School of Computer Sciences, Universiti Sains Malaysia

CST334 NETWORK MONITORING & SECURITY

Semester I, 2020/2021

ASSIGNMENT 2

Lecturer: Assoc. Prof Dr Aman Jantan, PhD

Literature Review on SQL Injection Attacks in Web Application

Name: Muhammad Iqrar Amin

Matric: 140659

E-mail: iqraramin@student.usm.my

Table of Contents

1 Introduction 4

2 Background 5

2.1 Overview of Structured Query Language (SQL) 5

2.2 Overview of Structured Query Language Injection Attacks (SQLIA) 6

3 Literature Review Methodology 7

4 Types of SQL Injection Attacks 9

5 SQL Injection Attacks Prevention 14

6 Conclusion 17

7 Acknowledgment 17

8 References 17

List of Figures

Figure 2.1: Figure 2.1: Different databases using SQL [6] 5

Figure 4.1: Error message from SQL injection attack 10

Figure 4.2: Tautology attack in the parameter fields in a login page 10

Figure 4.3: Union based injection attack result 11

List of Tables

Table 3.1: Inclusion and Exclusion Criteria 8

Table 3.2: Distribution of studies from various databases 8

Table 4.1: SQL Injection Types with attackers' goals 9

Abstract

. In this review I have performed a systematic literature review on SQL injection attacks that attackers use in exploiting web applications, mobile application and IoT devices. I have also reviewed some of the ways an individual can utilize to detect and prevent SQL injection attacks by using some tools and defensive coding practices. I have reviewed the literature and compiled different types of SQL injection attacks that exist in modern applications along with an example for each type. I also presented and analyzed existing detection and prevention techniques as well as defensive coding practices to counter against SQL injection attacks.

1 Introduction

. One of the most significant risks to Web applications has been established as SQL injection vulnerabilities [1]. SQL vulnerable Web applications can give an intruder direct access to their base databases. Applications Since these databases also contain personal information about customers or users, the resulting security breaches can include identity theft, loss, and fraud. In certain cases, a SQL injection loophole may also be exploited by attackers to manipulate and corrupt the web application hosting infrastructure.

This specific paper or rather literature review aims to answer the question: What are different types of SQL Injection techniques that are widely used among attackers? and What are some techniques that are used to detect and perhaps prevent SQL Injection attacks? A literature review is carried out to identify, analyze, and interpret available information (relevant papers) by using appropriate methods.

In this paper, I will perform a systematic literature review on different types of SQL Injection attacks used by attackers now a days. This study includes the analysis performed on information gathered from 25 different publications for the whole review and most of the publications answers the research questions. In Section 2, I have briefly introduced or rather overviewed Structured Query Language (SQL) and Structured Query Language Injection Attacks (SQLIA). In Section 3, I have discussed about the systematic review performed on the topic and the methodology/process used to carry out the successful review. Moving on to Section 4, I have analyzed the findings from all the publications and summarized them in tabular manner and discussed the types of SQL injection attacks in detail. Then in Section 5, I have suggested few possible solutions to counter the SQL injection attacks discussed in Section 4. Finally, I have concluded my study/review in Section 6.

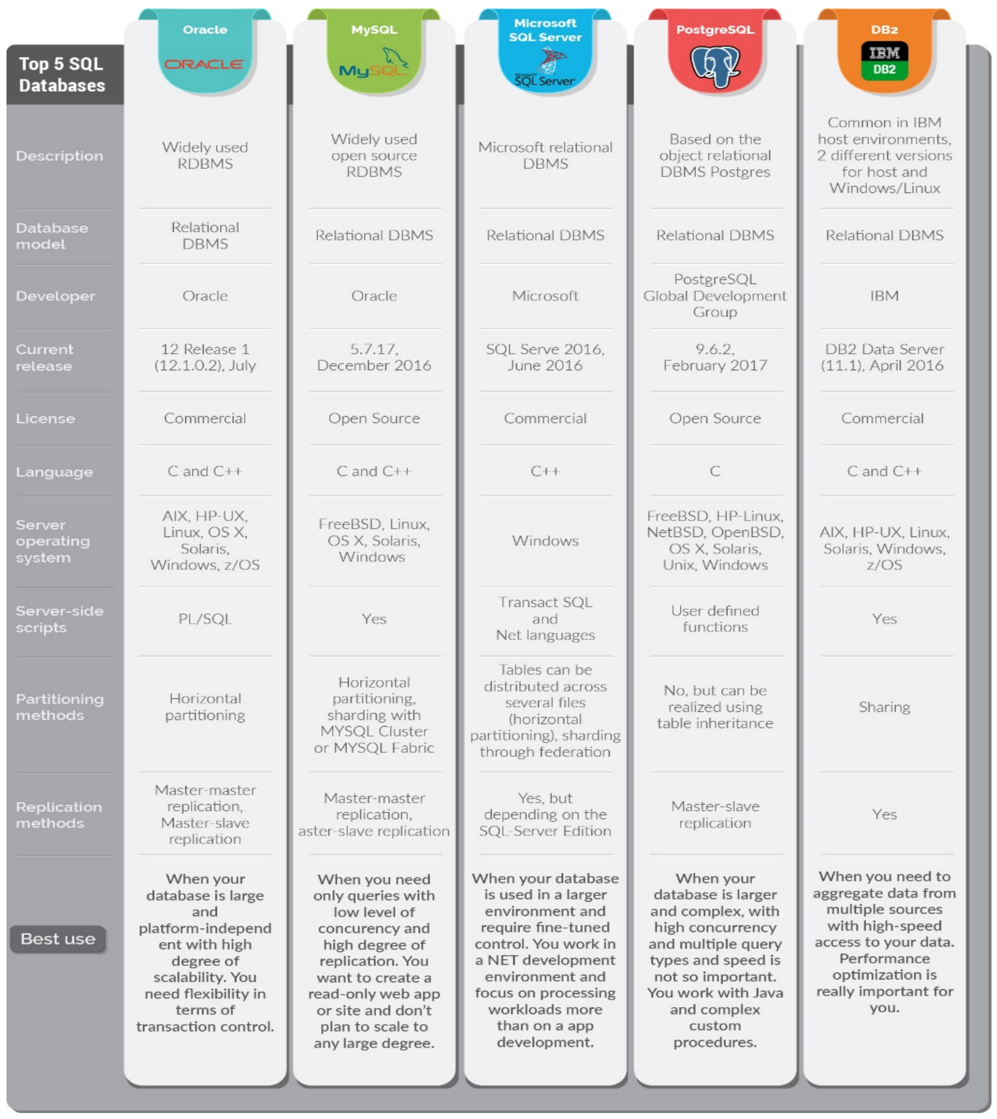
2 Background

This section of the review contains an overview of Structured Query Language (SQL) and overview of Structured Query Language Injection Attacks (SQLIA).

2.1 Overview of Structured Query Language (SQL)

Structural Query Language abbreviation for SQL is the primary method used for accessing, browsing, and maintaining a relation database. Thanks to its natural language-like commands and the consistency of operations it performs corresponding to particular language strings, SQL offers a well-structured language that is simple to understand [3].

In general, SQL is a high-performance language which is able to query structured data. Queries adopt a particular readable syntax which allow different operations to be done by users and database administrators, varying from constructing and removing tables to extracting data according to particular conditions [3]. shows different SQL databases and their properties.



Top 5 SQL Databases	Oracle	MySQL	Microsoft SQL Server	PostgreSQL	DB2
Description	Widely used RDBMS	Widely used open source RDBMS	Microsoft relational DBMS	Based on the object relational DBMS Postgres	Common in IBM host environments, 2 different versions for host and Windows/Linux
Database model	Relational DBMS	Relational DBMS	Relational DBMS	Relational DBMS	Relational DBMS
Developer	Oracle	Oracle	Microsoft	PostgreSQL Global Development Group	IBM
Current release	12 Release 1 (12.1.0.2), July	5.7.17, December 2016	SQL Serve 2016, June 2016	9.6.2, February 2017	DB2 Data Server (11.1), April 2016
License	Commercial	Open Source	Commercial	Open Source	Commercial
Language	C and C++	C and C++	C++	C	C and C++
Server operating system	AIX, HP-UX, Linux, OS X, Solaris, Windows, z/OS	FreeBSD, Linux, OS X, Solaris, Windows	Windows	FreeBSD, HP-Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows	AIX, HP-UX, Linux, Solaris, Windows, z/OS
Server-side scripts	PL/SQL	Yes	Transact SQL and Net languages	User defined functions	Yes
Partitioning methods	Horizontal partitioning	Horizontal partitioning, sharding with MYSQL Cluster or MYSQL Fabric	Tables can be distributed across several files (horizontal partitioning), sharding through federation	No, but can be realized using table inheritance	Sharing
Replication methods	Master-master replication, Master-slave replication	Master-master replication, aster-slave replication	Yes, but depending on the SQL-Server Edition	Master-slave replication	Yes
Best use	When your database is large and platform-independent with high degree of scalability. You need flexibility in terms of transaction control.	When you need only queries with low level of concurrency and high degree of replication. You want to create a read-only web app or site and don't plan to scale to any large degree.	When your database is used in a larger environment and require fine-tuned control. You work in a NET development environment and focus on processing workloads more than on a app development.	When your database is larger and complex, with high concurrency and multiple query types and speed is not so important. You work with Java and complex custom procedures.	When you need to aggregate data from multiple sources with high-speed access to your data. Performance optimization is really important for you.

Figure 2.1: Figure 2.1: Different databases using SQL [6].

2.2 Overview of Structured Query Language Injection Attacks (SQLIA)

The SQL Injection Attack (SQLIA) happens in an interactive manner by injecting new SQL keywords or operators into the query when an attacker alters an expected result of the SQL query. Both variants of SQLIAs mentioned in the literature and provided in this paper are meant to provide this informal description. An attack by SQLIA happens when a malicious user creates and sends a query that works differently from that intended for the programmer through specially constructed entries [4]. Let us take an example of a query which looks like using PHP code:

```
query = "SELECT * FROM accounts WHERE name=’"
+ $name + "’ AND password=’" + $password + "’";
```

This code will allow user to authenticate with the application by providing username and password and login to a website. However, an attacker can enter “attacker” into the name field and ‘“OR 1=1” in password field, and the query will become like this:

```
SELECT * FROM accounts WHERE name= ’attacker’ AND password =’’ OR 1=1
```

By using this query, the result is always evaluating to true allowing user to login every time despite the username and password. These injection vulnerabilities became very common to be discovered on large and real-world web application, and in fact they are regarded as the Top 1 vulnerability according to OWASP Top 10 [5], they also have severe effects on web application security.

3 Literature Review Methodology

I used the methods and strategies for the review proposed by Frâncila Weidt Neiva and Rodrigo Luis de Souza da Silva [6], and Kitchenham et al. [7]. Th following are the descriptions of the activities performed by me.

Research Questions: There are in total 2 research questions around which the whole SLR will circulate, the objective of this review is to identify the different types of SQL Injection attacks that exists in web applications and what are some techniques we can use to prevent such attacks. Hence, I have derived the following research questions to support my research:

RQ1: What are different types of SQL Injection techniques that are widely used among attackers?

RQ2: What are some techniques that are used to detect and perhaps prevent SQL Injection attacks?

Search Procedures: A Systematic Literature Review focuses on searching the scientific databases to get the desired information. The database utilized in this SLR search is listed below:

- IEEE Xplore
- ResearchGate
- Academy of Computing Machinery Digital Library (ACM)
- Science Direct
- Springer Link
- Others

Here is the query that was used when searching for relevant material in literature in search engines:

((“SQL Injection” OR “SQLIA” OR “Structure Query Language Injection Attacks”) AND (“types” OR “techniques” OR “prevention” OR “detection”) AND (“Web Applications” OR “Mobile Applications” OR “IoT”))

Inclusion and Exclusion Criteria: The studies were conducted on selected studies and research based on the inclusion and exclusion criteria. The inclusion criteria includes the research and studies on which the data extraction will be performed, while exclusion criteria includes the criteria which will be used to exclude the studies that are not related from the review. Table 3.2 shows the inclusion and exclusion criteria for selecting the appropriate studies.

Table 3.1: Inclusion and Exclusion Criteria

Type	Criteria
Inclusion Criteria	• Studies that depict different types of SQL Injection Attacks • Studies that discuss about different approaches to detect and prevent SQL Injection in an application
Exclusion Criteria	• Studies in languages other than English • Duplicated studies • Studies that restricts from accessing them, in other word full text is not available for them.

Quality Assessment: Once the studies were handpicked after executing inclusion and exclusion criteria, a comprehensive Quality Assessment was conducted to ensure the qualitative property of the search/researched data. In this specific SLR, I have considered the following four QA questions and a marking scheme was implemented on them which was Complete © = 1, Partial (P) = 0.5, None (N) = 0. The following is the list of QA questions:

1. Were the criteria (inclusion and exclusion) described before are well described and appropriate in the review?
2. Does the study mention the technique or any type of SQL Injection in the research paper?
3. Is the quality and validity of the study is included properly assessed by the reviewers?
4. Does the study discuss on research process?

Data Collection: The information that was taken from the study in the review of each paper is as follow:

- Journal or Conference Paper or References
- Study type classification
- Focus of the topic area
- A complete summary of every study
- The type of SQL Injection attack discussed in the study
- The technique used to detect and prevent SQL injection attack

Results: After the comprehensive review of the literature, 25 studies have been selected and the distribution of the studies selected are shown in Table 3.4, most are from IEEE Xplore (24%), Springer Link (28%), and ResearchGate (16%).

Table 3.2: Distribution of studies from various databases

Scientific Database	Stage 1	Stage 2	Stage 3	Stage 4	Weight
IEEE Xplore	250	75	20	6	24%
ResearchGate	69	60	10	4	16%
ACM	43	30	5	2	8%
Science Direct	38	10	5	2	8%
Springer Link	70	20	10	7	28%
Others	80	45	10	4	16%
Total	550	240	60	25	100%

4 Types of SQL Injection Attacks

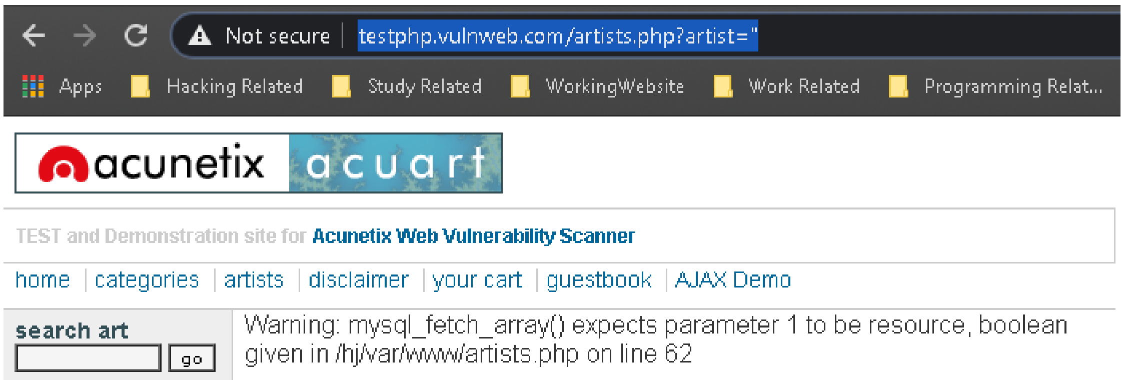
In this section of the review, I will try to answer the first research question **RQ1,** which will depict different types of SQL injection attacks in real world web applications, mobile applications and IoT devices. The Table 4.1 shows the finding for different SQL Injection attacks along with the attackers' goals of carrying out that specific attacks and reference from where that attack is referred from.

Table 4.1: SQL Injection Types with attackers' goals

Sr No.	SQL Injection Type	Attackers Goal	Reference
1	Error-based Injections	Identify table parameters, perform database fingerprinting and extracting data	[8] [9] [10]
2	Tautology Injections	Bypass authentication, identify injectable parameters and extracting data	[11] [8] [12]
3	Union-based Injections	Bypassing authentication and extracting data	[8] [10] [13]
4	Time-based Injections	Identify injectable parameters, extracting data and determining database schema	[14]
5	Piggy-backed Injections	Extracting data, adding or modifying data, performing DoS and remote code execution	[8] [10] [15]
6	Encoding alteration Injections	avoid and evade detection	[8] [15]
7	Blind SQL Injections	Identify injectable parameters, extracting data and determining database schema	[16]

All researched types of SQL injection attacks are discussed in more details with code examples in the following section with appropriate references:

1. **Error-based Injection:** In this type of attack, the attackers main goals are to identify injectable parameters, perform database fingerprinting to get useful information about the database and finally extracting data from the database itself. This attack helps an attacker to gain valuable details about the type and structure of a Web application's back-end database. This attack is a first phase in the compilation of knowledge for other attacks. The weakness of this attack is that the default application server error message is always too descriptive [10]. Let us take an example of this website which is vulnerable to SQL Injection attacks. By injecting a query such as " ' " in the URL parameter we will get the following error



Not secure
testphp.vulnweb.com/artists.php?artist="
Apps
Hacking Related
Study Related
WorkingWebsite
Work Related
Programming Relat...
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go

```
Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean
given in /hj/var/www/artists.php on line 62
```

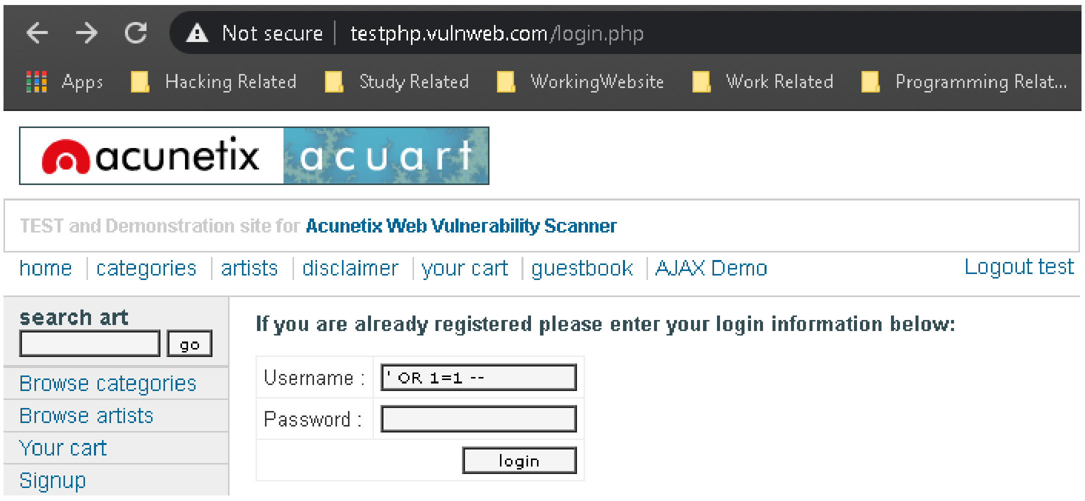
Figure 4.1: Error message from SQL injection attack

From this error we can easily get the information about the database and what kind of parameters the database query is expecting from the user. We can see that the database is MySQL and is accepting 1 parameter which is a number type.

2. **Tautology Injections:** In this attack, the attackers intent is to bypass the authentication along with identifying the injectable parameters and ultimately extracting data from the database. The key purpose of an attack on tautology is to insert code into one or more conditional declarations in order to always evaluate query to true. The effect of this attack depends on how the query results are used in the program. Attackers use this technique very commonly so that authentication pages are bypassed, and data can be collected [11]. For example, if we take the same website into account and try to login by using string in user as " ' OR 1=1 – " we get the following query result:

SELECT accounts FROM users WHERE login=' ' OR 1=1 -- AND password=''

Now the – will comment the rest of the query and will login no matter what we give in the parameters since OR 1=1 always evaluates to true.

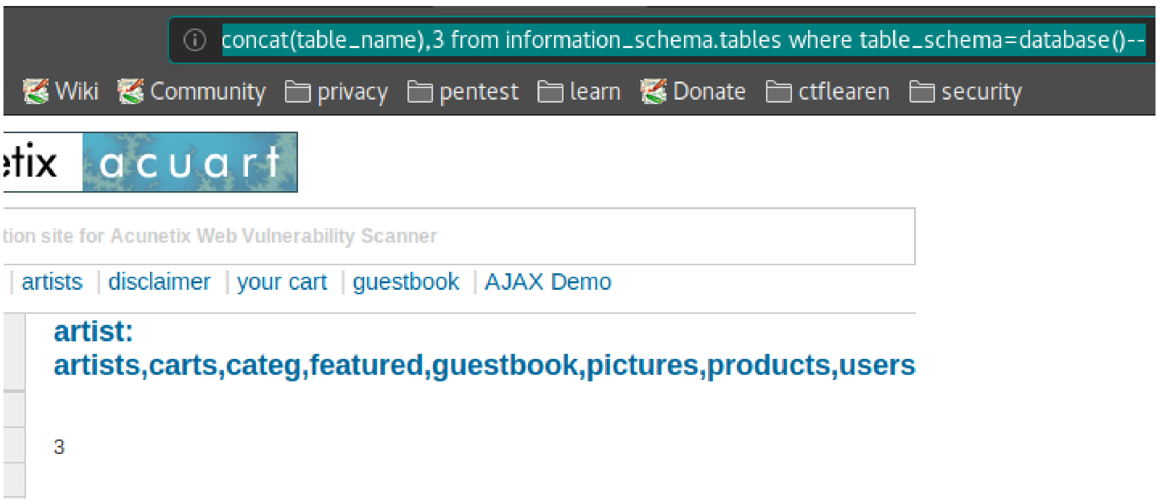


If you are already registered please enter your login information below:

Username :	' OR 1=1 --
Password :	

Figure 4.2: Tautology attack in the parameter fields in a login page

3. **Union-based Injections:** In this injection type, the attacker goal is to bypass authentication and get data out of the database without having to login to the database. An attacker uses a weak parameter in union query attacks to alter the data collection returned for a certain request. This approach helps an assailant to trick the program into returning data from a table other than that which the developer intended. Attackers are using a type declaration: `UNION SELECT <rest of the injected query>` `[13].` For demonstration we will inject the URL with query "`union select 1,group_concat(table_name),3 from information_schema.tables where table_schema=database() --`" as shown in the following figure to get the list of all available tables in the database.



concat(table_name),3 from information_schema.tables where table_schema=database()--

artist:
artists,carts,categ,featured,guestbook,pictures,products,users

3

Figure 4.3: Union based injection attack result

4. **Time-based Injections**: This is one of the Inference injections attacks where the attackers intent is to identify injectable paraments, extract useful information and determining database schema. A time attack helps an attacker to extract information from a database by detecting time gaps in the database response. The attacker designs the injected query for a time attack in the form of an if/then argument whose branch prediction is the same as an undefined knowledge about the database contents [14]. Let's take the following query as an example to demonstrate Timing attack:

   SELECT accounts FROM users WHERE login='user' and 1=0 -- ' AND pass=''

   An attacker can exploit this query by using the following in the place of *login* :

   ''user'' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 --''

   The attacker asks if the character's ASCII value is more or less than the X value. If the value is higher, the intruder is aware of this by detecting a further 5 second delay in the database response. The intruder will then use a binary check to find the first character value by modifying the X value.

5. **Piggy-backed Injections:** In this type of injection attacker wants to extract data from the database, add and modify data and perform a denial-of-service attack along with executing remote commands on the server. An attacker attempts to insert further requests in the initial query in this form of attack. In this case, an attacker does not attempt to change the initial intended query; instead, he tries to include new, distinct queries that "piggy-back" on the original query. This type is different from other ones. In this way, several SQL queries are submitted to the database. The vulnerability to this form of attack also depends on the layout of a database allowing several statements in a single string [15]. For example, if an attacker enters "'; drop table users -- " into the password parameter, it will generate the following query:

   SELECT accounts FROM users WHERE login='doe' AND password=''; drop table users -- '

   Now this query has two queries, when the first query will end at *password=''* , it will stop execution at the delimiter ';' and will execute the second query which starts with *drop* which will delete the table named the users resulting in unstable website which will ultimately result in denial of service.

6. **Encoding alternation Injections:** The main objective of attacker in this type of injection is to evade the detection from the server. The text injected is amended so that protective coding activities and certain automatic mitigation strategies are not identified during this attack. This form of attack is used for other attacks. In other words, alternative encoding is not a uniquely attackable solution to an application; it merely enables attackers to overcome techniques of detection and protection and manipulate bugs that would otherwise not be exploitable [15]. For instance, if "'OR 1=1 --" is blocked by the database server which can be by passed by using different encoding such as in

   HEX encoding in URL would be: *27%20%4f%52%20%31%3d%31%20%2d%2d*
   Base64 encoding would be: *JyBPUiAxPTEgLS0=*

   This will be regarded as valid by the database server which will allow the attacker to execute successful SQL injection attack in the remote database server and evade the detection from the system successfully.

7. **Blind SQL Injections:** This is one of the Inference injections attacks where the attackers intent is to identify injectable paraments, extract useful information and determining database schema. In this approach, the information from the page's actions must be inferred, asking true/false questions from the server. If the statement injected is true, the website will continue to work normally. In the particular instance of a false statement, the page differs significantly from that of the usual working page even when no descriptive error message exists [16]. Let’s take the following queries as an example:

   SELECT accounts FROM users WHERE login=’user’ and 1=0 -- ’ AND password=’’

   SELECT accounts FROM users WHERE login=’user’ and 1=1 -- ’ AND password =’’

   After running first query, the result will always be false and attacker will get login error messages each time, attack does not know if the credentials are wrong or is the attempt is blocked. The attacker can proceed with sending the second query which will always evaluate to true, if there is no error message in this case, then the attacker know that the attack went through, and login has been bypassed.

5 SQL Injection Attacks Prevention

Based on the review of the literature there, researchers have proposed wide range of tools and techniques to counter SQL injection attacks. These techniques include best coding practices, SQL injection detection and runtime SQL injection attacks prevention. These are discussed in more details in the upcoming sections.

1. **Coding Practices**: The major cause of SQL injection attacks is indeed the poor and insufficient input validation. The easiest way to remove such vulnerabilities is, thus, to introduce effective protective coding practices. Here is the list of few best practices that a developer can use to reduce SQL injection attacks:
    - *Input type checking*: SQL injection attacks on the web applications may be carried out in a string or numeric parameter by putting commands. Many attacks can also be avoided by simple checks of these inputs [12].
    - *Input encoding*: The use of meta characters to trick the SQL parser into reading user input as SQL tokens is also used to inject the string parameter. While a use of these meta characters may be forbidden, it limits the right of a non-malicious user to define legitimate inputs containing those characters. A safer approach is to use features that encrypt a string in order to encode and view all the characters as standard characters in the database [12].
    - *Valid Pattern Matching*: Developers can develop routines for input validation to recognize good input instead of bad input. In comparison to negative validation, which aims for the feedback of prohibited sets or square tokens, this method is usually considered a positive validation [12].
    - *Input source identification*: All inputs for the application must be reviewed by developers. These input sources may be a means for an attacker to add a SQL injection attack when constructing a query. Simply put, you must verify all input sources [12].

2. **SQL Injection Attacks' Detection**: There are a lot of tools that are available to detect the SQL Injection attacks in an application that can be used alongside with the existing application to help developers detect the incoming SQL Injection attacks. Some of them are discussed in the following section:

    - *AMNESIA* [17]: It integrates static analysis with runtime control, a model-based methodology. In its static level, AMNESIA uses static analytics to construct prototypes for the multiple type of queries an application may legally produce at each access point to the database. AMNESIA intercepts all requests in its dynamic process until it is submitted to the database and verifies each query against the statically constructed models.
    - *SQLCheck* [4]*:* It tests queries during runtime to verify if they fit a model of planned queries. The paradigm is represented in these methods as a grammar that accepts only lawful queries. In SQLCheck, the model is presented independently by the developers themselves instead of leaving it to the system. This uses a hidden key to restrict user feedback when parsing the run-time checker, so the protection of the solution relies on the ability of the attacker to locate the key.
    - *SQLrand* [18]*:* SQLrand is a randomization technique dependent on instructions. SQLrand offers a mechanism to allow users, rather than standard SQL keywords, to build queries using randomized instructions. In order to de-randomize the keywords, a proxy filter has been placed at the backend of a database server which will intercept the queries and do so. A randomized command set would not have created the SQL code injected by an attacker. Thus, injected commands will lead to an error in syntax. While this strategy can be extremely effective, it has many practical disadvantages.
    - *Tautology Checker* [4]*:* This approach is based solely on the forms of tautological attacks and cannot either identify or avoid any other form of SQL attacks. This strategy is thus limited to the form of attack and leaves the web app vulnerable to other attacks.

3. **SQL Injection Attacks' Prevention**: There are a lot of tools that are available to prevent the SQL Injection attacks in an application that can be used alongside with the existing application to help developers detect the incoming SQL Injection attacks. Some of them are discussed in the following section:

    - *PSIAQOP* [19]: It is also known as "Preventing SQL injection attacks based on query optimization process". The SQL Injection Attacks is prevented depending on the request optimization phase. The methodology is based on optimizing the user-input and responsive SQL queries during execution. PSIAQOP examines and recognizes hotspots represented by the most insecure SQL queries for attacks that are the source code used in web applications that is relying on users to supply the fields. These vulnerable queries then go through the mechanism of optimization based on the heuristic laws.
    - *Security Gateway* [20]*:* Security Gateway is a proxy filtering system to apply data validation rules in a web application. Security Gateway Developers have restrictions and define transformations to be used for device parameters by using their Own Security Policy Descriptor Language (SAPLL) as they switch from the web page to the app server.
    - *SQL DOM* [21]: It is a sophisticated framework which is suggested by McClure and Kruger in [21]. They closely consider existing flaws in relation databases from the language viewpoint of the OOP (Object-Oriented Program). They focus primarily on the identification of barriers to data interaction via CLIs (Call Level Interfaces). The SQL DOM object model is the solution proposed to deal with these problems by creating a secure communication environment.
    - *WebSSARI* [22]*:* It uses knowledge flow analysis to find errors relevant to input validation. Static analyzes are used in this approach to evaluate taint flows for critical functions against preconditions. The review detects the points where requirements were not satisfied and may recommend filters and sanitization functions which are applied to the framework automatically to satisfy these requirements.

6 Conclusion

In this specific paper, I carried out systematic literature review on SQL injection attack using Kitchenhams' systematic review as a guideline. There were debates about the causes and consequences of SQLIA.I have defined and listed the different forms of SQLIA techniques and exemplars from the syntax that was used for execution of such attacks. In addition, the most common strategies for SQLIA identification and prevention have been outlined along with some of the best defensive coding practices that a developer can use to reduce the risk of SQL injection attacks.

For future research, I will be researching evaluation methods based on resources that are required to implement the SQL injection attacks detection and prevention techniques. Moreover, some evaluation methods that are used to protect against web application from some other attacks such as Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and Denial of Service attacks will be researched.

7 Acknowledgment

I would like of thank Assoc. Prof Dr Aman Jantan, for giving me such an amazing opportunity to work on this literature review study and enhance my Affective Domain skills along with Cognitive skills.

8 References

[1] OWASP, "The Ten Most Critical Web Application Security Risks," OWASP, 2017.

[2] E. Galluccio, E. Caselli and G. Lombari, SQL Injection Strategies, Birmingham: Packt Publishing Ltd., 2020.

[3] Z. Su and G. Wassermann., "The Essence of Command Injection," in *In The 33rd Annual Symposium on Principles of Programming Languages*, 2006.

[4] OWASP, "OWASP Top Ten," OWASP, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 07 01 2021].

[5] F. W. N. Silva and R. L. d. S. da, "Systematic Literature Review in Computer Science - A Practical Guide," Federal University of Juiz de Fora, November 2016.

[6] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software.," University of Durham Tech. Rep. EBSE-2007-01, Durham, UK, July 2007.

[7] C. Anley., "Advanced SQL Injection In SQL Server Applications," Next Generation Security Software Ltd, 2002.

[8] D. Litchfield, "Web Application Disassembly with ODBC Error Messages," 2002. [Online]. Available: http://www.nextgenss.com/papers/webappdis.doc. [Accessed 07 01 2021].

[9] S. McDonald, "SQL Injection: Modes of attack, defense, and why it matters," GovernmentSecurity.org,, 2002.

[10] D. Ragesj and S. Shiva., "Runtime monitors for tautology based SQL injection attacks.," in *Cyber Security Cyber Warfare and Digital Forensic (CyberSec)*, 2012.

[11] Halfond, W., J. Viegas and A. Orso., "A classification of SQL injection attacks and countermeasures.," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA, 2006.

[12] S. Labs, "SQL Injection.," SPI Dynamics, Inc.,, 2002. [Online]. Available: http://www.spidynamics.com/assets/documents/WhitepaperSQLInjection.pdf. [Accessed 07 01 2021].

[13] C. Anley., "Advanced SQL Injection," Next Generation Security Software Ltd, 2002.

[14] M. Howard and D. LeBlanc, "Writing Secure Code," Microsoft Press, Redmond, Washington,, 2003.

[15] K. Spett, "Blind sql injection," SPI Dynamics, Inc.,, 2005. [Online]. Available: http://www.spidynamics.com/whitepapers/Blind SQLInjection.pdf.. [Accessed 07 01 2021].

[16] W. G. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," in *Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, USA, 2005.

[17] S. W. B. a. A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks and Network Security (ACNS) Conference," in *Proceedings of the 2nd Applied Cryptography*, 2004.

[18] A.-K. E., F. Al-Anzi and A. Salinan, "SIAQOP: Preventing SQL injection attacks based on query optimization process," in *Proceedings of the 2nd Kuwait Conference on E-Services and E-Systems*, Kuwait, 2011.

[19] D. Scott and R. Sharp, "Abstracting Application-level Web Security," in *Proceedings of the 11th International Conference on the World Wide Web (WWW 2002)*, 2002.

[20] R. McClure and I. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," in *27th International Conference Proceeding Software Engineering*, 2005.

[21] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee and S. Y. Ku, "Securing Web Application Code by Static Analysis and Runtime Protection," in *Proceedings of the 12th International World Wide Web Conference (WWW 04)*, 2004.

[22] F. Security, "What is Application Layer Security?," F5 Security, [Online]. Available: https://www.f5.com/services/resources/glossary/application-layer-security. [Accessed 10 11 2020].

[23] S. Magazine, "Cyberattacks, Application Vulnerabilities Increase by 40 Percent in September 2019," Security Magazine, 25 10 2019. [Online]. Available: https://www.securitymagazine.com/articles/91140-cyber-attacks-application-vulnerabilities-increase-by-40-percent-in-september-2019. [Accessed 25 11 2020].