A SURVEY ON SQL INJECTION: VULNERABILITIES, ATTACKS, AND PREVENTION TECHNIQUES

Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan
Department of Computer Science, International Islamic University Malaysia, Malaysia
diallo14@gmail.com and sakib@iium.edu.my

ABSTRACT

In this paper, we present a detailed review on various types of SQL injection attacks, vulnerabilities, and prevention techniques. Alongside presenting our findings from the survey, we also note down future expectations and possible development of countermeasures against SQL injection attacks.

1 INTRODUCTION

SQL Injection is a type of injection or attack in a Web application, in which the attacker provides Structured Query Language (SQL) code to a user input box of a Web form to gain unauthorized and unlimited access. The attacker's input is transmitted into an SQL query in such a way that it will form an SQL code [10], [9]. It is categorized as one of the top-10 2010 Web application vulnerabilities experienced by Web applications according to OWASP (Open Web Application Security Project) [1].

2 SQL INJECTION BACKGROUND

2.1 Why SQL Injection is a Threat?

Injecting a Web application is the synonym of having access to the data stored in the database. The data sometimes could be confidential and of high value like the financial secret of a bank or list of financial transactions or secret information of some kind of information system, etc. An unauthorized access to this data by a crafted user can threat their confidentiality, integrity, and authority. As a result, the system could bear heavy loss in giving proper services to its users or it may face complete destruction. Sometimes such type of collapse of a system can threaten the existence of a company or a bank or an industry. If it happens against the information system of a hospital, the private information of the patients may be leaked out which could threaten their reputation or may be a case of defamation. Attackers may even use such type of attack to get confidential information that is related to the national security of a country. Hence, SQL Injection could be very dangerous in many cases depending on the platform where the attack is launched and it gets success in injecting rogue users to the target system.

2.2 Types of Vulnerabilities in Web Programming

In this section, we present the most common security vulnerabilities found in Web programming languages [12] (see Table I). In fact, it is through these vulnerabilities that SQL Injection attacks are launched.

TABLE I. TYPES OF VULNERABILITIES AT A GLANCE

Vulnerability Types	Description
Type I	Lack of clear distinction between data types accepted as input in the programming language used for the Web application development.
Type II	Delay of operation analysis till the runtime phase where the current variables are considered rather than the source code expressions.
Type III	Weak concern of type specification in the design: a number can be used as a string or vice-versa.
Type IV	The validation of the user input is not well defined or sanitized. Inputs are not checked correctly.

TABLE II. TYPES OF SQLIAS AT A GLANCE

Types of Attack	Working Method
Tautologies	SQL injection queries are injected into one or more conditional statements so that they are always evaluated to be true.
Logically Incorrect Queries	Using error messages rejected by the database to find useful data facilitating injection of the backend database.
Union Query	Injected query is joined with a safe query using the keyword UNION in order to get information related to other tables from the application.
Stored Procedure	Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes.
Piggy-Backed Queries	Additional malicious queries are inserted into an original injected query.
Inference - ***Blind Injection*** - ***Timing Attacks***	An attacker derives logical conclusions from the answer to a true/false question concerning the database. Information is collected by inferring from the replies of the page after questioning the server true/false questions. An attacker collects information by observing the response time (behavior) of the database.
Alternate Encodings	It aims to avoid being identified by secure defensive coding and automated prevention mechanisms. Hence, it helps the attackers to evade detection. It is usually combined with other attack techniques.

2.3 Types of SQL Injection Attacks

Table II shows the most commonly known SQL Injection attacks with brief descriptions.

3. DETECTING SQL INJECTION

In order to protect a Web application from SQL Injection attacks [11], there are two major concerns. Firstly, there is a great need of a mechanism to detect and exactly identify SQL Injection attacks. Secondly, knowledge of SQL Injection Vulnerabilities (SQLIVs) is a must for securing a Web application. So far, many frameworks have been used and/or suggested to detect SQLIVs in Web applications. Here, we mention the prominent solutions and their working methods in brief.

SAFELI - [3] proposes a Static Analysis Framework in order to detect SQL Injection Vulnerabilities. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the White-box Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

Thomas et al.'s Scheme - Thomas et al., in [4] suggest an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities. They implement their research work using four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. Based on the experimental results, their prepared statement code was able to successfully replace 94% of the SQLIVs in four open source projects.

Ruse et al.'s Approach - In [5], Ruse et al. propose a technique that uses automatic test case generation to detect SQL Injection Vulnerabilities. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. Adding to that, the approach identifies the relationship (dependency) between sub-queries. Based on the results, the methodology is shown to be able to specifically identify the causal set and obtain 85% and 69% reduction respectively while experimenting on few sample examples.

Haixia and Zhihong's Scheme - In [7], Haixia and Zhihong propose a secure database testing design for Web applications. They suggest a few things; firstly, detection of potential input points of SQL Injection; secondly, generation of test cases automatically then finally finding the database vulnerability by running the test cases to make a simulation attack to an application. The proposed methodology is shown to be efficient.

Roichman and Gudes's Scheme - [8] suggests using a fine-grained access control to web databases. The authors develop a new method based on fine-grained access control mechanism. The access to the database is supervised and monitored by the built-in database access control. This is a solution to the vulnerability of the SQL session traceability. Moreover, it is a framework applicable to almost all database applications.

Shin et al.'s approach – [19] suggests SQLUnitGen, a Static-analysis-based tool that automate testing for identifying input manipulation vulnerabilities. The authors apply SQLUnitGen tool which is compared with FindBugs, a static analysis tool. The proposed mechanism is shown to be efficient as regard to the fact that false positive was completely absent in the experiments.

SQL-IDS Approach - Kemalis and Tzouramanis in [2] suggest using a novel specification-based methodology for the detection of exploitations of SQL injection vulnerabilities. The proposed query-specific detection allowed the system to perform focused analysis at negligible computational overhead without producing false positives or false negatives.

4. SQLI COUNTERMEASURES

Table III shows a summary of so far known countermeasures against SQL Injection.

TABLE III. SQL INJECTION COUNTERMEASURES

Countermeasure	Overview
SQL-IDS [2]	A specification based approach to detect malicious intrusions
Prepared Statements [4]	It is a fixed query "template" which is pre-defined, providing type–specific placeholders for input data.
AMNESIA [6]	This scheme identifies illegal queries before their execution. Dynamically-generated queries are compared with the statically-built model using a runtime monitoring.
SQLrand [13]	A strong random integer is inserted in the SQL keywords.
SQL DOM [14]	A set of classes that are strongly-typed to a database schema are used to generate SQL statements instead of string manipulation.
SQLIA prevention using Stored Procedures [15], [16]	Combination between static analysis and runtime monitoring.
SQLGuard [17]	The parse trees of the SQL statement before and after user input are compared at a run time. The Web script has to be modified.
CANDID [18]	Programmer-intended query structures are guessed based upon evaluation runs over non-attacking candidate inputs.
SQLIPA [20]	Using user name & password hash values, to improve security of authentication process.
SQLCHECK [21]	A key is inserted at both beginning and end of user's input. Invalid syntactic forms are attacks. The key strength is a major issue.
DIWeDa [22]	To detect various types of intrusions in Web Databases applications.
Manual approaches [23]	Defensive programming and Code review mechanisms are applied.
Automated approaches [23]	Static analysis FindBugs and Web vulnerability scanning frameworks are implemented.

Now, let us see what these schemes are actually about. The remaining texts will analyze the various aspects covered in the different types of countermeasures.

AMNESIA - In [6], Junjin proposes AMNESIA approach for tracing SQL input flow and generating attack input, JCrasher for generating test cases, and SQLInjectionGen for identifying hotspots. The experiment was conducted on two Web applications running on MySQL1 1 v5.0.21. Based on three attempts on the two databases, SQLInjectionGen was found to give only two false negatives in one attempt. The proposed framework is efficient considering the fact that it emphasizes on attack input precision. Besides that, the attack input is properly matched with method arguments. The only disadvantage of this approach is that it involves a number of steps using different tools.

SQLrand Scheme - SQLrand approach [13] is proposed by Boyd and Keromytis. For the implementation, they use a proof of concept proxy server in between the Web server (client) and SQL server; they de-randomized queries received from the client and sent the request to the server. This de-randomization framework has 2 main advantages: portability and security. The proposed scheme has a good performance: 6.5 ms is the maximum latency overhead imposed on every query.

SQL DOM Scheme - SQL DOM framework is suggested by McClure and Krüger in [14]. They closely consider the existing flaws while accessing relational databases from the OOP (Object-Oriented Programming) Languages point of view. They mainly focus on identifying the obstacles in the interaction with the database via CLIs (Call Level Interfaces). SQL DOM object model is the proposed solution to tackle these issues through building a secure environment for communication.

SQLIA Prevention Using Stored Procedures - Stored procedures are subroutines in the database which the applications can make call to [15]. The prevention in these stored procedures is implemented by a combination of static analysis and runtime analysis. The static analysis used for commands identification is achieved through stored procedure parser and the runtime analysis by using a SQLChecker for input identification. [16] proposed a combination of static analysis and runtime monitoring to fortify the security of potential vulnerabilities.

Parse Tree Validation Approach - Buehrer et al. [17] adopt the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement. They stopped the execution of statement unless there is a match. This method was tested on a student Web application using SQLGuard. Although this approach is efficient, it has two major drawbacks: additional overheard computation and listing of input (black or white).

Dynamic Candidate Evaluations Approach - In [18], Bisht et al. propose CANDID. It is a Dynamic Candidate Evaluations method for automatic prevention of SQL Injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer). Hence, it solves the issue of manually modifying the application to create the prepared statements.

Ali et al.'s Scheme - [20] adopts the hash value approach to further improve the user authentication mechanism. They use the user name and password hash values. SQLIPA (SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The user name and password hash values are created and calculated at runtime for the first time the particular user account is created.

SQLCHECKER Approach - Su and Wassermann [21] implement their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is applied for the user input delimitation [9]. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages.

DIWeDa Approach - Roichman and Gudes [22] propose IDS (Intrusion Detection Systems) for the backend databases. They use DIWeDa, a prototype which acts at the session level rather than the SQL statement or transaction stage, to detect the intrusions in Web applications. The proposed framework is efficient and could identify SQL injections and business logic violations too.

Manual Approaches – [23] highlights the use of manual approaches in order to prevent SQLI input manipulation flaws. In manual approaches, defensive programming and code review are applied. In defensive programming: an input filter is implemented to disallow users to input malicious keywords or characters. This is achieved by using white or black lists. As regards to the code review [24], it is a low cost mechanism in detecting bugs however it requires deep knowledge on SQLIAs.

Automated Approaches - Besides using manual approaches, [23] also highlights the use of automated approaches. The author notes that the two main schemes are: Static analysis FindBugs and Web vulnerability scanning. Static analysis FindBugs approach detects bugs on SQLIAs, gives warning when an SQL query is made of variable. However, for Web vulnerability scanning, it uses software agents to crawl and scans Web applications and detects the vulnerabilities by observing their behavior.

5. COMPARATIVE ANALYSIS

Table IV shows a chart of the schemes and their defense capability against various SQLIAs. In Table V, we note down the major approaches to deal with SQL Injection and classify them based on their features.

TABLE IV. VARIOUS SCHEMES AND SQL INJECTION ATTACKS

Scheme	Tautology	Logically Incorrect Queries	Union Query	Stored Procedure	Piggy-Backed Queries	Inference	Alternate Encodings
AMNESIA [6]	✓	✓	✓	X	✓	✓	✓
SQLrand [13]	✓	X	✓	X	✓	✓	X
SQLDOM [14]	✓	✓	✓	X	✓	✓	✓
WebSSARI[15, 16]	✓	✓	✓	✓	✓	✓	✓
SQLGuard [17]	✓	✓	✓	X	✓	✓	✓
CANDID [18]	✓	X	X	X	X	X	X
SQLIPA [20]	✓	X	X	X	X	X	X
SQLCHECK [21]	✓	✓	✓	X	✓	✓	✓
DIWeDa[22]	X	X	X	X	X	✓	X
Automated approaches [23]	✓	✓	✓	X	✓	✓	X

TABLE V. OBJECTIVE OF VARIOUS APPROACHES

Approaches	Goals	
	Detection	Prevention
SQL-IDS [2]	Yes	Yes
AMNESIA [6]	Yes	Yes
SQLrand [13]	Yes	Yes
SQL DOM [14]	Yes	Yes
WebSSARI [15], [16]	Yes	Yes
SQLGuard [17]	Yes	No
CANDID [18]	Yes	No
SQLIPA [20]	Yes	No
SQLCHECK [21]	Yes	No
DIWeDa [22]	Yes	No

6. CONCLUSIONS

In this paper we have reviewed the most popular existing SQL Injections related issues. We have presented a survey report on various types of SQL Injection attacks, vulnerabilities, detection, and prevention techniques.

7 REFERENCES

[1] http://www.owasp.org/index.php/Top_10_2010-A1-Injection, retrieve on 13/01/2010

[2] K. Kemalis, and T. Tzouramanis (2008). SQL-IDS: A Specification-based Approach for SQLinjection Detection. SAC'08. Fortaleza, Ceará, Brazil, ACM: pp. 2153 2158.

[3] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A Static Analysis Framework for Detecting SQL Injection Vulnerabilities, COMPSAC 2007, pp.87-96, 24-27 July 2007

[4] S. Thomas, L. Williams, and T. Xie, On automated prepared statement generation to remove SQL injection vulnerabilities. Information and Software Technology 51, 589–598 (2009).

[5] M. Ruse, T. Sarkar and S. Basu. Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs. 10th Annual International Symposium on Applications and the Internet pp. 31 – 37 (2010)

[6] M. Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of the 6th Int. Conf. on Information Technology: New Generations, Las Vegas, Nevada, pp. 1411-1414, April 2009.

[7] Y. Haixia, N. Zhihong, "A database security testing scheme of web application," Proc. of ICCSE '09 , pp. 953-955, 25-28 July 2009.

[8] Roichman, A., Gudes, E.: Fine-grained Access Control to Web Databases. In: Proc. of 12th SACMAT Symposium, France (2007)

[9] W. G. Halfond, J. Viegas, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. In Proc. of the Intl. Symposium on Secure Software Engineering, Mar. 2006.

[10] A. Tajpour; M. Masrom; M. Z. Heydari.; S. Ibrahim; "SQL injection detection and prevention tools assessment," Proc. Of ICCSIT 2010, vol.9, no., pp.518-522, 9-11 July 2010

[11] A. Tajpour; M. JorJor Zade Shooshtari , "Evaluation of SQL Injection Detection and Prevention Techniques," Proc. of CICSyN, 2010, pp.216-221, 28-30 July 2010

[12] N. Seixas; J. Fonseca; M. Vieira; H. Madeira, "Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study," Proc. of ISSRE '09, pp.129-135.

[13] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pages 292–302, June 2004.

[14] R.A. McClure, and I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 88- 96, 15-21 May 2005.

[15] K. Amirtahmasebi, S. R. Jalalinia, S. Khadem, "A survey of SQL injection defense mechanisms," Proc. Of ICITST 2009, vol., no., pp.1-8, 9-12 Nov. 2009

[16] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee and S.-Y. Kuo, "Securing Web Application Code by Static Analysis and Runtime Protection," 13th International Conference on World Wide Web, New York, NY, 2004, pp. 40-52.

[17] G. Buehrer, B.W. Weide, P.A.G. Sivilotti, Using Parse Tree Validation to Prevent SQL Injection Attacks, in: 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, 2005, pp. 106–113.

[18] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Trans. Inf. Syst. Secur., 13(2):1–39, 2010.

[19] Y. Shin, L. Williams and T. Xie, "SQLUnitGen: Test Case Generation for SQL Injection Detection," North Carolina State Univ., Raleigh Technical report, NCSU CSC TR 2006-21, 2006.

[20] S. Ali, SK. Shahzad and H. Javed, "SQLIPA: An Authentication Mechanism Against SQL Injection," European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4 (2009), pp 604-611.

[21] Z. Su and G. Wassermann "The essence of command injection attacks in web applications". In ACM Symposium on Principles of Programming Languages (POPL'2006), January 2006.

[22] A. Roichman, E. Gudes, "DIWeDa - Detecting Intrusions in Web Databases". In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 313–329. Springer, Heidelberg (2008).

[23] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of ITNG '09, pp.1411-1414, 27-29 April 2009.

[24] R. A. Baker, "Code Reviews Enhance Software Quality," In Proceedings of the 19th international conference on Software engineering (ICSE'97) pp. 570 - 571, Boston, MA, USA. 1997.