

Prevention of SQL Injection Attacks using RC4 and Blowfish Encryption Techniques

Sonakshi
M. Tech. Student
DCSA, KUK, Haryana

Rakesh Kumar
Professor
DCSA, KUK, Haryana

Girdhar Gopal
Assistant Professor
DCSA, KUK, Haryana

Abstract: SQL Injection Attacks (SQLIAs) are emerged nowadays as one of the most serious threats to the security of database-driven web applications. SQL injection attacks are one of the most critical vulnerabilities in web applications that software developers must address. The vulnerabilities can be harmful because they allow an attacker to access the database underlying an application. Using SQLIAs, an attacker may be able to read, modify, or even delete database information. In many cases, this information is sensitive and its loss can lead to problems such as identity theft and fraud. In this paper, the problem of SQL injection attack is addressed using three different prevention mechanisms. The first technique allows single word inputs only by matching malicious symbols with the list maintained which can prevent almost all types of SQLIAs. The second one is the well-known parameterized query used to handle these types of attacks & the last technique makes use of RC4 and blowfish encryption mechanism instead of AES which will drastically improve the performance by encrypting the confidential fields in much less time as compared to AES encryption method.

Keywords-Security breach; SQL injection; SQL injection detection; SQL injection prevention; Web application attacks; Web application Vulnerabilities

I. INTRODUCTION

Internet is the huge basket of information which is widely used for communication. Sometimes, this communication channel becomes insecure for exchanging information and hence, the principles of information security are violated. The database is the key element of most of the web applications as it contains important assets [1]. The attacker can get unauthorized access to the database with the help of crafted query. The security breach exposes the database worldwide which affects the reputation, liability and the clients of organizations [2]. Applications are vulnerable to new security threats. According to OWASP [3], SQL injection is ranked as number one in top 10 threats for web application security in 2013. These attacks not only make the attacker to breach the security and steal the contents of database but also to make changes and manipulate both the database schema and contents. No database is safe in web applications. The Structure Query Language (SQL) is used for interaction with relational database and the SQL query provides the result from backend database [1]. Third generation web technologies are mostly dependent upon parameter and its values. Also, the dynamic web applications accept user inputs through parameters i.e. request parameters [4]. Major threat to dynamic web applications are SQL Injection attacks (SQLIA). An SQL injection attack occurs when an input from user contain SQL keywords so that dynamically generated

SQL query changes the intended function of that SQL query [5]. In this attack, the attacker inserts a portion of SQL statement via not sanitized user input parameters into the original query and passes them to database server [6]. For example, a web application may invite the user to fill a form, post a comment, to submit a username and password for authentication. An attacker can enter the following input through user interface:

Username: 'OR 1=1- -' AND Password: 'OR 1=1- -'

This injected command would generate the following query: `SELECT user info FROM users WHERE id='1' OR 1=1- -' AND password='1' OR 1=1- -';`

Because the given input makes the WHERE clause in the SQL statement which is always true (a tautology) the database returns all the user information in the table. Therefore, the malicious user has been authenticated without a valid login id and password [5]. Lack of input validation in web applications causes hacker to be successful. Today, most of the web applications use a multi-tier design, usually three tiers: a presentation, a processing and a database tier. The presentation tier in the HTTP web interface, and it displays information related to such services as browsing, merchandise [7]. The data processing within the web application is done at the CGI tier. It can be programmed in various server script languages such as JSP, PHP, ASP etc. [8]. It implements the software functionality and the database users and the rejection of malicious users from the database. SQLI vulnerabilities and attacks occur between the Presentation tier and the CGI tier. Basically, the SQLIA process can be explained in three phases:

- i) An attacker sends the malicious HTTP request from a client to the web application as input.
- ii) Generates a SQL statement
- iii) Submits the SQL statements to the back end database server.

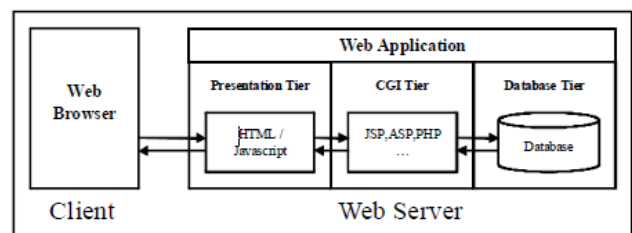


Fig. 1. Web Application Architecture [8]

When an authenticated user enters its Username and Password, the Presentation tier uses the GET or POST method to send the data to the CGI tier. The SQL query within the CGI tier connects to the back end database, processes the data and sends the result back to user [8].

This paper is organized in five sections as follows. Section 2 presents the types of attacks. Related work of SQLIA detection and prevention techniques is discussed in section 3. Proposed technique is provided in section 4 and the results are shown in section 5. Finally, conclusion is provided in section 6.

II. TYPES OF ATTACKS

For a successful SQLIA, the attacker should append a syntactically correct command to the original SQL query. The several types of attacks are discussed as follows [9] [10] [11]:

A. Tautology

This type of attack represents to the SQL manipulation category in which attacker can inject malicious code into more than one conditional query statement to be evaluated always true. It is mostly used to bypass authentication. For example, in this type the queries are of the form:
*Select * FROM accounts where name = 'sonakshi' OR 1= 1- - ;* which gives all the rows as output because '1=1' is always true.

B. Illegal/Logically Incorrect queries

Here the intention of attacker is to gather information about the type and structure of back end database that is being used in web applications [12]. These database error messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. This attack will return a default error webpage, which reveals important data. Mainly syntax errors are used to perform this attack. For Example: *Select * FROM accounts WHERE login='AND pwd='AND Pin = convert (int, (select table_name from information_schema.tables where xtype='u'))*. This is the mysql php syntax. Here, the attacker tries to extract the user table (xtype='u') from metadata table i.e. *tables*, after which the query is being converted to an integer, for that the system will throw an error because of illegal type conversion. The type of error message generated, will tell the attacker about the type of SQL Server being used. Also, the attacker comes to know about the value of string which caused the type conversion to occur [13] [14] [15].

C. Union query

This type of attack is mainly used to extract data. The output of this attack is that the database returns a dataset that is the union of the results of the original query with the results of the injected query [16]. Its format is; 'UNION SELECT < part of injected query>', where the query after UNION keyword is fully under control of the attacker so that he/she can retrieve data from any table which is not intended by the actual query [12]. For example: *SELECT * FROM user WHERE id= '1111' UNION SELECT * FROM member WHERE id='admin' - ' AND Password= '1234' ;* Injected query is concatenated with the original SQL query using the

keyword UNION in order to get information related to other tables from the application.

D. Piggy-backed query

Here, the additional query is injected into the original one. As a result, database has to process multiple queries simultaneously. Database treats this query as two different queries which are separated using the delimiter (;). Normally the first query is legitimate query, whereas the following query could be illegitimate. So, the attacker can inject virtually any type of SQL command to the database [17] [18] [19] [20]. For example, the attacker injects "O; drop table user" into the pin input field instead of logical value. Then the query will be produced as: *SELECT * FROM users WHERE id= 'admin' AND password = '1234'; DROP TABLE user;-- ;* Because of ";" character, database accepts both queries and executes them. The second query is illegitimate and can drop user table from the database.

E. Stored Procedure attacks

In this attack, the attacker tries to execute already present stored procedures in the database. Through this, the attacker identifies the current database being used, hence causing harmful effects. For example, Consider a SQL query; *SELECT id FROM users WHERE login= " + @Name + ", and pwd= " + @passwr + " and pin= " + @pin + " ' .* To perform this attack, the attacker injects ' ; SHUTDOWN; into either Name or password, which would result in shut down of database [21].

F. Inference attacks

Here, the attacker observes the response of the webpage after injecting some malicious code into it. Usually this attack takes place when attacker cannot use the error messages generated by the database. "Timing Attacks" and "Blind Injections" are the two kinds of inference attacks techniques. With these types of attacks, intruders change the behavior of a database or application [22]. These are as discussed as follows:

1) Blind Injection

The developers hide error details from the attackers, so that they won't get any help from the database. In this case, the attacker faces a generic page provided by developer, instead of an error message. An attacker can still steal the data by asking a series of True/False questions through SQL statements [1] [18].

2) Timing Attacks

A timing attack lets an attacker, gather information from a database by observing timing delays in database responses. This technique makes the use of "if-then" statement which causes the SQL engine to execute a long running query or a time delay statement depending upon the logic injected. In time based attacks, attacker introduces a delay by injecting an additional SLEEP(n) call into the query and then observing if the webpage was actually delayed by n seconds [23]. 'WAITFOR' is a keyword which causes the databases to delay its response by a specified time [16], [19]. For example, *declare @varchar (8000) select @s= db_name() if (ascii(substring(@s,1,1)) & (power(2,0)))>0 waitfor delay '0:0:5'*

In the above query, database will pause for five seconds if the first bit of the first byte of the name of current database is 1. Then the code is injected to generate a delay in response time when the condition is true.

III. RELATED WORK

The researchers have proposed various methods to address the SQL injection problem and there are many solutions proposed in the literature. These are discussed below:

Indrani Balasundaram, E.Ramaraj [26] proposed an authentication mechanism to prevent SQL injection attack using Advance Encryption Standard (AES). In this method, encrypted username and password are used to improve the authentication process with minimum overhead. The method has proposed three phases, in the first phase i.e. *registration phase*, server sends a registration conformation. In the second phase i.e. *login phase*, user can access the database from server. The username and password is encrypted by using Advance Encryption Standard (AES) algorithm by applying user secreta key and the SQL query is generated using encrypted username and password. Then the query will be sent to server. In the third phase i.e. *verification phase*, server gets the login query and verifies the corresponding users secreta key. If they matches then the decrypted username and password is checked from user account table. If it matches, then user is accepted otherwise rejected.

Mayank Namdev, Fehreen Hasan, Gaurav Shrivastav [24] have given a model to block SQL injections which is based on verification information. They have combined two approaches and created a new hybrid algorithm which works by applying the hash code with encryption for more security. In this approach, two extra columns are needed, one for storing the hash values of username and another one for storing the hash values of password. When the account of users is created for the first time, the hash values are calculated and stored in user table. The hash values are calculated at runtime using stored procedure when user logs into the database. The values which are calculated at runtime are matched with stored hash values in database table. Thus, if user tries to inject to the query, the proposed method will automatically detect the injections as malicious content & rejects the values. Therefore, it cannot bypass the authentication process. Its advantage is that hackers do not know about the hash value concept.

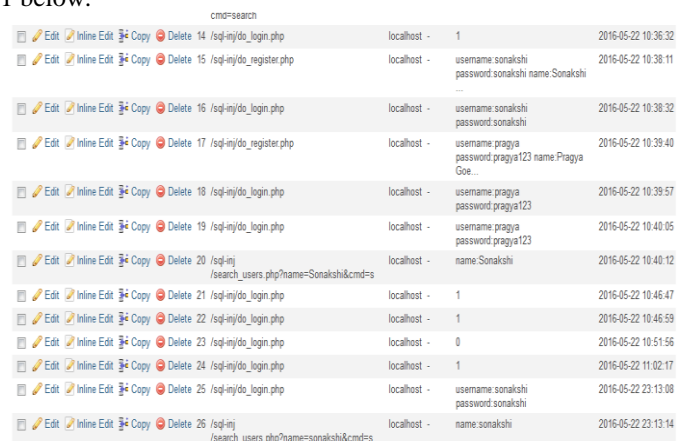
A similar approach to protect web applications against SQL Injection attacks is proposed by Neha Mishra, Sunita Gond [27] in which the authors has discussed some predefined methods and also proposed an integrated approach of encryption method with secure hashing. The technique works by creating two columns by DBA for storing username and password, in the same way as done in previous approach by [24] The secure hash values are generated at runtime using stored procedure if a user wants to login to the database. These values are stored in login table when the user's account is created first time. If a user wants to login to the database, his/her identity is verified via username, password and secure hash values. The combined approach of secure hashing on encryption is provided. The purpose of encrypting data is that it helps to change the data into a form that is not readable [28].

IV. PROPOSED METHOD

On studying the various SQL injection prevention techniques, a new technique is proposed in this paper for preventing the database against SQL injection attack. In the proposed approach, the security is provided in three different phases, such as:

1) *Input Validation using query tokenization and log maintenance at login phase*

At the login phase, user is allowed to enter single word instead of multi-words which is the basic cause of SQL injection attacks. By the use of single word, possibility of attack is reduced to minimum. In this way, malicious activity is detected if a user enters username or passwords using multi-words e.g. 'OR 1=1- -'. In this example, there is a space after "OR" so, it is treated as multi-word input. Almost all types of SQL injection attack types are prevented by allowing single word input only. At the same time, the input is tokenized, and list of various known malicious symbols or tokens are maintained in a log file. The validation process at login phase checks the entered input and matches it with malicious known symbol list. If a match is found, then further access is restricted and that user is considered as malicious one. In the proposed scheme, the IP address of malicious user's system along with date, time and page of application from which the entry is made is saved for future reference. It will help to keep a check on malicious activity in future as well. System will continuously observe user's login detail and compare it with saved database and looks for particular IP address who has visited the page given number of times & if found to be malicious is blocked for further access. The input entered by users is stored in logs table as shown in fig. 1 below.



| ID | Action | File | Host | Username/Password | Time |
|----|--------|---|-----------|---|---------------------|
| 14 | Copy | /sql-inj/do_login.php | localhost | 1 | 2016-05-22 10:36:32 |
| 15 | Copy | /sql-inj/do_register.php | localhost | username:sonakshi password:sonakshi name:Sonakshi | 2016-05-22 10:38:11 |
| 16 | Copy | /sql-inj/do_login.php | localhost | username:sonakshi password:sonakshi | 2016-05-22 10:38:32 |
| 17 | Copy | /sql-inj/do_register.php | localhost | username:pragya password:pragya123 name:Pragya Goe... | 2016-05-22 10:39:40 |
| 18 | Copy | /sql-inj/do_login.php | localhost | username:pragya password:pragya123 | 2016-05-22 10:39:57 |
| 19 | Copy | /sql-inj/do_login.php | localhost | username:pragya password:pragya123 | 2016-05-22 10:40:05 |
| 20 | Copy | /sql-inj /search_users.php?name=Sonakshi&cmd=s | localhost | name:Sonakshi | 2016-05-22 10:40:12 |
| 21 | Copy | /sql-inj/do_login.php | localhost | 1 | 2016-05-22 10:46:47 |
| 22 | Copy | /sql-inj/do_login.php | localhost | 1 | 2016-05-22 10:46:59 |
| 23 | Copy | /sql-inj/do_login.php | localhost | 0 | 2016-05-22 10:51:56 |
| 24 | Copy | /sql-inj/do_login.php | localhost | 1 | 2016-05-22 11:02:17 |
| 25 | Copy | /sql-inj/do_login.php | localhost | username:sonakshi password:sonakshi | 2016-05-22 23:13:08 |
| 26 | Copy | /sql-inj /search_users.php?name=sonakshi&cmd=s | localhost | name:sonakshi | 2016-05-22 23:13:14 |

Fig.2. Log entries of users is stored in logs table of database

2) *Use of Parameterized queries*

Using dynamic SQL queries is the root cause of SQL injection vulnerability. OWASP also recommends the use of parameterized queries as the first choice of prevention techniques for this vulnerability [29]. A dynamic query directly uses user's input into the query. While the SQL Parameterized Query forces the user to implement the logic of SQL query first and then inserting the user input values into it. This forces the SQL query to be built before entering any user input in it. The SQL query is sent as a query, and the database knows exactly what this query will do, and only

then it will insert the username and passwords as values. This means they cannot affect the query, because the database already knows what the query will do. So in this case it would look for a valid username and password not the malicious values. It won't allow direct insertion of user input value at the query creation step because it doesn't make use of concatenation symbols like "+", "." etc., which are most vulnerable to SQLIA. E.g. \$sql = "select `id` from users where username=? and password=?"; Another advantage of using SQL parameterized query is that it forces the data type of user input for a particular field in SQL query.

3) Encryption of Confidential data values at back end database using RC4 & blowfish algorithm

Encryption of confidential data helps to change the data into a form that is not readable [28], hence it helps to prevent database attacks. In the proposed method, the data is encrypted using RC4 & blowfish algorithm instead of AES, because these both have less time complexity as compared to AES i.e. they takes less time to encrypt the same lines of input if is done with AES algorithm. Another advantage is, RC4 algorithm is also symmetric i.e. the same key is used for encryption & decryption which takes less storage space to store a single public key. The process works by encrypting the user's registration & login details using RC4 algorithm or blowfish at backend. Whenever a new user registers his account, a unique secret key is generated and stored. At the time of login, the generated secret key is matched with the stored one, if a user is valid only then his/her information is decrypted otherwise it is present in encrypted form on backend server so that attacker cannot understand the exact data values.

Fig .3.Login form

In the above Fig. 1, the user has tried to login through injection, so if user tries to read the password of any other user through search page then he won't get succeed because the password is present in encrypted form at backend. The same can be seen in figure 2.

| Id | Username | Password | Name | Email | Mobile | Gender |
|----|----------|----------|-------------|---------------------|------------|--------|
| 5 | pragya | #_D&e5É | Pragya Goel | pragya123@gmail.com | 9876543210 | Female |

Fig.4. Encrypted password values at backend

It provides advantages in two ways. Firstly, the confidential data such as passwords and other personal information is also stored in encrypted form which ensures data confidentiality and integrity as in fig. 3. Secondly, if a user tries to enter some malicious keywords like 'OR 1=1 - -' in password field, it will be encrypted into some other form and hence user will not be able to access by using these keywords as done in tautology attack and an error message will be displayed such as in fig. 4.

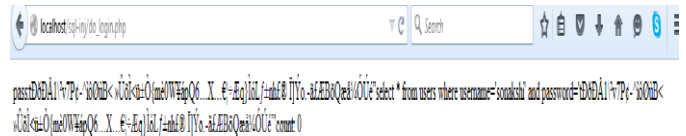


Fig 4: Error due to injected password field

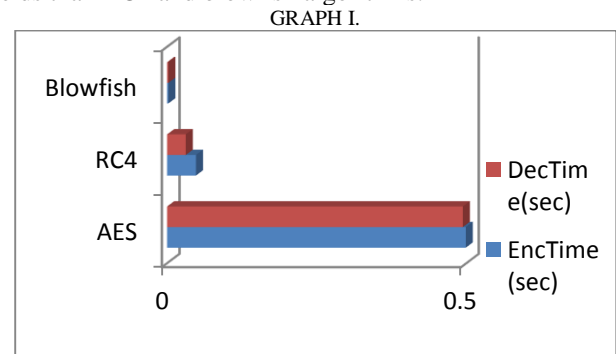
V. IMPLEMENTATION AND RESULTS

As an instance, a text file of size 5 KB approximately, is read, and its encryption and decryption time is noted corresponding to AES encryption algorithm, RC4 and Blowfish algorithm respectively in table 1 given below. Also, it can be seen in graph 1 which shows blowfish has least time complexity among all three encryption algorithms. So, it enhances performance by encrypting the confidential information faster than before.

TABLE I. Encryption, decryption and total time

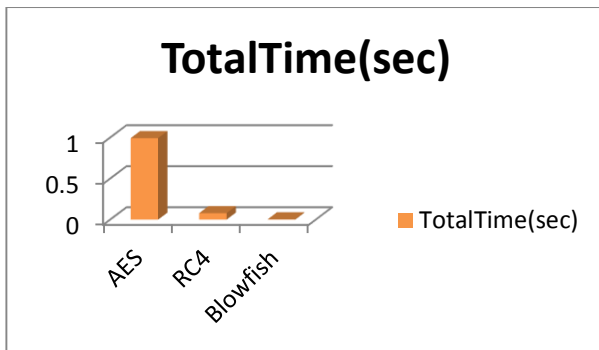
| Algorithm | Enc. Time(sec.) | Dec. time(sec.) | Total time (sec.) |
|-----------|-----------------|-----------------|-------------------|
| AES | 0.499 | 0.494 | 0.993 |
| RC4 | 0.048 | 0.031 | 0.079 |
| Blowfish | 0.002 | 0.002 | 0.004 |

The graph 2 shows the total time taken by all the three encryption algorithms given in table 1. Clearly, it can be seen that AES takes the more time to encrypt and decrypt the input fields than RC4 and blowfish algorithms.



Graph I. Representation of encryption and decryption time of Blowfish, RC4 and AES encryption algorithm

GRAPH II.



Graph II. Representation of total time in seconds

VI. CONCLUSION

The paper presented a novel and applicable technique for protecting web applications from SQLIAs. The approach consists of allowing single word user inputs only, that will automatically eliminate all sources of attack vulnerabilities. The next approach used is Rc4 and blowfish encryption methods which will enhance performance due to their less time complexity. AES is very complex encryption standard while RC4 is very old and simple. Both RC4 and Blowfish are faster in performance as compared to AES technique. Since, the technique is developed at application level, it requires no modifications in the existing runtime system and imposes less execution overhead to reduce SQL injection attacks almost in every way.

REFERENCES

- [1] Chandrashekhar Sharma and Dr. S.C.Jain, "Analysis and Classification of SQL injection vulnerabilities and Attacks on Web Applications," in IEEE International Conference on Advances in Engineering and Technology research(ICAETR), Kota, 2014.
- [2] [Online]. <http://www.w3.org/protocols>
- [3] (2013) The Open Web Applications Security Project, "OWASP TOP Project". [Online]. http://www.owasp.org/SQL_Injection
- [4] Venkatramulu Sunkari and Dr. C.V. Guru Rao, "Preventing Input type Validation Vulnerabilities using network based Intrusion detection System," in International Conference on Contemporary Computing and Informatics(IC3I), Warangal, 2014, pp. 702-706.
- [5] MeiJunjin, "Anon vulnerability detection approach for SQL inject," IEEE, pp. 1411-1414, 2009.
- [6] N. Antunes and M. Vieira, "Defending against Web Appliaction Vulnerabilities," vol. 45, no. 2, pp. 66-72, 2012.
- [7] Bojken Shehu and Aleksander Xhuvani, "A Literature review and Comparative Analyses on SQL Injection: Vunerabilities, Attacks and their Prevention and Detection Techniques," International Journal of Computer Science Issues(IJCSI), vol. 11, no. 4, pp. 28-37, July 2014.
- [8] Jeom-Goo Kim, "Injection Attack Detection using the Removal of SQL Query Attribute Values," IEEE, 2011.
- [9] Diallo Abdoulaye Kindy and Al-Shakib Khan Pathan, "A surevey on SQL injection : Vulnerabilities, Attacks and Prevention Techniques," in IEEE 15th International Symposium, 2011.
- [10] Atefeh Tajpour, Suthaimi, and Maslin Masrom, "SQL injection detection and prevention techniques," Inetrnational journal of Advancements in Computing Technology, vol. 3, no. 7, August 2011.
- [11] W.G. Halfond, J. Viegas, and A. Orso, "A classification of SQL injection Attacks and countermeasures," in International Symposium on Secure Software Engineering, 2016.
- [12] Sruthy Manmadhan and Manesh T, "A method of detecting SQL Injection attack to secure web applications," International Journal of distributed and Parallel Systems(IJDP), vol. 3, no. 6, November 2012.
- [13] S. McDonald, "SQL Injection: Modes of attack, defense, and why it matters," White paper, GovernmentSecurity.org, April 2012.
- [14] Haeng Kon Kim, ""Frameworks for SQL Retrieval on Web Application Security," in International Multiconference of Engineers and computer scientist, 2010.
- [15] K. Phalguna Rao, Dr. Ashish B.Sasankar, and Dr. Vinay Chavan, "Analysis of Detection and Prevention Techniques Against SQL Injection Vulnerabilities," IJCST, March 2013.
- [16] V. Nithya, R. regan, and J. Vijayaraghavan, "A survey on SQL injection attacks, their Detection and Prevention techniques," International Journla of Engineering and Computer Science, vol. 2, no. 4, pp. 886-905, April 2013.
- [17] Aanal Bhanderi and Nancy Rawal, "A Review n detection Mechanism for SQL Injection Attacks," International Journal of Innovative Research in Science, Engineering and Technology, vol. 4, no. 12, pp. 12446-12452, December 2015.
- [18] Pupendra Kumar and R.K. Pateriya, "A Survey on SQL injection attacks, detection and prevention techniques," IEEE ICCCNT, 2012.
- [19] Atefeh Tajpour, Suhaimi Ibrahim, and Mohammad Sharifi, "Web application Security by SQL Injection Detection Tools," International Journal of Computer Science Issues(IJCSI), vol. 9, no. 2, pp. 332-339, March 2012.
- [20] Mahima Srivastava, "Algorithm to Prevent Back end database against SQL Injection Attacks," IEEE, pp. 754-757, 2014.
- [21] Pankajdeep Kaur and Kanwalpreet kaur, "SQL Injection: tTudy and Augmentation," in International Conference on Signal Processing, Computing and Control (2015 ISPPC), Jalandhar, 2015.
- [22] Shubham Mukherjee, Sudeshna Bora, Pritam Sen, and Chittaranjan Pradhan, "SQL Injection: A Sample Review," in 6th ICCCNT 2015, Denton, U.S.A, 2015.
- [23] Debrata Kar and Suvasini Panigrahi, "Prevention of SQL Injection attack using query transformation and hashing," in 3rd IEEE International Advance Computing Conference(IACC) , 2013, pp. 1317-1323.
- [24] Mayank Namdev, Fehreen Hasan, and Gaurav Shrivastav, "Review of SQL Injection Attack and Proposed method for detection and Prevention of SQLIA," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 7, pp. 24-28, July 2012.
- [25] NTAGWABIRA Lambert and KANG Song Lin, "Use of Query Tokenization to detect and prevent SQL Injection Attacks," IEEE, pp. 438-440, 2010.
- [26] Balasundaram Indrani and E. Ramaraj, "An Authentication Mechanism to prevent SQL injection Attacks," International Journal of Computer Applications, vol. 19, no. 1, pp. 30-33, April 2011.
- [27] Neha Mishra and Sunita Gond, "Defenses to protect against SQL Injection Attacks," International Journal of Advanced research in Computer and Communication Engineering, vol. 2, no. 10, pp. 3829-3833, October 2013.
- [28] Priyanka and Vijay Kumar Bohat, "Detection of SQL Injection Attack and Various prevention Stategies," International Journal of Engineering and Advanced Technology, vol. 2, no. 4, April 2013.
- [29] Infosec Institute. [Online]. <http://resources.infosecinstitute.com/parameterized-sql-query-dynamic-sql-query/>