Conversion Method[6]
In this method user input is parsed and checked whether it is vulnerable, if there is any chance of vulnerability present then code conversion will be applied over that input [6].

C. SQL Filtering: An Effective Technique to Prevent SQL Injection Attack[8]
This technique is the integration of two techniques namely SQLIPA technique and use of proxy server to secure the database against SQLIAs [8].

IV. A CASE STUDY: SQLIA AT LOGIN PHASE

4.1 Demonstration
A new application has been designed in .NET architecture. A Login page has been designed to demonstrate SQLIA at login phase. This page takes input from user. Username and password are supplied by the user. These username and password are stored in the register table of the underlying database. An attacker supply specially crafted username and password. These specially crafted username and password become part of the dynamic SQL query and is fired to the database. A malicious user would like to gain control over database using specially crafted password.

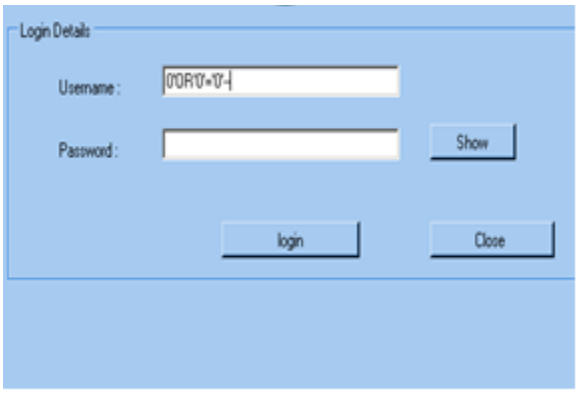

Fig. 2 Demonstration of SQLIA at Login Phase
In the above screen shot the SQLIA at login phase has been demonstrated. A specially moulded Username i.e. 0'OR'0'='0'-- is supplied by a malicious user. In this special case no password is needed. So when user press login button the following page is opened.

Fig. 3 Main Screen of Vulnerable Application

In this way the malicious user gets through the security check at login phase. In the proposed application several username and password combinations have been identified for which the malicious user can get unauthorized access to the database of the application.

When user supplies the username and password and press login button following dynamic sql query is generated in response of the submission:

SELECT * FROM register WHERE username='0'OR'0'='0'--' and password='anything';

This SQL query is sent to the underlying database and the malicious user gets through. the Password field has nothing to do in this query because it has no role to play due to comment operator.

4.2 Proposed Solution

This section describes the algorithms which are involved in the proposed dissertation. In the proposed solution four algorithms have been proposed to design an application:. These are,

1. Algorithm for Registration Process
2. Algorithm for Login Process
3. Algorithm for Validating Password
4. Algorithm for OTP Generation

4.2.1 Algorithm for Registration Process

The algorithm of the registration process is described in the following steps:

Step 1: Input the photo/image, username and email ID which is to be used for the secure identification.

Step 2: Fill the details provided in the registration form.

Step 3: Validate the Email ID and username in the database in order to cross validate whether the registration or the user already exists with the same email id or the user name.

Step 4: If the Details already exist then stop and provide the new details otherwise the photo provided as an input is validated from the user name database.

Step 5: The details will be saved in the database.

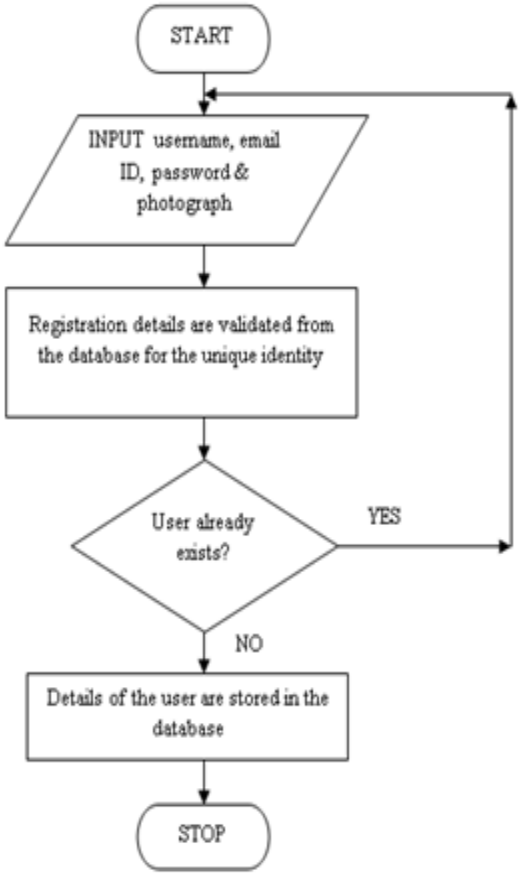

Fig. 4 Flowchart for Registration Process

4.2.2 Algorithm for Login Process

The algorithm of the login process is described in the following steps:

Step 1: Input the photo/image which is to be used for the secure identification.

Step 2: Fill the details provided in the Login Form including the username specified at the time of the registration.

Step 3: SHA code is generated and sent to registered email id.

Step 4: If the Details are OK and password matched Second page of secure query will get displayed

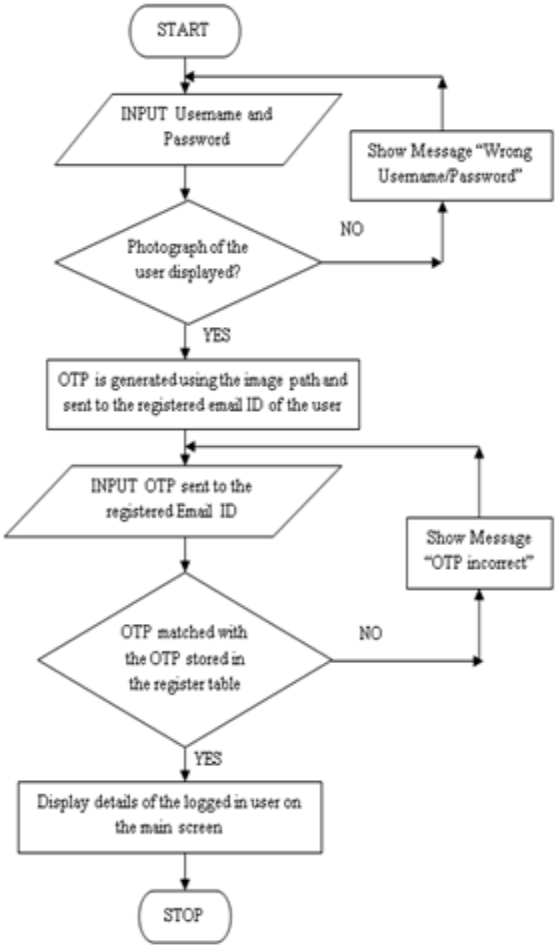

Fig. 5 Flowchart for Login Process

4.2.3 Algorithm for Validating Password

The algorithm for the validation process of password is described in the following steps:

Step 1: Take input of password and convert it into capital letters.

Step 2: Find the location of ‘ in the password and store its index into variable pos

Step 3: In case password does not contain ‘ validate username and password from database and exit

Step 4: In case password has ‘ find a substring of length 2 from left side and store it into variable s1

Step 5: Find the location of ‘ in the password from right side and store its index into variable pos2

Step 6: Find another substring of length 1 from right side and store it into variable s2

Step 7: In case s1 contains “OR” and s2 contains”=” display message “Wrong username/ password” and go to step 1

Step 8: Otherwise validate username and password from register table and exit.

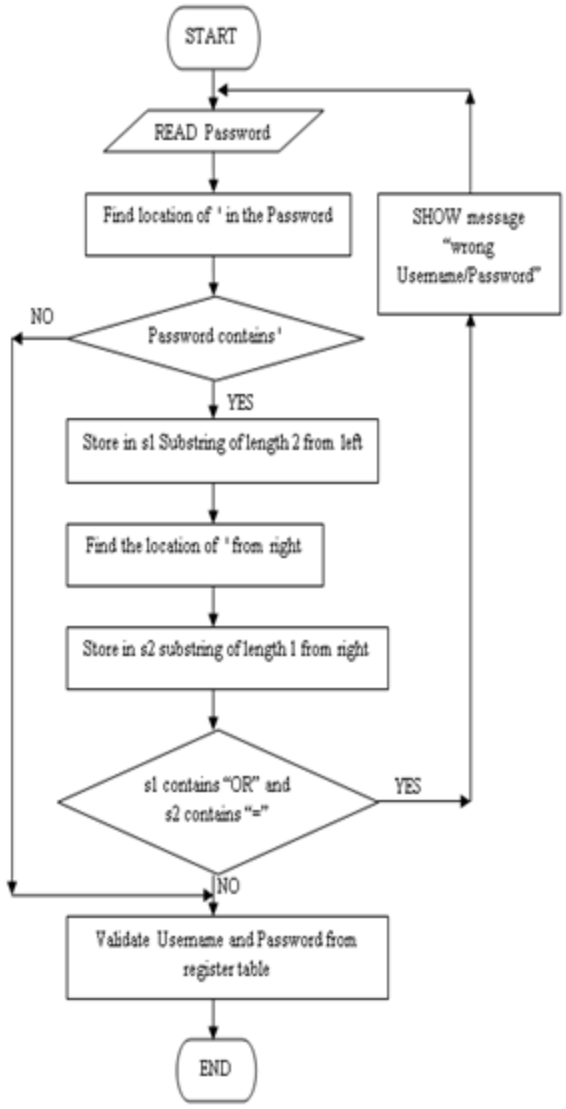

Fig. 6 Flowchart for Validation Process of Password

4.2.4 Algorithm for OTP Generation

The algorithm for OTP generation process is described in the following steps:

Step 1: Store image path and email ID into variables fname and emailis respectively.

Step 2: Generate 40 character long hash code by supplying fname as parameter to SHA1HashStringForUTF8String(String s) function and store hash code into a string variable otp

Step 3: Convert string variable otp into a character array otparray[] of 40 characters

Step 4: Generate 8 random 32-bit log integers and store them into variables r1,r2,r3,……..,r8 respectively

Step 5: Add otparray[0] to r1 and convert resultant into character, Add otparray[5] to r2 and convert resultant into character, Add otparray[10] to r3 and convert resultant into character, Add otparray[15] to r4 and convert resultant into character, Add otparray[20] to r5 and convert resultant into character, Add otparray[25] to r6 and convert resultant into character, Add otparray[30] to r7 and convert resultant into character, Add otparray[35] to r81 and convert resultant into character

Step 6: Append all the eight charaters obtained in the previous step to generate 8 character otp and store this otp into a string variable notp

Step 7: Display Image of the registered user stored at the path fname using which hash code/otp was generated

Step 8: Email this otp to the registered email id of the user stored at variable emailed.

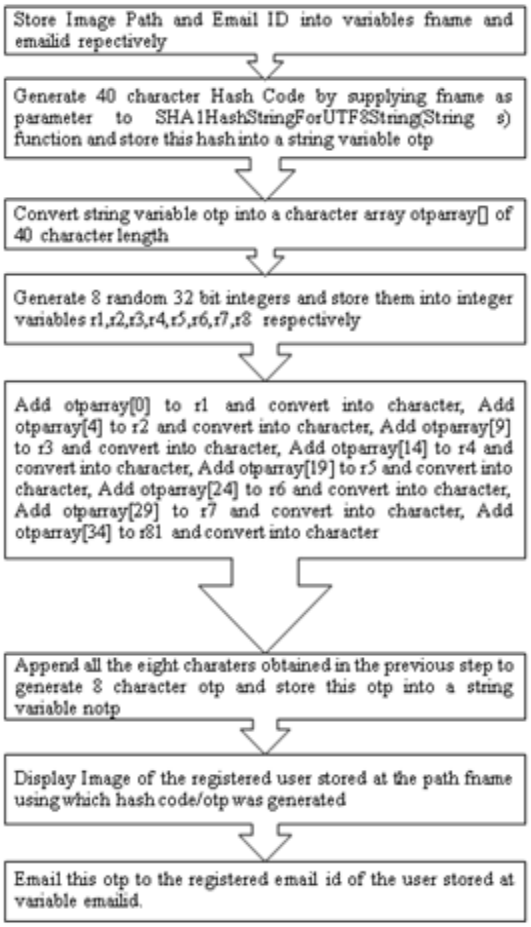

Fig. 7 Flowchart for OTP Generation Process

4.3 Implementation

The above proposed algorithms have been implemented in C# programming language in .NET framework. A new login form has been generated for the secure login. In this login form two strategies have been used: defensive coding for identification for the SQLIA and SHA-1 based OTP generation for the authentication of the user credentials. Following form is a screen shot of the newly proposed login form:

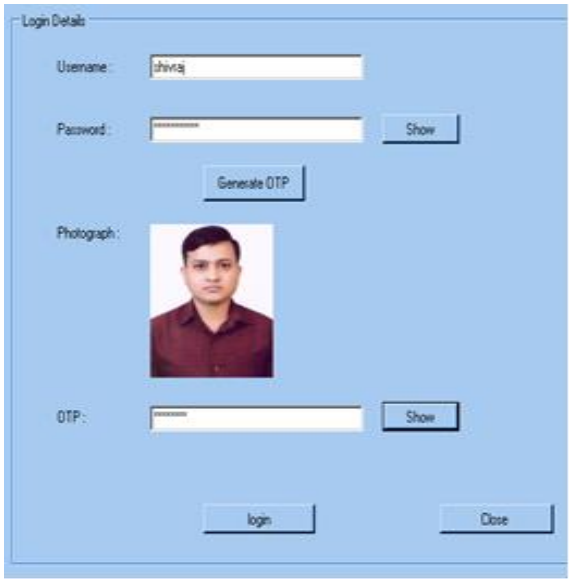

Fig. 8 Proposed Login Form

In this form when the user enters the username and password and press generate OTP button, an OTP is generated and photograph of the particular user is displayed on the screen. This photograph is actually used in the SHA code generation. OTP is sent to the email address of the particular user. Once the OTP is entered by the user the login permission is granted. This newly proposed login form eliminates all the possibility of the SQLIA at login phase.

4.4 Result Analysis

In this section all the possible SQL Injection username and passwords have been collected and presented in a tabular form. All these combinations are used to launch a SQLIA at login phase. These combinations are successfully used to login in the vulnerable login page. However, these combinations have been identified and our new login page does not allow them to be used successfully.

In the proposed login form when a user tries to launch a SQLIA, It displays a message that the username or password is wrong. The following

image is a screenshot of the SQLIA at the login page of the proposed application:

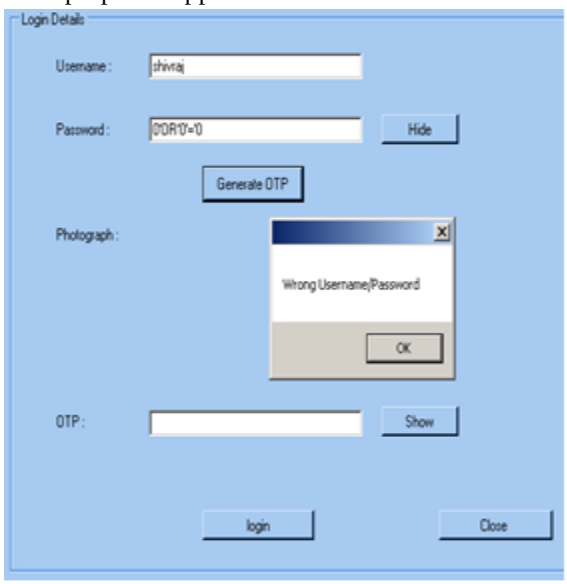

Fig. 9 An Attempt of SQLIA at the Proposed Login Form

Table I. Comparison of the vulnerable and proposed Login approach

Sr. No.	Username	Password	Result Description	
			Non-Secure Mode	Secure Mode
1	0'OR'0'='0'--	-	Login Successful	Login Failed
2	'OR'x'='x'--	-	Login Successful	Login Failed
3	shivraj	'OR''='	Login Successful	Login Failed
4	shivraj	0'OR''='	Login Successful	Login Failed
5	shivraj	x'OR'x'='x	Login Successful	Login Failed
6	-	'OR''='--	Login Successful	Login Failed

V.CONCLUSION

In this paper a novel approach to detect and prevent SQLIA at login phase has been closely analyzed. Four algorithms to carry out the detection and prevention of SQLIA at login phase have been proposed. A new application has been designed in .NET framework. This application has a linked database. Two separate login pages have been designed. One page demonstrates the vulnerability of the application that can be explored by a malicious user to carry out a SQLIA. Our proposed work, however, overcomes these vulnerabilities and has a demonstration at the second login page.

In the future work, this designed application will be augmented to detect all the possible SQL Injection attack queries. A sample database would be designed and feature based approach would be evolved to carry out the main dissertation work.

REFERENCES

[1] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures", IEEE, 2006

[2] Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan, "A Survey on SQL Injection: Vulnerabilities, Attacks and Prevention Techniques", IEEE 15th International Symposium on Consumer Electronics (ISCE), vol. 11, pp. 468-471, 14-17 June 2011

[3] Mihir Gandhi, Jwalant Baria, "SQL Injection Attacks in Web Application", International Journal of Soft Computing and Engineering (IJSCE), vol. 2, Issue 6, pp. 189-191, January 2013

[4] Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan, "A Detailed Survey on various aspects of SQL Injection in Web Applications: Vulnerabilities, Innovative Attacks and Remedies", International Journal of Communication Networks and Information Security (IJCNIS), vol. 5, no. 2, pp. 80-92, August 2013

[5] Chandershekhar Sharma, Dr. S. C. Jain, "SQL Injection Attacks on Web Applications", International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), vol. 4, Issue 3, pp. 1268-1272, March 2014

[6] Ashish John, "SQL Injection Prevention by adaptive algorithm", IOSR Journal of Computer Engineering, vol. 17, pp. 19-24, January 2015.

[7] Pankajdeep Kaur, Kanwal Preet Kour, "SQL Injection: Study and Augmentation", IEEE International Conference on Signal Processing, Computing and Control (ISPCC), pp. 102-107, 24-26 September 2015

[8] Rhythm Dubey, Himanshu Gupta, "SQL Filtering: An Effective Technique to Prevent SQL Injection Attack", IEEE 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 312-317, 7-9 September 2016

[9] Krit Kamtuo, Chitsutha Soomlek, "Machine Learning for SQL Injection Prevention on Server-Side Scripting", IEEE International Computer Science and Engineering Conference(ICSEC), pp. 1-6, 14-17 December 2016

[10] Dr. Ahmad Ghafarian, "A Hybrid Method for Detection and Prevention of SQL Injection Attacks", IEEE Computing Conference, pp. 833-838, 18-20 July 2017

[11] OWASP Top 10 - 2017, "OWASP Top 10 Most Critical Web Application Security Risks", pdf of the document is available at https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf [last accessed on 21 July 2018]