IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 06, Issue 02 (February. 2016), ||V1|| PP 06-17

Classification of Sql Injection Detection And Prevention Measure

Muhammad Saidu Aliero*, Abdulhamid Aliyu Ardo, Imran Ghani, Mustapha Atiku
Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, 81310 Skudai, Johor Bahru Makaysia

Abstract: SQL injection vulnerability is the one of the most common web-based application vulnerabilities that can be exploited by SQL injection attack to gain access to restricted data, bypass authentication mechanism, and execute unauthorized data manipulation language. Defensive coding is a simple and affordable way to tackle this problem, however there are some issue regarding use of defensive coding which makes the system in effective, less resistant and resilience to attack. In this paper we provide detailed background of SQLIA (SQL Injection Attack), classified defensive coding to different categories, reviewed existing technique that are related to each techniques, state strength and weakness of such technique, evaluate such technique based on number of attacks they were able to stop and evaluate each category of approach based on its deployment requirements related to inheritance. The goal of this paper is to provide programmers with common issues that need to be considered before choosing a particular technique and to raise awareness of issues related to such techniques as many of those techniques were not meant for the purpose of protection of SQLIA. In addition, we hope to provide researchers by shedding light on how to develop good SQLI (SQL Injection) protection tools as most of the SQLI protection tools were developed using combination a of two or more defensive coding techniques. Lastly we provide recommendations on to avoid such issues.

***Keywords**: SQL Injection, Defensive Coding, Injection Parameter, SQLI vulnerability.*

I. INTRODUCTION

Technological advancement and network development has allowed enterprises to change the way they conduct rout of conducing their day to day business for instance, E-commerce, health care, transportation, social activities are all now available on web-based database driving application. These applications process data and store the result in back-end database server where enterprise related data are stored. Depending of the specific purpose of application, most communicate frequently with customers, users, employees to enable them used the service offered by the enterprise. The fact that these applications can be invoked by anyone worldwide attracted attackers to possibility of benefitting from these vulnerabilities. According to a survey report published on security by open web application security project (OWASP) and system administration, network and security (SANS) security expert report from 2014 report, over 63% of web applications worldwide are at risk of being hacked a as result of their vulnerabilities [1].

SQLI (SQL injection) vulnerability is the one of web-based database driving application vulnerabilities that presents high risk for organizational assets. SQLI vulnerability results from inappropriate validation of input from user, which enables the attacker to manipulate programmer intended queries by adding new SQL operator, command, keyword, or clause thereby bypassing authentication mechanism and unauthorized database modification.

To overcome the problem of SQLIA many programmers use defensive coding practice to secure their queries from manipulation by malicious users. Defensive codes provide an affordable and straightforward method by which programmers can effectively secure their applications; however, there are some common issues that programmers must keep in mind while developing security mechanism using defensive coding practice. These issues begin with common programmers error to approach inherit limitation.

In this regard we decided to review the most common defensive code practices and to provide recommendation as to how programmers should tackle such issues. Figure 1 below shows the organization of this paper.

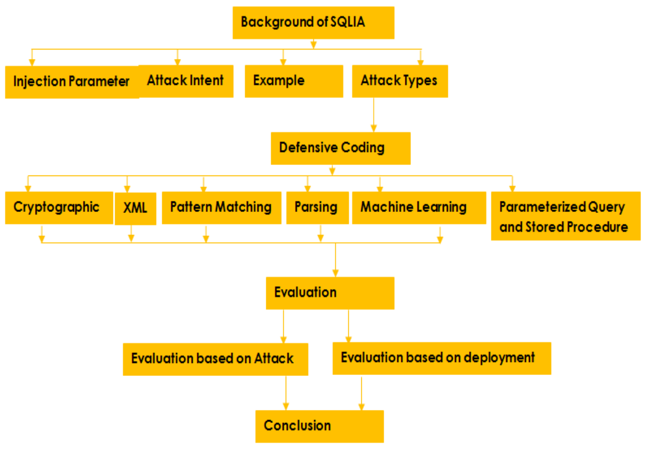

Figure1 Orgnization of paper

II. BACKGROUND OF SQLIA

2.1 Injection Parameters

Attackers inject malicious queries in through input text field, cookies, server variables and using second order SQL injection attacks [22]. This section provide the description of injection parameters

Through User input field**:** web application requests are made through user input fields which transfer user request from client side to server side and back to client side with help of HTTP PO and GET. Because these inputs are connected with backend database using SQL statement to retrieve and render requested information for users or allows users to connect to the system. Attackers can inject malicious input to change intended query if these inputs are not properly validated.

Injection through cookies: Cookies are structures that maintain persistence of web application by storing state information in the client's machine. When a client returns to a Web application, cookies can be used to restore the client's state information. If a Web application uses the cookie's contents to build SQL queries, then attackers can takes this opportunity to modify cookies and submit to the database engine.

Injection through server variables: Server variables are a collection of variables that contain HTTP, network headers, and environmental variables. Web applications use these server variables in different ways, such as session usage statistics and identifying browsing trends. If these variables are logged to a database without sanitization, this could create SQL injection vulnerability. Because attackers can forge the values that are placed in HTTP and network headers-by entering malicious input into the client-end of the application or by crafting their own request to the server.

Second order injection: In second-order injections, attackers plant malicious inputs into a system or database to indirectly trigger an SQLIA when that input is called at a later time so when the attack occurs, the input that modifies the query to construe an attack does not come from the user, but from within the system itself.

2.2 Attack Intent

Attacks can also be classified based on what the attacker wants to achieve. Therefore, each of the attack types that we provide in Section 2.3 includes a list of one or more of the attack intentions defined in this section.
Identifying Inject able Parameters: Inject able parameters are text-input that allows user to request information from the database. This query request is sent to the database server though HTTP request, for example ULR; search box and authentication entries are considered as text-input. When these text-inputs send user requests, to database without proper validation they are considered as inject able parameters which allow attackers to inject SQL query attack. Identifying injection parameters is the first step in performing an attack[22]

Performing database fingerprinting**:** after identifying the injection parameter the second step is to know the database engine type, knowing version of database is very important to an attacker because it enables him to know how to construct query format supported by that database engine(as vendors of database engine employs a different proprietary SQL language dialect).

Determine database schema: schema is the structure of database. It includes table names, relationship, and column name. Knowing this information about database makes it easier for an attacker to construct an attack to perform database extraction or data manipulation language.
Database manipulation and extraction: depending on the intent, most attackers are more interested in getting customers bank information, creating invalid data modifying friend's salaries.

Evading detection: this is a technique that attackers use to avoid being detection by security mechanism in the sense that their action cannot be detected or traced.
Executing Remote Commands: Remote commands are executable code resident on the compromised database server. Remote command execution allows an attacker to run arbitrary programs on the server. Attacks with this type of intention could compromise the entire network.
Bypassing authentication: this is the most popular attack because it allows attackers to get access to the database with user privileges.
Performing privilege escalation: These attacks take advantage of implementation errors or logical flaws in the database in order to escalate the privileges of the attacker. As opposed to bypassing authentication attacks, these attacks focus on exploiting the database user privileges.

2.3 Example of PHP Vulnerable Code
Before describing types of SQL injection attack, we present an example of PHP code that is vulnerable to SQL injection attack. We use this example in section 2.3 to provide attack examples.

```
$connection=mysql_connect('localhost','root','') or die('connection error');
 $dbselect= mysql_select_db('hr')or die('connection error');
$uname='msaidu';
$pword='123';
$query="SELECT * FROM login WHERE username='$uname'
 AND password='$pword' ";
$dquery=mysql_query($query);
$tquery=mysql_num_rows($dquery);
if($tquery==Null){
echo 'User Does not exist';
}
else{
echo ‘You are now connected ';
}
$Get_Rows = GetRow(Run("SELECT * FROM employee"));
$Get_Rows = GetRow(Run("SELECT * FROM salary"));
```

2.4 SQLIA Types
Tautology attack: this type of attack takes advantage of “WHERE” clause in SQL statement to evaluate the results returned by Query in relational database to be always true. Attacker uses this type of attack to achieve authentications bypassing in web application or perform unauthorized database extraction.

Authentication bypassing all relational database management system with no exception evaluate SQL query with “OR 1=1” where clauses as always true. Also in relational database management system anything followed by comment (--) will not be processed. For example valid user login to the system by using:

(SELECT * FROM login WHERE username=admin AND password=admin123);// ------------statement (1)
However attackers always find ways to bypass authentication to have access to the system and this can be achieved by :
(SELECT * FROM login WHERE username=admin or 1=1--- and password =nothing). ------statement (2).
As shown in statement 2 admin was used as a user or 1=1 meaning that connect admin or whoever exist in the system and comment (---) was used to ignore password which means password will not be processed.

Data extraction: most of the users in the system are only allowed to perform certain actions for example viewing their own profile details. However malicious users or attackers always want more and trying to access restricted data.
For example, system was design for employee to view their own personal details. Thus employees can only view their profile and nothing more.

(SELECT * FROM username, password UNION SELECT salary, cardnumber FROM pay WHERE username=admin ---statement 1
This statement displays username, password, salary and credit card number of admin. But what would happen for people with bad intentions? Because they are only trying to find information that would help them making money illegally so they might think of injecting something like this:

(SELECT * FROM username, password UNION SELECT salary, cardnumber FROM pay WHERE username=admin OR 1'='1--statement 2

As shown, the "OR 1=1" transforms the "WHERE" clause to true which results in displaying at least one record that exists in database, if not all, with the following details about employee: Username, password, salary and credit card number

Illegal or incorrect logical query: knowing the names of the server schema tables, and column make it easy for attackers to have unauthorized access to the system.
For example http://localhost/?EmpId=10'
As shown at the end of the ULR quote was inserted after 10. This is kind of disturbing the database engine because when you type something within quote it is used to tell the database this is a query and to process it. Thus after processing http://localhost/?EmpId=10' the database engine returns the following message saying:
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\" at line 1.
This tells the attacker that backend database engine used is MySQL and the user is asked to check the syntax in line one.
Inference Attack: this attack can be classified into Blind SQL injection and Timing Attack.

Blind SQL injection attack: this is another method of doing database fingerprinting .Sometimes database engines can be configured to hide database error messages and return generic errors to the user when there is SQL syntax error in the users SQL statement. This can serve as a method to prevent attackers from database fingerprinting by using illegal or incorrect query method. However this does not mean the database is secure; it only conceals the return default error message making it difficult for attackers who rely on database finger printing as the first step in carrying out an attack. Thus blind SQL injection attack can be used to deduced if there is a security mechanism implemented in the web application or not. Blind SQL injection attack can be achieved by asking a series of true or false queries in the database. In this the case attacker tries to inject the following statements:

SELECT * FROM emp_name, emp_address, gender, from employee where 1=0; drop employee ------------------------Statement (1)

SELECT * FROM emp_name, emp_address, gender, from employee where 1=1; drop employee -----------------------Statement (2)

After executing above Boolean malicious SQL query an attacker will get to know if the database is secure or not. A same response (return generic error message) means that there is a protection mechanism that detects an attack and blocks the execution of the query and returns an error message to the user because the statement contains malicious keywords. A different response means that the query has reached the inside of the database engine and has been and executed. Therefore the first query will returned with an error message because it is an incorrect query while the second may or may not return any error message because it is a correct query.

Timing Attacks: In this type of attack the response time by which the database takes to respond to users queries are noticed which helps to deduced some information about a database. This method uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example an attacker can extract information from database using a vulnerable parameter.

declare @ varchar (8000) select @s = db_name () if (ascii (substring (@s, 1, 1)) & (power (2, 0))) > 0 waitfor delay '0:0:8
Union attack: this is most common type of attack used by attackers in gaining access to restricted data in other tables. Malicious SQL query can be appended with valid SQL query in order to secure unauthorized access to extra data.
Consider example where online human resource in a particular company allows employees to view only their personal details online. A malicious user can access extra information such as employee credit card number.
For Example:

SELECT * FROM emp_name, emp_address, gender, from employee where emp_id=7;
The above statement displays employee name, address and gender of with identification number 7. The below statement extracts more data about an employee:
SELECT * FROM emp_name, emp_address, gender, from employee where emp_id=7UNION SELECT credit_card FROM salary;

The above statement provides an attacker with employee details including credit card number of employees.

Piggery-backend query attack: some of the database engines support stacked query by default. This feature creates opportunity for attacker to perform dangerous actions in the database. In this case valid queries will be terminated by (;) and malicious query is added. After processing of valid query the malicious query is executed unlike in union query where malicious queries will be combined with valid query and processed as a single joined query.
For example:

SELECT * FROM emp_name, ep_address, gender, from employee where emp_id=7; drop employee
The above query will drop the employee table after displaying employee information.

Stored Procedure: Stored procedure is a part of database where programmers could set an extra abstraction layer as security to prevent SQL injection attack. As stored procedure could be coded by the programmer, so, this part is as inject able as web application forms. Depend on specific stored procedure on the database there are different ways to attack.

Alternate encoding**:** most of the SQL injection mechanism that uses filters prohibits the use of quote (‘) in the SQL statement which can be used in constructing different kind of malicious query request to the database. In this case for attacker to bypass such filter he has to convert SQL query into it alternative encoding such as hexadecimal, ASCII or Unicode. by converting SQL query into alternate encode enable them to carry out their attacks. For example

"0; exec (0x73587574 64 5f177 6e), " and the result query is: SELECT accounts FROM login WHERE username=" AND password=0; exec (char (0x73687574646j776e))
The above example use the char () function and ASCII hexadecimal encoding. The char () function takes hexadecimal encoding of character(s) and returns the actual character(s). The stream of numbers in the second part of the injection is ASCII hexadecimal encoding of the attack string. This encoded string is translated into the shutdown command by database when it executed.

III. DEFENSIVE CODING

Due to lack of resources and technical knowledge some enterprises prefer programmers to directly use scripting to embed security codes inside their source code of the web applications. One of the objectives of defensive coding is to write secure queries so that it behaves in a predictable manner despite unexpected inputs or user actions. It is based on the idea that every program module is solely responsible for itself. Normally secure codes are been added to the application after web development was done and tested and it works perfectly. Issues sometimes arise after a secure code are been added to web application due to lack of knowledge or improper testing by the development team which results in malfunction of the website and limiting right of some to use application.

Basically defensive coding can be characterized as shown in figure 1 below. Programmer uses one or more approach to patch up their application depending of the need on the enterprise.
For example some enterprise do not allow search box in the website to reduce the risk of being attacked. In such case they provide the user with predefined search input where he can select his search, so there is no need to apply a security code in the search box.
In this section we categorised existing methods based on their approach. We also stated strength and weakness of reviewed existing methods based on implementation as well as inherent feature of such methods.

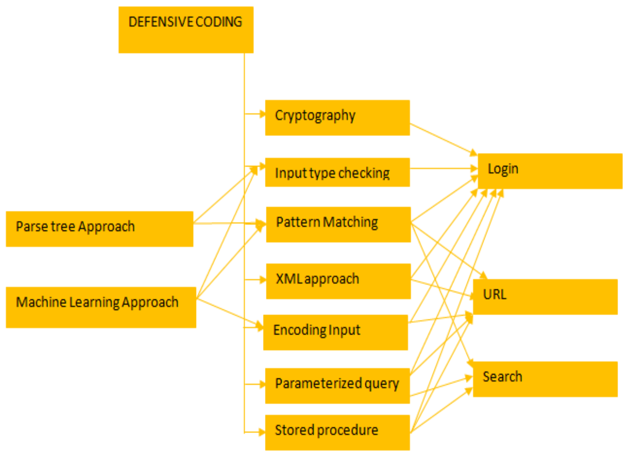

Figure 1 Categorization of Defensive Coding.

3.1 Cryptographic Approach

Literature [2] and [3] describe types of SQL injection attack and propose a runtime approach that uses cryptographic hash function algorithm to prevent attackers from bypassing the login authentication on the system. The idea was to prevent database engines from processing statements like “OR 1=1-- that attacker use to bypass authentication mechanism. Because all relational database engines with no exception process query with ‘OR 1’=’1’-- as always true, after “WHRE” condition, which give attackers opportunity to gain access to the system without proper authentication and authorization.

In this method when users try to log into the system, the credential will be converted into the corresponding hash value and compared with hash value stored in database which was created by the user the first time. In this case, when attacker try to log in to the system with muhd ‘OR’1= ‘1—as his credential. The system will automatically prevent access because the hash value of two different credentials can never be the same (unless if collision occurs which is very rare and it only happens in MD5). The importance of this method is that the hashed values of credentials are very difficult to be reverse even if attacker managed to gain access to the database table. This method does not prevent database finger printing attack because the approach does not really detect any malicious value, having only transformed it into a different format to prevent fooling the relational database engine.

Literature [4] and [5] also describe types of SQL injection attack and proposed the similar method in [2] and [3]. The only difference is that hash function is applied to encrypted credentials and comparison is being made between hashed encrypted credentials produce dynamically at runtime. The problem with this approach is that it is time consuming. User must wait for the system to compute the hash value of credentials to let him access the system. The advantage of this approach is that it adds security to user credential such that even if the user account is compromised the attacker would not be able to deduce anything since valuable information is not stored in clear text.

3.2 XML Approach

Literature [6] proposed hybrid method to detect and prevent SQLIA. In this method XML was used to authenticate system users. The use of XML here is similar in concept to the cryptographic approach by not allowing relational database engines to directly process queries with tautology attack (OR1 =1--) in which result is always evaluated to be true. When users try to log into the system XML file maker intercepts the user query and converts it into XML format which will then be sent to the XML file. XML file then pass user credential to the Xschema validator and Xschema validator compares the user query that is produced by XML maker with user query threshold that was already defined as a legitimate query in Xschema file. If the dynamic query matches the query defined in Xschema file then the user is allowed to connect to the systems; otherwise access is denied and error returns to the user. This method has a number of drawbacks, First, storing data for processing in XML format is not a good idea because the processing speed is very slow in which each tree nodes of tree has to be visited. Second, using predefined patterns in protection mechanism has limitation as users can inject values that programmers have not been addressed in user threshold which results in successful attack. Third, blocking SQL keywords, operators and clauses is not effective because sometimes we might have a query containing keywords blocked by programmer while they are allowed to be used in the system query. However, blocking some known characters and keywords can help to prevent database finger printing.

Literature [7] and literature [8] proposed similar methods that detect and prevent SQL injection attack in databases using web services. In this approach functions were created that generate user credentials from database and temporary store them in XML file authentication.

In this approach if user credential found to be equal with one in XML the user is allowed to connect to the system; otherwise they are blocked. Using XML to authenticate is not a good idea for two reasons. One causes delays generating and authenticating users second, attackers can inject XML to get user’s credentials.

Literature [9] also provides an overview of type SQL injection attack and proposed similar literature [8] and literature [9]. In these methods XML was used to filters suspicious characters, operators and SQL keywords. The advantage of using XML is that after XPath authentication user credentials are encrypted to prevent password sniffing. Also, in terms of database storage, this method used copy of login table only for authentication in case of any successful attack on login table the original login table remain unaffected.

3.2 Pattern Matching Approach

Literature [10] proposed a method to defend against SQL injection attack using ASCII Based String Matching as an alternative to using hash or XML method to prevent database engine from executing commands like “1 ‘OR 1’=’1--. When a user is created his credentials are converted in to ASCII based and stored in database. When user tries to log into the system his credentials are encoded at runtime and compared with one stored in database. This method is not time consuming, similar to hashing approach and XML approach. However a disadvantage over hashing and encryption method is that it is very easy to bypass and incase the login table is compromised as an attacker does not need to know the key or algorithm used in conversion since no algorithm type no key is required for encoding strings into ASCII based.

Literature [11] proposed a method that uses Query Matching to detect and prevent SQL injection attack. This method uses three (3) library files to perform validation. One of the file is called SQLMF which contains all legitimate SQL queries in the website, file 2 contain the number of user defined illegitimate queries and file 3 contains format of legitimate dynamic SQL queries. When users issue a dynamic SQL query the XML file maker intercepts the dynamic query and converts it into XML format and makes an exact match comparison with query in file 1. If matches with master file query, then query is said to be legitimate and is passed to the database for processing. Otherwise it sends for an approximate match if it matched with the format in file 3 it is consider as valid and query is added in master file and sent to a database engine otherwise blocked and prevented from executing in database engine. This method requires source code analysis to generate query model which make it error prone because all static queries must to be carefully addressed in master file.

Literature [12] proposed a method that uses pattern matching algorithm to detect and prevent SQL injection attack. This method required users to define set of query related attacks that would be used to compare with dynamic queries issued by the user. In this case user queries would not be transformed into XML format or broken into SQL keyword, operator or characters but rather dynamic SQL query entered by user will be compared with one defined in anomaly detection pattern. If the anomaly of the dynamic query is equal to the one defined by user threshold, then the query is rejected and if anomaly score is high than the anomaly defined in user threshold then an alarm will be sent to the administrator to analyze the query manually. If the query is found to be a new attack then it is added to the anomaly detection library. The drawback of this method is that queries have to be analyzed manually by administrator and manually added into the anomaly detection library.

3.3 Parsing Approach

Literature [13] proposed a method to detect and prevent SQL injection attack in the URL of the application. In this method query model was created (SQL_statement_safe) as a library that contains syntax grammar of SQL statement. This syntax grammar was built on two perspectives, one for single query and one for stacked query. It also contains tree structure of SQL query. When user issues SQL query from the website URL the query will first be checked in SQL_statement_safe to see if the query is single and conforms to the semantics of a legitimate SQL statement. If a query was found to be single and conforms with defined syntax, it is allowed to be processed in the database engine, and if it is single and does not conform to syntax in SQL_statement_safe it rejected. If the query is found to be a stacked query it then pass to parse tree comparison is done to check each action in each query. If both actions are found to be legitimate, an action query is allowed; otherwise it is blocked indicating that the user has modified one of the stacked queries.
The problem of this method is that it allows an attacker to perform database fingerprinting by executing an inference attack. The importance of this method is that there is no need of adding URL when new a page is created.

Literature [14] and literature [15] proposed similar methods that use the similar semantic of SQL statement as was used in building the SQL_statement_safe [13]. This method named as Benin_query. The same procedure was used in detecting malicious queries entered by users. The only difference is that in this method authors added a store procedure mechanism to prevent attackers from performing database fingerprinting. The advantage of this method makes SQL injection attack very difficult if not impossible as attackers do not have information on how the database was designed This stored procedure hides database design secret such as table, schema and column name from potential attackers. However using stored procedure to conceal database secret design is problematic in nature as it depends on how accurately the procedure was designed as it can be vulnerable as pieces of code.

Literature [16] provide an overview of SQL injection attack types and proposed methods in which static queries of web applications are built in the form of a tree-like structure. The assumption was all legitimate queries have same semantic syntax. If the query entered by the user does not the match structure of the defined query, the query will be blocked. In addition the queries model uses filter layers to prevent database finger printing. Using filters to prevent database fingerprinting is more effective than using stored procedures as attackers can exploit store procedure to perform malicious action in database.

Literature [17] proposed a method to detect and prevent SQL injection attack that uses lexical analysis to break SQL queries in number of tokens called parameter. The number of parameters in SQL query used to detect whether query contains malicious input or not. The idea behind this method was that tautology attack cannot be carried out without blank space, special characters and equal sign. Thus if number of tokens in dynamic query entered by users is high than the number of token expect by programmer, then the query is considered as malicious. This method can only prevent attacks that requires space, = sign and special characters.

Literature [18] proposed a method that uses parse tree structure to determine the legitimate structure of SQL statement. In this method query model needs to be constructed in the form of tree-like structure and stored inform of a library which will be used for later comparison when users submit a query for processing. This method requires manual intervention if the anomaly detected is higher than the anomaly detected by the approach.

3.3 Machine Learning Approach

Literature [19] proposed a machine learning method using Bayesian algorithm to detect and prevent SQL injection attack. When user issue a dynamic SQL query from website URL, the monitor intercepts SQL query, break it into number of keywords based on blank space in the dynamic query and calculates length of dynamic SQL query. In addition it also calculated calculate the number of keywords present in such query and send numeric values of length and keyword of dynamic query to classifier. The classifier then calculates the probability of SQL injection in dynamic query based on result received by converter, and then compares probability of SQL injection calculated with one defined by user threshold as training dataset which help in computing of the probability of legitimate query and probability of malicious query. If probability of dynamic SQL query calculated by classifier matched probability of legitimate query compute in training dataset the query is allowed; otherwise it is blocked. One important thing in this method is it simulates a high number of attack patterns in training data including blind SQL injection attack which is very difficult to address.

Literature [20] method to detect and prevent SQL injection attacks which overcome some weakness in [19]. In this method similar blank space method was used in breaking SQL statement into keywords, by calculating the number of keywords in dynamic query. Each keyword is assigned score number defending on the severity of the keywords. This method do not addressed the issue of stacked query which result in false positive if supported by database engine.

Literature [21] proposed a method that uses machine learning to detect and prevent SQL injection attack. In this method training dataset was constructed by analyzing source code program of the application and calculating the entropy of static SQL query. The main purpose of entropy is to count the average amount of information needed to identify the class model of a training dataset. In this case entropy of all static queries that are implemented in a website was calculated which was used to construct a training dataset which will be used later for comparison. When user issue a SQL query the entropy of dynamic SQL query is calculated and compared with entropy in training data. If a match is found the query is allowed to execute in database engine; otherwise it is blocked and prevented from parsing to the database engine. Using entropy in machine learning to classify queries has advantages over using probability as used in [19] and [20] because it produced better results when data are categorized instead of using continuous-valued. Also small changes in SQL query will have a great effect when entropy of that query is calculated. The disadvantage of this method is that it requires analysis of source code of application.

3.3 Parameterized Query and Stored Procedure

Parameterized query is a type of query which has some placeholders. In these queries instead of making dynamic queries by concatenating the parameters with SQL statement, it will replace the placeholders with the value of parameters at runtime. Using stored procedures can also be effective to protect SQL injection attack. Because they check the type of parameters, if the attacker passes a wrong type of value to the stored procedure, then an exception will be thrown but these exceptions should handled properly; otherwise some design secrets of database are exposed to attackers. In fact stored procedures independently cannot eliminate the SQL injection but they do hide the structure of the database from the attacker[22].

IV. EVALUATION

This section, we evaluate the method described in section 3 with respect to deployment, injection mechanism and attack type each method was able to address.

4.1 Evaluation Based on Deployment

We analyze each method as shown in Table 1 with its ability to be implemented in injection mechanisms that we described in Section 3. Because not all of the approaches can be implemented in all injection parameter we also analyzed each of the methods based on different deployment requirements. We evaluated each method with respect to the following criteria:

(1) Does the method require developers to modify their code base?

(2) What is the degree of automation of the detection aspect of the approach?

(3) What is the degree of automation of the prevention aspect of the approach?
(4) What is the degree of attack resistance associated with that approach?
(5) What is the degree throughput effect related to data processing when such approaches are implemented in particular application?

4.2 Evaluation Base on the Attack Type

We analyzed and evaluated each proposed method as shown in table 2 in this section to assess whether it is capable of addressing a particular attack we described in section 2.3. Evaluation was done analytically based on our experience. We have not assessed any of the methods in real time practices as most of method implementation codes are not available or some methods are not implemented.
Therefore, our assessment of these methods is optimistic compared to what their performance may be in practice. We used " ✓ " to indicate that a technique can successfully stop all attacks of that type and

Table 1: Evaluation Based on Deployment

METHOD	URL	LOGIN	SEARCH	DETECT	PREVENT	MODIFY CODE BASE	RESISTANCE TO ATTACK	ACCURACY
CRYPTOGRAPHY	X	✓	X	X	✓	✓	HIGH	HIGH
PATTERN MATCHING	✓	✓	✓		✓	X	MEDIUM	DEPEND ON FILTER PATTERN
XML	✓	✓	✓	X	✓	X	LOW	HIGH
MACHINE LEARNING	✓	✓	✓	✓	✓	✓	MEDIUM	DEPEND ON TRAINING DATA
PARSING	✓	✓	✓	✓	✓	✓	HIGH	DEPEND ON TREE STRUCTURE

✓ " indicate method can be deployed to that injection parameter.
"x" indicate method cannot be deployed to that injection parameter

Table 2: Evaluation Based on Attack Type

METHOD	Tautology	Illegal/incorrect	Piggy-backend	Inference	Alternate encode	Stored procedure	Union
Literature[2]	✓	✓	X	✓	X	X	X
Literature [3]	✓	✓	X	✓	X	X	X
Literature [4]	✓	✓	X	✓	X	X	X
Literature[5]	✓	✓	X	X	X	X	X
Literature[6]	✓	✓	✓	x	✓	x	✓
Literature[7]	✓	✓	✓	x	✓	x	✓
Literature[8]	✓	✓	✓	x	✓	x	✓
Literature[9]	✓	✓	✓	x	✓	✓	✓
Literature[10]	✓	✓	X	✓	X	✓	✓
Literature[11]	✓	✓	✓	✓	✓	X	✓
Literature[12]	✓	✓	✓	X	✓	X	✓
Literature[13]	✓	✓	✓	X	✓	X	✓
Literature[14]	✓	✓	✓	X	✓	✓	✓
Literature[15]	✓	✓	✓	X	✓	✓	✓
Literature[16]	✓	✓	✓	✓	✓	✓	✓
Literature[17]		✓	✓	X	✓	X	✓
Literature[18]	✓	✓	✓	X	✓	X	✓
Literature[19]	✓	✓	✓	x	✓	✓	✓
Literature[20]	✓	✓	✓	x	✓	x	✓
Literature[21]	✓	✓	✓	X	✓	X	✓

✓ Indicate method can stop attack of that type.

"x" indicate method cannot stop attack of that type.

V. RECOMMENDATION

Cryptographic hash algorithm was never meant to protect against SQIA. Rather it was designed to protect confidentiality and data integrity. It is now use to achieve SQLIA security. However there are some issues that are needed to be considered when using cryptographic method to protect attack against bypassing authentication in web applications.

1. There is no advantage in mixing encryption with hash to protect bypassing application because we are not trying to preserved confidentiality or integrity of data but rather we are trying to prevent database engines from executing statement like "1 OR 1'='1--, that is being used by attackers to bypass web applications. The idea was to convert user credentials into another form before sending to the database engine in the sense that even if input contain "1 OR 1'='1--, will not be execute as intended by attacker.
2. Using AES encryption to achieve this goal will consume much more time and resources as it will cost database engines average of 2.5 time's normal processing time.
3. The Last issue to consider is that equality and order, which must be follow. For example hashing then encrypt is not equal to encrypt then hash and also encrypt hash is not equal to hash and vice-verse.

H(E(credential) != E(H(credential) and E(H(credential)!=(H(credential)

XML approach; Implementing multiple layers of security is the best way to improve the security of the system. However such layers need to be design in such way that each layer is independent of each other. This mean that even if an attacker manages to breach one layers it will not result in compromising subsequent layers, second issue is using XML as validation and data storage. Despite the advent of JSON (JavaScript Object Notation) as lightweight means of communicating data between client and database engine, XML remains a viable and popular alternative that is often supported in parallel to JSON by web services API. Some issues that need to be considered include:

1. Avoiding default handling of XML parser needs to be considered when using XML-based authentication, to avoid injecting code directly into the Xpath query evaluation engine which allows attackers to bypass authentication, accessing restricted data from the XML data source.
2. Avoiding using XML as temporary storage of user credentials and retrieving user credentials at runtime validation is far better than using XML to store user credentials permanently and using XML schema to validate format of user query in term of security and performance.

Machine learning, is the one of the common methods used in classifying data to the class into which it belongs. When using machine learning for example Bayesian classification method that predict result based on training dataset defined and unknown dataset, these are some issues to consider.

1. Bayesian classifier produced better results when the attribute values are categorically defined compared to using continuous-valued.

2 Since the accuracy of this method depends on training dataset it is important to accurately define sets of pattern and their associated class model.

3 during training phase it is very important to develop sub class model for each users of the system since different user have different privilege on the system.

When it comes to time consideration, parse tree validation take less time compared to hash or XML approach. However, there are some issues that need to be considered before building SQL tree-like structure to validate dynamic query.

1. As we mentioned in machine learning method sub class model is required in order to differentiate user role into the system, but in parse tree method it is mandatory to have two SQL tree-like structures with syntax that address malicious and non-malicious queries to prevent attackers from executing piggy backend query because some database engines support execution of stacked query.
2. When using parse tree method, the problem of database fingerprinting attack might not be addressed. Thus, it is recommended to add one more layer of security that does not allow revealing secrets design of database. This goal can be achieved by turning off or changing default configuration error messages return by database server when anomaly occurred. Another option is to use stored procedures to conceal the server default error and provide customized error to user but also be aware store procedures are vulnerable as source code and option three use filter method to filter all keywords characters that attackers used to perform database finger fingerprinting.

Pattern matching method is similar to the machine learning method in which pattern model has to be developed that would be used for later comparison. Issues to consider when using pattern matching securing application are as follows.

1. Attack representation: same types of attack have different representation and knowing different attack representation help to design an effective pattern model. A Slight mismatch pattern might results in successful attack.
2. Always group similar attack and create pattern for such attack and lastly create master pattern model which will help in management of pattern incase if new attack types is found.

VI. CONCLUSION

In this paper we present background of SQLIA. We introduced and classify, survey different existing methods implementing such approach, identified some common issues related to each approach as well as programmer mistakes. We also evaluated each approach based on it inherited deployment requirement and evaluated method based on type of attack it is able to address. We also provide recommendation that helps programmer to reduced error in implementing defensive code in web applications source codes.

VII. ACKNOWLEDGEMENT

This research was fully supported by "Universiti
Teknologi Malaysia" Johor Bahru, Malaysia. Authors would like to acknowledge Faculty of Computing for supporting this work.

REFERENCES

[2] Mihir Gandhi, JwalantBaria 2013. SQL INJECTION Attacks in Web Application International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6.
[3] Raghav Kukreja , Nitin Garg 2014. OVERVIEW OF SQL INJECTION ATTACK international journal of innovative research in technology Volume 1 Issue 5.
[4] Ms. Mira K. Sadar et al 2014. Securing Web Application against SQL Injection Attack: a Review International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169 Volume: 2 Issue: 3
[5] Neha Mishra , Sunita Gond 2013. Defenses To Protect Against SQL Injection Attacks International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 10
[6] R. Joseph Manoj et al 2014. An Approach to Detect and Prevent Tautology Type SQL Injection in Web Service Based on XSchema validation. International Journal Of Engineering And Computer Science ISSN: 2319-7242 Volume 3 Issue 1
[7] Shrivastava, Rahul, Joy Bhattacharyji, and RoopaliSoni2013."SQL INJECTION ATTACKS IN DATABASE USING WEB SERVICE: DETECTION AND PREVENTION–REVIEW." Asian Journal of Computer Science & Information Technology 2.6
[8] IndraniBalasundaram, E. Ramaraj 2011. "An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service." IJCSNS International Journal of Computer Science and Network Security 11.1 95-100.
[9] Borade, Monali R., and Neeta A. Deshpande2014. "Web Services Based SQL Injection Detection and Prevention System for Web Applications." International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 4, Issue
[10] Indrani, B., and E. Ramaraj.(2011) "X–LOG AUTHENTICATION TECHNIQUE TO PREVENT SQL INJECTION ATTACKS." International Journal of Information Technology and Knowledge Management 4.1 : 323-328.
[11] Das, Debasish, Utpal Sharma, and D. K. Bhattacharyya (2010) "An Approach to Detection of SQL Injection Attack Based on Dynamic Query Matching."International Journal of Computer Applications 28-34.
[12] Prabakar, M. Amutha, M. Karthikeyan, and K. Marimuthu 2013. "An efficient technique for preventing SQL injection attack using pattern matching algorithm." Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on. IEEE,
[13] Narayanan, Sandeep Nair, AlwynRoshanPais, and Radhesh Mohandas 2011."Detection and Prevention of SQL Injection Attacks Using Semantic Equivalence." Computer Networks and Intelligent Computing. Springer Berlin Heidelberg, 2011. 103-112.
[14] Ms. Zeinab Raveshi and Mrs. Sonali R. Idate 2013 Efficient Method to Secure Web applications and Databases against SQL Injection Attacks
[15] Manmadhan, Sruthy, and T. Manesh(2012) "A method of detecting sql injection attack to secure web applications." International Journal of Distributed and Parallel Systems 3 1-8.

[16] Kumar, Kuldeep, Debasish Jena, and Ravi Kumar(2013). "A Novel Approach to detect SQL injection in web applications." International Journal of Application or Innovation in Engineering & Management (IJAIEM) Volume 2, Issue ISSN 2319 - 4847

[17] Bangre, Shruti, and AlkaJaiswal(2012) "SQL Injection Detection and Prevention Using Input Filter Technique." International Journal of Recent Technology and Engineering (IJRTE)145-149.

[18] Shi, Cong-cong, et al.(2012) "A New Approach for SQL-Injection Detection."Instrumentation, Measurement, Circuits and Systems. Springer Berlin Heidelberg,. 245-254.

[19] Cheon, Eun Hong, Zhongyue Huang, and Yon Sik Lee(2013). "Preventing SQL Injection Attack Based on Machine Learning." International Journal of Advancements in Computing Technology 5.9

[20] Joshi, Anamika, and V. Geetha.(2014) "SQL Injection detection using machine learning." Control, Instrumentation, Communication and Computational Technologies (ICCICCT), International Conference on. IEEE, 2014.

[21] Shahriar, Hossain, and Mohammad Zulkernine. "Information-theoretic detection of sql injection attacks." High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on. IEEE, 2012.

[22] Aliero, Muhammad Saidu, et al. (2015). "REVIEW ON SQL INJECTION PROTECTION METHODS AND TOOLS." Jurnal Teknologi Vol. 77.No.13