NUML International Journal of Engineering and Computing

Volume: 2 Issue: 2

https://numl.edu.pk/journals/nijec
Print ISSN: 2788-9629
E-ISSN: 2791-3465
DOI:https://doi.org/10.52015/nijec.v2i2.50

A Systematic Literature Review on SQL Injection Attacks

Maryam Mehmood[1,2], Asad Ijaz[3]

1 Department of Computer Software Engineering, College of E&ME, National University of Science and Technology, NUST, Pakistan
2 Department of Software Engineering, NUML, Islamabad
3 Department of Mechanical Engineering, MUST AJK, Pakistan
Corresponding author: First Author (e-mail: Maryam.mehmood@numl.edu.pk).

Submitted	**Revised**	**Published**
01-Dec-2023	13-Dec-2023	21-Feb-2024

Abstract

With the increasing use of web applications, concerns for data integrity and security have increased manifolds in the current time. The growth in quantity of internet clients and sites has made the web security circumstances progressively extreme. Structured Query Language Injection Attack (SQLIA) is a major threat to web applications. Over the time, many studies have explored the reasons and techniques of these attacks, and also ways to detect and prevent them from happening. This study presents a Systematic Literature Review (SLR) based on the methodology proposed by Kitchenham in 2007. The focus of study is on determining how and why SQLIA are done and how can they be avoided or mitigated. The literature is considered for a time period of four years; 2016 to 2023. Moreover, evaluation has been done, based on limitations and priorities proposed by each technique studied. Attack types with their severity has been reviewed that may help researchers propose new techniques in order to make web applications more secure against SQLIAs**.**

Keywords: SQL injection, SQL injection detection, SQL injection prevention, SQL injection types, SQL injection techniques

1. Introduction

In order to reach out to potential customers and users across the globe, most of organizations have web-based applications enacting as their connection to the rest of the world. Database driving web based solutions are now considered as a backbone of global software market. In general, most of the software applications are now web based. It is understood that web based applications need to be accessed over network by multiple devices. Database connected at the backend of application needs to be accessed by the application for each data transaction; depending upon the purpose and requirements the software. Applications need to process data from connected databases, as well as data received from users (maybe another system) for every transaction. Any vulnerability in the system may cause catastrophic damage to an organization; tarnishing the reputation in the global market. Attackers having intentions of benefitting from these vulnerabilities may attack organizations' applications, thus, compromising the data integrity and security.

Structured Query Language Injection Attack (SQLIA) is amongst the top threats, as rated by organizations like OWASP [1] and MITRE [2]. It targets connected databases by inserting malicious code, to be processed for achieving desired intention. For exploiting the web applications, attackers initiate attack and get access to unauthorized data; to be manipulated according to their own will. One of the major reasons of SQLIA is poor input validation on web applications. Imperva's web application attack report [3] talked about 6,800 SQLIAs per hour made in their 6th edition.

Researchers and practitioners have proposed many solutions to overcome such problems, but each of these solutions has its own limitations due to scope, tools, and technology constraints. In general, researchers worked on SQL injections by considering two broad categories: detection and prevention. In detection category, researchers try to provide multiple solutions for detecting a potential SQLIA; one that aims to differentiate attackers from users. While in the prevention category, researchers try to provide solutions for the SQLIA under consideration.

The main focus of this study is to assess the effectiveness of already existing tools/frameworks in terms of their detection and prevention efficiency. A Systematic Literature Review (SLR) was conducted by Lawal and et al. on SQLIA in 2016 [1]; SLR is discussed in detail in the next section. It covered in-depth literature from 2005 onwards on various SQL injection techniques and methods of detecting and preventing SQLIA. The paper laid a good foundation of the topic and hence, was chosen to act as a base study for the current research.

The current study extends the research of the base paper to include literature from 2016 to 2023 to gain insight on the latest trends of SQLIA. The next section states the methodology adopted for creating the SLR. The findings are discussed in detail in the discussion section, followed by conclusion. The results produced by this study will help researchers to enhance existing strategies and introducing new tools or frameworks to overcome deficiencies.

2. Research Methodology

This research uses an SLR approach, as used in the base study. SLR is now a well-known and enriched review method in the domain of research. Instead of randomly searching the web for relevant data, SLR uses a defined approach of finding, sorting, analyzing and interpreting available research on a topic. Broadly, it can be categorized into three main steps:

a. Planning the research
b. Conducting the research
c. Reporting the findings

Each step can be further divided into a number of activities to make the process of SLR more methodical. Even though it requires more effort, as compared to normal research, a structured SLR makes the whole review process to be less biased and ensures coverage of a wide range of settings and empirical methods. Given below is one possible breakdown of the three steps of SLR as suggested in an EBSE Technical Report [2], Figure 1 shows the steps in a refined manner.

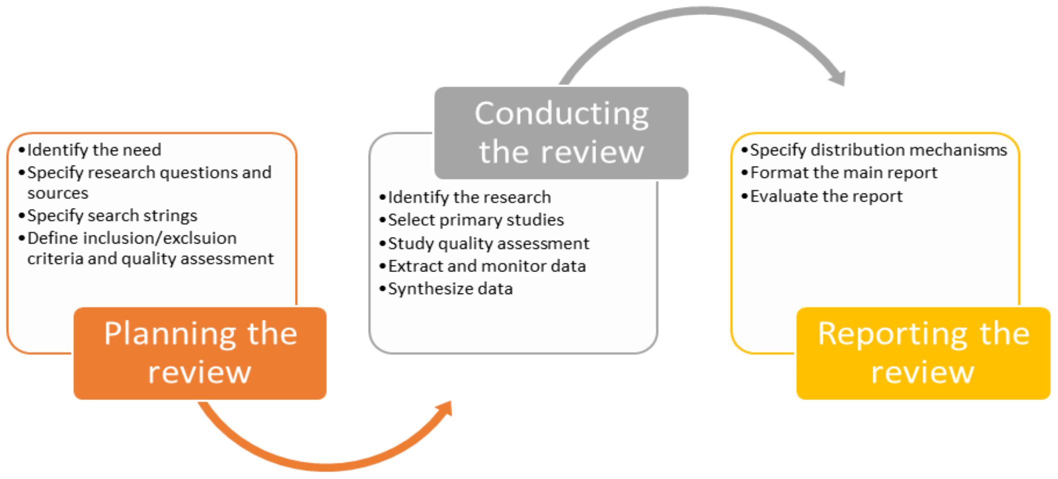

Figure 1: SLR Process

A. *Planning the review*

i. Identify the need for conducting a review: this step requires researchers to review all existing information in a detailed and unbiased manner.

ii. Specify the research question(s): this step requires researchers to think of the exact questions they would like their study to answer.

iii. Develop review protocol: this step identifies the methodology to be used; including keywords to be searched, repositories or resources to be searched, quality assessment criteria and checklists, and data extraction and synthesis strategy.

iv. Evaluate the review protocol: this step requires a quality and feasibility check of all things specified in the methodology.

B. *Conducting the review*

i. Identify the research: this step refers to determining and following a search strategy that suits the methodology selected for the SLR.

ii. Select primary studies: this step is about finding studies that fall directly within the scope of the research questions and provide direct evidence about them, based on inclusion/exclusion criteria.

iii. Study quality assessment: this step defines means of measuring the importance of a selected study to the research questions.

iv. Extract and monitor data: guidelines to extract and record data are defined in this step.

v. Synthesize data: this step is about processing and organizing the results gathered from every study found.

C. *Reporting the review*

i. Specify distribution mechanisms: this step is about planning the dispersal of information gathered i.e. reporting study in any journals and/or conferences.

ii. Format the main report: this step is about formulating templates for writing the final report of SLR.

iii. Evaluate the report: this step is about evaluating the quality of the study conducted and reported.

3. Research Questions

Phase 1: Planning the Review

The base study for his research was addressing four research questions (RQ) given below:

RQ1: What are the reasons and effects of SQL injections?

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 injections attacks?

For the sake of clarification, the original questions were rephrased and divided into parts to form a better understating of the topic; Table 1 explains the motivation behind each RQ.

Data Sources

This study was conducted on four famous repositories. Table 2 shows the search criteria applied on the repositories. Four authors took the responsibility of searching data in one repository each and hence, each source was checked for all search strings.

Table 1: Motivation and Research Questions

Research Question	**Motivation**
RQ1: What are the reasons behind SQLIA?	Exploration of potential motives behind SQLIAs, with their consequences.
RQ2: What are the types and/or techniques of SQLIA?	Identification of various types of SQLIAs on web based applications.
RQ3: How SQLIA are done?	Investigation of techniques used to make SQLIAs
RQ4: What are some SQLIA detection and prevention techniques?	Examination of techniques proposed by researchers in detecting and combating SQLIAs

Table 2: Data Source and Search Criteria

Electronic repositories	1. IEEE Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp) 2. Springer (https://www.springer.com/in) 3. ACM Digital Library (https://dl.acm.org/) 4. ScienceDirect (https://www.sciencedirect.com/)
Language	English

Publication period	January 2016 to March 2023
Searched items	Books, journals and conference papers having any of the search strings in their title or keywords

Search Strings

The search strings were taken from the base study and were also derived from keywords used while rephrasing the research questions. Table 3 shows the search strings and their alternatives used to conduct the study.

Table 3: Search Strings and Alternatives

Keywords	**Alternatives**
SQL injection	1. SQLIA 2. SQL injection attacks 3. Structured Query Language Injection Attacks 4. Reasons for SQLIA
SQL injection detection	1. SQL injection attacks detection and prevention techniques 2. Detect SQL Injection Attacks
SQL injection prevention	1. SQL Injection attacks and defense 2. SQL Injection defense mechanisms 3. SQL injection attacks detection and prevention techniques 4. Prevent SQL Injection Attacks
SQL injection types	1. Why SQLIA
SQL injection techniques	1. SQL injection attacks detection and prevention techniques 2. Executing SQLIA

Inclusion and Exclusion Criteria

After the basic search, only the articles matching the inclusion criteria were considered for the study and the remaining or those falling in the exclusion criteria were dropped from the analysis.

Inclusion Criteria

i. Articles published in journals, workshops and conferences related to SQLIA
ii. Articles having the search strings in their titles
iii. Articles having the search strings in their keywords
iv. Articles that deal with the research questions
v. Articles published "between" January 2016 to December 2023
vi. Articles having English as primary language

Exclusion criteria

i. Articles that are not related to the research objective of the study
ii. Articles not in English
iii. Articles not falling in the decided timeline
iv. Articles not having any of the search strings in their title and/or keywords.

Quality Evaluation (QE)

The finalized articles were evaluated by each author via a pre-defined quality checklist. Each selected article was evaluated on a scale of 0 to 1 for all five research questions, in order to check its importance in providing answer(s) to any or all of the RQs identified for the study.

- An article was assigned a '1′ against a question it answered completely.
- An article was assigned a '0.5' against a question it answered partially.
- An article was assigned a '0' against a question it failed to answer at all.

Phase 2: Conducting the Review

Primary Study Selection

A total of 113 articles were found against the search strings in all four repositories. Each author applied the inclusion and exclusion criteria on the articles found in the repository he was working on. After the application of the criteria, a total of 72 articles were left for the primary study; which comprises of almost 64% of the total articles found by all four authors. The selected primary study articles included 37 journal papers (51%), 30 papers from conference proceedings (42%) and 4 book chapters (6%).

Table 4 gives a summary of each repository in terms of total articles found it in for the primary study. It also shows how many articles contributed to each of the research questions identified for the study. Figure 2 compares the contribution of each repository for the research questions.

Table 4: Primary Study Stats

Source	Primary Articles	RQ1	RQ2	RQ3	RQ4
Springer	25	14	5	11	20
IEEE Xplore	7	6	4	0	5
ACM Digital Library	17	9	17	0	16
ScienceDirect	23	13	5	17	11

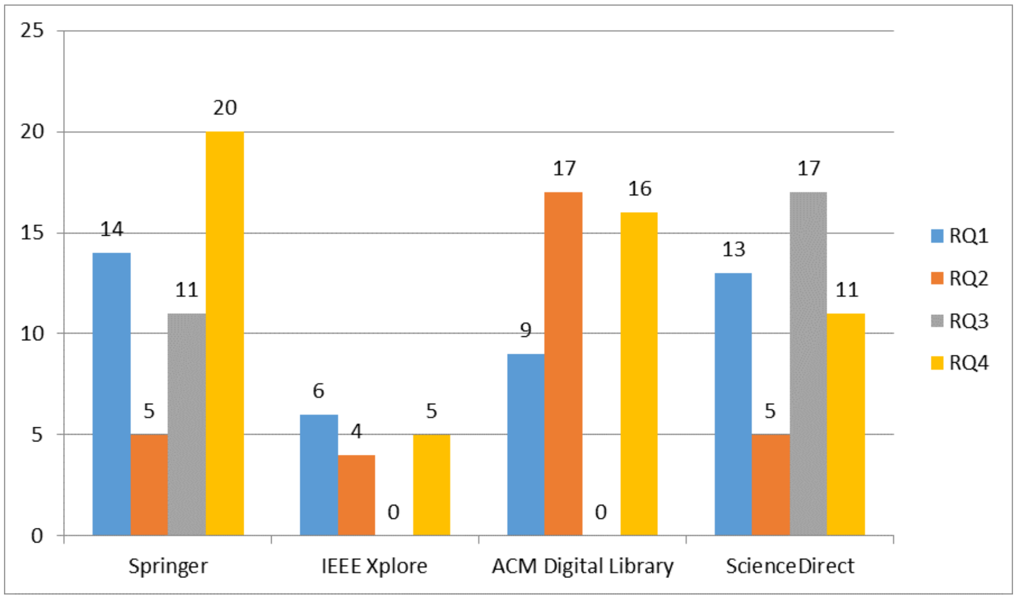

Figure 2: Repository contributions for each RQ

Data Extraction

Each author was responsible for extracting relevant data from the primary articles found in his respective repository. Each article was evaluated on the quality criteria defined in section "Quality Evaluation (QE)" above. Table 5 provides the list of selected articles along with their total QE scores. The last column in the table gives a percentage of the contribution each article made to the SLR.

Table 5: QE of Selected Articles

Paper ID	**Q1**	**Q2**	**Q3**	**Q4**	**Total**	**%**
Article#1 [3]	0	0	0.5	1	1.5	38
Article#2 [4]	1	0	0	1	2	50
Article#3 [5]	1	0	0	1	2	50
Article#4 [6]	0	0	0	1	1	25
Article#5 [7]	1	0.5	1	0.5	3	75
Article#6 [8]	1	0	1	0.5	2.5	63
Article#7 [9]	1	0.5	0	1	2.5	63
Article#8 [10]	0	0	0.5	1	1.5	38
Article#9 [11]	1	0	0	1	2	50
Article#10 [12]	0	0	0.5	1	1.5	38
Article#11 [13]	1	0	0.5	1	2.5	63
Article#12 [14]	1	1	1	1	4	100
Article#13 [15]	0	0	0	0	0	0
Article#14 [16]	0.5	0.5	0	0.5	1.5	38
Article#15 [17]	0	0.5	0	0.5	1	25
Article#16 [18]	0.5	1	0	0.5	2	50
Article#17 [19]	1	1	0	1	3	75
Article#18 [20]	0.5	0	0	1	1.5	38
Article#19 [21]	1	1	1	1	4	100
Article#20 [22]	0	0	0	1	1	25
Article#21 [23]	0	0	0	0	0	0
Article#22 [24]	0.5	0.5	0	1	2	50
Article#23 [25]	0.5	0	0.5	1	2	50
Article#24 [26]	0.5	0.5	0	0.5	1.5	38
Article#25 [27]	0	0	0	0.5	0.5	13
Article#26 [28]	0.5	0	0	0	0.5	13
Article#27 [29]	0	1	0	0.5	1.5	38
Article#28 [30]	0.5	0.5	0	0.5	1.5	38

Article#29 [31]	1	0.5	0	0.5	2	50
Article#30 [32]	0	0.5	0	1	1.5	38
Article#31 [33]	1	1	1	1	4	100
Article#32 [34]	0	0	0.5	0	0.5	13
Article#33 [35]	0	0	0	0	0	0
Article#34 [36]	0.5	0	0.5	0	1	25
Article#35 [37]	0	0	0	0	0	0
Article#36 [38]	0	0	0.5	0.5	1	25
Article#37 [39]	0	0	0.5	0.5	1	25
Article#38 [40]	1	0	0	1	2	50
Article#39 [41]	1	1	0	0.5	2.5	63
Article#40 [42]	0	0	0	1	1	25
Article#41 [43]	0.5	0	0	0	0.5	13
Article#42 [44]	0	1	0	0.5	1.5	38
Article#43 [45]	0	0	0.5	1	1.5	38
Article#44 [46]	1	0.5	0.5	1	3	75
Article#45 [47]	1	0	0.5	1	2.5	63
Article#46 [48]	1	1	0	1	3	75
Article#47 [49]	1	0	0.5	0	1.5	38
Article#48 [50]	1	0	1	0	2	50
Article#49 [51]	0.5	1	0	1	2.5	63
Article#50 [52]	0.5	0	0	1	1.5	38
Article#51 [53]	0.5	0	0.5	0.5	1.5	38
Article#52 [54]	1	1	0	0	2	50
Article#53 [55]	0	0.5	0	0.5	1	25
Article#54 [56]	0.5	0	0.5	0.5	1.5	38
Article#55 [57]	0	1	0	1	2	50
Article#56 [58]	1	0.5	0	0.5	2	50
Article#57 [59]	0	1	0	1	2	50
Article#58 [60]	0.5	0	0	0	0.5	13
Article#59 [61]	0	0	1	0.5	1.5	38
Article#60 [62]	0	1	0	1	2	50
Article#61 [63]	1	1	0.5	1	3.5	88
Article#62 [64]	1	1	0	0	2	50
Article#63 [65]	0.5	0	0.5	0	1	25
Article#64 [66]	0	0	0.5	0	0.5	13

Article#65 [67]	1	1	1	1	4	100
Article#66 [68]	1	0.5	0	0	1.5	38
Article#67 [69]	0	0	0.5	0	0.5	13
Article#68 [70]	0	0	0	0.5	0.5	13
Article#69 [71]	0	0	0.5	0	0.5	13
Article#70 [72]	0	0	0	0	0	0
Article#71 [73]	0	0	0	0	0	0
Article#72 [74]	0	1	0	0.5	1.5	38

Data Synthesizing

Figure 3 is based on the last two columns of Table 5 and it provides a summarized view of the overall contribution the primary study articles have made to the SLR. The QE score had an average of 40% approximately and hence, the articles included in reporting were selected on a threshold of at least 30% contribution. This means that from the selected 73 articles of the primary study, 48 (almost 67%) were included in the final reporting of the SLR.

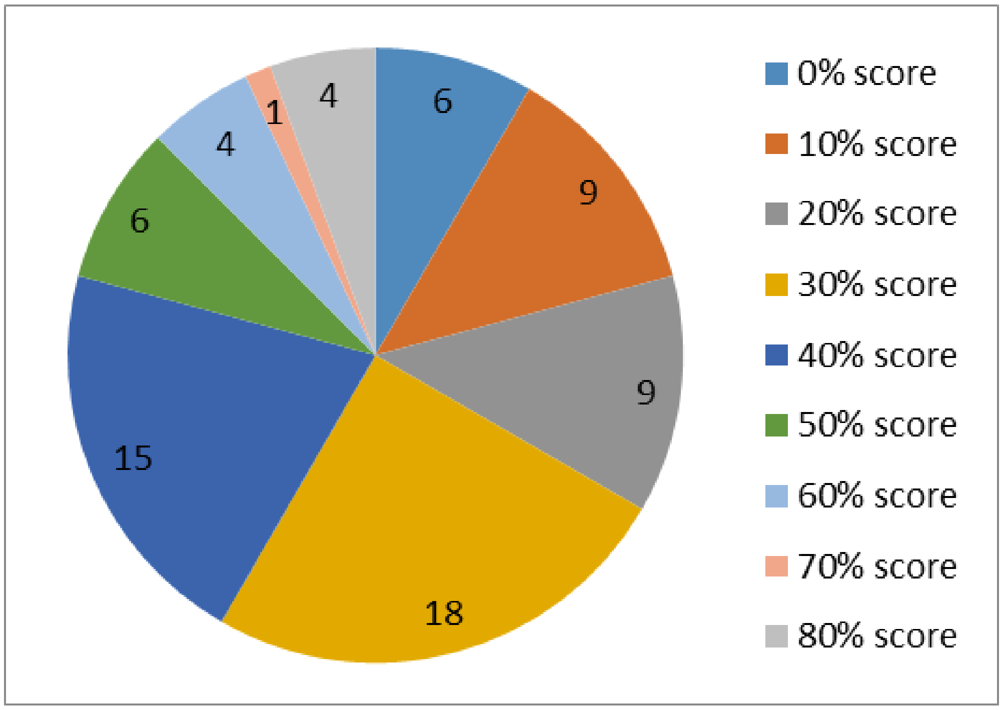

Figure 3: Collective QE score of primary study articles

4. DISCUSSION

This section discusses the data gathered from all the selected articles mentioned above, in relation to the RQs identified by the study. SQLIAs are considered a champion amongst the commonly occurring web application hazards [7].

RQ1. What are the types/techniques of SQLIA?

The researchers have classified SQLIAs in many different ways. SQL statements can be altered very easily by the intruders [64]. A broad categorization [14][49] simply divides immediate execution of a malicious code as First Order Attack and a time-based triggered execution as Second order attack. Another, more precise classification [5],[64],[9],[13] is given in Table 6, where the attacks are classified into eight types along with their severity levels.

Table 6: Different types of SQLIAs

SQLIA Type	**Process**	**Query**	**Target**	**Risk level**
Tautology	▪ Identify injectable parameters ▪ Bypass authentication ▪ Extract data	SELECT * FROM userdetails WHERE login=‘anyone’ or ‘1=1’ and password= anything’ or ‘x’=‘x’	Returns all users details	Medium
Incorrect Logical Query	▪ Identify injectable parameters ▪ Retrieve database fingerprint from generated error message	SELECT * FROM userdetails WHERE login='kao''' AND password =	Returns some key information of the server like database server, version, platform etc.	Low
Piggyback	▪ Extract data ▪ Modify dataset by appending malicious query at the end of valid query using semi-colon(;) ▪ Execute remote commands	SELECT * FROM userdetails WHERE userid=‘admin’ and password=‘admin’; drop table user_details–	Performs database operations as deletion, update, and addition	High
Boolean-based Blind SQL Injection	▪ Extract data ▪ Inject series of true/false queries to database ▪ Error message shows presence of protection mechanism, otherwise successful attack	SELECT * FROM emp_name, emp_address, gender from employee where 1=0; drop employee SELECT * FROM emp_name, emp_address, gender from employee where1=1;drop employee	An attacker gets insight about the database whether it is secure or not.	Low

Time based Blind SQL Injection	▪ Identify injectable parameters ▪ Bypass authentication ▪ Extract data ▪ Attacker performs time intensive operations using time based command “SLEEP”	SELECT name, price FROM store_table WHERE id= ‘46’ and if(1=1, sleep(10) , false)	Return response time database has taken to respond to the user’s query.	Low
UNION-based SQL Injection	▪ Gain unauthorized access ▪ Extract data ▪ Malicious SQL query appended with valid SQL query by using UNION command	SELECT * FROM table WHERE login=’’ UNION SELECT ** FROM table WHERE No=12500 -- AND password =’’ AND pin=	Combine the result sets of two or more statements	Medium
Stored Procedure	▪ Gain unauthorized access ▪ To execute the stored procedure SHUTDOWN	SELECT * FROM userdetails WHERE login= 'kao' AND password ='lai'; SHUTDOWN;--;	Execute built-in functions using malicious SQL codes	Medium, high
Alternate Encodings		SELECT * FROM table WHERE login= 'kao';exec(char(0x73697 574646f776e)) –‘ AND password ='lai' AND pin =; SHUTDOWN;--;	Modify the injection statement by alternating encoding to escape from detection	Medium, high

RQ2. What are the reasons behind SQLIA?

Simple SQLIAs are not difficult to initiate, human errors and negligence during development make an attackers work easy [63]. Successful SQLIAs are often very harmful. There’s no limit to what proportion of harm can be caused by an intruder. Given below is a list of most common reasons reported behind a successful SQLIA in the literature [7][15][33]; most of which narrow down to human errors or bad development practices:

- The data compromised was very sensitive and/or worthwhile to the attacker.
- The application had insecure development architecture.
- The application had poorly filtered strings and incorrect type handling.
- Database access rights were not properly assigned to the correct authorities.
- Security design was compromised due to budget shortfalls.
- Mismatched data types were common.
- Insufficient input validation protocols were implemented.

- Detailed error messages were not displayed to give proper warnings.
- Dynamic SQL statements were constructed using input content to access the database.
- The code used stored procedures, which are passed as strings containing unfiltered user input.

RQ3: How SQLIA are done?

The SQL injection attacks may be carried out manually [3], or through a few software tools [75] like: jSQL Injection, Whitewidow, Blind-Sql-Bitshifting, SQLMap, SQLNinja, etc. But no matter whether the attack is manual or tool based, it always begins by constructing a special statement to observe the SQL vulnerabilities existing in the web program [3]. There are a few things that can help an attacker formulate a query suitable for conducting an SQLIA [33],[21]:

- Identify Injectable Parameters: these textual parameters allow users to request information from the database via an HTTP request. When this done without proper validation, an attacker can most likely inject an SQL query in it.
- Perform database fingerprinting: knowing the version of a database enables an attacker to construct a supported query format.
- Determine database schema: knowing the structure of the database makes it easier for an attacker to extract or manipulate data.

Literature reports [14],[50],[61],[21] that the process of SQLIA is performed using different input parameters like:

- User input: web applications receive inputs from user via HTTP (GET or POST) requests. An attacker can inject SQL command disguised as user input.
- Cookies: an attacker can easily tamper a cookie's contents and embed a malicious code in it.
- Server variables: an attacker can place an SQLIA directly into the header of the variable, which can be triggered as soon as the query to log the server variable is issued.

RQ4: What are some SQLIA detection and prevention techniques?

Researchers talk about many development practices and fully automated techniques for the detection and prevention of SQLIAs. Sooner an injection attack or a potential chance of an injection attack can be detected, lesser will be the damage that follows.

Development Practices

Following are some commonly preached development practices that can help web application designers to prevent SQLIAs from happening.

Defensive Coding Practices [19]

Such practices focus on writing the code in a manner that can prevent insertion of malicious code [13]. Although they do involve chances of human error and are not as reliable as automated techniques, they can still help the cause.

a. ***Input examination:*** verification of user input ensures that the input criteria defined for the application is being met. Usually, a type-check against a parameter or a regular expression to validate an input fall within this category. For example, numeric inputs restrict the usage of alphabets or special characters.

These checks are often ignored while working with strings; hence, a validation mechanism is mandatory to prevent SQLIAs.

b. ***Encoding of inputs:*** meta-characters inserted in input strings can be interpreted as valid SQL tokens by the parser. A practice of either restricting such meta-characters or using functions to encrypt all meta-characters to be interpreted as general characters can help.
c. ***Pattern matching (positive validation):*** this routine checks good inputs against bad inputs. Unlike negative validation, which searches for forbidden patterns, it specifies all inputs which are legal.
d. ***Identification of input source(s):*** insertion of data is done via multiple sources and thus, these sources can act as a path for an attacker to introduce a SQLIA.

Defense at Platform-Level

Imprecise configurations in the database increase the chances of an SQLIA. Thus, careful practices while handling database platforms can help against SQLIAs.

a. Correct Configuration of Web Server: a web server can be properly configured in three ways [7]:
    i. Change initial configuration
    ii. Install security patches in a timely manner
    iii. Turn off error messages
    iv. Correct Configuration of Database: the database can be safely configured by using principle of least privilege i.e. grant access to authorized user only. This can be done by [7]:
b. Modifying the initial configuration of the database
c. Upgrading the database timely
d. Proxy filters: Security Policy Descriptor Language (SPDL) provides a security gateway for input data to flow to web applications. Being highly expressive, SPDL allows developers to apply constraints and special transformations to the input. One drawback to this approach is that it relies heavily on human-knowledge to know which data and/or patterns need to be filtered.

Automated Techniques:

In contrast to development practices, there are a few automated techniques to prevent an SQLIA from causing havoc.

a. Machine learning: Some researchers [7],[13],[4],[5] utilize machine learning technologies to detect SQLIAs. This approach has two stages

i. Learning stage: it uses training sets to build detection models
ii. Classification stage: it uses the formulated models to label a query as an injection attack

The quality of the training set determines how well the model will perform. Support Vector Machine (SVM) follows this strategy by analyzing original query and suspicious query and based on the confusion matrix formulated, it prevents a likely SQLIA from being executed [12],[61].

b. Stored Procedure Approach: This technique is a mixture of static and dynamic analysis and works in a similar manner. The static module checks the source code and the dynamic module checks the queries at runtime [13].

i. Static analysis: detects all possible weaknesses in an application which can aid an SQLIA before deployment. This technique's efficiency depends on how accurately the input validity module has worked. This method has no run time overhead. But the analysis of code has two major constraints:
- It makes the method very host language-specific and cannot detect all types of SQLIAs.
- It requires access to source code, which is itself a risk.

ii. Dynamic analysis: used for analysis of runtime SQL queries and it processes every query before posting it to the database server.

c. Taint Analysis: This technique modifies the PHP interpreter to track user input and verify if it does not modify SQL queries. The restriction to this technique is it uses certain types of filters to judge the input and considers it sufficient to detect an input as an attack [5] and [19].

d. Aho–Corasick Algorithm: This technique looks for a string of a particular arrangement inside the query and calculates the suspicion level of it being an SQLIA [7] and [8].

Table 7: SQL Prevention and Detection Techniques

Prevention Technique	**Description**
Defensive Coding Practices	
Input Examination	Verifies user input against predefined criteria, such as type-checking or regular expressions also prevents SQL injection.
Encoding of Inputs	Restricts meta-characters and encrypts them in preventing interpretation as SQL tokens by the parser.
Pattern Matching (Positive Validation)	Specifies all legal inputs and compares against them and identifies potentially malicious inputs.
Identification of Input Sources	Identifies and secures all sources through which data is inserted to prevent potential paths for SQL injection attacks.
Defense at Platform-Level	
Correct Configuration of Web Server	Changes initial configurations, installs security patches, and turns off error messages
Configuration of Database	Applies the principle of least privilege by modifying initial configurations, upgrading databases, and restricting access to authorized users.
Proxy Filters (SPDL)	Uses Security Policy Descriptor Language (SPDL) and applies constraints and transformations to input data, though reliance on human knowledge may be a drawback.

Automated Techniques	
Machine Learning	Utilizes machine learning, e.g., Support Vector Machine (SVM), with a learning stage using training sets and a classification stage to label queries as injection attacks based on formulated models.
Stored Procedure Approach	Combines static and dynamic analysis; static analysis detects weaknesses pre-deployment, and dynamic analysis processes queries at runtime.
Taint Analysis	Modifies PHP interpreter to track user input and verifies its impact on SQL queries, using certain filters to detect potential attacks.
Aho–Corasick Algorithm	Searches for specific string arrangements in queries and calculates suspicion levels of SQL injection attacks

5. Conclusion

This systematic literature review has been produced based on guidelines provided by Kitchenham et. al [2]. It explored four repositories for relevant literature; including 113 articles. Based on our exclusion and inclusion criteria, 72 articles were shortlisted for the primary study and then the quality analysis applied on the primary study yielded 48 articles to be finally included in the analysis of the research questions. The study talks about primary reasons behind conducting SQLIAs. Different types of SQL Injection attacks have been reviewed in light of their classification category, process of conduction, query format, target and risk level. A detailed discussion on SQLIA detection and prevention technique has also been done by the study. The procedures and steps have been discussed to help new researchers in proposing advanced techniques to detect and prevent the SQLIAs. The findings of the study show that all proposed SQLIA prevention and detection techniques have their own limitations and thus, relying on a single technique might not solve the problem. Instead, multiple techniques must be combined and deployed in order to deal with diverse kinds of SQLI attacks.

References

[1] Lawal, M.A., Sultan, A.B.M., & Shakiru, A.O. (2016). Systematic Literature Review on SQL Injection Attack. International Journal of Soft Computing, 11(1), 26-35.

[2] Kitchenham, B. A. & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering (EBSE 2007-001). Keele University and Durham University Joint Report.

[3]Chen, T. W. C., & Sun, X. (2016). A Countermeasure to SQL Injection Attack for Cloud. Wireless Personal Communications.

[4] Kao, D.-Y., Lai, C.-J., & Su, C.-W. (2018). A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics. Paper presented at the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC).

[5] Ghafarian, A. (2017). A hybrid method for detection and prevention of SQL injection attacks. Paper presented at the 2017 Computing Conference.

[6] Saoudi L., Adi K., Boudraa Y. (2020) A Rejection-Based Approach for Detecting SQL Injection Vulnerabilities in Web Applications. In: Benzekri A., Barbeau M., Gong G., Laborde R., Garcia-Alfaro J. (eds) Foundations and Practice of Security. FPS 2019. Lecture Notes in Computer Science, vol 12056. Springer, Cham

[7] Bokey, V., Datar, K., Jabalpure, D., Suryawanshi, K., Lokhande, V., & Kale, P. (2018). A Review on Different Methodologies to Counter SQL Injection Attack. International Journal of Scientific Research in Science and Technology (IJSRST), 4(2), 1528-1535.

[8] Yeole , R., Ninawe, S., Dhore, P., & Tembhare, P. U. (2017). A Study on Detection and Prevention of SQL Injection Attack. International Research Journal of Engineering and Technology (IRJET), 4(3), 435-438.

[9] Faker, S. A., Muslim, M. A., & Dachlan, H. S. (2017). A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities. International Journal of Computer Engineering and Information Technology, 9(12), 284-291.

[10] Zhu, Y., Zhang, G., Lai, Z., & Niu, B. (2018). A Two-Tiered Defence of Techniques to Prevent SQL Injection Attacks, 1.

[11] Saidu, M., Imran, A., Kashif, G., Qureshi, N., & Fo, M. (2019). An algorithm for detecting SQL injection vulnerability using black-box testing. Journal of Ambient Intelligence and Humanized Computing, 0(0), 0. http://doi.org/10.1007/s12652-019-01235-z

[12] Awasthi, R., & Mangal, D. (2016). An Approach Based on SVM Classifier to Detect SQL Injection Attack. International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), 2(3), 256-260.

[13] Agarwal, R., & Sirsikar, S. (2019). An Efficient Technique for finding SQL Injection using Reverse Proxy Server. International Research Journal of Engineering and Technology (IRJET), 6(9), 1564-1569.

[14] Kumar, A. & Taterh, S. (2016). ANALYSIS OF VARIOUS LEVELS OF PENETRATION BY SQL INJECTION TECHNIQUE THROUGH DVWA. Journal of Advanced Computing and Communication Technologies, 4(2), 28-32.

[15] Priyadharshini, S. and Rajmohan, R. (2017). Analysis on Database Security Model Against NOSQL Injection. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2(2), 168-171.

[16] Yadav, N., & Shekokar, N. (2018). Analysis on Injection Vulnerabilities of Web Application Á Injection vulnerability Á Attack Á Security. Springer Singapore. http://doi.org/10.1007/978-981-10-8339-6

[17] Lokhande, P. S., & Meshram, B. B. (2016, March). Analytic Hierarchy Process (AHP) to Find Most Probable Web Attack on an E-Commerce Site. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (pp. 1-6).

[18] Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2007, October). CANDID: preventing sql injection attacks using dynamic candidate evaluations. In Proceedings of the 14th ACM conference on Computer and communications security (pp. 12-24).

[19] Mavromoustakos, S., Patel, A., Chaudhary, K., Chokshi, P., & Patel, S. (2016, December). Causes and prevention of sql injection attacks in web applications. In Proceedings of the 4th International Conference on Information and Network Security (pp. 55-59).

[20] Khanna, S., & Verma, A. K. (2018). Classi fication of SQL Injection Attacks Using Fuzzy Tainting, 463–469. http://doi.org/10.1007/978-981-10-3373-5

[21] Aliero, M. S., Ardo, A. A., Ghani, I. & Atiku, M. (2016). Classification of Sql Injection Detection And Prevention Measure. IOSR Journal of Engineering (IOSRJEN), 6(2), 06-17.

[22] Zhao J., Dong T., Cheng Y., Wang Y. (2020) CMM: A Combination-Based Mutation Method for SQL Injection. In: Miao H., Tian C., Liu S., Duan Z. (eds) Structured Object-Oriented Formal Language and Method. SOFL+MSVL 2019. Lecture Notes in Computer Science, vol 12028. Springer, Cham

[23] Kesarwani, M., Kaul, A., Singh, G., Deshpande, P. M., & Haritsa, J. R. (2018). Collusion-Resistant Processing of SQL Range Predicates. Data Science and Engineering, 3(4), 323-340.

[24] Stasinopoulos, A., Ntantogian, C., & Xenakis, C. (2018). Commix : automating evaluation and exploitation of command injection vulnerabilities in Web applications. International Journal of Information Security. http://doi.org/10.1007/s10207-018-0399-z

[25] Das, D., & Sharma, U. (2017). Defeating SQL injection attack in authentication security : an experimental study. International Journal of Information Security. http://doi.org/10.1007/s10207-017-0393-x

[26] Chenyu, M., & Fan, G. (2016, August). Defending SQL injection attacks based-on intention-oriented detection. In 2016 11th International Conference on Computer Science & Education (ICCSE) (pp. 939-944). IEEE.

[27] Antunes, N., & Vieira, M. (2017). Designing vulnerability testing tools for web services: approach, components, and tools. International Journal of Information Security, 16(4), 435-457.

[28] Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D. C., Staples, J., & Krause, L. (2019). Detecting web attacks with end-to-end deep learning. Journal of Internet Services and Applications, 10(1), 1-22.

[29] Basta, C., Elfatatry, A., & Darwish, S. (2016). Detection of SQL injection using a genetic fuzzy classifier system. International Journal of Advanced Computer Science and Applications, 7(6), 129-137.

[30] Rauti, S., Teuhola, J., & Leppänen, V. (2015, August). Diversifying SQL to prevent injection attacks. In 2015 IEEE Trustcom/BigDataSE/ISPA (Vol. 1, pp. 344-351). IEEE.

[31] Khalid, A. &Yousif, M. F. M. (2016). Dynamic Analysis Tool for Detecting SQL Injection. International Journal of Computer Science and Information Security (IJCSIS), 14(12), 224-232.

[32] Joshi, P. N., Ravishankar, N., Raju, M., & Ravi, N. C. (2018). Encountering SQL Injection in Web Applications. Paper presented at the 2018 Second International Conference on Computing Methodologies and Communication (ICCMC).

[33] Sharma, C., Jain, S. C. & Sharma, A. K. (2016). Explorative Study of SQL Injection Attacks and Mechanisms to Secure Web Application Database- A Review. International Journal of Advanced Computer Science and Applications, 7(3), 79-87.

[34] Safianu, O., Twum, F. and Hayfron-Acquah, J. B. (2016). Information System Security Threats and Vulnerabilities: Evaluating the Human Factor in Data Protection. International Journal of Computer Applications, 143(5), 08-14.

[35] Gbedawo, V., Agbesi, K. and Adukpo, T. (2017). Intrusion Detection on Campus Network, the Open source approach: Accra Technical University Case Study. International Journal of Computer Applications, 164(6), 20-27.

[36] Zech, P., Felderer, M., & Breu, R. (2019). Knowledge-based security testing of web applications by logic programming. International Journal on Software Tools for Technology Transfer, 21(2), 221-246.

[37] Ramachandra, A. C., & Bhattacharya, S. (2020). Literature Survey on Log-Based Anomaly Detection Framework in Cloud. In Computational Intelligence in Pattern Recognition (pp. 143-153). Springer, Singapore.

[38] Hazaa, M. A. S., Algabry, M. A. S. and Altayar, M. M. Q. (2016). METHODS OF SAFEGUARDING THE SITES FROM SQL INJECTION. Saba Journal of Information Technology and Netwroking (S.J.I.T.N), 4, 20-28.

[39] Geneiatakis, D. (2015, December). Minimizing databases attack surface against sql injection attacks. In International Conference on Information and Communications Security (pp. 1-9). Springer, Cham.

[40] Feng K., Gu X., Peng W., Yang D. (2019) Moving Target Defense in Preventing SQL Injection. In: Sun X., Pan Z., Bertino E. (eds) Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science, vol 11635. Springer, Cham

[41] Ross, K., Moh, M., Moh, T. S., & Yao, J. (2018, March). Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. In Proceedings of the ACMSE 2018 Conference (pp. 1-8).

[42] Eassa, A. M., Elhoseny, M., & El-bakry, H. M. (2018). NoSQL Injection Attack Detection in Web Applications Using RESTful Service 1, 44(6), 435–444.

[43] Pathak, M. P., Khan, N. K. and Tantak, T. C. (2016). Novel Approach To Detect and Prevent Web Attacks. International Research Journal of Engineering and Technology (IRJET), 3(5), 504-510.

[44] Relan, K., & Singhal, V. (2016, March). Pentest Ninja: XSS And SQLi Takeover Tool. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (pp. 1-2).

[45] Arumugam, C. (n.d.). Prediction of SQL Injection Attacks in Web Applications (Vol. 1). Springer International Publishing. http://doi.org/10.1007/978-3-030-24305-0

[46] Bittal, V., & Banerjee, S. (n.d.). Prevention Guidelines of SQL Injection Database Attacks : An Experimental Analysis, 23–32. http://doi.org/10.1007/978-81-322-2553-9

[47] Agarwal, S., & Singh, U. (2017). PREVENTION OF SQL INJECTION ATTACK IN WEB APPLICATION WITH HOST LANGUAGE. International Research Journal of Engineering and Technology (IRJET), 4(11), 1468-1470.

[48] Castillo, R. E., Caliwag, J. A., Pagaduan, R. A., & Nagua, A. C. (2019, March). Prevention of SQL injection attacks to login page of a website application using prepared statement technique. In Proceedings of the 2019 2nd International Conference on Information Science and Systems (pp. 171-175).

[49] George, T. K., James, R. & Jacob, P. (2016). Proposed Hybrid model to detect and prevent SQL Injection. International Journal of Computer Science and Information Security (IJCSIS), 14(6), 441-448.

[50] Manjunatha, K. M. & Kempanna, M. (2019). RAMIFICATION ANALYSIS OF SQL INJECTION DETECTION IN WEB APPLICATION. International Journal of Computer Science and Information Security (IJCSIS), 17, 132-138.

[51] Singh, T., & Aksanli, B. (2019, November). Real-time Traffic Monitoring and SQL Injection Attack Detection for Edge Networks. In Proceedings of the 15th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (pp. 29-36).

[52] Bherde, G. P., & Pund, M. (2016). Recent attack prevention techniques in web service applications. Paper presented at the 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT).

[53] Chen, Z., Li, M., Cui, X., & Sun, Y. (2019). Research on SQL Injection and Defense Technology. Springer International Publishing. http://doi.org/10.1007/978-3-030-24268-8

[54] Ma, L., Zhao, D., Gao, Y., & Zhao, C. (2019). Research on SQL Injection Attack and Prevention Technology Based on Web. Paper presented at the 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA).

[55] Sadasivam, G. K., & Hota, C. (2015, February). Scalable honeypot architecture for identifying malicious network activities. In 2015 international conference on emerging information technology and engineering solutions (pp. 27-31). IEEE.

[56] Nagpal, B., Chauhan, N., & Singh, N. (2016). SECSIX : security engine for CSRF , SQL injection and XSS attacks. International Journal of System Assurance Engineering and Management.

[57] Liu, M., Li, K., & Chen, T. (2019, July). Security testing of web applications: a search-based approach for detecting SQL injection vulnerabilities. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 417-418).

[58] Maheshwarkar, B., & Maheshwarkar, N. (2016, March). SIUQAPTT: SQL Injection Union Query Attacks Prevention Using Tokenization Technique. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (pp. 1-4).

[59] Ceccato, M., Nguyen, C. D., Appelt, D., & Briand, L. C. (2016, September). SOFIA: An automated security oracle for black-box testing of SQL-injection vulnerabilities. In 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 167-177). IEEE.

[60] Kozik, R., & Choraś, M. (2016, April). Solution to data imbalance problem in application layer anomaly detection systems. In International Conference on Hybrid Artificial Intelligence Systems (pp. 441-450). Springer, Cham.

[61] Ladole, A., & Phalke, D. A. (2016). SQL Injection Attack and User Behavior Detection by Using Query Tree, Fisher Score and SVM Classification. International Research Journal of Engineering and Technology (IRJET), 3(6), 1505-1509.

[62] Zhang, H., Zhao, B., Yuan, H., Zhao, J., Yan, X., & Li, F. (2019, October). SQL Injection Detection Based on Deep Belief Network. In Proceedings of the 3rd International Conference on Computer Science and Application Engineering (pp. 1-6).

[63] Kaur, D., & Kaur, P. (2017). SQLI Attacks : Current State and Mitigation in SDLC, 673–680. http://doi.org/10.1007/978-981-10-3153-3

[64] Som, S., Sinha, S., & Kataria, R. (2016). Study on sql injection attacks: Mode detection and prevention. International Journal of Engineering Applied Sciences and Technology, Indexed in Google Scholar, ISI etc., Impact Factor: 1.494, 1(8), 23-29.

[65] Sahasrabuddhe, A., Naikade, S., Ramaswamy, A., Sadliwala, B. and Futane, P. (2017). Survey on Intrusion Detection System using Data Mining Techniques. International Research Journal of Engineering and Technology (IRJET), 4(5), 1780-1784.

[65] Kumari, Y. (2019). Survey on Web Application Vulnerabilities. International Research Journal of Engineering and Technology (IRJET), 6(9), 922-925.s

[67] Saidu, M., Kashif, A., Qureshi, N., & Fermi, M. (2020). Systematic Review Analysis on SQLIA Detection and Prevention Approaches. Wireless Personal Communications. Springer US. http://doi.org/10.1007/s11277-020-07151-2

[68] Silva, R., Barbosa, R., & Bernardino, J. (2016, July). Testing Snort with SQL Injection Attacks. In Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering (pp. 129-130).

[69] Bhosale, T., More, S. and Mhatre, S. N. (2019). Testing Web Application using Vulnerability Scan. International Research Journal of Engineering and Technology (IRJET), 6(5), 265-267.

[70] Nembhard, F. D., Carvalho, M. M., & Eskridge, T. C. (2019). Towards the application of recommender systems to secure coding. EURASIP Journal on Information Security, 2019(1), 9.

[71] Mahdi, M. and Mohammad, A. H. (2016). USING HASH ALGORITHM TO DETECT SQL INJECTION VULNERABILITY. INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS, 4(1), 26-32.

[72] Yadav, S. and Kumar, S. (2018). Web Application Security: Protection from Advanced Persistent Threat. INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN TECHNOLOGY (IJIRT), 4(12), 954-960.

[73] Rane, V., Chaitrali Rane, C., Shelar, M. and Pinjarkar, V. (2016). Website Security Tool. International Research Journal of Engineering and Technology (IRJET), 3(3), 299-304.

[74] Fang, Y., Peng, J., Liu, L., & Huang, C. (2018, March). WOVSQLI: Detection of SQL injection behaviors using word vector and LSTM. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy (pp. 170-174).

[75] https://www.kitploit.com/p/sql-injection-tools.html (Friday, June 26,2020 08:28 PM)