Indonesian Journal of Electrical Engineering and Computer Science
Vol. 21, No. 2, February 2021, pp. 1121~1131
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v21.i2.pp1121-1131

SQL injection attacks countermeasures assessments

Mamdouh Alenezi[1], Muhammad Nadeem[2], Raja Asif[3]
[1]College of Computer and Information Sciences, Prince Sultan University, Saudi Arabia
[2,3]Department of Computer Engineering, Faculty of ICT, BUITEMS, Quetta, Pakistan

Article Info

Article history:

Received Jun 26, 2020
Revised Aug 11, 2020
Accepted Aug 21, 2020

Keywords:

CVE
Database security
Injection attack
SQLIA
SQLIV

ABSTRACT

SQL injections attacks have been rated as the most dangerous vulnerability of web-based systems over more than a decade by OWASP top ten. Though different static, runtime and hybrid approaches have been proposed to counter SQL injection attacks, no single approach guarantees flawless prevention/ detection for these attacks. Hundreds of components of open source and commercial software products are reported to be vulnerable for SQL injection to CVE repository every year. In this mapping study, we identify different existing approaches in terms of the cost of computation and protection offered. We found that most of the existing techniques claim to offer protection based on the testing on a very small or limited scale. This study dissects each proposed approach and highlights their strengths and weaknesses and categorizes them based on the underlying technology used to detect or counter the injection attacks.

Corresponding Author:

Mamdouh Alenezi
Department of Computer Science
Prince Sultan University
P.O.Box No. 66833 Rafha Street, Riyadh 11586 Saudi Arabia
Email: malenezi@psu.edu.sa

1. INTRODUCTION

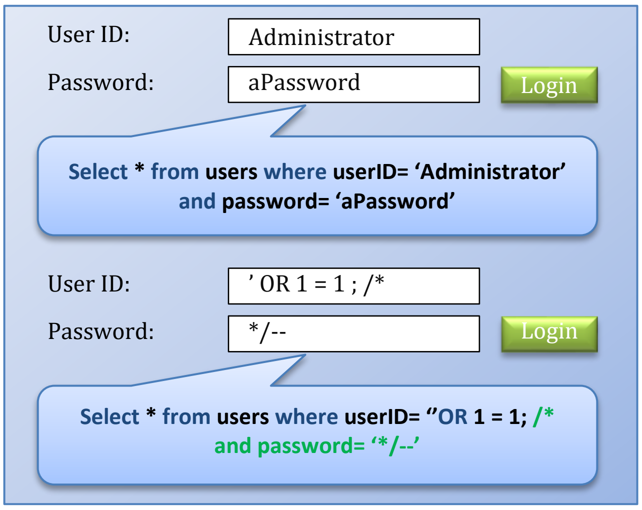
In recent years SQL injections have emerged as one of the most dangerous types of attacks to web-based systems and are ranked number one among the Open Web Application Security Project's (OWASP) top ten vulnerabilities [1]. SQL injection attacks (SQLIAs) are launched by entering malicious characters into the input fields of web applications resulting in a modified SQL query. There has not been a long history of SQL injection attacks. During the early days of the Internet, most content was static and not prone to injection attacks. The first large scale SQL injection attack was launched in Feb 2002 on Guess.com customers' database permitting the attacker to retrieve more than 200,000 customer names, credit card numbers, and expiration dates. More recently, the National Vulnerability Database (NVD) reports that 7 percent of the total reported incidents in 2011 were caused by SQL injections [2]. Moreover, it was discovered that an average of 71 SQL injection attacks is attempted every hour [3].

SQLIAs are used by attackers to gain access to confidential information in a database. SQLIAs can result in a financial loss as well as a damaged reputation for the impacted organization. There is no sound technique or combination of techniques that fully protects against all types of SQLIAs [4]. To help protect confidential data from being illegally accessed by attackers using SQLIA, the effectiveness of different techniques and combinations of techniques need to be evaluated to determine a robust set of combined techniques to lessen the impact SQLIAs have on an organization. Our strategy for determining which techniques and combinations of techniques are most effective consists of performing a systematic mapping study on the subject of SQLIA mitigation strategies. By analyzing the effectiveness of various techniques, we will be able to identify the best protection mechanisms that also consume the least human and computing resources. The rest of the paper is organized as follows: Section 2 contains background information on SQL injections; Section 3 contains our research methodology; Section 4 is our synthesis/ results; Section 5 is our conclusion and possible future work.

2. BACKGROUND

SQL, Structured Query Language, is used to retrieve and manipulate data in a database. SQL can be used to execute queries to retrieve, insert, delete, and edit data within a database. Most of today's applications require data storage and access to both local and remote databases. To launch an SQLIA, an attacker inserts malicious commands into an input string.

By changing the executed statement, a database can divulge data that deemed private by its creators. SQLIA can also be used to gain access to sites without prior knowledge of acceptable usernames or passwords. Attackers may also comment out certain parts of the query as shown in Figure 1.



```
Select * from users where userID= 'Administrator'
and password= 'aPassword'
```

```
Select * from users where userID= ''OR 1 = 1; /*
and password= '*/--'
```

Figure 1. Example of SQL injection attack [5]

SQLIAs are the most common website vulnerability. There have been many high-profile attacks performed through the years; one prominent SQLIA was performed by LulzSec on SonyPictures.com where the organization stole millions of digital coupons and passwords stored in plain text [6].

2.1. Types of target systems

We found existing techniques to counter SQLIAs that were empirically evaluated on systems of different types and sizes. We categorized these systems into three groups, "toy systems," "limited scale," and "large-scale" systems. These terms are defined as follows:

a) A "toy system" is a system created for the sole purpose of testing code as opposed to a system that was already performing a useful service. Toy systems are contrived and, therefore, may provide inaccurate data as they may not be a good representation of real conditions.

b) A "limited-scale" system for testing SQLIA prevention techniques is limited in either the features that can be tested on it, its reliability, or it may just not be maintained. These provide better results than toy systems but are still not realistic representatives of industrial systems.

c) A "large-scale" system is an established codebase that performs many useful functions and is regularly maintained. These are the types of systems the prevention techniques are designed to protect; therefore, the results obtained from testing them are the most accurate.

2.2. Is SQL injection an old day's problem?

The SQL injection is not an old day's problem, we found the problem reported in most recent literature as well. For instance, a study conducted on security vulnerabilities related to web-based data [7] in 2019 argues that mechanisms currently in place are insufficient.

Similarly, a study conducted in 2020 proposes improved ways of storing passwords and other critical information [8] as the breaches due to injections and other related attacks reveal the passwords and other critical information.

The attacks are not limited to traditional database systems rather smart grids and other cyber-physical systems have been the target of attackers in recent years. Another study conducted in 2020 highlights the essentials security features for advanced energy management systems [9]. Although hundreds of studies may be listed to highlight the importance of the improved security mechanism to counter vulnerabilities, it is not the focus of current research.

3. RESEARCH METHODOLOGY

Our research consisted of reading research papers on the current body of techniques and evaluating those techniques based on the following attributes:

a) Overhead to implement/ latency to use the technique
b) Coverage offered for different types of SQLIAs
c) Multi-language/ platform support offered
d) Detection or success rate
e) False-positive or true negative detection
f) Empirical evaluation of the technique

We chose these attributes based on existing research aimed at arguing the usefulness of their techniques [6-=10, 24=delete, 30=delete]. Because of the large number of sources for mitigation techniques (>50) identified in our literature search, it is not feasible to independently evaluate all the techniques as a part of this study. Therefore, we are using the evaluation results provided in the respective studies. We found that the selected techniques empirically evaluated systems of different types and sizes and used multiple testing strategies. Accordingly, we cannot develop one common scale to measure each technique’s characteristics.

3.1. Research questions

Our research focuses on the following questions:

a) Which techniques have relatively lower overhead than others?
b) Which techniques have high coverage of SQLIAs (i.e., various types of SQLIAs it finds)?
c) Which techniques have reliably a few false positives?

The motivation behind these research questions is to figure out those existing techniques which have lower overhead, higher coverage, and lower false positive detection rates, etc. The goal is to combine techniques to form hybrid approaches that have the highest detection rate and coverage; lowest false positive rate; and are the most efficient.

3.2. Data collection & metrics

The first step when performing a systematic literature review is to collect data from all relevant sources to analyze. The main objective is to find evidence addressing the research questions. The search starts by figuring out what keywords best describe the research questions. These keywords will then be searched in different research document databases. The databases used in this research are IEEE Xplore, ACM Digital Library, Google Scholar, Elsevier, and Scopus. Table 1 categorizes papers source type and publication database.

Table 1. Breakdown of papers by content type

Source Type	IEEE	ACM	Google Scholar	Elsevier	Scopus	Total
Conference	106	19	15	1	-	141
Journal	16	13	6	19	31	85
Symposium	21	15	4	-	1	41
Workshop	5	9	2	-	1	17
Others	-	-	9	-	1	10
Total	148	56	36	20	34	294

Keywords used to search relevant literature were “SQL injection”, “Injection attack”, “SQL injection attack”, and “SQLIA”. The total number of papers found by searching the databases was 326. After filtering duplicates, there were 294 papers to analyze. Several papers discussed multiple types of injections. Table 2 breaks down the types of injections that were discovered and provides a short description of each type.

Table 2. Breakdown of papers by injection type

Source Type	Total
SQL Injection – insert SQL characters/ keywords in unrestricted user input parameters	151
Code Injection – a small piece of code is inserted and executed via vulnerabilities such as buffer overflow	21
Data Injection – false data is injected in monitoring systems or sensor networks	18
Fault Injection – attempts to deviate a targeted device/system from its normal functionality	21
Profile Injection – insertion of biased profiles into rating database for altering the system's behavior	4
Script Injection – inject a malicious script to a website e.g., wiki, and execute it via the browser	2
Command Injection – issuing unauthorized commands to the underlying system (e.g., OS)	1
Packet Injection – false packets are injected especially in ad hoc networks to cause a denial of service	1
Other Related	75
Total	294

Papers were further filtered out and following exclusions were made (number of papers excluded in parentheses):

a) Papers not focused on SQL injections (142)
b) Papers not providing defense/ detection technique (14)
c) Papers not providing experimental validation (24)
d) Papers based on summaries of existing literature (25)
e) Exclusion made after reading a paper in detail (18)
f) Similar studies by the same authors in different journals or conferences (10)

The flowchart, given in Figure 2, helps to understand how every selected study was analyzed.

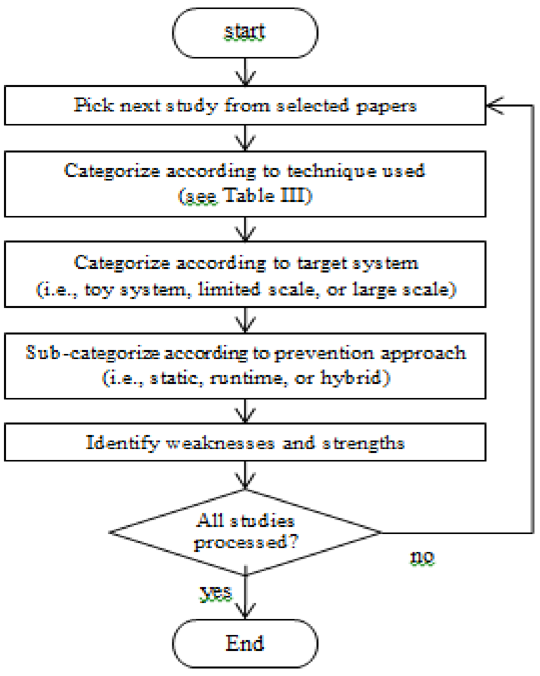

start
Pick next study from selected papers
Categorize according to technique used (see Table III)
Categorize according to target system (i.e., toy system, limited scale, or large scale)
Sub-categorize according to prevention approach (i.e., static, runtime, or hybrid)
Identify weaknesses and strengths
All studies processed?
no
yes
End

Figure 2. The process for analyzing selected studies

4. SYNTHESIS/ RESULTS

In this section, the findings of our survey are discussed, and the data is presented. Section 4.1 discusses the categorization of literature whereas section 4.2 discusses the weaknesses and strengths identified in the existing literature.

4.1. Categorization of selected literature

Clustering the papers based on the approach used to counter SQL injection attacks resulted in categories that are shown in Table 3. The table is divided into three columns. The first column provides the name of the technique, the second column describes the technique, the next column contains the citations, and the last column specifies the percentage of studies focused on the technique.

The findings from our literature survey have been summarized in Table 4. The "Name of Technique" header describes the name of the technique for named techniques or a short description of the technique for unnamed techniques. The "Type" header differentiates between static or runtime techniques; the static technique is one applied before the system is put into execution mode e.g., static analysis of code, analysis of recorded network traffic or mining logs, etc., whereas the runtime techniques are applied while the system is in execution mode, e.g., comparing each query at runtime with a set of predetermined legit queries and block all unauthorized ones.

The next two columns give short descriptions of the strengths and weaknesses of the technique. The techniques are evaluated in terms of their success rate, false-positive/ true negative detections, code coverage, platform and language support, and overheads/latencies, etc. Lastly, the "Testing applied" header describes the size of the system with which the given approach was tested, see section 2A.

It can be observed from Table 3 that code transformation techniques need more focus. Code transformation is a complicated task of replacing the vulnerable code with a secure code with the exactly same functionality. However, once it is done properly it can make a system capable of self-healing.

Table 3. Categorizing the existing literature

Name of the technique	Description	Reference to the studies	Overall contribution
Code transformation	These static approaches scan the source code and hunt for SQL code vulnerable to SQL injection attacks and then replace the code with secure code	[4, 10-14]	11.1% of total studies focused on code transformation techniques
Keyword randomization	These runtime approaches replace SQL keywords with secret random words, so that attacker's input containing SQL keywords may be detected, the secret words are then replaced by actual SQL keywords before query execution	[15-17]	5.6% of total studies focused on keyword randomization techniques
Query comparison	These approaches determine the SQL statements at different stages, e.g., by removing user attributes in the query and comparing it with predetermined one, or by determining the change in the intent of the query, or sanitizing the queries or inputs, etc.	[18-40]	42.6% of total studies focused on query/ information comparison techniques
Testing/ attacking system using automated tools	These approaches use automated tools or test case generators to exploit SQL injection vulnerability in the target system. Reports generated by these tools help developers to fix these vulnerabilities.	[41-54],	25.9% of total studies focused on automated tools
Trusted tests	These approaches have a preliminary stage that is dedicated to learning the profile of valid input. After the preliminary stage, the profile is used during runtime to determine the validity of the input.	[55-62]	14.8% of total studies focused on trusted tests

4.2. Identification of weaknesses and strengths

The selected studies were then sorted based on the nature of the target systems on which they were tested, which has been illustrated in Figure 3. Sorting the studies based on the nature of the target system is important because testing a technique on a tiny system (we call it a toy system) does not guarantee rigorous testing for a technique; hence, achieving a specific level of coverage on a toy system does not guarantee that the technique will have same coverage when used on a large scale open-source system.

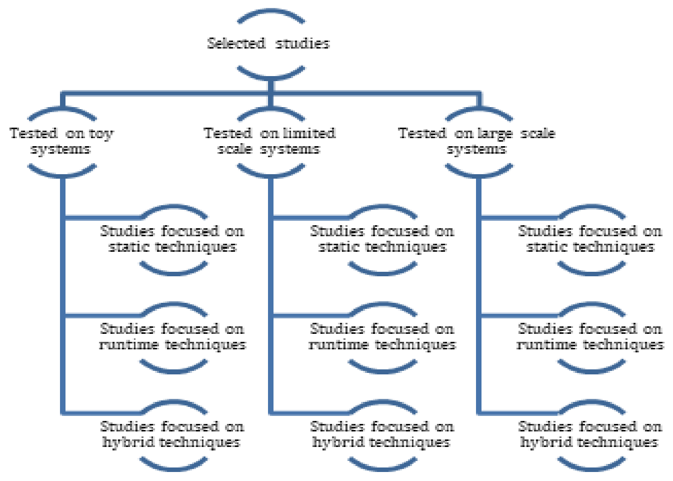

Selected studies
Tested on toy systems
Tested on limited scale systems
Tested on large scale systems
Studies focused on static techniques
Studies focused on runtime techniques
Studies focused on hybrid techniques
Studies focused on static techniques
Studies focused on runtime techniques
Studies focused on hybrid techniques
Studies focused on static techniques
Studies focused on runtime techniques
Studies focused on hybrid techniques

Figure 3. Hierarchy of literature organization

The studies are then sub-categorized according to the technique which can be static (i.e., design time), runtime, or hybrid. This leaves us with nine (09) categories in total, the result of each category has been synthesized into tables i.e., from Table 4 to Table 12. The notable studies which use static analysis techniques on toy systems are summarized in Table 4.

Important studies that apply runtime analysis techniques on toy systems are summarized in Table 5. The prominent studies which use hybrid analysis techniques on toy systems are summarized in Table 6. The notable studies which use static analysis techniques on limited-scale systems are summarized in Table 7. Important studies that apply runtime analysis techniques on limited-scale systems are summarized in Table 8. The prominent studies which use hybrid analysis techniques on limited-scale systems are summarized in Table 9. The notable studies which use static analysis techniques on large-scale systems are summarized in Table 10. Important studies that apply runtime analysis techniques on large-scale systems are summarized in Table 11. The prominent studies which use hybrid analysis techniques on large-scale systems are summarized in Table 12.

Table 4. Static analysis techniques on toy systems

Name of Technique	Strengths	Weaknesses
SAFELI: Analyze Microsoft Symbolic Intermediate Lang. (MSIL) to detect SQLIAs [11]	- Can identify vulnerabilities missed by black-box testing	- Limited to Microsoft platform - Needs code transformation in MSIL
An algorithmic approach for replacing insecure SQL statements in the code with secure ones [13]	- 94% reported accuracy	- Limited to Java -Code transformation - Coverage issues: e.g., Batch queries
Mining input sanitization patterns for predicting SQLIVs [47]	- 85% reported detection rate	- High false positives

Table 5. Runtime analysis techniques on toy systems

Name of Technique	Strengths	Weaknesses
SQLIMW [19], a middleware specifically designed for detection SQL injection attacks	- Efficient - Transparent to programmer	- Limited coverage, works best for single sign-on systems
SVM for prediction of SQLIA [34]	- Very low overhead - High detection rate (96%)	- May produce false positives
TransSQL [40]	- Multiplatform - No code changes - Easy to deploy - Automated	- High latency: every query is executed twice
Security testing scheme based on automatic test case generation and simulated tests [43]	- Effective	- Coverage problems (attack rule library needs improvement)
SQL-IDS (SQL injection detection system) [58]	- No code changes - Low overhead - High coverage - No false positives	- Limited to Java platform
Artificial Neural Network based web application firewall for SQL injection [62]	- Platform independent - Effective, the high detection rate	- May produce erroneous results

Table 6. Hybrid analysis techniques on toy systems

Name of Technique	Strengths	Weaknesses
A technique for defending against SQLIAs targeting stored procedures [10]	- Automated, used only as needed - Very low overhead - Effective	- Limited to stored procedures
Obfuscation-based Analysis of SQLIAs [17]	- No code transformation - Effective	- Obfuscation/de-obfuscation overhead - Coverage issues: dynamic queries
Build a model based on valid queries; at runtime check the compliance of queries to the model [23]	- May not predict every possible query at the first stage	- May generate false negatives - Latency: Check query using NDFA
Blocking of SQLIAs by comparing static and dynamic queries [24]	- Complexity: O(n) - High coverage - No false positives	- Overhead of 78ms per query
SQL injection elimination based on Regular Expression matching [30]	- No code transformation - Effective	- Coverage problem: Doesn’t detect dynamically generated queries
Transparent defense mechanism for eliminating SQLIA [34]	- No code transformation - Seamless integration - High coverage	- May generate false positives

Table 7. Static analysis techniques on limited scale systems

Name of Technique	Strengths	Weaknesses
SQL DOM [12]	- High coverage (except Stored Proc.)	- High overhead - Coverage problems - Requires new programming models
Automated fix generator for SQLIAs [14]	- Automated - Effective	- Needs code transformation - Limited to PHP - Overhead
Signature and auditing method to prevent SQLIAs using [18]	- Low execution overhead - No code transformation - Automated	- Restricted to web apps
An anomaly-based system that uses different detection models to detect unknown attacks [21]	- Low overhead	- Limited to PHP - Coverage problems - Generates false-negative results
Automatic creation of SQL injection attacks for uncovering SQL injection vulnerabilities [46]	- Automated - No runtime overhead - No modification in the target system	- May generate false positives - Tool limited to PHP/ MySQL
A method for hunting SQL injection vulnerabilities [53]	- Effective, the high detection rate - Very low overhead	- Complex involves different stages

Table 8. Runtime analysis techniques on limited scale systems

Name of Technique	Strengths	Weaknesses
Generation of SQL-injection free secure algorithm to detect and prevent SQLIA [4]	- Low overhead	- Low coverage (piggybacked queries & stored procedures) - Limited to Java
SQL injection detection by K-Centers [20]	- Effective, high detection rate	- May produce false negative
The proposed method learns valid & invalid inputs. These are taken into account for future inputs [22]	- Effective	- The learning period is needed, and new exploits could be found.
Hot query bank approach: records most occurring queries and skips their analysis in future [29]	- Efficient - 45% improvement in performance	- Tested in a simulated environment - overhead to integrate with detectors
SQLStor, SQL injection detection in stored procedures [33]	- Effective, high detection rate	- Limited to Java platform
NVS, Network Vulnerability Scanner [44], to detect SQL injection attacks	- Efficient - No false positives	- Complex to setup
An alternate way to parse SQL statements [46]	- Implementation minimizes the effort required by the programmer	- Currently limited to Java
IDEA, a testing approach for information leakages through error messages [49]	- High coverage - Effective	- Involves overhead
A learning-based approach to secure web services from SQL injection attacks [55]	- Efficient - Works transparently - Low implementation overhead	- Produces false positives - The success rate varies (Lowest 68%)
Event-based alert correlation system to detect SQL injections attacks [57]	- High detection rate	- May generate false positives - Tested with simulated attacks
SQLProb [59]	- No code transformation - Very low overhead - Effective	- Currently limited to Java/ MySQL
SDriver [60], SQL query signatures are stored, at runtime, signatures are used to judge valid queries	- Effective	- If the application is altered the driver will need to go back through learning
idMAS-SQL [61]	- Automated - Effective - Can also detect other vulnerabilities	- Complex setup - May become resource hungry

Table 9. Hybrid analysis techniques on limited scale systems

Name of Technique	Strengths	Weaknesses
A technique for detection and prevention of SQLIA using ASCII based string matching [25]	- High success rate	- Limited to .NET - Prototype implementation
IPAAS (Input Parameter Analysis System) [36]	- Very low overhead	- Low prevention rate (83%) - Limited to PHP - Produces false positives
Information-Theoretic detection of SQLIAs [37]	- Very effective - High success rate - No false positives	- Overhead of comparison entropies - Coverage issues - Limited to PHP
Positive Tainting and Syntax-Aware evaluation to counter SQLIAs [38]	- High coverage - Effective - No false positives	- Limited to Java - Overhead could be high
Removing attribute values to detect SQLIAs [39]	- Effective - No false positives - High coverage - Automated	- Limited to web applications - Involved overhead
Automated protection of PHP applications against SQL injection attacks [45]	- Automated - Effective	- Limited to PHP - Requires code transformation
SQLUnitGen, based on static analysis, runtime detection and automated testing [50]	- Automated - No false positives	- Produces false negatives - Tested on a limited scale
AMNESIA: static analysis and runtime monitoring technique [51]	- Low overhead - Fully Automated - Low (or No) false positives	- Success dependent on the accuracy of query models

Table 10. Static analysis techniques on large-scale systems

Name of Technique	Strengths	Weaknesses
ASSIST (Automatic and Static SQL Injection Sanitization Tool) [28]	- Effective - Low overhead (2%)	- Currently limited to Java - Requires code transformation
Hidden web crawling for SQL injection detection [52]	- Effective, high detection rate	- May produce false positives and true negatives

Table 11. Runtime analysis techniques on large-scale systems

Name of Technique	Strengths	Weaknesses
SQLRand [15]	- 6.5 ms latency per query on average - Platform independent	- Secrecy of keywords is a must - Needs proxy for de-randomization - Coverage issues
Random4, randomized encryption of input parameters [16]	- Good performance - Automated	- Overhead to randomize and de-randomize input parameters
SMART [32]	- Effective - Easy to deploy - No code transformation	- High overhead - Limited to Java - May produce false positives
Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection [41]	- Automated - High coverage	- May generates false positives - Currently supports Java platform
Object-oriented approach to SQL injection prevention [42]	- Efficient - Minimal deployment time	- May produce false positives
V1p3R ("viper") [48]	- Good performance - Automated - Multi platform support	- Complex to setup
A vulnerability scanner for web services [54]	- Effective	- May produce false positives - Has coverage issues

Table 12. Hybrid analysis techniques on large-scale systems

Name of Technique	Strengths	Weaknesses
Method for SQL injection attack detection based on removing SQL query attribute values [26]	- High success rate - High coverage - Efficient O (1) - Platform independent	- May need changes in source code
CANDID: transforms insecure Java byte code to secure code and applies a runtime checker [27]	- High success rate - No false positives - Automatic	- Currently limited to Java - Needs code transformation
Securing web applications with static and dynamic information flow tracking [31]	- Effective, - Fully automated - High detection rate (up to 99%)	- Complex to setup

Table 4 until 12 are the major contribution of our research; it helps one to develop a broad understanding of existing techniques to counter SQLIAs in terms of their strengths and flaws.

5. CONCLUSION AND FUTURE WORK

The paper presented a thorough survey of current approaches to counter SQL injection attacks. This research confirms that a single technique to fully counter the SQL injection attacks does not exist; therefore, further research is needed on combining different static and runtime approaches to get maximum possible protection with affordable computing power. In addition, our study revealed that current approaches have certain constraints; lack of generalization where most of the current approaches to counter the SQL injection is platform-dependent, lack of rigorous testing, and lack of empirical validation.

As another contribution of our study, we list the strengths and weaknesses of various approaches for the detection and prevention of SQL injection attacks. This information may help one find the right technique. For instance, very few techniques offer protection to a SQL injection attack on stored procedures, the reason being the dynamic nature of SQL queries generated by stored procedures during execution. Similarly, if a web application has been developed using multiple languages, all SQL injection prevention techniques might not work for such applications because techniques are often platform dependent.

Future work will include a new approach to counter the SQL injection attacks by combining the tailored versions of existing static and runtime approaches. In addition, further study of the existing approaches is needed that examines different aspects of the techniques (e.g., detection rate, overhead, and false-positive rate, etc.). Efforts must also be placed on developing general criteria for the assessment of these approaches.

REFERENCES

[1] *OWASP top ten security vulnerabilities*, Apr. 2013. [Online]. Available: http://www.owasp.org/ index.php/Top_10.
[2] *National Vulnerability Database*, 2013. [Online]. Available: http://nvd.nist.gov.
[3] *SQL injections*, Apr. 2013. [Online]. Available: http://en.wikipedia.org/wiki/SQL_injection.
[4] K. Natarajan and S. Subramani, "Generation of Sql-injection Free Secure Algorithm to Detect and Prevent SQL-Injection Attacks," in *Procedia Technology*, vol. 4, pp. 790–796, 2012.
[5] R. Shan, "LulzSec Hacker Arrested, Group Leaks Sony Database," The Epoch Times, http://www.theepochtimes.com/n2/technology/lulzsec-member-arrested-group-leaks-sony-database-57296.html, accessed 9, 2013.
[6] S. Tamada, "Hack your Own Web Project? SQL Injection," 9 Lessons, http://www.9lessons.info/2008/12/sql-injection.html, accessed 2013.
[7] M. Awad, M. Ali, M. Takruri, and S. Ismail, "Security vulnerabilities related to web-based data," in *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 2, pp. 852-856, 2019.
[8] J. Polpong and P. Wuttidittachotti, "Authentication and password storing improvement using SXR algorithm with a hash function," in *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 6, pp. 6582-6591, 2020.
[9] R. Muzzammel, R. Arshad, S. Mehmood, and D. Khan, "Advanced energy management system with the incorporation of novel security features," in *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 4, pp. 3978-3987, 2020.
[10] K. Wei, M. Muthuprasanna, and S. Kothari, "Preventing SQL Injection Attacks in Stored Procedures," *Proc. of the IEEE Software Engineering Conference*, Australian, p. 8, 2006.
[11] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, "A Static Analysis Framework For Detecting SQL Injection Vulnerabilities," *Proc. of the 31st International Conf. on Computer Software and Applications, COMPSAC'07*, 2007, pp. 87-96.
[12] R. A. McClure and I. H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," *Proc. of the 27th International Conference on Software Engineering, ICSE'05*, pp. 88-96, 2005,.
[13] S. Thomas, L. Williams, and T. Xie, "On Automated Prepared Statement Generation to Remove SQL Injection Vulnerabilities," in *Information and Software Technology*, vol. 51, no. 3, pp. 589-598, 2009.
[14] F. Dysart and M. Sherriff, "Automated Fix Generator for SQL Injection Attacks," *Proc. of the 19th International Symposium on Software Reliability Engineering*, 2008, pp. 311-312.
[15] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks," in *Applied Cryptography and Network Security*, pp. 292-302, 2004.
[16] S. Avireddyet *et al*., "Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection," *Proc. of the 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1327-1333, 2012,.
[17] R. Halder and A. Cortesi, "Obfuscation-based analysis of SQL injection attacks," *Proc. of the IEEE Symposium on Computers and Communications (ISCC'10)*, pp. 931-938, 2010.
[18] R. Ezumalai and G. Aghila, "Combinatorial Approach for Preventing SQL Injection Attacks," *Proc. of the International Advance Computing Conference (IACC '09), IEEE Computer Society*, pp. 1212-1217, 2009.
[19] G. Jiao, C. M. Xu, and J. Maohua, "SQLIMW: A New Mechanism against SQL-injection," *Proc. of the International Conference on Computer Science & Service System (CSSS 2012)*, pp. 1178-1180, 2012.
[20] X. Wu and P. Chan, "SQL injection attacks detection in adversarial environments by K-centers," *Proc. of the 2012 International Conference on Machine Learning and Cybernetics*, pp. 406-410, 2012.
[21] F. Valeur, D. Mutz, and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," *Proc. of the Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA)*, Vienna, Austria, 2005.
[22] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," *Proc. of the 5th ACM international workshop on Software engineering and middleware*, Lisbon, Portugal, pp. 1-8, 2005.
[23] W. G. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL injection attacks," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1-7, 2005.
[24] J. Minhas and R. Kumar, "Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries," in *International Journal of Computer Network and Information Security*, 2013.
[25] I. Balasundaram and E. Ramaraj, "An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching," in *Procedia Engineering*, vol. 30, pp. 183-190, 2012.
[26] I. Lee, S. Jeong, S. Yeo, and J. Moon, "A Novel Method for SQL Injection Attack Detection Based on Removing SQL Query Attribute Values," in *Mathematical and Computer Modeling*, vol. 55, no. 1, pp. 58-68, 2012.
[27] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan., "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," in *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, p. 14, 2010.
[28] R. Mui and P. Frankl, "Preventing SQL Injection through Automatic Query Sanitization with ASSIST," in *TAV-WEB (EPTCS)*, vol. 35, pp. 27-38, 2010.
[29] Y. C. Chung *et al*., "A Hot Query Bank Approach to Improve Detection Performance against SQL Injection Attacks," in *Computers and Security*, vol. 31, no. 2, pp. 233-248, 2012.
[30] M. Wan and K. Liu, "An Improved Eliminating SQL Injection Attacks Based Regular Expressions Matching," *Proc. of the International Conference on Control Engineering and Communication Technology (ICCECT'12)*, 2012.

[31] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, "Securing Web Applications with Static and Dynamic Information Flow Tracking," *Proc. of the PEPM'08*, San Francisco, California, Jan. 2008, pp. 3-12.
[32] H. Wu and G. Miao, "Test SQL Injection Vulnerabilities in Web Applications Based on Structure Matching," *Proc. of the IEEE International Conference on Computer Science and Network Technology*, Dec. 2011, pp. 935-938.
[33] M. T. S. Mamadhan and V. Paul, "SQLStor: Blockage of Stored Procedure SQL Injection Attack Using Dynamic Query Structure Validation," *Proc. of the 12th IEEE International Conference on Intelligent Systems Design and Applications*, 2012, pp. 241-245.
[34] R. Romil and S. Shrivastav, "SQL injection attack Detection using SVM," *International Journal of Computer Applications*, pp. 1-4, Mar. 2012.
[35] M. Muthuprasanna, K. Wei, and S. Kothari, "Eliminating SQL Injection Attacks - A Transparent Defense Mechanism," *Proc. of the Eighth IEEE International Symposium on Web Site Evolution (WSE'06)*, 2006.
[36] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "Preventing Input Validation Vulnerabilities inWeb Applications through Automated Type Analysis," *Proc. of the IEEE 36th International Conference on Computer Software and Applications*, pp. 233-243, 2012.
[37] H. Shahriar and M. Zulkernine, "Information-Theoretic Detection of SQL Injection Attacks," *Proc. of the 14th IEEE International Symposium on High-Assurance Systems Engineering*, pp. 40-47, 2012.
[38] W. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL injection attacks," *Proc. of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 175-185, 2006.
[39] J. Kim, "Injection Attack Detection using the Removal of SQL Query Attribute Values," *Proc. of the 2011 International Conference on Information Science and Applications (ICISA)*, pp. 1-7, 2011.
[40] K. Zhang, C. Lin, S. Chen, Y. Hwang, H. Huang, and F. Hsu, "TransSQL: A Translation and Validation-based Solution for SQL-Injection Attacks," *Proc. of the First International Conference on Robot, Vision and Signal Processing*, pp. 248-251, 2011.
[41] Y. Kosuga, K. Kernel, M. Hanaoka, M. Hishiyama, and Yu Takahama, "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection," *Proc. of the 23rd annual conference on Computer Security Applications, ACSAC'07*, pp. 107-117, 2007.
[42] D. R. Giri, S. P. Kumar, L. Prasannakumar, and R. N. V. V. Murthy, 'Object Oriented Approach to SQL Injection Preventer," *Proc. of the Third International Conference on Computing Communication & Networking Technologies (ICCCNT 12)*, pp. 1-7, 2012.
[43] Y. Haixia and N. Zhihong, "A database security testing scheme of web application," *Proc. of the 4th International Conference on Computer Science & Education, ICCSE '09*, pp. 953-955, 2009.
[44] A. K. Singh and S. Ro, "A Network Based Vulnerability Scanner for Detecting SQLI Attacks in Web Applications," *Proc. of the 1st International Conference on Recent Advances in Information Technology (RAIT 2012)*, 2012.
[45] E. Merlo, D. Letarte, and G. Antoniol, "Automated Protection of PHP Applications against SQL-injection Attacks," *Proc. of the 11th European Conference on Software Maintenance and Reengineering, CSMR '07*, pp. 191-202, Mar. 2007.
[46] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL Injection and cross-site scripting attacks," *Proc. of the IEEE 31st International Conference on Software Engineering, ICSE 2009*, pp. 199-209, 2009.
[47] L. K. Shar and H. B. K. Tan, "Mining Input Sanitization Patterns for Predicting SQL Injection and Cross Site Scripting Vulnerabilities," *Proc. of the Internation conference on Software Engineering, ICSE 2012*, Zurich, Switzerland, pp. 1293-1296, 2012.
[48] A. Ciampa, C. A. Visaggio, and M. D. Penta, "A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications," *Proc. of the SESS'10*, Cape Town, South Africa, pp. 43-49, 2010.
[49] B. Smith, L. Williams, and A.Austin, "Idea: Using System Level Testing for Revealing SQL Injection-related Error Message Information Leaks," in *Engineering Secure Software and Systems*, pp. 192-200, 2010.
[50] Y. Shin, L. Williams, and T. Xie, "SQLUnitGen: Sql Injection Testing using Static and Dynamic Analysis," *Proc. of the 17th IEEE International Conference on Software Reliability Engineering (ISSRE'06)*, 2006.
[51] W. G. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," *Proc. of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, 2005.
[52] X. Wang, L. Wang, G. Wei, D. Zhang, and Y. Yang, "Hidden web crawling for SQL injection detection," *Proc. of the 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT'10)*, pp. 14-18, 2010.
[53] W. Tian, J. Xu, K. Lian, Y. Zhang, J. Yang, "Research on mock attack testing for SQL injection vulnerability in multi-defense level web applications," *Proc. of the 2nd International Conference on Information Science and Engineering (ICISE'10)*, pp. 1-5, 2010.
[54] N. Antunes and M. Vieira, "Detecting SQL Injection Vulnerabilities in Web Services," *Proc. of the Fourth Latin-American Symposium on Dependable Computing*, pp. 17-24, 2009.
[55] N. Laranjeiro, M. Vieira, and H. Madeira, "A Learning-Based Approach to Secure Web Services from SQL/XPath Injection Attacks," *Proc. of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 10)*, pp. 191-198, 2010.
[56] N. Laranjeiro, M. Vieira, and H. Madeira., "Protecting Database Centric Web Services against SQL/XPath Injection Attacks," *Database and Expert Systems Applications*, 2009.

[57] F. Alserhani, M. Akhlaq, I. U. Awan, and A. J. Cullen, "Event-Based Alert Correlation System to Detect SQLI Activities," *Proc. of the IEEE International Conference on Advanced Information Networking and Applications (AINA 2011)*, pp. 175-182, 2011.
[58] K. Kemalis and T. Tzouramanis., "SQL-IDS: A Specification-based approach for SQL-injection Detection," *Proc. of the 2008 ACM symposium on Applied Computing*, 2008.
[59] A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou, "SQLProb: A Proxy-based Architecture towards Preventing SQL Injection Attacks," *Proc. of the 24th Annual ACM Symposium on Applied Computing*, Honolulu, Hawaii, pp. 2054-2061, 2009.
[60] M. Dimitris and D. Spinellis, "SDriver: Location-specific signatures prevent SQL injection attacks," in *Computers & Security*, vol. 28, pp. 121-129, 2009.
[61] P. Cristian, J. De Paz, Á. Herrero, E. Corchado, J. Bajo, and J. Corchado, "idMAS-SQL: Intrusion Detection Based on MAS to Detect and Block SQL injection through data mining," *Information Sciences*, vol. 231, pp. 15-31, 2011.
[62] M. Asaad, "Artificial Neural Network based Web Application Firewall for SQL Injection," *World Academy of Science, Engineering & Technology*, vol. 64, pp. 12-21, 2010.