International Journal of Soft Computing 11 (1): 26-35, 2016
ISSN: 1816-9503

Systematic Literature Review on SQL Injection Attack

M.A. Lawal, Abu Bakar Md. Sultan and Ayanloye O. Shakiru
Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

Abstract: SQL injection attack is a common threat to web applications that utilizes poor input validation to implement attack on a target database. It is becoming a very serious problem in web applications as successful execution leads to loss of integrity and confidentiality and this makes it a very sensitive issue of software security. The study presents a Systematic Literature Review (SLR) on SQL Injection Attacks (SQLIA) following Kitchenham's procedure of performing systematic literature review. This study gives a review on SQL injection attack, detection and prevention techniques. In the end, an evaluation of the techniques is carried out to check the effectiveness of each technique based on how many method of attack it can detect and prevent. It is imperative to note that a good number of the evaluated techniques were able to detect and prevent all types of SQLIA based on the selected criteria. To determine the best technique resources such as memory and processing time need to be considered in the evaluation.

Key words: SQL injection attack, detection, prevention, software security, systematic literature review

INTRODUCTION

SQL Injection Attack (SQLIA) is a type of vulnerability that target database connected web applications where by malicious codes are inserted, processed and executed. This vulnerability exists when the web application does not perform a proper input validation. A poorly designed web application can be attacked by inserting malicious code in order to get access to the database. There are several places where users can input data in web applications, each of which can provide a SQL injection attack opportunity that can lead to loss of confidentiality, integrity and market value of an organization (Kumar and Pateriya, 2012).

The broad implementation of security infrastructures such as intrusion prevention systems for protecting enterprise information systems has recently made remote unauthorized access of connected enterprise applications difficult. However, these safeguarding steps are being invaded and beaten without difficulty using simple scripting language. SQL injection is an example of this technique. In such a susceptible application, an SQL command injection attack uses crooked inputs that changes the SQL query and establish illegal connection to the database. These types of attacks are intensely common and graded as one of the most common form of attack on web applications (Bisht *et al*., 2010).

Many security procedures on the server side exist but they are not helpful in large scale because of the deployment complexity. And on the client side, installing of security applications deteriorates the client systems performance which thus decreases the web browsing satisfaction of the client (Suguna *et al*., 2014).

Detection of SQL injection before it happen is difficult. Several cases of such illegal intrusion is carried out by the attacker using valid user credentials or by using built-in characteristics of database application such as virulent modification of current SQL queries of web application that are penetrating the delicate parts of the changed databases (Johari and Sharma, 2012).

The purpose of this study is to review the issue of SQL injection attacks detection and prevention. The review is based on four questions asked and answered.

MATERIALS AND METHODS

This study has been undertaken as a systematic literature review based on the guidelines provided by Kitchenham *et al*. (2009). The steps followed in this study are elaborated below.

Research questions: This study attempts to address the research questions:

- RQ1: what are the reasons and effects of SQL injection attacks?
- RQ2: what are the currently and widely used SQL injection techniques?
- RQ3: what are the widely used SQLIA detection and prevention techniques?
- RQ4: how effective are these techniques in detecting and preventing the SQL injection attacks?

Corresponding Author: M.A. Lawal, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

Regarding RQ1, the study will address the reasons why a malicious user would use SQL injection to attack a database. It also addresses the effects of such attacks. This is will give an insight of the consequences if such attacks are successful.

In RQ2, the study looks at various methods used in executing attacks on database by utilizing user input fields on web applications. RQ3 will discuss the SQLIA detection and prevention techniques that are currently and widely used.

Our final research question RQ4 will compare the techniques to check their effectiveness by checking how many methods of attacks mentioned in RQ2 a particular technique in RQ3 is able to detect and prevent.

Search procedure: The study used for the SLR are obtained through online search (Universit Putra Malaysia (UPM) Lib and Google scholar). The search was done with keywords like, software security, SQL Injection Attack (SQLIA), SQL injection detection and prevention. Only relevant articles were saved for further review. Table 1 shows a summary of the source and number of articles selected from the digital libraries.

Inclusion and exclusion criteria

Inclusion:

- Journals and conferences related to SQLIA detection or prevention
- Journals and conferences with our search keywords
- Surveys which are related to SQLIA
- Surveys that are related to SQLIA detection or prevention
- One study not related to SQLIA but included in Kitchehams' SLR which is used as a guideline for this study

Exclusion:

- Articles that were publish before 2005
- Articles publish in another language

Quality assessment: Since, the study follows Kitchenham *et al.* (2009) method for conducting systematic literature review, all articles were evaluated using Center for Reviews and Dissemination (CDR) and Database of Abstracts of Reviews of Effects (DARE) criteria. The (QA) questions for the quality assessment are focused on four main criteria:

Table 1: Summary of selected articles from digital libraries

Sources	No. of articles selected
IEEE Xplore	7
Google Scholar	6
Acm Digital Library	4
Scopus	1
Science Direct	2
Total Relevant Articles	20

- Q1: were the inclusion and exclusion criteria well described and appropriate in the reviews?
- Q2: as all related and relevant studies covered during the literature search?
- Q3: as quality and validity for the studies included properly assessed by the reviewers?
- Q4: was there adequate description of the vital information/studies?

Scores of the questions are as follows:

- QA1: Y (Yes), the criteria for inclusion is properly explained in the study clearly. P (Partly), criteria for inclusion is contained in the study. N (No), the criteria for inclusion are not explained and cannot be inferred anytime
- QA2: Y (Yes), at least 4 digital libraries have been searched by the author and other search strategies exhausted or every journal that is addressing the area of interest identified and referenced. P (Partly), 3 or 4 digital libraries have been searched by researcher without attempting other search strategies or search restricted articles that are defined. N (No), at most 2 digital libraries were looked up by researchers or highly restricted journals were used for the literature search
- QA3: Y (Yes), quality and validity criteria were clearly defined and carefully separated from the initial study. P (Partly), quality and validity issues discussed by the paper are in the research question. N (No), quality and validation assessment of the initial studies was not attempted clearly
- QA4: Y (Yes), full information on all studies carried out is produced. P (Partly), summary information of the initial studies was the only thing produced. N (No), there was no specified results of the individual initial studies carried out

The grading process used for the evaluation is Y = 1 (represents information that has been clearly and fully specified), P = 0.5 (represents information that has been specified partly or shows some effort), N = 0 (represents information that is not specified). The process of quality evaluation is done by researchers using kitchenham SLR Method as a reference and guideline.

Data collection: Information taken from the study of each paper is:

- Journal or conference sources and references
- Study type classification
- Focus of the topic area
- A full summary of each study
- The technique used for detecting and preventing SQL injection attacks
- The analysis of the effectiveness of the technique if provided

Data collection was carefully done by the researchers. The data was first extracted and then reviewed over to ensure its eligibility to be included in this systematic literature review.

Data analysis:

- The data is summarized and organized tabular form
- The reasons and effects of SQLIA (RQ1) and the techniques used for detecting and preventing these attacks (RQ3) are summarized
- The widely used techniques for executing the attacks (RQ2) and the analysis of the effectiveness of the methods proposed for detecting and preventing the attacks (RQ4) are tabulated

RESULTS AND DISCUSSION

Search results: Table 2 is the lists all the resources obtained after performing the search procedure. There are articles that proposed a technique for SQL injection attack.

Table 2: List of selected studies

ID	Authors	Type of article
S1	Kumar and Pateriya (2012)	Conference
S2	Halfond and Orso (2006)	Conference
S3	Liu *et al.* (2009)	Conference
S4	Bisht *et al.* (2010)	Journal
S5	Al-Khashab *et al.* (2011)	Conference
S6	Rahul *et al.* (2012)	Journal
S7	Fu and Qian (2008)	Conference
S8	Grazie (2008)	Thesis
S9	Indrani and Ramaraj (2011)	Journal
S10	Jiao *et al.* (2012)	Conference
S11	Tajpour *et al.* (2010a)	Conference
S12	Indrani and Ramaraj (2011)	Conference
S13	Natarajan and Subramani (2012)	Conference
S14	Halfond and Orso (2006)	Conference
S15	Ezumalai and Aghila (2009)	Conference
S16	Halder and Cortesi (2010)	Conference
S17	Johari and Sharma (2012)	Conference
S18	Amirtahmasebi *et al.* (2009)	Conference
S19	Kindy and Pathan (2011)	Conference
S20	Tajpour *et al.* (2010b)	Conference

While some articles made a survey of the techniques used in mitigating the attack. The list of selected relevant studies is shown in Table 2.

Quality evaluation of studies: Quality and validity assessment was done based on the DARE criteria mentioned earlier in quality assessment of the study. Table 3 shows the grade of each study. For the representation of the fields in the table, refer to quality assessment. Based on the outcome of the result, 13 out of 20 articles scored 4 points, 2 scored 3.5 and 5 of the studies scored 3 points. This show a higher number of the articles satisfy the quality assessment questions on the DARE scale.

The focus of this study is answering our research questions. RQ1: What are the reasons and effects of SQL injection attack?

Reasons for SQL injection attack: Below are various reasons or intent of an attacker for using SQL injection (Halfond *et al.*, 2006a, b).

Data extraction: The main goal of data extraction is to get access to data values using a back end of the database. The sensitivity of the information is extracted is dependent on the web application type. Medical records, credit cards, personal information are all examples of data that are very valuable to an attacker.

Data modification: The intent of an attacker here is to get access to the database in such a way that he has the permission to ADD, REMOVE, INSERT, DELETE or UPDATE the values of any record or data in the database.

Database finger printing: An attacker would finger print a database for the purpose of retrieving certain technical

Table 3: Quality evaluation

ID	QA1	QA2	QA3	QA4	Total
S1	Y	Y	Y	Y	4.0
S2	Y	Y	Y	Y	4.0
S3	Y	Y	Y	Y	4.0
S4	Y	Y	Y	Y	4.0
S5	Y	Y	Y	Y	4.0
S6	Y	Y	P	Y	3.5
S7	Y	Y	P	Y	3.5
S8	Y	Y	Y	Y	4.0
S9	Y	Y	Y	Y	4.0
S10	Y	Y	N	Y	3.0
S11	Y	Y	Y	Y	4.0
S12	Y	Y	N	Y	3.0
S13	Y	Y	Y	Y	4.0
S14	Y	Y	Y	Y	4.0
S15	Y	Y	N	Y	3.0
S16	Y	Y	N	Y	3.0
S17	Y	Y	N	Y	3.0
S18	Y	Y	Y	Y	4.0
S19	Y	Y	Y	Y	4.0
S20	Y	Y	Y	Y	4.0

information that are vital and specific to that database. For example, the version of the database that a particular web application is running. Different database respond in different ways to attacks and queries. This information gives so much detail that it is used to finger print the database. Once knowledge of the database and its version is known, an attacker can execute attacks.

Bypassing authentication: This is one of the most popular intents of attackers. The aim of this attack is to bypass particular web application and gain access to the database. Once access is gained, it would give an attacker the same rights and privileges as a genuine user of the web application.

Identifying injectable parameters: The intent of an attacker here is to test or check which user inputs fields are exploitable or vulnerable to SQL injection attack. Identification of these vulnerabilities can be done by using an automated tool called vulnerability scanner.

Determining database schema: The intent of an attacker here is to get information on the database schema. Database schemas contain precise information on the name of the tables, names of columns and the data type contained in the columns. Attackers use knowledge of this information for successful extraction of data from the database. Attackers use special tools like penetration tester and vulnerability scanners to achieve their goal.

Evading detection: Many SQL database try to implement some form of detection or prevention technique as defense in attempt to stop attackers from exploiting their system. The intent of an attacker here is trying to hide or avoiding detection. The defensive mechanism attacker try to hide from are usually, defensive coding practices and most automated detection and prevention techniques.

Performing Denial-of-service: The intent of an attacker here would be to interrupt or disturb the services of a system by executing some command in the database to shutdown the web application which then causes a Denial of service. Dropping and locking database tables are also considered Denial of service attacks.

Executing remote commands: The intent of an attacker here would be attempting to gain complete control of the whole system. This is the most dangerous form of attack. This is done by executing illogical commands on the database. Database users have functions or stored procedure commands available to them. This feature is exploited by attackers. For example, take stored procedure xp-cmdshell in Microsoft SQL servers which allows execution of illogical commands. If this command is successfully executed by the attacker, it is unavoidable that the integrity of the server is lost and the server would utterly be compromised.

RQ. 1.2 (Effects of SQL injection): Successful SQL injection attack can be very devastating. There is no limit to how much harm can be done by an attacker. This research question discusses the effect of SQL injection attack and what is at risk if a system is successfully breached using SQL injection attack. Johari and Sharma (2012) have outlined the consequences of SQLIA below.

Authorization: A successful SQLIA can allow an attacker to change user privilege on the web application. The authorization to carry certain operations on the database can be changed. Vital information stored on database may be altered if unauthorized access to the SQL database is gained through vulnerabilities in the database.

Authentication: When username and password are not validated properly, the consequences would be devastating. Anyone would be capable of gaining access to the system without knowing the right username and password.

Confidentiality: A successful SQLIA would violate the confidentiality expected to be derived from storing data in a database because databases are usually used to store delicate and important information. This information must be kept out of the wrong hands. If a system fails, confidentiality of the database is lost and this becomes a problem.

Integrity: When an attacker is able to change or remove the contents of a database, this results to loss of system integrity. When integrity is compromised, false records can also be created.

RQ2. What are the currently and widely used SQL injection techniques?: Table 4 shows the techniques, description and examples of the widely used SQL injection techniques as described by Halfond *et al.* (2006a, b), Kumar and Pateriya (2012), Tajpour *et al.* (2011, 2010a) and Kindy and Pathan (2011).

RQ3. What are the widely used SQLIA detection and prevention techniques?: Most developers try to achieve SQLIA detection or prevention by doing what is known as defensive coding or OS hardening. Usually, these will not be enough to stop SQLIA in web applications. We will discuss the dominant techniques.

Table 4: Techniques used for executing SQLIA

Techniques	Description	Example
Tautologies	This attack focuses on manipulation of the SQL database. It injects a query into the database by forcing the condition statement to return all true values. This gives an attacker all the information of the table. Once access to the table is gained, picking out the user name and password of the users stored on the database becomes easy	Example of a normal login query for a User Info table: SELECT * FROM User_Info WHERE UserName = 'johndoe1' and Password = '8882906' Example of a MALICIOUS login query for a User Info table: SELECT * FROM User_Info WHERE UserName = "OR 1 = '1' and Password = 'I 4444 The statement for the malicious login attempt is forcing the table to return all true values for UserName and nullifying the password requirement
Logically Incorrect Queries (LIQ)	SQL database servers return error messages, these messages can be exploited in a malicious way. An attacker inputs a manipulated query into the database with the intention of making it fail so it can generate an error message. The error message generated always shows some information about the cause of the error. Error messages generated most time show table names, columns and sometimes the conditions for failure. This information is too much as it already gives an attacker an idea of what you database schema looks like which shouldn't be the case	Example of a query required for extraction of table name from a database: SELECT *FROM User_info WHERE UserName = "HAVING 1 = '1' and Password = '554181' Attackers can get error message like these with the use of this type of queries: "Column 'User_info. UserID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause" Name of table which is "User_info" and name of column which is "User_info. UserID" are both clearly being indicated in the error message. This should not be the case as it gives away too much information. Example of a query required for extraction of column names from a table: SELECT * FROM User_info WHERE UserName = "GROUP BY UserID HAVING 1 = '1' and Password = '554181' Attackers can get error message like these with the use of this type of queries: "Column 'User_info.Username' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause" Name of table column which is "User_info. Username" and name of column which is "User_info. UserID" are both clearly being indicated in the error message. This should not be the case as it gives away too much information
Union query	Attackers use this type of attack to get the information on the data type in the table columns from the database. Usually code injection and manipulation of information are the motivation behind attacks like these. This attack adds malicious codes or inserts codes which are safe query	Example of a union query attack on a database server is: SELECT * FROM User_info WHERE UserName = "UNION SELECT SUM (Username) from User_info. BY UserID HAVING 1 = '1' and Password = '551244'. This would generate an error message which looks like the one shown: "Operand data type varchar is invalid for s urn operator." It is very clear from the error message that the datatype of Username from the table is VARCHAR. Repeating this several times would help an attacker determine the datatype of all fields on the table.
Stored procedure	Attacks using stored procedures are carried out by implementing in-built procedure calls. It is usually done by injecting SQL function calls into the database	Example procedure stored in SQL server is shown below: ALTER PROCEDURE shaxAccessProc (@select NVARCHAR(500), @tableName NVARCHAR(500), @where NVARCHAR(500) AS BEGIN EXEC (SELECT'+@select+ 'FROM'+@tableName+ 'WHERE'+@where) END GO EXEC shaxAccessProc, *FROM INFORMATION SCHEMA. TABLES FOR XML RAW-',' TEST','PUSPENDRA'. Vulnerability in this procedure shows when an attacker uses an INFORMATION_SCHEMA.TABLES. "<rowTABLE CATALOG = "Users" TABLE SCHEMA = "adp" TABLE NAME = "User_info" TABLE TYPE = "BASE TABLE" />" Above is an example of a XML data that contains table information and database schema
Piggy-backed queries attacks	What happens here is the attacker exploits conventional queries by injecting them in a malicious way and performs manipulative activities by using the DELETE, INSERT and UPDATE clause	Example of a piggy-backed query attack is shown below: "SELECT * FROM User_info WHERE UserName = ";INSERT INTO User_info VALUES('pbqa','6543'}-" For as long as the table name is known and is correctly identified, the query above would be inserted into the table in the database. All other SQL operations for manipulation can be done and injected in the same way
Alternate encodings	Attackers manipulate queries by using special characters like ASCII, unicode and hexadecimals because they can evade the filters created by developers. This style of attack is known as alternate encoding. Most times these characters are considered by the filter as bad characters	

Table 4: Continue

Techniques	Description	Example
Inference based attacks	Databases behave differently depending on the result of the queries it produces. There are two types of inference attacks which are TIMING injection and BLIND injection	Timing injection attacks: time delay is the element being exploited by an attacker to collect database information Blind injection attacks: using SQL statements, data is stolen from the database by posing true or false questions to the database

Using positive tainting and syntax-aware evaluation: It is a dynamic and highly automated technique used in protecting web applications from SQLIA. This approach use a dynamic tainting, the application's reliable strings are recognized and this reliable strings are let on to generate a portion of the SQL query. This technique is implemented using a tool developed in java known as Web Application SQL Injection Preventer (WASP) (Kumar and Pateriya, 2012; Tajpour *et al*., 2010a, b; Halfond and Orso, 2006).

SQLProb (A proxy-based architecture towards preventing SQL injection attacks): This method uses a proxy that merges current operational background to protect the backend and front end web servers. It dynamically recognizes and removes malicious SQL control structure from user inputs (Liu *et al*., 2009).

PSIAQOP (Preventing SQL injection attacks based on query optimization process): It prevents SQL Injection Attacks (SQLIAs) that depends on the query optimization process. The technique is based on optimizing the SQL queries that gets input from users and vulnerable in the execution phase. PSIAQOP analyze the source code which is used in the web applications and identify the hotspots that represent the most vulnerable SQL queries to attacks and depends on users to supply the input fields. After that these vulnerable queries go through the optimization process which is based on the heuristics rules. In this case, the query parser first generates the initial internal representation of the vulnerable queries which is mostly in the forms of relational algebra expression to be handled by the optimization engine. The optimization engine then generates a number of valid execution plans according to the heuristics rules to select the optimal plan. Finally, PSIAQOP replace each hotspot with its optimized query in the web application code. The web application is then run its code normally and prevents any SQLIAs and thwarts them from achieving their harmful goals (Al-Khashab *et al*., 2011).

SQLIMW (A new mechanism against SQL-Injection): This technique use defense tool against SQLIA which is SQL injection middle ware, it depends on a system that allows one sign-in on web application.

The combination of hash and encryption components together with XOR validates system and user credentials through a middleware. This middleware is place between the user verification system and the system that stores the user information. This middleware can be placed at any level of web application that interacts with the database (Jiao *et al*., 2012).

X-LOG authentication technique to prevent SQL injection attacks: In this method SQL injection threat is checked at runtime. This method oversees the dynamically created queries with the data model which is created by X-Log generator at runtime and examines them for concurrence. If the data similarity is not the same with the model then it shows a potential SQLIA and stopped from executing on the database and reported (Indrani and Ramaraj, 2011).

An efficient technique for detection and prevention of SQL injection attack using ASCII based string matching: The technique combines static and dynamic analysis in the static analysis stage, the prevention method is divided into three stages:

- The wildcard used by the attacker is stopped by the text detector
- Restriction of log in details, i.e., number of characters to be used as login and password
- ASCII value is created in database for the delicate data as user id ASCII and password ASCII to shield the database (Balasundaram and Ramaraj, 2012)

Amnesia: This method is completely automated in stopping SQLIA. Invalid queries are recognized before being accomplished on the target with help of an approach based on a model. It comprises of a static part in which it automatically builds a model of legitimate queries using program analysis and also a dynamic part in which it inspects and authenticate the queries dynamically created with the statically built model with the help of runtime monitoring. Queries that breach the model are possible SQLIAs and are blocked from execution. The technique comprises of four stages which include recognizing the hotspot, creating SQL-query models, instrument application, runtime, monitoring (Kumar and Pateriya, 2012; Johari and Sharma, 2012; Tajpour *et al*., 2010a, b; Halfond *et al*., 2006b; Rahul *et al*., 2012).

Candid (Dynamic candidate evaluations for automatic prevention of SQL injection attacks): It dynamically and

automatically assemble structure of intended queries for any input (contender inputs) from the legal user and the format of query obtained is checked against that of the malicious input. This method makes the web applications protect themselves from SQLIA (Kumar and Pateriya, 2012; Johari and Sharma, 2012; Kindy and Pathan, 2011; Tajpour *et al*., 2010a, b; Bisht *et al*., 2010).

SQL Rand: This approach utilizes a proxy server placed at the middle of the web server and database server for cracking the queries exchanged between the client and database server. Database server receives set of accepted keywords of the de-randomize SQL queries for computation. The error created by the database server which consist of illegitimate queries is hidden to avoid giving the hacker a chance of getting the architecture of the database tables and schema (Kumar and Pateriya, 2012; Kindy and Pathan, 2011; Tajpour *et al*., 2010a, b; Amirtahmasebi *et al*., 2009).

SQL Dom: This technique uses Call Level Interface (CLI) which is put at the middle of application and database. This solution uses an API (SQL dom gen) which analyses the database. The use of API prompts a systematic procedure of putting queries together. Hence, API administers proper input validation which is verified at compilation (Kumar and Pateriya, 2012; Kindy and Pathan, 2011; Tajpour *et al*., 2010a, b; Amirtahmasebi *et al*., 2009).

Webssari: This approach uses a static analysis to confirm taint flows counter to an arrangement for delicate process. In an event of exposure to threat the runtime guard will be at alert. This technique uses an automated tool that works depending on certified input that goes through a filter (Kumar and Pateriya, 2012; Tajpour *et al*., 2010a, b).

SQL Injection Protector for Authentication (SQLIPA): This technique uses hash value to get better execution performance of validation mechanism for web application. Validation parameters (user ID and password) for the hash value are set up at runtime when accounts are generated (Kumar and Pateriya, 2012; Kindy and Pathan, 2011).

Obfuscation-based analysis of SQL injection attacks: This technique detects the threat before routing the query to the database. It consists of three stages:

- Static stage: SQL Queries of the web application code are changed into obfuscated format
- Dynamic stage: this stage combines the user input and the obfuscated query at runtime. Verification of the obfuscated query at atomic level is carried out to check for SQL injection threat
- The reverse of stage one will be carried out if a threat is not found (Kumar and Pateriya, 2012; Johari and Sharma, 2012)

Context Sensitive String Evaluation (CSSE): This is an automated procedure to stop SQL injection attacks. It uses channel to prevent SQL injection attack that consist of assignment of metadata to user input and metadata preserving string operation and context sensitive string evaluation. Intercommunication with the developer is irrelevant in this technique and alteration in code. The technique needs difference in fundamental framework of the language. Context Sensitive String Evaluation Model works with PHP language (Kumar and Pateriya, 2012).

Generation of SQL-injection free secure algorithm to detect and prevent SQL-Injection attacks (SQL-IF): The proposed algorithm depends on dynamic technique which withstands SQL injection attacks. This algorithm concentrate on to forming Injection Free (IF) attacks where a special type of test suite is generated to detect and prevent SQL injection attacks. It generates algorithm which is merged into the runtime environment while the execution was carried out using Java (Natarajan and Subramani, 2012).

Combinatorial approach for preventing SQL injection attacks: This technique uses an innovative highly automated approach for preventing web applications from SQLIAs. This approach consists of three steps:

- Analyzing hot spot from the application
- Protecting web application code by static analysis and runtime using Hirschberg algorithm to detect the SQL injection attacks
- DBMS is then used for auditing methods to find out the transactions. Hirschberg algorithm uses divide and conquer approach, it decreases time and space factor by discovering SQL injection threats and it supports user authentication to stop the SQL injection attacks execution after analyzing the DBMS auditing (Ezumalai and Aghila, 2009)

Tautology checker: This technique only focuses on tautology attack types and cannot help in detection or prevention of any other type of SQL injection attacks. This makes this technique limited to specific to type of attack and leaving the web application vulnerable to other attacks (Kumar and Pateriya, 2012; Wassermann and Su, 2004).

SAFELI: This is a static analysis framework design. It works by recognition of SQL injection attack

vulnerabilities during compilation time. By performing symbolic execution, SAFELI will statistically observe or monitor what is called, Microsoft Symbolic Intermediate Language (MSIL) byte code for the ASP. NET web application. Analysis of the source code can be done by SAFELI and then it will be possible for SAFELI to perform a proper identification of sensitive weaknesses or vulnerability which could not be found using black-box vulnerability scanners. The downside to this technique is it only detects SQL injection attacks carried out specifically on Microsoft products (Fu and Qian, 2008).

SQL guard and SQL check: These methods examine the query during runtime, relying on a model that only allows legal queries that are expressed as grammars. In the case of SQL guard, the query structure is checked by SQL guard before and after user inputs are added based on the model. While in the case of SQL check, the developer specifies the model independently for SQL check. During parsing by the runtime checker both techniques uses a secret key to restrict a users' input. In both approach a developer should do some modification to the code using an extraordinary intermediate library or manually slot in extraordinary markers in the program code where input required from user is included to a dynamically generated query (Buehrer *et al*., 2005; Su and Wassermann, 2006).

SQL-IDS: This approach concentrates on writing web application specifications which explain the intended

Table 5: Evaluation of detection and prevention techniques

Techniques	Detection/Prevention	Tautologies	Logically incorrect queries	Alternet encoding	Union queries	Piggy-backed	Stored procedure queries	Inferences
Using positive tanting	Detection	√	√	√	√	√	√	√
and syntax evaluation	Prevention	√	√	√	√	√	√	√
SQL-Prob: A proxy	Detection	√	√	√	√	√	√	√
based architecture	Prevention	√	√	√	√	√	√	√
Webssari	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Using ASCII	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
AMNESIA	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL guard	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL rand	Detection	√	X	√	√	√	X	√
	Prevention	√	X	√	√	√	X	√
CSSE	Detection	√	√	√	√	X	X	√
	Prevention	N	N	N	N	N	N	N
SQL prevent	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
SQL-IF	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Obsfucation	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
SQL IPA	Detection	√	√	X	√	√	√	√
	Prevention	√	√	X	√	√	√	√
SQL DOM	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
CANDID	Detection	P	P	P	P	P	P	P
	Prevention	√	X	X	X	X	X	X
SQL-MW	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
X-LOG	Detection	√	N	N	√	√	N	N
	Prevention	√	N	N	√	√	N	N
PSIAQOP	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Tautology checker	Detection	N	√	X	X	X	X	X
	Prevention	X	X	X	X	X	X	X
SAFELLI	Detection	X	√	√	√	√	√	√
	Prevention	N	N	N	N	N	N	N
SQL check	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL IDS	Detection	√	√	√	√	√	√	√
	Prevention	N	N	N	N	N	N	N
Swanddler	Detection	P	P	P	P	P	P	P
	Prevention	N	N	N	N	N	N	N
Combina torial	Detection	N	N	N	N	N	N	N
Approach for preventing SQLIA	Prevention	N	N	N	N	N	N	N

structure of SQL statements produced by the application and automatically monitor SQL statement that are executed for violation in respect to the specification (Kemalis and Tzouramanis, 2008).

Swaddler: This technique examines interior state of web applications. It is very impressive in the way it shows resistance to complex attacks on a web application by using both single and multiple variables. The technique starts by describing normal variables to the application components. It then goes on to monitor the execution of the application in other to find irregular states; this is done during the detection phase (Cova *et al*., 2007).

SQL prevent: This technique uses a HTTP request interceptor. Modification of the main data flowing is done when SQL prevent is installed into a web server. The process starts by saving the HTTP requests into a current thread-local storage then a SQL interceptor is used in intercepting SQL statements created by a the web application and transfers these SQL statements to a SQL injection attack detector module. Reason for this is so the HTTP request from thread-local storage can be gotten and checked to see if SQL injection attack is contained in it. It would then prevent whatever malicious SQL statement found in it from being sent to the database, if it is suspected to be a SQL injection attack (Grazie, 2008).

RQ4. How effective are the techniques for preventing the SQL injection attacks?: Effectiveness of the techniques was evaluated by checking how many attacks a particular technique is able to detect and prevent as shown in Table 5. As discussed in RQ2, we have seven type of attacks. Most of the techniques used a testbed of web application for the evaluation:

- ✓ signifies that the technique can detect or prevent the type of attack
- X signifies that the technique cannot detect or prevent the attack
- N signifies that the study did not define any form of evaluation using any type of attack
- P signifies that the technique can partially detect or prevent the attack

However, some of the articles used the performance overhead to evaluate the techniques, i.e., the system resources used in execution. Example: memory, processing time.

CONCLUSION

This study presents the result of a systematic literature review on SQL injection attack using Kitchenhams' systematic literature review as a guideline.

The reasons and effects of SQLIA were discussed. The various types of SQLIA techniques were identified and described and also examples of the syntax used for execution were given. Furthermore, the most common detection and prevention techniques for SQLIA were summarized. Finally an evaluation was done to check the effectiveness of the SQLIA prevention and detection techniques against the SQLIA types. However, the evaluation was based on prevention and detection only without considering the requirements for implementing the technique.

Our future concern is an evaluation based on resources needed to implement the SQLIA prevention and detection techniques. Evaluating methods used to protect web applications from other threats suchas cross site scripting and buffer overflow is also part of our research direction.

ACKNOWLEDGEMENT

Researchers gratefully acknowledge the support of Faculty of Computer Science and Information Technology, Universiti Putra Malaysia.

REFERENCES

Al-Khashab, E., F.S. Al-Anzi and A.A. Salman, 2011. PSIAQOP: Preventing SQL injection attacks based on query optimization process. Proceedings of the 2nd Kuwait Conference on E-Services and E-Systems, April 5-7, 2011, Kuwait, USA.

Amirtahmasebi, K., S.R. Jalalinia and S. Khadem, 2009. A survey of SQL injection defense mechanisms. Proceedings of the International Conference for Internet Technology and Secured Transactions, November 9-12, 2009, London, pp: 1-8.

Balasundaram, I. and E. Ramaraj, 2012. An efficient technique for detection and prevention of SQL injection attack using ASCII based string matching. Procedia Eng., 30: 183-190.

Bisht, P., P. Madhusudan and V.N. Venkatakrishnan, 2010. Candid: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Trans. Inf. Syst. Security, 5: 1-39.

Buehrer, G., B.W. Weide and P.A.G. Sivilotti, 2005. Using parse tree validation to prevent SQL injection attacks. Proceedings of the 5th International Workshop on Software Engineering and Middleware, September 5-6, 2005, Lisbon, Portugal, pp: 106-113.

Cova, M., D. Balzarotti, V. Felmetsger and G. Vigna, 2007. Swaddler: An approach for the anomaly-based detection of state violations in web applications. Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection, September 5-7, 2007, Gold Goast, Australia, pp: 63-86.

Ezumalai, R. and G. Aghila, 2009. Combinatorial approach for preventing SQL injection attacks. Proceedings of the International Advance Computing Conference, March 6-7, 2009, Patiala, India, pp: 1212-1217.

Fu, X. and K. Qian, 2008. SAFELI: SQL injection scanner using symbolic execution. Proceedings of the Workshop on Testing, Analysis and Verification of Web Services and Applications, July 20-24, 2008, Seattle, WA., USA., pp: 34-39.

Grazie, P., 2008. SQL prevent. Ph.D. Thesis, University of British Columbia, Vancouver, Canada.

Halder, R. and A. Cortesi, 2010. Obfuscation-based analysis of SQL injection attacks. Proceedings of the IEEE Symposium on Computers and Communications, June 22-25, 2010, Riccione, Italy, pp: 931-938.

Halfond, W.G.J. and A. Orso, 2006. Preventing SQL injection attacks using AMNESIA. Proceedings of the 28th International Conference on Software Engineering, May 20-28, 2006, Shanghai China, pp: 795-798.

Halfond, W.G.J., A. Orso and P. Manolios, 2006a. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. Proceedings of the 14th ACM Sigsoft International Symposium on Foundations of Software Engineering, November 5-11, 2006, Portland, OR, USA., pp: 175-185.

Halfond, W.G.J., J. Viegas and A. Orso, 2006b. A classification of SQL injection attacks and countermeasures. Proceedings of the International Symposium on Secure Software Engineering, March 2006, New York, USA.

Indrani, B. and E. Ramaraj, 2011. X-Log authentication technique to prevent SQL injection attacks. Int. J. Inform. Technol. Knowledge Manage., 4: 323-328.

Jiao, G., C.M. Xu and J. Maohua, 2012. SQLIMW: A new mechanism against SQL-injection. Proceedings of the International Conference on Computer Science and Service System, August 11-13, 2012, Nanjing, pp: 1178-1180.

Johari, R. and P. Sharma, 2012. A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. Proceedings of the International Conference on Communication Systems and Network Technologies, May 11-13, 2012, Rajkot, pp: 453-458.

Kemalis, K. and T. Tzouramanis, 2008. SQL-IDS: A specification-based approach for SQL-injection detection. Proceedings of the 2008 ACM Symposium on Applied Computing, March 16-20, 2008, Fortaleza, Ceara, Brazil, pp: 2153-2158.

Kindy, D.A. and A.S.K. Pathan, 2011. A survey on SQL injection: Vulnerabilities, attacks and prevention techniques. Proceedings of the IEEE 15th International Symposium on Consumer Electronics, June 14-17, 2011, Singapore, pp: 468-471.

Kitchenham, B., O.P Brereton, D. Budgen, M. Turner, J. Bailey and S. Linkman, 2009. Systematic literature reviews in software engineering-A systematic literature review. Inform. Software Technol., 51: 7-15.

Kumar, P. and R.K. Pateriya, 2012. A survey on SQL injection attacks, detection and prevention techniques. Proceedings of the 3rd International Conference Computing Communication and Networking Technologies, July 26-28, 2012, Coimbatore, India, pp: 1-5.

Liu, A., Y. Yuan, D. Wijesekera and A. Stavrou, 2009. SQLProb: A proxy-based architecture towards preventing SQL injection attacks. Proceedings of the ACM Symposium on Applied Computing, March 8-12, 2009, Honolulu, HI., USA., pp: 2054-2061.

Natarajan, K., and S. Subramani, 2012. Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks. Procedia Technol., 4: 790-796.

Rahul, S., J. Bhattacharyji and R. Soni, 2012. SQL injection attacks in database using web service: Detection and prevention. Asian J. Comput. Sci. Inform. Technol., 2-6: 162-165.

Su, Z. and G. Wassermann, 2006. The essence of command injection attacks in web applications. Proceedings of the 33rd ACM Symposium on Principles of Programming Languages, January 11-13, 2006, Charleston, South Carolina, USA., pp: 372-382.

Suguna, R., T. Kujani, N. Suganya and C. Krishnaveni, 2014. Hunting pernicious attacks in web applications with XProber. Am. J. Applied Sci., 11: 1164-1171.

Tajpour, A., M. Massrum and M.Z. Heydari, 2010a. Comparison of SQL injection detection and prevention techniques. Proceeding of the 2nd International Conference Education Technology and Computer, June 22-24, 2010, Shanghai, pp: 174-179.

Tajpour, A., Z. JorJor and M. Shooshtari, 2010b. Evaluation of SQL injection detection and prevention techniques. Proceeding of the 2nd International Conference Computational Intelligence, Communication Systems and Networks, July 28-30, 2010, Liverpool, pp: 216-221.

Tajpour, A., S. Ibrahim and M. Masrom, 2011. SQL injection detection and prevention techniques. Int. J. Adv. Comput. Technol., 3: 82-91.

Wassermann, G. and Z. Su, 2004. An analysis framework for security in web applications. Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, October 2004, Atlanta, GA., pp: 70-78.