Wireless Personal Communications (2020) 112:2297–2333
https://doi.org/10.1007/s11277-020-07151-2

Systematic Review Analysis on SQLIA Detection and Prevention Approaches

Muhammad Saidu Aliero[1] · **Kashif Naseer Qureshi**[2] · **Muhammad Fermi Pasha**[1] · **Imran Ghani**[3] · **Rufai Aliyu Yauri**[4]

Published online: 30 January 2020

Abstract
SQL injection attack (SQLIA) is one of the most severe attacks that can be used against web database driving applications. Attackers use SQLIA to get unauthorized access and perform unauthorized data modification. To combat problem of SQLIA, different researchers proposed variety of tools and methods that can be used as defense barrier between client application and database server. However, these tools and methods failed to address the whole problem of SQL injection attack, because most of the approaches are vulnerable in nature, cannot resist sophisticated attack or limited to scope of subset of SQLIA type. With regard to this different researcher proposed different approach (experimental and analytical evaluation) to evaluate the effectiveness of these existing tools based on type SQLIAs they can detect or prevent. However, none of the researcher considers evaluating these existing tool or method based on their ability to be deployed in various injection parameters or development requirements therefore, in this study *Kitchenham's* guidelines of performing systematic review of software for conducting our study. In this paper, we reviewed the tools and methods that are commonly used in detection and prevention of SQLIA, Finally, we analytically evaluated the reviewed tools and methods based on our experience with respect to SQIAs types and injection parameters. The evaluation result showed that most researchers focused on proposing approaches to detect and prevent SQLIAs, rather than evaluating the efficiency and effectiveness of the existing SQLIA detection and prevention tools/methods. The study also revealed that more emphasis was given by the previous studies on prevention measures than detection measures in combating problem of SQLIAs. An analysis showed that these tools and methods are developed to prevent subset of SQLIAs type and only few of them can be deployed to various injection parameters to be considered in examining SQLIAs. It further revealed that none of the tools or methods can be deployed to prevent attacks that can take advantage of second order (server side SQLIA) SQLI vulnerability. Finally, the study highlights the major challenges that require immediate response by developers and researchers in order to prevent the risk of being hacked through SQLIAs.

✉ Kashif Naseer Qureshi
Kashifnq@gmail.com

Extended author information available on the last page of the article

Keywords SQLIA prevention · SQLIAs detection · Detection method · Detection tool · Types of SQLIAs · Injection parameters · Analytical evaluation

1 Introduction

Technology and networks enable organizations to have adopted web-based applications as a backbone to conduct their day to day activities. Different domains like Intelligent Transportation Systems [1], Healthcare Systems [2], Industrial Technologies [3], E-commerce [4], social activities are all now available on web-based databases driving applications and where the security of web applications is, in general, quite poor and demanding [5, 6]. These applications process the data and store the result in a back-end database server where the organization's related data are stored. Depending on the specific purpose of the application, most of the communication with customers and users use the services offered by the organization. The fact that these applications can be invoked by anyone worldwide drew the attention of attackers who wish to benefit from these vulnerabilities. One of the techniques to exploit these applications (web-based driving database applications) is called SQLIA (SQL injection attack). SQLIA is a situation whereby attack modifies programmer intended queries to have access to restricted data or perform unauthorized data manipulations. SQLIA comes in a variety of types depending on what attacker wants to accomplish, but the main cause of SQLIA is improper validation of input by user which programmer should take care of while developing the application [5–7]. To tackle the problem of SQLIA, researchers have been proposed different techniques to handle SQLIA. These techniques have limitations starting from research scope to address particular type of SQLIA, deployment capability to the approach, tool or technique.

Most of the researches regarding the evaluation of SQLI prevention measure have focused on evaluation ability to address the particular type of attacks. Similarly, the need to evaluate such SQIAs and taken prevention measures or develop a new approach in various injection parameters is also important. Because the SQLI prevention measures can affect the effectiveness of the tool to address the SQIAs types. If the tool cannot be deployed in a particular injection parameter, it implies that attack injected through that injection parameters would be carried out successfully without any detection or prevention by tool or technique. Thus, the focus of this review is to assess the effectiveness of current SQL injection prevention (SQLIP) tools and techniques based on their ability to address SQLIAs with respect to development approach and ability to be deployed in different injection parameters and also determine the new trend of the research in the field of SQLIA. The results of our evaluation will help new researchers who want to improve the current trends in SQLIAs prevention measures.

2 Research Material and Review Method

Systematic Literature Review (SLR) is a type of review that follows a sequence of precise methodological steps for reducing bias in research. This SLR on SQLIA prevention measures is based on well-established and evaluated review protocols to extract, analyze, and report the results as shown in Fig. 1. We adopted the guidelines provided by [8] with a three-step review process that includes planning, conducting and documenting.

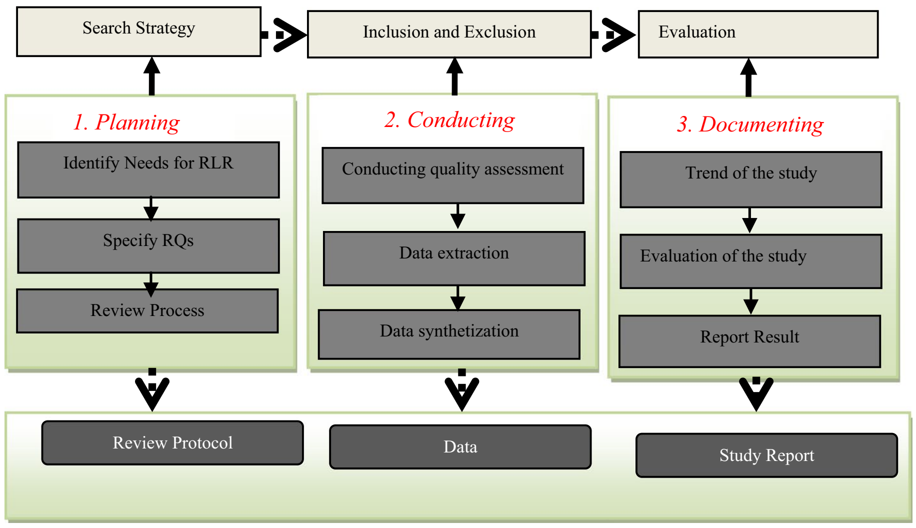


Fig. 1 Research methodology steps

Figure 1 summarizes the steps of the review process; planning, conducting and documenting the methodology adopted in this SLR. The results of the analysis are documented in terms of a data summary in Sect. 2.3, and findings and research implications in Sect. 2.4.

2.1 Planning Phase

In this SLR of SQL injection detection and prevention measures, the planning phase begins with an identifying need for SLR, identification of the research questions and describing the review process. Having these parameters defined, we can formulate a review protocol.

2.1.1 Identifying Need for SLR

There are several studies on SQLIAs detection and prevention tools and methods such as [5, 9–14]. None of these methods provide a systematic way of conducting a literature review on tools and methods. This gives us the motivation to conduct SLR on SQLIAs detection and prevention tools based on research questions (Table 1).

2.1.2 Specifying Research Questions

This study is mainly based on four (4) research questions (RQs). Most of the proposed approaches have focused on evaluating the effectiveness of existing SQLIAs detection and prevention tools which are based on SQIAs types where each tool or method can detect or prevent. In addition, this study tries to investigate further by considering the effectiveness of tools and methods which are based on tool or method ability to be deployed on particular injection parameters (Sect. 2.3.2). Based on development requirements, proposed a method or tool (i.e. anomaly-based that are prone to false

Table 1 Research questions and motivation

Research question	Motivation
RQ1: What are the types of SQLIA?	The objectives are to identify various techniques of performing SQL injection attacks against web-based database driving applications
RQ2: What are the possible injection parameters by which attackers inject the SQLIA to a database?	The aim is to identify the path by which attack injects as a malicious query on web-based database driving applications
RQ3: What are the current tools and techniques used to detect and prevent SQLIA?	The aim is to investigate the current tools and techniques proposed by different researchers in combating SQLIA against web-based database driving applications
RQ4: How effective these techniques are with respect to the development approach and their ability to be deployed in various injection parameters?	The aim is to evaluate tools and techniques considered in this review based on their ability to detect different types of SQLA and their ability to be deployed in identified injection parameters in RQ2

positive and false negative alarm). Concerning the aforementioned motivation, we defined four research questions that represent the foundation for deriving the search strategy for literature extraction (See Table 1).

2.1.3 Review Process

After identifying the need for SLR, RQs are the next step to refine the review process. This begins with related studies retrieval from different databases, studies selection, extracting the result from data in the selected studies and information synthesis. Figure 2 shows the follow of processes followed in the review process.

Searching Related studies query retrieval covered six different databases (Table 2). The search terms and guidelines adopted from which a composition of 453 different search strings have used (Fig. 3).

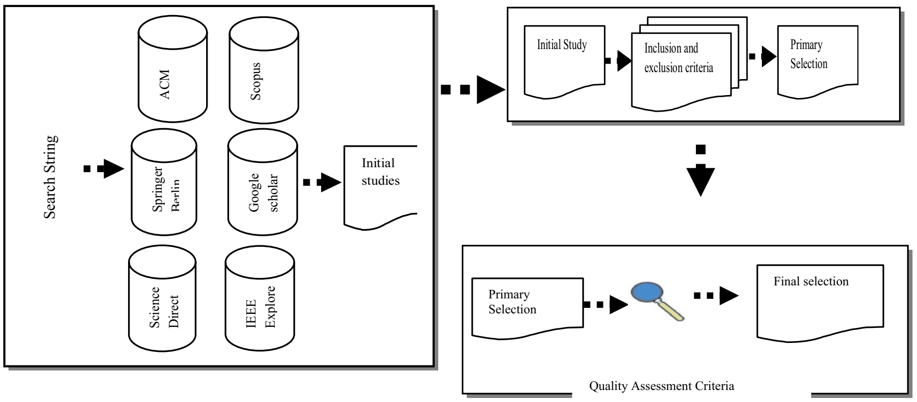


Fig. 2 Study selection process

Table 2 Search query result from different databases

S/N	Databases searched	
	Sources	Result
1	IEEE explore	119
2	Google scholar	435
3	ACM digital library	95
4	Scopus	378
5	Springer berlin	145
6	Science direct	89
	Total relevant article	1261

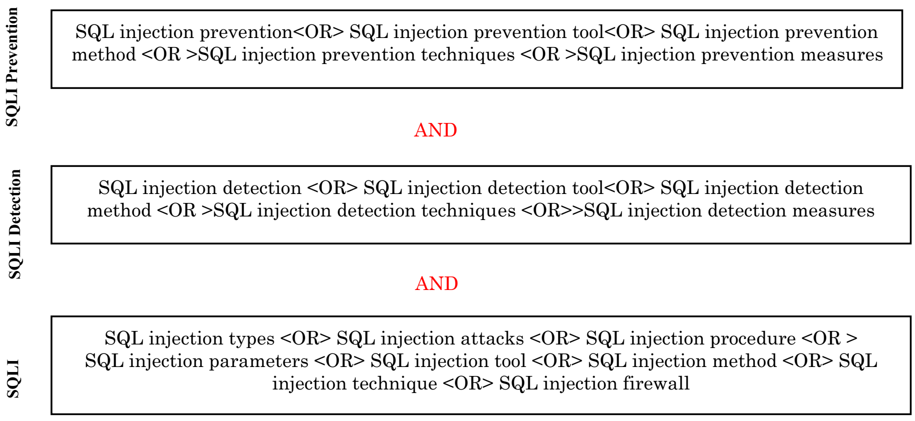

SQLI Prevention
SQL injection prevention<OR> SQL injection prevention tool<OR> SQL injection prevention method <OR >SQL injection prevention techniques <OR >SQL injection prevention measures
AND
SQLI Detection
SQL injection detection <OR> SQL injection detection tool<OR> SQL injection detection method <OR >SQL injection detection techniques <OR>>SQL injection detection measures
AND
SQLI
SQL injection types <OR> SQL injection attacks <OR> SQL injection procedure <OR > SQL injection parameters <OR> SQL injection tool <OR> SQL injection method <OR> SQL injection technique <OR> SQL injection firewall

Fig. 3 Used search strings

Based on the above strings (search term used), we retrieved 1261 peer-reviewed literatures, methods and techniques from the years 2006 to 2019 (inclusive) from six sources (Table 2).

Initial Selection This activity is carried out by screening the titles and abstracts of potential primary studies performed by the researchers against inclusion/exclusion criteria in Table 3. For almost 30% of studies, no decision could be made. In such cases, exclusion or proceeding for final selection is involved in examining the full text.

Inclusion and Exclusion Criteria Table 3 provides the summary of inclusion and exclusion criteria adopted in this study.

Final Selection After scanning the result from initial selection validation ware made on studies based on proposed SQL methods, techniques, and firewall/tool and SQL types to support the evaluation approach. By considering inclusion criteria we selected 46 studies based on selection criteria adopted from Guidelines [8] snowballing which is based on data and result presented in the considered study (Quality assessment criteria). Out of these 83 studies, one is guideline is not related to the field of the study, therefore, the total number of studies considered is 83 as shows in Table 4.

Table 3 Inclusion and exclusion criteria

	Criteria	Rationale
Inclusion	Studies in the form of a scientific peer-reviewed paper related to SQLI detection and prevention	A scientific paper guarantees a certain level of quality through a peer review and contains a substantial amount of content
	Studies that proposed a method or solution, experience, or evaluation of SQLI detection and prevention tools	We are interested in specific solutions, metrics, and analysis of SQLI detection and prevention measures
	Kitchenham's guideline also considered in this study	This work is a systematic review so we use *Kitchenham's* guideline
Exclusion	Studies that do not explicitly discuss SQLI detection and prevention	This study objective is to study the SQLIA detection and prevention approaches
	Studies that investigate the detection of SQLI vulnerabilities	These studies are not associated with detection or prevention SQLIA. Instead; they explored the SQLI vulnerability detection approaches
	Studies that do not explicitly propose a method, technique or tool to detect or prevent SQLIA	These studies are not directly enabling detection or prevention or even review analysis but rather consider conceptual SQLIA such as discussion of SQLIA types or its techniques
	Non-peer-reviewed studies, white papers, or non-English manuscripts, thesis and chapters	Although there are numerous white papers in SQLIA but we decided to exclude them because they are situational. So also these studies usually are associated with conference or journal papers and for the entire retrieved thesis or book chapters, we included the most related paper of the corresponding authors
	Related studies that do not have an evaluation results	These studies do not provide reasonable amount of information for an objective decision
	Related articles published before 2006	These studies aim to provide the current issues or trend or SQLI detection and prevention tools and we expect issues before 2006 are solved

Table 4 Search material

Selected studies	
Sources	No of selected articles
IEEE explore	16
Google scholar	9
ACM digital library	8
Scopus	7
Springer berlin	37
Science direct	6
Total relevant article	83

2.2 Conducting Study

Section 2 describes the procedure followed in SLR planning on SQLIAs detection and prevention measures using guidelines in [8] and describes the procedure in conducting the SLR of SQLIAs study. This phase begins with conducting quality assessment criteria.

2.2.1 Conduct Quality Assessment (QA)

We used the Center for Reviewer and Dissemination (CRD) and Database of Abstract of Reviews of Effect (DARE) criteria to evaluate each technique. The following four questions are asked for quality assessment and the answers to these questions are summarized in Table 5.

Based on above questions and answers in Table 5, where we evaluated selected studies as summarized in Table 6.

Table 6 shows the list of selected studies, Eighty-three (83) studies are selected out of 1261 studies. This SLR indicated that more journal articles are published than conference proceedings. Out of these studies, Sixty-nine (69) proposed tools and methods ([S2-23, S36-S83]) Twelve (12) are survey (seven analytical analyses [S24-S30] while five are experimental analysis [S31-S35]) related to SQLIAs detection and prevention measures and one [S1] is not related to this study. This analysis shows that the researchers are more focusing on solution for the detection and prevention of SQLIAs than evaluating efficiency and accuracy of existing tools and methods (See Figs. 4, 5, 6) which show demand for evaluating the current SQLIAs detection and prevention tools.

Because we used DARE criteria for evaluating the quality of study as described (See Sect. 2.2.1 Table 5) therefore, information in Table 6 shows that Nineteen (19) out of Eighty three (83) related studies [S4, S5, S6, S7, S8, 10, S11, S12, S13, S17, S19, S20, S21, S22, S24, S24, S497, S49, S60] satisfied the quality assessment using DARE scale by scoring 4/4, while Thirty two (32) related studies [S2, S3, S9, S14, S15, S18, S25, S36, S38, S39, S41, S42, S44, S45, S48, S52, S58, S59, S62-69, S74-S77, S79, S80] score 3.5/4, likewise Twenty four (24) [S16, S26, S27, S28, S29, S37, S43, S46, S50, S51, S53-S57, S61, S70-S73, S78, S81-S83] out of Eighty three (83) score 3/4, while One (1) [S35] study score value 2.5/4 and also Six (6) [S30, S31, S32, S33, S34, S40] studies score value

Table 5 Quality assessment procedure

QA	QA question	Motivation	Ev Evaluation process using DARE	Evolution assessment using DARE

QA1	Does criteria for inclusion and exclusion are well described and appropriately added in review?	The aim is to determine whether the inclusion criteria is clearly defined and discussed in the study, or partially implicit; or not defined and cannot be readily inferred	Y (yes) indicates, inclusion criteria are clearly defined.
			P (partly) indicates, partially defined.
			N (no) indicates, not defined, and cannot be readily inferred
				The following rating factor are assigned when answering QA1 question.
				Y = 1,
				P = 0.5
				N N = 0

QA2	Is domain search possibly covered all related work as literature search caryout?	Aim to determine whether four (4) or more digital libraries have been searched and some search strategies are added or all journals addressing the area considered which are identified and referenced by authors	Y(yes) indicate, either four (4) or more digital libraries have been searched and some search strategy are added or all journals addressing the considered areas which are identified and referenced by authors;
			P (partially), indicate only three (3) or four (4) digital libraries which have been searched with no addition of any search strategies, or defined searched is used with restriction of set of journals and conference proceedings,
			N (no) indicates, the authors have search up to 2 digital libraries or an extremely restricted set of journals
				The following rating factor are assigned when answering QA2.
				Y = 1,
				P = 0.5
				N = 0

QA3	Does the quality and validity of included studies have been asses by the reviewers?	Aim is to determine whether quality criteria considered are clearly defined and separated from result.	Y (yes) indicate, Quality criteria considered are clearly defined and separated from,
			P (partially), indicate that the research question involves quality issues that are addressed by the study;
			N (no) indicates that quality assessment of first result was no clearly attempted
				The following rating factor are assigned when answering QA3.
				Y = 1,
				P = 0.5
				N = 0

Table 5 (continued)

QA	QA question	Motivation	Ev Evaluation process using DARE	Evolution assessment using DARE

QA4	Does the information/studies of concern were described adequately?	The aim is to determine the detail information presented in study	Y (yes) indicates detail Information about study is presented;
			P (partially) indicates summary information about first studies is presented;
			N (no) indicates no results of individual primary studies are specified
				The following rating factor are assigned when answering QA4 Y = 1,
				P = 0.5
				N = 0

Table 6 Quality evaluation using DARE scale

Authors	ID	Type	QA1	QA2	QA3	QA4	Score
[8]	S1	Journal	0	0	0	0	0
[15]	S2	Journal	P	Y	Y	Y	3.5
[16]	S3	Journal	Y	Y	Y	P	3.5
[17]	S4	Conference	Y	Y	Y	Y	4
[18]	S5	Journal	Y	Y	Y	Y	4
[19]	S6	Conference	Y	Y	Y	Y	4
[20]	S7	Conference	Y	Y	Y	Y	4
[21]	S8	Journal	Y	Y	Y	Y	4
[22]	S9	Conference	Y	Y	Y	P	3.5
[23]	S10	Journal	Y	Y	Y	Y	4
[24]	S11	Journal	Y	Y	Y	Y	4
[25]	S12	Journal	Y	Y	Y	Y	4
[26]	S13	Journal	Y	Y	Y	Y	4
[27]	S14	Journal	Y	Y	Y	P	3.5
[28]	S15	Journal	Y	Y	Y	P	3.5
[29]	S16	Journal	Y	Y	Y	N	3
[30]	S17	Journal	Y	Y	Y	Y	4
[31]	S18	Conference	Y	Y	Y	P	3.5
[32]	S19	Journal	Y	Y	Y	Y	4
[33]	S20	Journal	Y	Y	Y	Y	4
[34]	S21	Journal	Y	Y	Y	Y	4
[35]	S22	Journal	Y	Y	Y	Y	4
[36]	S23	Journal	Y	Y	Y	Y	4
[5]	S24	Journal	Y	Y	Y	Y	4
[9]	S25	Journal	Y	Y	P	Y	3.5
[10]	S26	Journal	Y	Y	N	Y	3
[14]	S27	Conference	Y	Y	N	Y	3
[7]	S28	Journal	Y	Y	N	Y	3
[11]	S29	Journal	Y	Y	N	Y	3
[37]	S30	Journal	P	P	N	Y	2
[38]	S31	Journal	P	P	N	Y	2
[39]	S32	Journal	P	P	N	Y	2
[40]	S33	Journal	P	P	N	Y	2
[41]	S34	Journal	P	P	N	Y	2
[42]	S35	Journal	Y	P	N	Y	2.5
[43]	S36	Conference	Y	P	Y	Y	3.5
[44]	S37	Journal	Y	P	Y	P	3
[45]	S38	Journal	P	Y	Y	Y	3.5
[46]	S39	Conference	P	Y	Y	Y	3.5
[47]	S40	Journal	P	P	N	Y	2
[48]	S41	Journal	P	Y	Y	Y	3.5
[34]	S42	Journal	P	Y	Y	Y	3.5
[49]	S43	Journal	P	P	Y	Y	3
[50]	S44	Journal	P	Y	Y	Y	3.5
[51]	S45	Journal	P	Y	Y	Y	3.5
[52]	S46	Conference	P	Y	P	Y	3

Table 6 (continued)

Authors	ID	Type	QA1	QA2	QA3	QA4	Score
[6]	S47	Journal	Y	Y	Y	Y	4
[53]	S48	Journal	Y	P	Y	Y	3.5
[54]	S49	Journal	Y	Y	Y	Y	4
[55]	S50	Conference	P	Y	P	Y	3
[56]	S51	Conference	P	Y	P	Y	3
[57]	S52	Journal	Y	Y	P	Y	3.5
[58]	S53	Conference	P	Y	P	Y	3
[59]	S54	Conference	P	Y	P	Y	3
[60]	S55	Conference	P	Y	P	Y	3
[61]	S56	Conference	P	Y	P	Y	3
[62]	S57	Conference	P	Y	P	Y	3
[63]	S58	Journal	P	Y	Y	Y	3.5
[64]	S59	Journal	P	Y	Y	Y	3.5
[58]	S60	Journal	Y	Y	Y	Y	4
[65]	S61	Conference	P	Y	Y	P	3
[66]	S62	Journal	P	Y	Y	Y	3.5
[67]	S63	Journal	P	Y	Y	Y	3.5
[57]	S64	Journal	P	Y	Y	Y	3.5
[68]	S65	Journal	P	Y	Y	Y	3.5
[69]	S66	Journal	P	Y	Y	Y	3.5
[70]	S67	Journal	P	Y	Y	Y	3.5
[71]	S68	Journal	P	Y	Y	Y	3.5
[72]	S69	Journal	P	Y	Y	Y	3.5
[73]	S70	Conference	P	Y	P	Y	3
[70]	S71	Conference	P	Y	P	Y	3
[74]	S72	Conference	P	Y	P	Y	3
[75]	S73	Conference	P	Y	P	Y	3
[76]	S74	Journal	P	Y	Y	Y	3.5
[77]	S75	Journal	P	Y	Y	Y	3.5
[78]	S76	Journal	P	Y	Y	Y	3.5
[79]	S77	Journal	P	Y	Y	Y	3.5
[80]	S78	Conference	P	Y	Y	p	3
[81]	S79	Journal	P	Y	Y	Y	3.5
[82]	S80	Journal	P	Y	Y	Y	3.5
[83]	S81	Conference	P	Y	P	Y	3
[84]	S82	Conference	P	Y	P	Y	3
[85]	S83	Conference	P	Y	P	Y	3

of 2/4, One study S1 score 0/4 as is not related to this study which is used as a guideline. In summary, the information in Table 6 shows that the maximum number of related studies considered are satisfied with quality assessment questions.

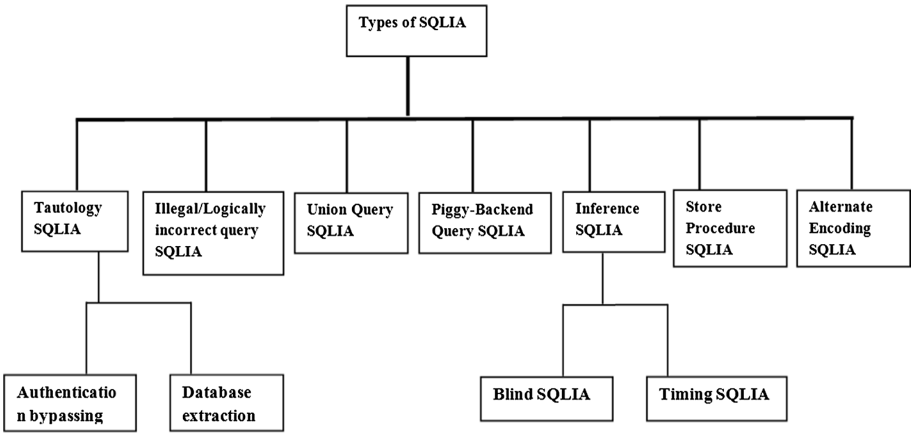
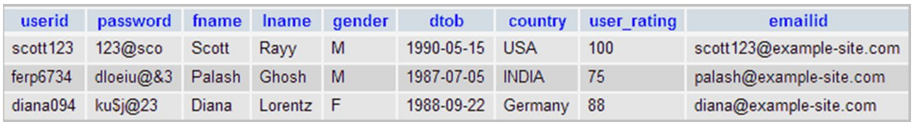
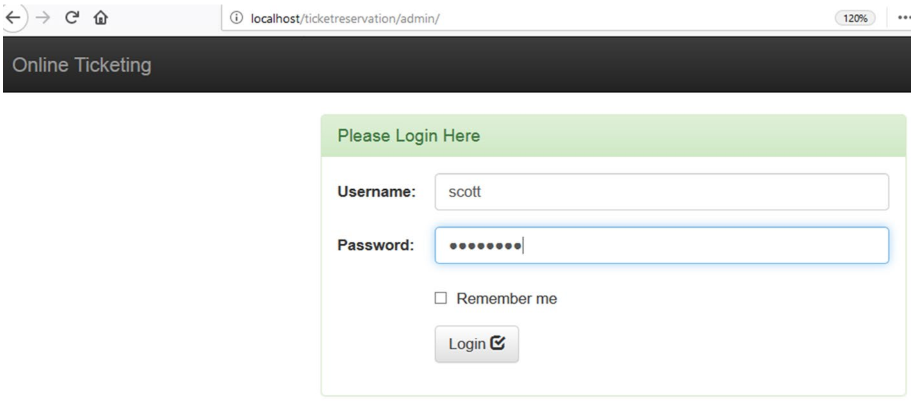


Fig. 4 SQLIA types



userid	password	fname	lname	gender	dtob	country	user_rating	emailid
scott123	123@sco	Scott	Rayy	M	1990-05-15	USA	100	scott123@example-site.com
ferp6734	dloeiu@&3	Palash	Ghosh	M	1987-07-05	INDIA	75	palash@example-site.com
diana094	kuSj@23	Diana	Lorentz	F	1988-09-22	Germany	88	diana@example-site.com

Fig. 5 Users database records



localhost/ticketreservation/admin/

Online Ticketing

Please Login Here

Username: scott

Password: ●●●●●●●●

☐ Remember me

Fig. 6 User input credential form to use the system with valid credential

2.2.2 Data Extraction and Synthesis

We carefully extracted and synthesized the data from each study for collecting the following information:

1. Classification of study and its topic domain.
2. Types of SQLIAs

3. Injection parameters
4. Assessment of the proposed technique's effectiveness, if available

Having extracted data from the considered studies, we analytically evaluate them, without any experimental proof (using experience), based on the classification of the study domain and with respect to attack the injection parameters. Basically, during the evaluation, we combine the classification for domain types of SQLIAs and tools effectiveness which is one of the evaluation strategy entitled "evaluation with respect to attack types" which enable us to answer research question four (RQ4).

2.3 Documenting

The aforementioned sections described the procedure followed in planning of this SLR using guideline in [8]. This phase begins with conducting quality assessment criteria.

2.3.1 Types of SQLIAs Used for Attacks

In view of RQ1 (See Table 1), the study explored Eighty-two (82) studies. Out of the 82 reviewed studies, 44 fully presented the data on different techniques by which attackers use SQLIAs for attacking the web application database (See Table 6). Attackers use SQLIA to attack web applications and these attacks are fall in different types, depending on what attackers want to achieve. Moreover, these attacks are classified into Seven (7) types [5, 9–14] as represented in Fig. 4.

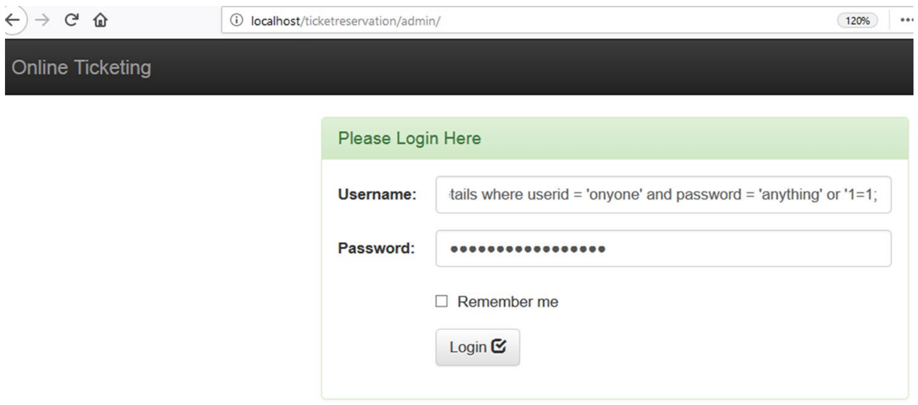
2.3.1.1 Tautology Attack This is a type of attack that takes advantage of "WHERE" clause in SQL statement to evaluate the results returned by Query in a relational database which is always true. Attackers use this type of attack to achieve authentication bypassing in web applications or perform unauthorized database extraction [5, 9, 10, 56] Authentication bypassing: all relational database management system with no exception evaluate SQL query with "OR 1 = 1" where clause is always true. Also, in a relational database management system, anything followed by comment (–) will not be processed. For example, consider the Fig. 5 database records presenting users credential and personal details and Fig. 6 shows the client sides that takes input credential from user for authentication to use system (Table 7).

Upon pressing the login button as shown in Fig. 6, the scott details would be submitted and passed to admin.php script (`$ucredential=$_POST [' ucredential'];` `$pcredential=$_POST['pucredential'];` `$Query="select * from user_ ucredential where userid='$u ucredential' and password='$p ucredential' ";` `$result=mySQL_query($SQL);`) where it would be validating against scott credential details stored in fig. if the input from Fig. 44 matched with one stored in the scott would be grand access to the system and logical record output presented in Table and if the details did not match, invalid user name or password would be received.

In the above Fig. 7, the input username credential contains malicious SQL injection attack codes; input by malicious user and password could be anything. When this code is passed to $ucredential and $pcredential and executed by the database server, this might result in a serious threat. This means that the interpreter is fetched all the records which exist from the users_details table and returned them into $result which will be presented

Table 7 Logical output of successful authentication of scott credential

User ID	Password	First name	Last name	Gender	Date of birth	Country	User rating	Email ID
Scott	123@sco	Scott	Rayy	M	1990-05-15	USA	100	scott123@example-site.com


localhost/ticketreservation/admin/

Online Ticketing

Please Login Here

Username: tails where userid = 'onyone' and password = 'anything' or '1=1;

Password: ●●●●●●●●●●●●●●●●●

Remember me

Login

Fig. 7 User input credential form to use the system with tautology SQL injection attack

by a malicious user. The above code can be interpreted as $SQL = ”select * from user_details where ucredential = ’select * from user_details where userid = ’onyone’ and password = ’anything’ or ‘1=1;’ and password = ’anything’ or ‘x’ = ‘x’ “;

With the help of WHERE clause the statement of 1=1 or x=x is always returning to true for every row, therefore the query will return all the records. In this way, an attacker able to view all the personal information of the users for example of output presented in Table 8.

2.3.1.2 Illegal or Incorrect Logical Query Knowing the server, schema, table, and column names make it easy for attackers to gain unauthorized access to the system [5, 9, 10]. For example, consider the Fig. 8 below with URL input as http://localhost/ticketreservation/reserved.php/ select * from user_details where userid = ’onyone’ and password = ’anything’ or ‘1=1;’.

If you notice at the end of the ULR http://localhost/ticketreservation/reserved.php/ a malicious code is introduced. This is local host website, where we test the malicious node activity. This disturbs the database engine because when you type something within the quote it is used to tell the database that this is a query and to process it. So after processing makes the database engine returns the error message in Fig. 9.

As can be seen information in Fig. 9, it indicates database server, version, platform and other vital information which helps the malicious users to gather the required information to lunch devastating attacks to the target system.

2.3.1.3 Inference Attack This attack can be classified into Blind and Timing SQLIA [5].

A. Blind SQL Injection Attack

This is another method of doing database fingerprinting. Sometimes database engines can be configured to hide database error messages and return a generic error to the user when there is an SQL syntax error in the user’s SQL statement. This can serve as a method to prevent attackers from database fingerprinting by using illegal or incorrect query methods. However, this does not mean the database is secure; it only conceals the return default error message which will be difficult for attackers who rely on database fingerprinting as a first step in carrying out an attack. Thus blind SQL injection attack

Table 8 Logical output of successful tautology SQL injection attacks on user authentication form

User ID	Password	First name	Last name	Gender	Date of birth	Country	User rating	Email ID
ferp6734	dloeiu@&3	Palash	Ghosh	M	1987-07-05	INDIA	75	palash@example-site.com
diana094	ku$j@23	Diana	Lorentz	F	1988-09-22	Germany	88	diana@example-site.com
Scott	123@sco	Scott	Rayy	M	1990-05-15	USA	100	scott123@example-site.com

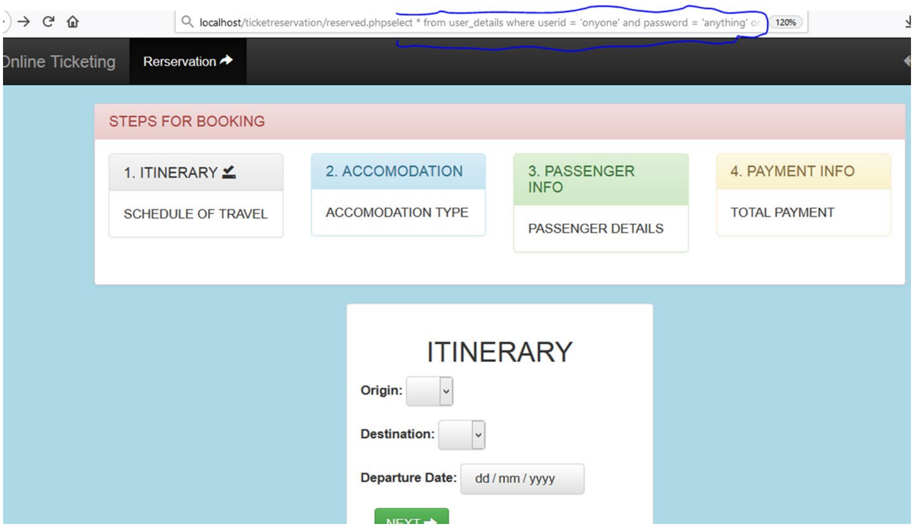
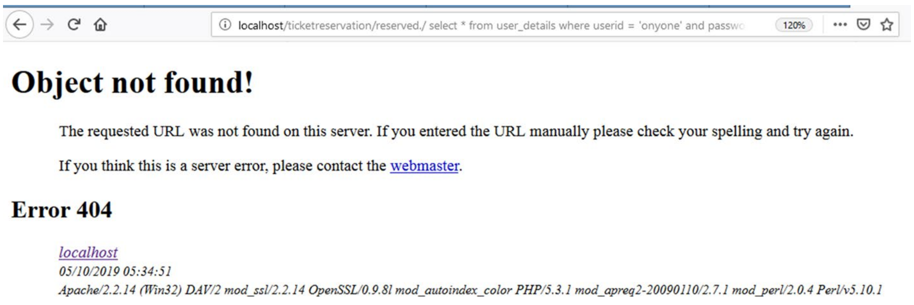


Fig. 8 Example of illegal or incorrect logical query


localhost/ticketreservation/reserved./ select * from user_details where userid = 'onyone' and passw
120%

Object not found!

The requested URL was not found on this server. If you entered the URL manually please check your spelling and try again.

If you think this is a server error, please contact the webmaster.

Error 404

localhost
05/10/2019 05:34:51
Apache/2.2.14 (Win32) DAV/2 mod_ssl/2.2.14 OpenSSL/0.9.8l mod_autoindex_color PHP/5.3.1 mod_apreq2-20090110/2.7.1 mod_perl/2.0.4 Perl/v5.10.1

Fig. 9 Output of illegal or incorrect logical query

can be used to deduce if there is a security mechanism implemented in the web application or not. Blind SQL injection attacks can be achieved by asking a series of true or false queries in the database. In this case, the attacker tries to inject the following statements:

*SELECT * FROM emp_name, emp_address, gender, from employee where 1=0; drop employee//-----------Statement (1)*
*SELECT * FROM emp_name, emp_address, gender, from employee where1=1; drop employee//-----------------------Statement (2)*

After executing the above Boolean malicious SQL query, an attacker knows about the database is secure or not. If the same response is delivered (return a generic error message) there is protection mechanism that has detected an attack and blocked the query from executing and returned an error message to the user because all of the statement contains malicious words. A different response means that the query has reached inside the database engine and has been executed. Therefore, the first query returns an error message because it is an incorrect query while the second mayor may not return any error message because it is a correct query.

B. Timing Attacks

In this type of attack, the response time which the database takes to respond to the user's query is noticed which helps to know some information from a database. This method uses an if–then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, an attacker can extract information from a database using a vulnerable parameter.

declare @ varchar (8000) select @s = db_name () if (ascii
(substring(@s, 1, 1)) & (power (2, 0))) > 0 waitfor delay '0:0:8

2.3.1.4 Union Attack This is the most common type of attack used by attackers in gaining access to restricted data in other tables. The malicious SQL query can be appended by an attacker to combine with valid SQL queries to gain unauthorized access to extra data [5, 9, 10]. For an example of a malicious attack, consider the following example where online human resources in a particular company allow employees to view only their details online. A malicious user can access extra information such as employee salary and phone number from Fig. 10.

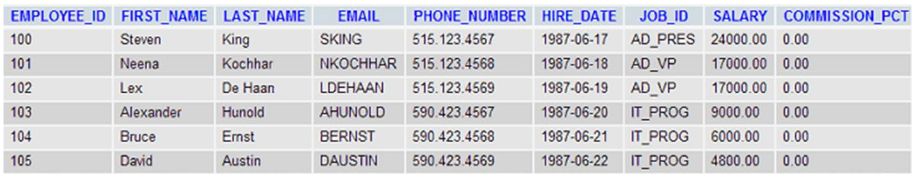
`The information on fig can be interpreted as`

SELECT * FROM user_details WHERE userid='' UNION SELECT * FROM EMP_ DETAILS – ' and password='admin'

This means that after successful authentication, the malicious user has access to Emp_ details table with the help of two dashes (–) comments.

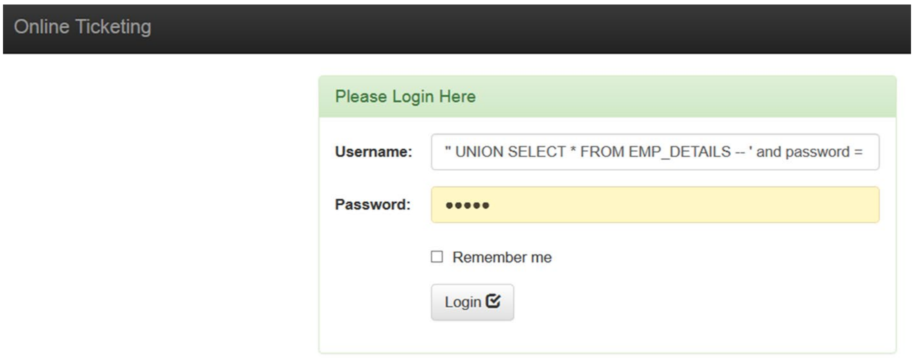
This feature creates an opportunity for an attacker to perform dangerous action in the database. In this case, a valid query is terminated by (;) and a malicious query is added. After processing the valid query, a malicious query is then executed, unlike in piggy back-end query where a malicious query has joined with a valid query and processed as a single joined query (Fig. 11).

2.3.1.5 Alternate Encoding Most of the SQL injection mechanisms that use filters prohibit the use of quote (') in the SQL statement which can be used in constructing different kinds of malicious query requests to the database. In this case for an attacker to bypass such a filter and has to convert SQL query into alternatives encodings such as hexadecimal, ASCII or Unicode. Converting SQL query into alternate encode enables them to carry out their attacks. For example



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00
105	David	Austin	DAUSTIN	590.423.4569	1987-06-22	IT_PROG	4800.00	0.00

Fig. 10 Employee details database records


Online Ticketing

Please Login Here

Username: " UNION SELECT * FROM EMP_DETAILS -- ' and password =

Password: •••••

☐ Remember me

Login

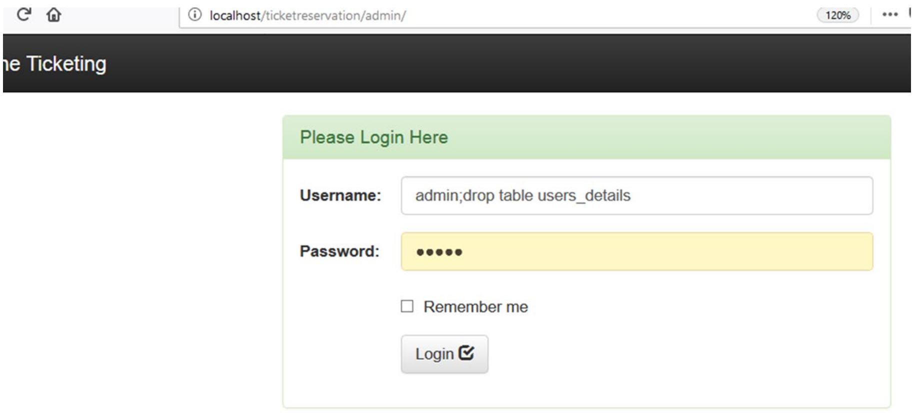
Fig. 11 Example of union SQL injection attack

"0; exec (0x73587574 64 5f177 6e), " and the result query is: SELECT accounts FROM login WHERE username=" AND password=0; exec (char (0x73687574646j776e))

The above example uses the char () function and ASCII hexadecimal encoding. The char () function takes hexadecimal encoding of character(s) and returns the actual character(s). The stream of numbers in the second part of the injection is ASCII hexadecimal encoding of the attack string. This encoded string is translated into the shutdown command by the database when it is executed.

2.3.1.6 Piggery-Backend Query Attack Some of the database engines support stacked queries by default. This feature creates an opportunity for an attacker to perform dangerous actions in the database. In this case, a valid query is terminated by (;) and a malicious query is added. After processing the valid query, a malicious query is then executed, unlike in a union query where a malicious query is joined with a valid query and processed as a single joined query. For example: consider in Fig. 12 below.

The information on Figure can be interpreted as


localhost/ticketreservation/admin/

ne Ticketing

Please Login Here

Username: admin;drop table users_details

Password: •••••

☐ Remember me

Login

Fig. 12 Example of Piggy Backend query SQL injection attack

```
select * from user_details where userid='admin' and
password='admin'; drop table user_details - '.
```

Once the first query executed then, the database server would use the query delimiter(";") and process the injected second query. The result of executing the second query would be to drop table users_details table, which would destroy valuable information.

2.3.1.7 Stored Procedure A stored procedure is a part of the database where programmers could set an extra abstract layer on the database as security to prevent SQL injection attack. As the stored procedure could be coded by the programmer, so, this part is known as an injectable web application. Depending on specific database storage procedure there are different ways to attack [5, 9].

2.3.2 Injection Parameters

In view of RQ2 (See Table 1), this section provides a detailed description of the injection parameter (HTTP GET, HTTP POST, Cookies, etc.) where attackers craft malicious queries to the application databases through a client application. Two (2) [5, 9] out of 83 studies explored in this study have fully presented different injection parameters by which attackers inject malicious queries in web-based driving database applications as discussed in Sects. 2.3.2.1 to 2.3.2.4 below.

2.3.2.1 Injection Through User Input Field User input fields are provided in web applications to enable web application users to request information from the backed databases to the user with the help of HTTP POST and GET (See Fig. 6). These inputs are connected with the backend database using SQL statements to retrieve and render the requested information for users or to allow users to connect to the system. User input fields are vulnerable to SQL injection attacks if input provided by the user is not sanitized before sending it to the database server for processing, which enables attackers to modify intended queries to perform malicious action in the system.

2.3.2.2 Injection Through Cookies Cookies are structures that maintain the persistence of web applications by storing state information on the client machine. When a client returns to a Web application, cookies can be used to restore the client's state information. If a Web application uses the cookie's contents to build SQL queries, then an attacker can take this opportunity to modify cookies and submit to the database server.

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

2.3.2.4 Second Order Injection In second-order injections, attackers plant malicious inputs into a system or database to indirectly trigger an SQLIA. When that input is called at a later

Fig. 13 Trend of the study

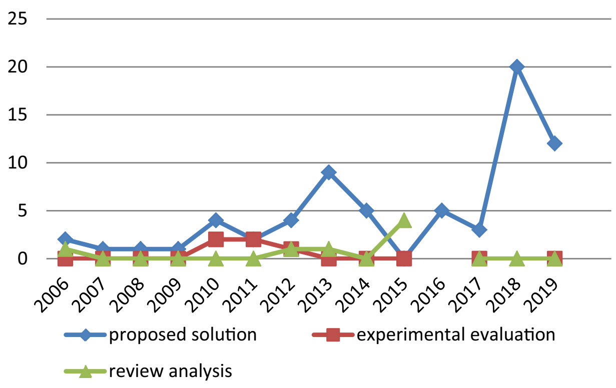
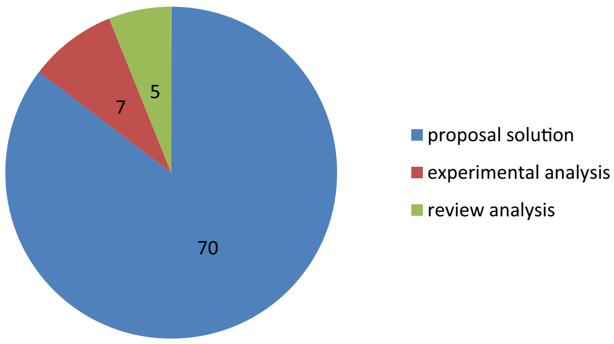


Fig. 14 Percentage selected studies with respect to Focus



time when an attack occurs, the input that modifies the query to construe an attack does not come from the user, but from within the system itself.

3 SQLIAs Detection and Prevention Approaches

In view of RQ3 (See Table 1), the study provides the trend of current SQLIAs detection and prevention tools and methods proposed by various researchers to handle problems of SQL injection attacks. These methods start from the development of best practices to automatic tools for detecting and preventing SQL injection attacks. We also considered considering studies that evaluate the effectiveness of these proposed tools and methods (both experimentally and analytically) as to be summarized in Fig. 13 below.

Result of analysis in Fig. 14, presents that there is a side by side effort by different researchers in trying to evaluate the effectiveness of existing SQLIAs detection and prevention tools and methods from 2006 to 2009 where the experimental evaluation goes high in the year 2010–2011 and again goes down in the year 2013–2015 compared to analytical evaluation (review analysis). While in case of propose tools and methods it shows are searchers are putting more effort into finding the way of combating with the problem of SQLIAs, which shows a significantly increasing number of the proposed method each year unless the year 2011 and 2015 with 2018 with highest proposed methods and tools.

Figure 13 shows the trends of the studies related to SQLIAs detection and prevention measures; similarly, the Fig. 6 shows the percentage (%) number of studies extracted and selected from six different databases related to SQLIAs detection and prevention measures in this study with proposed solution with 70 studies or 85.4%, review analysis 5 studies or 6.1% and experimental evaluation 7 or 8.5%. In summary, the study shows that researchers focus more on proposing a solution to tackle the problem of SQLIAs rather than evaluating the efficiency and accuracy of existing tools and methods (Figs. 13 and 14).

3.1 Discussion of Reviewed SQLIAs Detection and Prevention Approaches

In view of RQ4 (See Table 1), we assess the effectiveness of current SQLIA detection and prevention measures with respect to development approach and the ability to be deployed in various injection parameters considered (Sect. 2.3.2).

To achieve that, the following questions were asked:

- What are the scopes of current techniques to address particular attack type?
- How effective is this technique is with respect to deployment requirements?
- Do current techniques be deployed in each injection parameters?
- Does techniques required code modification when new web page is added?

3.2 Discussion on SQLIAs Prevention Tools Based with Respect to Attack Types

We analyzed and evaluated each proposed method as shown in Tables 9 and 10. To ensure a particular tool or method is capable of addressing a particular attack type described (Sect. 2.3.1); we used analytical evaluation based on experience. We have not assessed any of the tools or methods in real-time practice for the reason that most tools or method's implementation codes are not available or some methods are not implemented. Table 9 presents evaluations of SQLIAs detection tools and methods considered in this study.

As indicated in Table 9 out of the tools and methods considered, only Three (3) of them, [S4], [S6] and [S7] focus on addressing all types of SQLIAs considered the rest of proposed tools and method focusing on addressing a subset of SQLIAs. However, the effectiveness of these tools and methods considered for addressing particular types of SQLIAs varies depending on the approach used, in developing tools or method, and its ability to be deployed in various injection described parameters, (See Sect. 2.3.2 for injection parameters consider in this study). For example, we used four different symbols "•", "×","◦" and "–" to describe the effectiveness of the tool or method considered in Table 8, with "•" indicates that a method can successfully stop all attacks of that type, "×" indicates that a method is not able to stop all attacks of that type and "◦" indicates that a method can address the attack type considered, but cannot provide any guarantee of completeness."–" indicates that a method can partially address the attack type considered, but cannot provide a guarantee of completeness.

For example, tick dot symbol ("•") as can be seen in Table 8 is used for [S3], [S5], [S12], [S13], [S14], [S19], [S20], [S21], [S22], [S23] which indicates this method or tool can guarantee protection of particular SQLIAs type which they are developed to addressed (but cannot prevent out of their scope). However, out of these tools and methods, none of the tools can successfully be deployed to prevent all injection parameters considered (See Table 9). The ("◦") and ("–") symbols are used in Table 9 to indicate that method or tool can partially detect and prevent SQLIAs type considered without guaranteeing that a given

Table 9 Evaluation of prevention tools and methods based on attack types

ID	Tautology	Illegal/incorrect	Union query	Piggy-Backend	Inference	Alternate encode	Stored procedure
SQLIA Prevention Tools and Methods							
[S2]	●	●	●	●	×	●	●
[S3]	●	●	●	●	×	●	●
[S4]	–	–	–	–	–	–	–
[S5]	●	●	●	●	×	●	×
[S6]	○	○	○	○	○	○	○
[S7]	○	○	○	○	○	○	○
[S8]	○	○	○	○	×	○	×
[S9]	○	○	○	○	×	○	×
[S10]	○	○	○	○	×	○	×
[S11]	○	○	○	○	×	×	×
[S12]	●	●	●	×	×	×	×
S13]	●	●	●	×	×	×	×
[S14]	●	●	●	×	×	×	×
[S15]	–	–	–	–	–	–	×
[S16]	○	○	○	○	○	○	×
[S17]	–	–	–	×	×	–	×
[S18]	–	–	–	×	×	×	–
[S19]	●	●	●	×	×	●	×
[S20]	●	●	●	×	\x	×	×
[S21]	●	●	●	×	×	×	×
[S22]	●	●	●	×	×	×	×
[S23]	●	●	●	×	×	×	×
[S48]	○	○	○	○	×	×	×
[S49]	●	●	●	×	×	×	×
[S50]	●	●	●	×	×	×	×
[S53]	●	●	●	×	×	×	×

Table 9 (continued)

ID	Tautology	Illegal/incorrect	Union query	Piggy-Backend	Inference	Alternate encode	Stored procedure
[S54]	–	–	–	–	–	–	x
[S55]	◦	◦	◦	◦	◦	◦	x
[S56]	–	–	–	x	x	–	x
[S57]	●	●	●	x	x	x	x
[S58]	●	●	●	x	x	x	x
[S59]	●	●	●	x	x	x	x
[S60]	–	–	–	–	–	–	x
[S61]	●	●	●	x	x	x	x
[S62]	●	●	●	x	x	x	x
[S63]	●	●	●	x	x	x	x
[S64]	◦	◦	◦	◦	x	x	x
[S70]	●	●	●	x	x	x	x
[S71]	●	●	●	x	x	x	x
[S72]	●	●	●	x	x	x	x
[S73]	–	–	–	–	–	–	x
[S74]	●	●	●	x	x	x	x
[S75]	●	●	●	x	x	x	x
[S76]	●	●	●	x	x	x	x

"●" indicates that a method can successfully stop all attacks of that type

"×" indicates that a method was not able to stop all attacks of that type

"◦" indicates that a method can address the attack type considered, but cannot provide any guarantee of completeness

"–" indicates that a method can partially address the attack type considered, but cannot provide guarantee of completeness

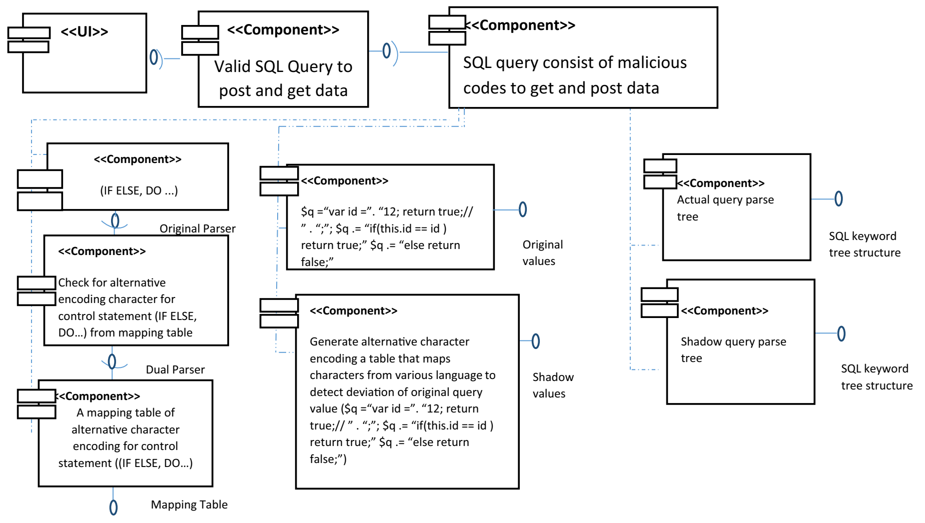

<<UI>>
<<Component>> Valid SQL Query to post and get data
<<Component>> SQL query consist of malicious codes to get and post data
<<Component>> (IF ELSE, DO ...)
Original Parser
<<Component>> Check for alternative encoding character for control statement (IF ELSE, DO...) from mapping table
Dual Parser
<<Component>> A mapping table of alternative character encoding for control statement ((IF ELSE, DO...)
Mapping Table
<<Component>> $q ="var id =". "12; return true;// " . ";"; $q .= "if(this.id == id) return true;" $q .= "else return false;"
Original values
<<Component>> Generate alternative character encoding a table that maps characters from various language to detect deviation of original query value ($q ="var id =". "12; return true;// " . ";"; $q .= "if(this.id == id) return true;" $q .= "else return false;")
Shadow values
<<Component>> Actual query parse tree
SQL keyword tree structure
<<Component>> Shadow query parse tree
SQL keyword tree structure

Fig. 15 Component of the solutions proposed in studies S4

method prevents the future attack of similar addressed type. We used ("∘") for methods that implement anomaly or machine learning-based approach to detect and prevent SQLIAs for example [S6], [S7], [S8], [S9], [S10], [S11], and [S16], this is because these approaches use sets of typical application queries as input data set to train the protection model, thus any query that goes against the model might result in false positive or false negative. Therefore, the effectiveness of these tools and methods is highly dependent on the quality of training data set used and how good the model trained, as poor training data set and model result in false-positive and negative. Thus, the effectiveness of methods and tools implementing these approaches is considered partial using circle ("∘") symbol as shown in Table 9. Other methods considered as partial are [S4], [S15], [S17] and [S18] methods that use SQL query related errors (first-order SQLI vulnerability) to detect prevent SQLIAs as SQL query related errors is only one of the many possible ways to prevent of SQLIAs. We used ("–") to represents tools and methods implementing such an approach (Table 9).

Diglossia is tool that is able to partially adress all type of SQL injection attack considered in this study (Table 9). Diglossia consist of two major conponet (Fig. 15) that intecept user queries (valid and malacious) break it into SQL keyword. This enable the tool look for malicious keyword or character in the user request to database.

Alternate encoding and stored procedure are the most important case of SQL injection attacks that are hard to defend by many of the proposed tools and methods considered. However, S4 provides a partial solution with a filter that detects and prevent the use of quote (') in the user input, to avoid malicious request that is being constructed with ('). While in the case of the stored procedure, S4 can examine code that generates the query when stored is executed on the database unlike most of the methods considered focus on preventing an attack on queries that are generated with applications.

3.3 Discussion on SQIAs Detection Tools Based with Respect to Attack Types

Table 9 above represents an evaluation of SQLIAs prevention tools and method while Table 10 below represents an evaluation of SQLIAs detection tools. In Table 10 we

Table 10 Evaluation of detection tools and methods based on attack types

ID	Tautology	Illegal/incorrect	Union query	Piggy-Backend	Inference	Alternate encode	Stored procedure
SQLIA Detection Tools And Methods							
[S36]	✓	✓	✓	✓	x	✓	x
[S37]	✓	✓	✓	✓	x	✓	x
[S38]	✓	✓	✓	✓	✓	✓	x
[S39]	✓	✓	✓	✓	✓	✓	x
aS40]	✓	✓	✓	✓	✓	✓	x
[S41]	✓	✓	✓	✓	x	x	x
[S42]	✓	✓	✓	✓	x	✓	x
[S43]	✓	✓	✓	✓	✓	✓	x
[S44]	✓	✓	✓	✓	x	✓	x
[S45]	✓	✓	✓	✓	✓	✓	x
[S46]	✓	✓	✓	✓	✓	✓	x
[S51]	✓	✓	✓	✓	✓	✓	x
[S52]	✓	✓	✓	✓	x	x	x
[S65]	✓	✓	✓	✓	x	✓	x
[S66]	✓	✓	✓	✓	✓	✓	x
[S67]	✓	✓	✓	✓	x	✓	x
[S68]	✓	✓	✓	✓	x	x	x
[S69]	✓	✓	✓	✓	x	✓	x
[S77]	✓	✓	✓	✓	✓	✓	x
[S78]	✓	✓	✓	✓	x	✓	x
[S79]	✓	✓	✓	✓	✓	✓	x
[S80]	✓	✓	✓	✓	✓	✓	x
[S81]	✓	✓	✓	✓	✓	✓	x
[S82]	✓	✓	✓	✓	✓	✓	x
[S83]	✓	✓	✓	✓	✓	✓	x

✓ “indicate tool or method can address of attack considered

x “indicate tool or method cannot address particular attack considered described effectiveness of each tools and method considered using Four different symbols while in Table 8 we used only Two different symbols (and X) this is because in detection approach considered in different researchers uses similar approach (dynamic approach or penetration testing) in trying to resolve problem of SQLIAs while in prevention approach different researchers employed different approaches (i.e. anomaly-based, machine learning-based blacklisting and white listening, etc.) in developing these tools and methods and some of these methods are problematic in nature i.e. anomaly-based (prone to false and negative alarm) some cannot be deployed in every injection parameters considered in this study i.e. whitelisting and blacklisting approaches.

Table 10 shows that most of the detection tools and method considered in this study are able to resolve" tautology, illegal or incorrect query, union query and alternate encoding SQLIAs" while inference and stored procedure attack seems to be a difficult attack to be addressed by many of the tools and methods considered this because the code that generates the query is stored and executed on the database and most of the methods considered focus on preventing attack on queries that are generated with applications. However, it is important to note that we did not take precision into account for evaluation, that is to say, many methods and tools considered are based on conservative analysis that may result in false positive.

3.4 Evaluation of SQLIAs Detection and Preventions Tools and Methods with Respect to Injection Parameters

In this section we combine evaluation of SQLIAs detection and prevention tools and methods together (Table 11), this is because every attacker who wants to perform SQLIAs against web-database driving applications has to use one or more injection parameters considered in this study, therefore, there is no need of separation of evaluation since this injection parameter are same to any web-database driving application. In this regard, we analyzed each tool and method considered with respect to their handling of the various injection mechanisms described (Sect. 2.3.2). We used "Yes" to indicate a tool or method that can be deployed to that injection parameter and "No" to indicate that the tool cannot be deployed that parameter injection parameter (Table 11).

Table 11 shows only [S2, S3, S4, S6, S7, S8, S9, S42, S47, S63, S64, S66, S75, S78, S79] can be deployed in "URL login, search, and cookies input fields, while [S5, S10, S11, S16, S17, S19, S20, S21, S37, S38, S40, S41, S43, S45, S46, S47-S49, S54, S55, S57-S59, S65, S81, S82] can be deployed in "URL, login and search input fields and [S12, S13, S14, S15, S18, S22, S23, S36, S39, S44, S50-53, S56, S60-S62] can only be deployed in "URL and login" input fields. This shows that none of the studies (tool or method) considered can be deployed to detect or prevent an attack that exploits the server-side vulnerability. This is due to the fact that server-side is vulnerable to second-order SQLIV which is not a problem of sanitizing sensitive function but is intentionally created by attackers through vulnerable parts of the application (not necessarily through Login. Add user page or ULR attacker may also use file inclusion attack to exploit dynamic file include) and reside in application database. In summary, it is important to know that all of the tools and method considered can address attacks through URL and login input fields, halve of the tools and method considered can examine queries in search input field, average number of the tools and methods

Table 11 Evaluation of detection and prevention tools and methods based on injection parameters

ID	URL	Login	Search	Cookies	Server side
[S2]	Yes	Yes	Yes	Yes	No
[S3]	Yes	Yes	Yes	Yes	No
[S4]	Yes	Yes	Yes	Yes	No
[S5]	Yes	Yes	Yes	No	No
[S6]	Yes	Yes	Yes	Yes	No
[S7]	Yes	Yes	Yes	Yes	No
[S8]	Yes	Yes	Yes	Yes	No
[S9]	Yes	Yes	Yes	Yes	No
[S10]	Yes	Yes	Yes	No	No
[S11]	Yes	Yes	Yes	No	No
[S12]	Yes	Yes	No	No	No
[S13]	Yes	Yes	No	No	No
[S14]	Yes	Yes	No	No	No
[S15]	Yes	Yes	No	No	No
[S16]	Yes	Yes	Yes	No	No
[S17]	Yes	Yes	Yes	No	No
[S18]	Yes	Yes	No	No	No
[S19]	Yes	Yes	Yes	No	No
[S20]	Yes	Yes	Yes	No	No
[S21]	Yes	Yes	Yes	No	No
[S22]	Yes	Yes	No	No	No
[S23]	Yes	Yes	No	No	No
[S36]	Yes	Yes	No	No	No
[S37]	Yes	Yes	Yes	No	No
[S38]	Yes	Yes	Yes	No	No
[S39]	Yes	Yes	No	No	No
[S40]	Yes	Yes	Yes	No	No
[S41]	Yes	Yes	Yes	No	No
[S42]	Yes	Yes	Yes	Yes	No
[S43]	Yes	Yes	Yes	No	No
[S44]	Yes	Yes	No	No	No
[S45]	Yes	Yes	Yes	No	No
[S46]	Yes	Yes	Yes	No	No
[S47]	Yes	Yes	Yes	Yes	No
[S48]	Yes	Yes	Yes	No	No
[S49]	Yes	Yes	Yes	No	No
[S50]	Yes	Yes	No	No	No
[S51]	Yes	Yes	No	No	No
[S52]	Yes	Yes	No	No	No
[S53]	Yes	Yes	No	No	No
[S54]	Yes	Yes	Yes	No	No
[S55]	Yes	Yes	Yes	No	No
[S56]	Yes	Yes	No	No	No
[S57]	Yes	Yes	Yes	No	No
[S58]	Yes	Yes	Yes	No	No
[S59]	Yes	Yes	Yes	No	No

Table 11 (continued)

ID	URL	Login	Search	Cookies	Server side
[S60]	Yes	Yes	No	No	No
[S61]	Yes	Yes	No	No	No
[S62]	Yes	Yes	No	No	No
[S63]	Yes	Yes	Yes	Yes	No
[S64]	Yes	Yes	Yes	Yes	No
[S65]	Yes	Yes	Yes	No	No
[S66]	Yes	Yes	Yes	Yes	No
[S75]	Yes	Yes	Yes	Yes	No
[S78]	Yes	Yes	Yes	Yes	No
[S79]	Yes	Yes	Yes	Yes	No
[S81]	Yes	Yes	Yes	No	No
[S82]	Yes	Yes	Yes	No	No
[S83]	Yes	Yes	No	No	No

considered can examine queries in cookie fields, and none of the tools or method considered can detect or prevent attacks that take advantage of server-side SQLI vulnerability (See Table 11).

4 Conclusion

This SLR on SQLIAs detection and prevention measures adopt guideline in [8] on conducting a systematic literature review on software, our study explores different studies from six different studies published the database, we carefully selected Eighty-two (82) studies out of initial 1261 based on inclusion and exclusion criteria defined in a study. Out of these eighty-two (82) studies, our study shows that seventy 70 or 85.4% are methods and tools proposed by different researchers to mitigate the problem of SQLIAs, while 7 or 8.5% are proposed experimental evaluation 5 or 6.1% are analytical analysis.

The evaluation result showed that a few of these proposed SQLIAs detection and prevention tools and methods are developed to address all types of SQLIAs while others focused on addressing a subset of particular SQLIAs type considered. Similarly, the result showed that a few of these tools can examine malicious SQL queries injected through cookies with no tool or method considered be able to detect or prevent attacks from server-side vulnerability.

In conclusion, one of the reasons why researchers have not been able to find the ultimate solution for the problem of SQLIAs is that each proposed methods and tools have a limitation on how it addresses a particular attack, starting from scope of the proposed method to its weakness in the development approach. For example, as can be seen in Fig. 16, almost each of the proposed SQLIAs prevention tools and methods reviewed in this study provides the guarantee of protection tautology, illegal/incorrect query, and union query SQLIA by 62.2%, alternate encoding attack by 42.2%, piggy-backend query attack by 22.2%, inference, and stored procedure attack by 11.1%. Likewise, SQLIAs detection tools and methods considered in this study provides the guarantee of protection tautology, illegal/incorrect query, and union query SQLIA by 20%, alternate encoding and piggy-backend query attack by 17.7%,

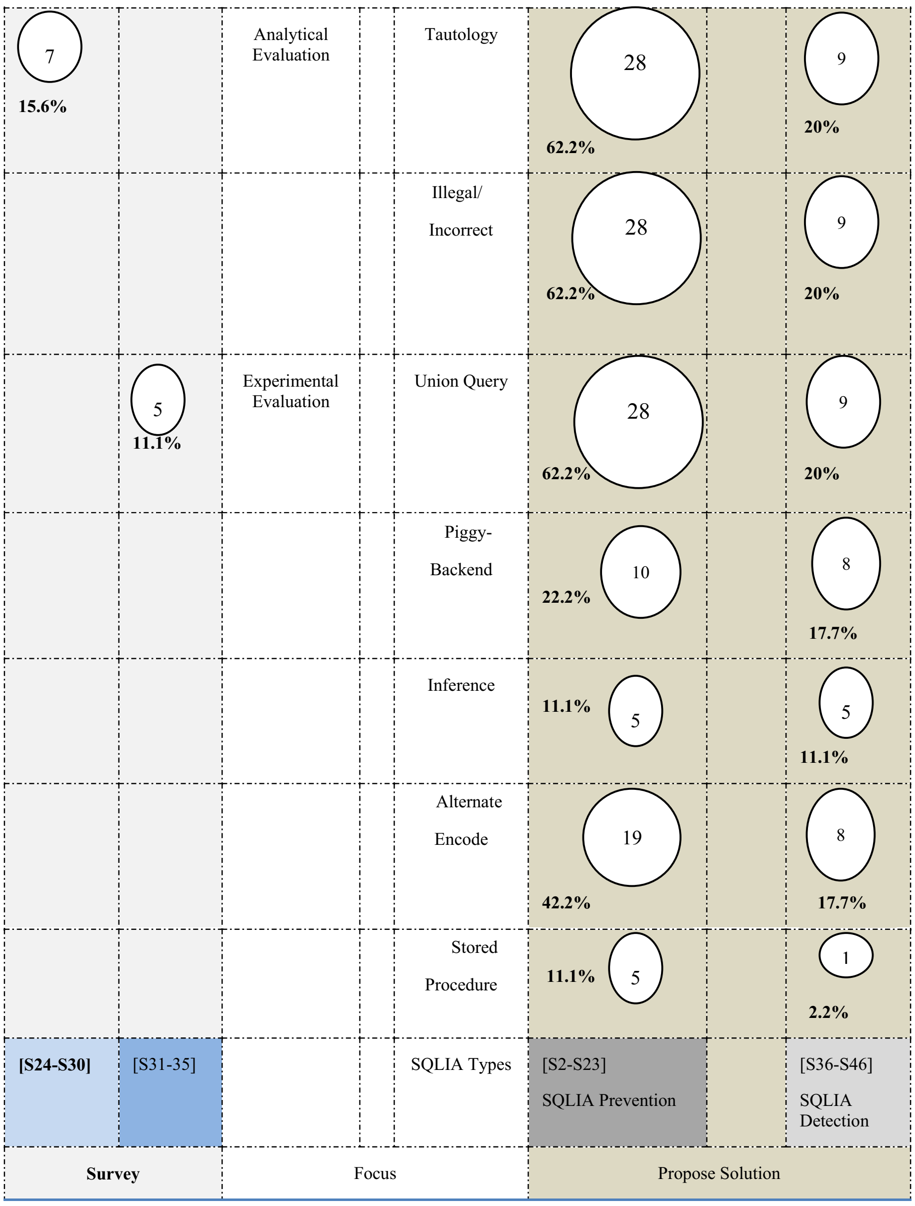
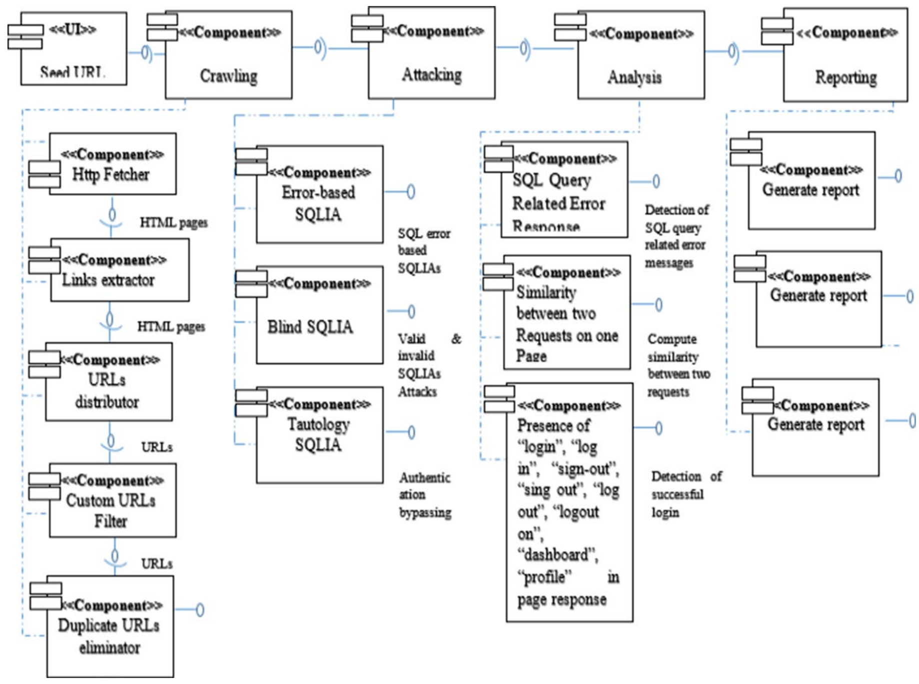


Fig. 16 Research contributions regarding SQLIAs

inference attack by 11.1% and stored procedure attack by 2.2%. On the other hand, the study shows that the number of studies that proposed an evaluation of existing SQLIAs detection and prevention tools and methods is quite low, which is around 26.7%, 15.6% for analytical evaluation, and 11.1% for experimental evaluation. Lastly, this study highlights the major challenges that required immediate response by developer and researchers in order to prevent the risk of being hacked through SQLIAs lack of capability to detect attacks that can exploit



<<UI>> Seed URL
<<Component>> Crawling
<<Component>> Attacking
<<Component>> Analysis
<<Component>> Reporting
<<Component>> Http Fetcher
HTML pages
<<Component>> Links extractor
HTML pages
<<Component>> URLs distributor
URLs
<<Component>> Custom URLs Filter
URLs
<<Component>> Duplicate URLs eliminator
<<Component>> Error-based SQLIA
SQL error based SQLIAs
<<Component>> Blind SQLIA
Valid & invalid SQLIAs Attacks
<<Component>> Tautology SQLIA
Authentication bypassing
<<Component>> SQL Query Related Error Response
Detection of SQL query related error messages
<<Component>> Similarity between two Requests on one Page
Compute similarity between two requests
<<Component>> Presence of "login", "log in", "sign-out", "sing out", "log out", "logout on", "dashboard", "profile" in page response
Detection of successful login
<<Component>> Generate report
<<Component>> Generate report
<<Component>> Generate report

Fig. 17 Components of the solutions proposed in studies S47

server-side SQLI vulnerability, poor prevention of inference and stored procedure attacks lack of ability to be deployed in various SQL injection parameter used in target applications.

5 Future Work

The study provides a comprehensive overview of SQL injection detection and defensive tools and method to combat unauthorized access and data modification on web-based database-driven applications. However, these tools and methods have weaknesses ranging from development practice to deployments capabilities. Our study reveals that none of the tools can fully detect or prevent all SQL injection attacks types. Tools [S4, S6, and S7] attempts to stops SQL injection attacks of all type, however, their accuracy highly dependent on the quality of training data set used and how good the model was trained, as poor training data set and model result in false-positive and negative. Therefore, these tools can only partially depend against a subset of SQL injection attacks considered as a result of common development errors and attackers are continually inventing ways of bypassing anomaly-based approach detection and prevention mechanism. Therefore, the effectiveness of these tools and methods is highly dependent on the quality of training data set used and how good the model was trained, as poor training data set and model result in false positive and negative.

Furthermore, the study recommends S47 for future improvements as a tool can be deployed in various injection parameters to detect SQL injection attack types except for stored procedure and time SQL injection attacks type. S47 is designed with the concept of components based software engineering practice as described in Fig. 17 below, which allows easier and efficient future improvement, maintenance and reuse much complexity as the system becomes complex. The proposed tool has four major components, namely: crawling, attacking analysis and reporting in addition to this, each component has sub-components indicating activities performed by the component. In the attack component, the tool claimed to detect SQL injection attack type considered except stored procedure SQL injection attack.

Finally, the study recommends designing of hybrid SQL injection attack tool that detects and block SQL injection attacks using the static and dynamic approach to have a more accurate result with high efficiency. In future, our focus on most recent studies of internet of vehicles, vehicular ad hoc networks and wireless sensor networks security analysis [86–88].

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical Approval This article does not contains any studies with human participants performed by any of the authors.

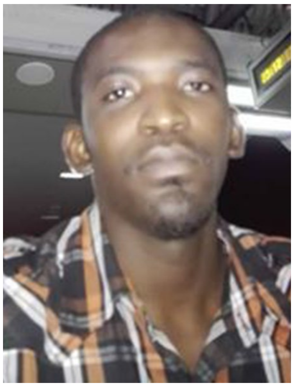
References

1. Qureshi, K. N., Bashir, F., & Abdullah, A. H. (2019). Distance and signal quality aware next hop selection routing protocol for vehicular ad hoc networks. Neural Computing and Applications, 1–14.
2. Anwar, M., et al. (2018). Securing data communication in wireless body area networks using digital signatures. *Technical Journal, 23*(02), 50–55.
3. Qureshi, K. N., & Abdullah, A. H. (2014). Adaptation of wireless sensor network in industries and their architecture, standards and applications. *World Applied Sciences Journal, 30*(10), 1218–1223.
4. Iqbal, S., et al. (2018). Critical link identification and prioritization using Bayesian theorem for dynamic channel assignment in wireless mesh networks. *Wireless Networks, 24*(7), 2685–2697.
5. Aliero, M. S., Ghani, I., Zainudden, S., Khan, M. M., & Bello, M. (2015). Review on SQL injection protection methods and tools. *Jurnal Teknologi, 77*(13), 49–66.
6. Aliero, M. S., et al. (2019). An algorithm for detecting SQL injection vulnerability using black-box testing. *Journal of Ambient Intelligence and Humanized Computing*, *11*, 1–18.
7. Thiyagarajan, A., et al. (2015). Methods for detection and prevention of SQL attacks in analysis of web field data. *International Journal of Computer Science and Mobile Computing*, *4*(4), 657–662.
8. Kitchenham, B., et al. (2009). Systematic literature reviews in software engineering—A systematic literature review. *Information and Software Technology, 51*(1), 7–15.
9. Halfond, W. G., & Orso, A. (2007). *Detection and prevention of sql injection attacks, in Malware Detection* (pp. 85–109). Berlin: Springer.
10. Sadeghian, A., Zamani, M., & Manaf, A. A. (2013). A taxonomy of SQL injection detection and prevention techniques. In *2013 international conference on informatics and creative multimedia* (pp. 53–56). IEEE.
11. Tiwari, Y., & Tiwari, M. (2015). A study of SQL of injections techniques and their prevention methods. *International Journal of Computer Applications*, *114*(17), 31–33.
12. Tajpour, A., Ibrahim, S., & Sharifi, M. (2012). Web application security by SQL injection detection-tools. *IJCSI International Journal of Computer Science, 9,* 2.
13. Kindy, D. A., & Pathan, A. S. K. (2011). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In *2011 IEEE 15th international symposium on consumer electronics (ISCE)* (pp. 468–471). IEEE.
14. Tajpour, A., & zade Shooshtari, M. J. (2010). Evaluation of SQL injection detection and prevention techniques. In *2010 2nd international conference on computational intelligence, communication systems and networks* (pp. 216-221). IEEE.

15. Doshi, J. C., Christian, M., & Trivedi, B. H. (2014). SQL FILTER–SQL Injection prevention and logging using dynamic network filter. In *International symposium on security in computing and communication* (pp. 400-406). Springer, Berlin, Heidelberg.
16. Medhane, M. (2013). R-WASP: Real time-web application SQL injection detector and preventer. *International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2*(5), 327–330.
17. Son, S., McKinley, K. S., & Shmatikov, V. (2013). Diglossia: Detecting code injection attacks with precision and efficiency. In *Proceedings of the 2013 ACM SIGSAC conference on computer & communications security*. ACM.
18. Shin, Y., Williams, L., & Xie, T. (2006). Sqlunitgen: SQL injection testing using static and dynamic analysis. In *17th IEEE proceedings of the international symposium on software reliability engineering (ISSRE)*.
19. Bandhakavi, S., et al. (2007). CANDID: Preventing SQL injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM conference on computer and communications security*. ACM.
20. Liu, A., et al. (2009). SQLProb: A proxy-based architecture towards preventing SQL injection attacks. In *Proceedings of the 2009 ACM symposium on applied computing*. ACM.
21. Cheon, E. H., Huang, Z., & Lee, Y. S. (2013). Preventing SQL injection attack based on machine learning. *International Journal of Advancements in Computing Technology, 5*(9), 967–974.
22. Joshi, A., & Geetha, V. (2014). SQL injection detection using machine learning. In *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)*. IEEE.
23. Shahriar, H., & Zulkernine, M. (2012). Information-theoretic detection of SQL injection attacks. In *2012 IEEE 14th international symposium on high-assurance systems engineering*. IEEE.
24. Gubbi, J., et al. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems, 29*(7), 1645–1660.
25. Johari, R., & Sharma, P. (2012). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. In *2012 international conference on communication systems and network technologies*. IEEE.
26. Mishra, N., & Gond, S. (2013). Defenses to protect against SQL injection attacks. *International Journal of Advanced Research in Computer and Communication Engineering*, *2*(10), 3829–3833.
27. Manoj, R. J., Chandrasekhar, A., & Praveena, M. A. (2014). An approach to detect and prevent tautology Type SQL injection in web service based on XSchema validation. *International Journal Of Engineering And Computer Science*, *10*, 2319–7242.
28. Lee, I., et al. (2012). A novel method for SQL injection attack detection based on removing SQL query attribute values. *Mathematical and Computer Modelling, 55*(1), 58–68.
29. Indrani, B., & Ramaraj, E. (2011). X-log authentication technique to prevent SQL injection attacks. *International Journal of Information Technology and Knowledge Management, 4*(1), 323–328.
30. Das, D., Sharma, U., & Bhattacharyya, D. (2010). An approach to detection of SQL injection attack based on dynamic query matching. *International Journal of Computer Applications, 1*(25), 28–34.
31. Prabakar, M. A., Karthikeyan, M., & Marimuthu, K. (2013). An efficient technique for preventing SQL injection attack using pattern matching algorithm. In *2013 IEEE international conference on emerging trends in computing, communication and nanotechnology (ICECCN)*. IEEE.
32. Narayanan, S. N., Pais, A. R., & Mohandas, R. (2011). Detection and prevention of sql injection attacks using semantic equivalence. In *International conference on information processing* (pp. 103–112). Springer, Berlin.
33. Kumar, K., Jena, D., & Kumar, R. (2013). A novel approach to detect SQL injection in web applications. *International Journal of Application or Innovation in Engineering & Management (IJAIEM), 2*(6), 37–48.
34. Zhang, X. H., & Wang, Z. J. (2010). A static analysis tool for detecting web application injection vulnerabilities for ASP program. In *2010 2nd international conference on e-business and information system security (EBISS)*.
35. Tongshu, L., Jing, Z., & Jianzheng, L. (2013), SQL injection prevention. Google Patents.
36. Randive, P. U., Khatke, M. B., & Reddi, M. B. (2014). An Approach for Prevention of SQL Injection Attacks on Database: A Review. *International Journal of Innovative Research in Advanced Engineering, 1*(3), 38–41.
37. Masri, W., & Sleiman, S. (2015). SQLPIL: SQL injection prevention by input labeling. *Security and Communication Networks, 8*(15), 2545–2560.
38. Antunes, N., & Vieira, M. (2009). Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities in web services. In *2009 15th IEEE pacific rim international symposium on dependable computing*. IEEE.

39. Antunes, N., & Vieira, M. (2011). Enhancing penetration testing with attack signatures and interface monitoring for the detection of injection vulnerabilities in web services. In *2011 IEEE international conference on services computing*. IEEE.
40. Antunes, N., & Vieira, M. (2015). Assessing and comparing vulnerability detection tools for web services: Benchmarking approach and examples. *IEEE Transactions on Services Computing, 8*(2), 269–283.
41. Khoury, N. et al. (2011). An analysis of black-box web application security scanners against stored SQL injection. In *2011 IEEE third international conference on privacy, security, risk and trust (PASSAT) and 2011 IEEE third international conference on social computing (SocialCom)*. IEEE.
42. Antunes, N., & Vieira, M. (2012). Evaluating and improving penetration testing in web services. In *2012 IEEE 23rd international symposium on software reliability engineering*. IEEE.
43. Djuric, Z. (2013). A black-box testing tool for detecting SQL injection vulnerabilities. In *2013 Second international conference on informatics & applications (ICIA)*. IEEE.
44. Liban, A., & Hilles, S. M. (2014). Enhancing MYSQL Injector vulnerability checker tool (MYSQL Injector) using inference binary search algorithm for blind timing-based attack. In *2014 IEEE 5th control and system graduate research colloquium*. IEEE.
45. Doupé, A.et al. (2012). Enemy of the state: A state-aware black-box web vulnerability scanner. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*.
46. Shakhatreh, A. Y. I. (2010). *SQL-injection vulnerability scanner using automatic creation of SQL-injection attacks (MySqlinjector)*, Universiti Utara Malaysia).
47. Ciampa, A., Visaggio, C. A., & Di Penta, M. (2010). A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. In *Proceedings of the 2010 ICSE workshop on software engineering for secure systems*. ACM.
48. Fu, X. et al. (2007). A static analysis framework for detecting SQL injection vulnerabilities. In *31st annual international computer software and applications conference (COMPSAC 2007)*. IEEE.
49. Cho, Y.-C., & Pan, J.-Y. (2015). Design and implementation of website information disclosure assessment system. *PLoS ONE, 10*(3), e0117180.
50. Falcove. (2007) Falcove web vulnerability scanner and penetration testing. http://www.ramsayfalcove.com/htdocs/Welcome.html. Accessed June 29, 2015
51. Singh, A. K., & Roy, S. (2012). A network based vulnerability scanner for detecting sqli attacks in web applications. In *2012 1st international conference on recent advances in information technology (RAIT)*. IEEE.
52. Aliero, M. S., & Ghani, I. (2015). A component based SQL injection vulnerability detection tool. In *2015 9th Malaysian software engineering conference (MySEC)*. IEEE.
53. Seyyar, M. B., Çatak, F. Ö., & Gül, E. (2018). Detection of attack-targeted scans from the Apache HTTP Server access logs. *Applied Computing and Informatics, 14*(1), 28–36.
54. Eassa, A. M., et al. (2019). NoSQL injection attack detection in web applications using RESTful service. *Programming and Computer Software, 44*(6), 435–444.
55. Taylor, C., & Sakharkar, S. (2019). DROP TABLE textbooks: An Argument for SQL injection coverage in database textbooks. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 191–197). ACM.
56. Basit, N., Hendawi, A., Chen, J., & Sun, A. (2019). A learning platform for SQL injection. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 184–190). ACM.
57. Batista, L., et al. (2018). Fuzzy neural networks to create an expert system for detecting attacks by SQL Injection. *The International Journal of Forensic Computer Science, 13*(1), 8–21.
58. Khanna, S., & Verma, A. K. (2018). Classification of SQL injection attacks using fuzzy tainting. In *Progress in intelligent computing techniques: Theory, practice, and applications* (pp. 463-469). Springer, Singapore.
59. Uwagbole, S. O., Buchanan, W. J., & Fan, L. (2016). Numerical encoding to Tame SQL injection attacks. In *NOMS 2016–2016 IEEE/IFIP network operations and management symposium* (pp. 1253–1256). IEEE.
60. Ross, K. et al. (2018). Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. In *Proceedings of the ACMSE 2018 conference* (pp. 1–8). ACM.
61. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP* (pp. 108–116).
62. Moh, M. et al. (2016). Detecting web attacks using multi-stage log analysis. In *2016 IEEE 6th international conference on advanced computing (IACC)* (pp. 733–738). IEEE.

63. Iqbal, S., et al. (2016). On cloud security attacks: A taxonomy and intrusion detection and prevention as a service. *Journal of Network and Computer Applications, 74,* 98–120.
64. Deepa, G., & Thilagam, P. S. (2016). Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology, 74,* 160–180.
65. Yadav, N., & Shekokar, N. (2018). Analysis on injection vulnerabilities of web application. In *Proceedings of international conference on wireless communication* (pp. 13–22). Springer, Singapore.
66. Buro, S., & Mastroeni, I. (2018). Abstract code injection. In *International conference on verification, model checking, and abstract interpretation* (pp. 116–137). Springer, Cham.
67. Deshpande, G., & Kulkarni, S. (2019). Modeling and mitigation of XPath injection attacks for web services using modular neural networks. In *Recent findings in intelligent computing techniques* (pp. 301–310). Springer, Singapore.
68. Schwichtenberg, H. (2018). Reading and modifying data with SQL, stored procedures, and table-valued functions. In *Modern Data Access with Entity Framework Core* (pp. 305–315). Apress, Berkeley, CA.
69. Heled, J., et al. (2018) Research on SQL injection detection technology based on SVM. In *MATEC web of conferences.*
70. Yan, R., et al. (2018). New deep learning method to detect code injection attacks on hybrid applications. *Journal of Systems and Software, 137,* 67–77.
71. Wang, X., & Zhao, Y. (2018). Order-revealing encryption: File-injection attack and forward security. In *European symposium on research in computer security*. Springer, Cham.
72. Thomé, J., et al. (2018). Security slicing for auditing common injection vulnerabilities. *Journal of Systems and Software, 137,* 766–783.
73. Stasinopoulos, A., Ntantogian, C., & Xenakis, C. (2019). Commix: Automating evaluation and exploitation of command injection vulnerabilities in Web applications. *International Journal of Information Security, 18*(1), 49–72.
74. Kaur, G., et al. (2018). Efficient yet robust elimination of XSS attack vectors from HTML5 web applications hosted on OSN-based cloud platforms. *Procedia Computer Science, 125,* 669–675.
75. Irmak, E., & Erkek, İ. (2018). An overview of cyber-attack vectors on SCADA systems. In *2018 6th international symposium on digital forensic and security (ISDFS)*. IEEE.
76. Barzegar, M., & Shajari, M. (2018). Attack scenario reconstruction using intrusion semantics. *Expert Systems with Applications, 108,* 119–133.
77. Babiker, M., Karaarslan, E., & Hoscan, Y. (2018). Web application attack detection and forensics: A survey. In *2018 6th international symposium on digital forensic and security (ISDFS)*. IEEE.
78. Nadeem, R. M., et al. (2017). Detection and prevention of SQL injection attack by dynamic analyzer and testing model. *International JournalOURNAL of Advanced Computer Science and Applications, 8*(8), 209–214.
79. Rahman, T. F. A., et al. (2017). SQL injection attack scanner using Boyer-Moore string matching algorithm. *JCP, 12*(2), 183–189.
80. Baror, S. O., & Venter, H. (2019). A Taxonomy for cybercrime attack in the public cloud. In *International conference on cyber warfare and security* (pp. 505). Academic Conferences International Limited.
81. Mukherjee, S. (2019). Popular SQL server database encryption choices. *arXiv preprint* arXiv:1901.03179.
82. Zheng, L. et al. (2019). Research and implementation of web application system vulnerability location technology. In *The international conference on cyber security intelligence and analytics*. Springer, Cham.
83. Awad, M., et al. (2019). Security vulnerabilities related to web-based data. *Telkomnika, 17*(2), 852–856.
84. Deshpande, D. S., Deshpande, S. P., & Thakare, V. M. (2019). Detection of online malicious behavior: An overview. In *Ambient communications and computer systems* (pp. 11-24). Springer, Singapore.
85. Kozik, R., Choras, M., & Keller, J. (2019). Balanced efficient lifelong learning (B-ELLA) for cyber attack detection. *Journal of Universal Computer Science, 25*(1), 2–15.
86. Qureshi, K. N., Bashir, F., & Abdullah, A. H. (2017). Provision of security in vehicular ad hoc networks through an intelligent secure routing scheme. In *2017 international conference on frontiers of information technology (FIT)*. IEEE.
87. Qureshi, K. N., Bashir, F., & Islam, N. U. (2019). Link aware high data transmission approach for internet of vehicles. In *2019 2nd international conference on computer applications & information security (ICCAIS)*. IEEE.
88. Qureshi, K. N., Abdullah, A. H., & Iqbal, S. (2016). Improving quality of service through road side backbone network in Vanet. *Jurnal Teknologi*, *78*(2), 7–14.



Muhammad Saidu Aliero He is Lecturer at Kebbi State University of Science And Technology Aliero and Data Entry Operator at Goggo Global Link Company Limited. Currently he is student in cyber security department, Monash University, Malaysia. Muhammad Saidu Aliero, B.Sc. Information Technology, Kebbi State University of Science and Technology Aliero 2012, Master of Information Security Universiti Tecknologi Malaysia 2016 and Ph.D. student Monash University Malaysia.

Dr. Kashif Naseer Qureshi is currently a Senior Assistant Professor with Bahria University, Islamabad. He received his doctorate from the University of Technology Malaysia (UTM). His research interest focuses on but are not limited to Ad hoc communication, Internet of Vehicles, Body area Networks, Cloud Computing and Internet of Things and provision of security in these domains. He is a Cisco and Microsoft Certified Network Professional. He has been a reviewer for various reputable academic journals. He is an active researcher with around 80 international publications in various renowned journals with the cumulative impact factor of 50. He has lead Ministry of Higher Education Malaysia Project by Ministry of Education Malaysia (MOE) and conducted in collaboration with Research Management Center (RMC) at Universiti Teknologi Malaysia (UTM) in Malaysia. Currently working with Ministry of Planning Commission and Higher Education Commission Project Cyber Reconnaissance and Combat (CRC) in Bahria University. This is a three years funded project on specialized field of Cyber Security and its practical applications, which are important components of Pakistan Vision 2025. Apart from his day job startup, he is also focusing that how to integrate the research with industry and to new start-ups, SME's and businesses.

Dr. Muhammad Fermi Pasha is graduated and earned his Ph.D. from Universiti Sains Malaysia in 2010. He then continued working as a research fellow at the same university for the next 5 years working on several multidisciplinary research projects with community engagement activities. Before that, while pursuing his Ph.D., he spent 4 years working as a Senior Software Engineer and Architect in the software industry. He is currently with Monash University, Malaysia, as a Lecturer attached to the School of Information Technology. Dr. Fermi is a passionate software developer and researcher. He has received various awards recognizing the software solutions and the research projects that he was involved with both as member as well as team lead. Presently, his research focuses on computational neuroimaging, intelligent network security traffic analysis, and healthcare and radiology IT with emphasis on big data.

Dr. Imran Ghani He is working as Senior Lecturer of Software Engineering at Monash University Malaysia. He was previously worked in Universiti Teknologi Malaysia (UTM), Studied Ph.D Business Information Technology at Kookmin University. He did his Ph.D. from Business Information Technology, Seoul, Korea.

Rufai Aliyu Yauri Associate Professor Rufai Aliyu Yauri, Ph.D. and M.Sc. Universiti Putra Malaysia. Deputy Dean Faculty of Engineering, Head of Department of ICT. Kebbi State University of Science and Technology.

Affiliations

Muhammad Saidu Aliero[1] · **Kashif Naseer Qureshi**[2] · **Muhammad Fermi Pasha**[1] · **Imran Ghani**[3] · **Rufai Aliyu Yauri**[4]

1 School of IT, Monash University, Malaysia, Subang Jaya, Malaysia

2 Department of Computer Science, Bahria University, Islamabad, Pakistan

3 Indiana University of Pennsylvani, Philadelphia, USA

4 Department of ICT, Kebbi State University of Science and Technology Aliero, Aliero, Nigeria