A Design Review: Concepts for Mitigating SQL Injection Attacks

Ed Pearson III
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS, USA
ep602@msstate.edu

Cindy L. Bethel
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS, USA
cbethel@cse.msstate.edu

Abstract— Recently, it is not unusual to notice media coverage of some major breach in some large organization's cyber security. A large number of said breaches are due to vulnerabilities in their software or system. Once an in-depth analysis of these vulnerabilities was performed, it came to light that a large number of these vulnerabilities were the result of development issues. To be more specific, either the developers or the design process was the cause of the vulnerabilities. A particular vulnerability initiated by developers or a subpar design process is injection attacks.

In particular SQL injection attacks (SQLIA) have been the culprit of most organizational cyber security breaches. This form of attack could have a detrimental impact on a business or organization. These impacts could range from monetary loss, exposure of confidential business information, exposure of customer data, a decrease in company stock value, or some combination of these four. SQL injection attacks are relatively common in interactive web applications. Not only are SQL injection attacks common they are easily detectable and are reasonably simple to mitigate.

There is a plethora of literature on defending against SQL injection attacks once a system or software is functional. The goal of this work is to address the issue of SQL injection attacks starting in the design process. The contribution of this paper is a proposed design review methodology that allows designers to examine the user interface (UI) and user experience (UX) in the design phase to expose any attack surfaces that allow for an injection attack to occur. In particular, the method proposed in this work combines human-computer interaction concepts along with cyber security principles and software security techniques to design a user interface that is not subject to SQL injection attacks. Because injection attacks occur from malicious user input, this method concentrates on the design of the interface to eliminate all entry points that allow for injection attacks.

Keywords—SQL injection attacks; SQL injection attack mitigation; SQL injection attack prevention; human-computer interaction, software security; information security; cyber security

INTRODUCTION

Over the years there has been a dramatic increase in web applications for business purposes. These applications have become crucial tools of society, being used for everything, from banking and commerce to communications and education. As demonstrated through media coverage with Target, Wal-Mart, Home Depot, and Apple breaches, just to name a few, these applications frequently contained vulnerabilities. The leading vulnerability in most instances of web applications are SQL injection attacks. Ranked number one on the Open Web Application Security Project (OWASP) 2013 list of vulnerabilities was injections attacks, which contained the domain of SQL injection attacks. This vulnerability is a result of insufficient design practices or unsatisfactory implementation habits [9].

In essence, according to OWASP, a SQL injection attack occurs when untrusted data arrives at the interpreter incorporated into a command or query. The malicious input can cause the interpreter to execute involuntary commands or grant access to data without proper authorization [9]. Bing and Xin's discussion of SQL injection attacks state that attackers input precise SQL commands in either the URL, form fields, or some other input field to alter the query properties, which misleads the verification of the application and management, resulting in the exposure of sensitive information. Malicious input can be entered that can damage the contents and structure of the database. It is even possible to use certain capabilities of the database itself to manipulate the server's operating the system [1].

Most of the literature revolving around SQL injection attacks attempt to rectify the attack once the system is up and running. The goal of our work is to remedy the SQL injection attacks in the design phase. The method presented in this paper utilizes techniques from human-computer interaction (HCI) for designing user interfaces (UI) and user experiences (UX) and harnesses the tools of perception and cognition along with ideas on sketching to design software/interfaces to ensure there are no attack vectors for SQL injection attacks. This method is designed to work as a team effort and is composed of multiple steps. In addition, this method requires the team to already have knowledge and training in the area of software/cyber security. This paper has the following sections: Section I discusses *What is an SQL injection attack?*; Section II outlines the different types of injection attacks; Section III reviews current measures used to mitigate SQL injections attacks; Section IV presents the theory behind the HCI method; Section V gives a comparison of mitigation methods; and Section VI provides conclusions and future work derived from this research.

I. What is a SQL Injection Attack?

According to Wei, Muthuprasanna, and Kothari [11], a SQL injection attack represents a vital class of attacks aimed at web applications. These authors also point out that this form of attack is possible due to inadequate input validation. By taking advantage of this vulnerability an attacker may gain access to the underlying database used for the application. It is also Wei, Muthuprasanna, and Kothari's belief that any web application that accesses the internet or is contained within a corporate intranet is vulnerable to SQLIAs [11]. In their discussion on SQLIA, Sharma and Jain state this form of attack includes the injection of keywords inside of SQL queries that alter the logic of the query [7]. Shan, Xiaorui, and Hong describe a SQLIA as an attack where malicious code is implanted within a string that will eventually be passed as a request to the SQL Server for deconstruction and execution [6].

All the definitions given to exemplify SQLIAs commonly express that this form attack exists due to a lack of or improper validation of inputs inserted into a web application. According to Sharma and Jain this type of attack is the result of a vulnerability in the design of the web application [7]. Shan, Xiaorui, and Hong take it a step further by conveying the fact that any web application that fabricates SQL statements is subject to the vulnerability of injection since SQL Servers will execute all queries that have the correct syntax [6]. So, it is reasonable to conclude that this vulnerability is caused by the designer as well the programmer (to be concise both designers and programmers will be referred to as developers) of the system. There are a number of reasons why developers make mistakes when it comes to security. However, a study conducted by Xie, Lipford, and Chu exposed that the core reasons stem from the developers having common erudition and awareness of software security, but gaps reside amid the knowledge, routines, and behaviors exhibited by the developer [12].

The essentials needed for an attacker to mount an SQLIA are a web browser, the knowledge of SQL queries, and clever guesses of significant table and field names. There are two approaches for executing SQLIAs, which are via user inputs and URLs [7]. The procedure for launching an attack can be summarized in four steps. The first step entails the detection of whether an application is vulnerable to a SQLIA. This is achieved by determining if special characters are accepted as input. The second step of launching a SQLIA is the determination of what type of database is being used by the web application. Establishing the database type is beneficial because different database management systems have different injection procedures. Gathering all possible information about the database is the third step. This step depends on the attacker's ability to guess tables, field names, and stored procedures in the database. The last step is to mount the attack, which is now simple because the attacker has done all of the reconnaissance. Now that the attacker has knowledge of the system's vulnerability and which type of database management is in place, the attacks could scale from adding administrator accounts, opening the 3389 Remote Terminal Services, to uploading web-page Trojans [1].

II. Different Types of SQL Injection Attacks

Currently, there are two primary causes of SQLIAs being discussed in research. The first is that some web development languages contain vulnerabilities as a result of syntactical restrictions. The second and primary catalyst are inadequate design and implementation methods such as poor input validation, type checking, escalated privilege accounts, and error messages with too much information. Due to the nature of SQLIAs, the attacks can be grouped in three taxonomies: (1) orderwise, (2) blind, and (3) against the database [7].

Orderwise SQLIAs occur outside and are injected within the user's code, which can happen directly or indirectly to obtain unlimited and unauthorized access. Orderwise attacks are also broken into three subclasses, which are first-order injection attack, second-order injection attack, and lateral injection attack. First-Order Injection Attacks are basic in nature. The assailant is given the anticipated result instantly by direct response from the application. Second-Order Injection Attacks happen when the injected code does not expose or commit any malicious action directly after injection, but waits for certain inputs by the user. Second-Order Injection Attacks can also be divided into a subset of four categories, which is beyond the scope of this paper. Lateral Injection Attacks occur when the assailant exploits PL/SQL routines that do not accept inputs from users [7].

Blind SQLIAs are like playing the guessing game with a web application. This form of attack stems from insufficient results from the third step of launching a SQLIA. Other contributors are defensive tactics used by developers. Generally, when developers obfuscate error messages, error message detail, or the error message page, it becomes difficult for attackers to craft the injection statements necessary to extract results. Thus the attacker uses a series of true and false statements to acquire information about and from the system [7].

Last are against database SQLIAs, which are stated to be the most common attacks deployed as a means to obtain data from the database server in the form of SQL queries. This form of attack capitalizes on vulnerabilities involving user input validation. It is highly difficult to detect or protect against this type of attack because intrusion detection systems (IDS) and firewalls do not catch these attacks because the attacks are being crafted from SQL statements that have the correct syntax for both the IDS and the firewalls [7].

III. Current Methods to alleviate SQL Injection Attacks

Researchers have derived a multitude of approaches to combat against SQLIAs. Most of the methods are designed for the defense against SQLIAs are meant for use after an application is functional and each vary in the range of

difficulty to deploy. One method proposed by Shan, Xiaorui, and Hong, incorporated the use of a protective shell around the database, which is expected to prevent SQL injections in the data layer, and is achievable by spending a small amount of extra time resources. The protective shell is composed of three layers: (1) the character list, (2) association list, and (3) the command structure tree [6].

Bing and Xin [1] presented a multi-level prevention approach for defending against SQLIAs. This method has five levels. The first level focuses on the client side and it places limits on the interaction between the application and the user. The next level concentrates on the server side where the task is to validate and filter submissions to the server. The third level is a database configuration, because most database management systems already incorporate security functionality, the goal is to configure these features in such a manner that yields optimal performance. Level four is associated with web application development, in which features used to develop and configure the application are examined. The authors stated that internet information services (IIS) security configurations help to avoid SQLIAs. Level four is also where browser error configurations, directory permissions, and other general configurations come into effect. General configurations may include the following instances: eliminating unwarranted application mapping, removal of virtual directories, etc. The last level is SQLI vulnerability scanning, which involves analysis of the application and website for vulnerabilities. Using a tool for this level is highly recommended [1].

Wei, Muthuprasanna, and Kothari [11] presented a method for the prevention of SQLIAs in stored procedures. The method Wei, Muthuprasanna, and Kothari proposed was fully automated and designed to detect, prevent, and report SQLIA occurrences in stored procedures. The essence of their technique constituted taking the proposed SQL query and isolating its activities in an application and representing this as a SQL-graph offline as a single instance through the use of a static analysis of the stored procedure code. Next, the developed graph was used to compare user inputs for malicious intent before implementation. The SQL-graph was beneficial in that it apprehended a wide variety of models and forms for the execution of SQLIAs [11]. This method serves as an approach to defend against the class Sharma and Jain [7] labeled Against Database SQLIA. Sharma and Jain also offer simple and efficient methods for defending against SQLIAs that designers and developers alike need to know in order for the method presented in our paper to be effective.

IV. HCI Method for Mitigation

It has been stated repeatedly throughout the literature that programmers and designers make mistakes related to vulnerabilities in software and applications [2][12]. Inherently, the liability of SQLIAs reside with the practices of developers and designers as well [4][11][1][7][6]. Stoica and Islam suggest that teaching courses in software and information security are essential, and they presented a framework for developing this type of curriculum [5]. Most of the literature on SQLIAs comment in some way that the education of designers and programmers is the key to the creation of secure defenses against this type of attack. The rest of this section is dedicated to the presentation of the theory and foundations necessary for the development of a method that is beneficial for defending against SQLIAs.

The HCI-SQLIA Mitigation Method (HCI-SQLIAM^2) is designed to be integrated into the development life cycle of web applications. HCI-SQLIAM^2 requires that the designers and developers have knowledge of SQLIAs. In addition to understanding SQLIAs, the design and development team should possess techniques for the detection and prevention of SQLIAs [13]. Xie, Lipford, and Chu insist that simply having knowledge of the vulnerability along with detection and prevention methods, is not enough and that something must be done to change the way developers and designers practice application/software/interface development [12].

HCI-SQLIAM^2 is geared toward web application development. In the current stages of this research, the focus is on the design phase, specifically the design of the user interface. Analyses are currently being conducted with HCI concepts related to cognition and perception in conjunction with the investigation of inclusion in the phases of interface design (sketching, designing, prototyping, and implementation) that results in the minimal number of attack surfaces for SQLIAs to occur [10]. This process involves all members of the application development team (designers and programmers). In order to optimize this phase, all members should undergo educational training in SQLIAs. The details of this phase involve designing the interface using a sketching approach [3]. After deciding on a final sketch concept all members of the development team review the design and inspect it for any possible SQLIA surfaces. For each plausible vulnerability, the team member should sketch an alternative, this will initiate the HCI Design Funnel [3]. While in the HCI Design Funnel, a metric for determining security benefits over usability benefits, dictates when to exit the Design Funnel [3] [8]. This is the basis for HCI-SQLIAM^2, and further research is essential to finesse this methodology.

V. Comparative Analysis

Earlier three methods for SQLIA mitigation were presented, and was followed by a proposed method for prevention. The first method discussed was a protective shell method which defends against SQLIAs in three layers. The three levels consist of: (1) a character list protection layer, (2) an association list protection layer, and (3) a command structure tree. The character list is composed of a list of characters that if included in a query, that query could possibly be malicious. The association list contains a list of sensitive data fields and the accompanied modified fields, for example when a database updates a password it also modifies the “revised time” field. The command structure tree

assembles the SQL commands according specific rules, which are designated by the programmer's coding style [6].

The next approach presented was a multi-level technique that had five levels. The five levels included the client side protection, server side protection, web application deployment protection, database configuration, and SQL injection vulnerability scanning [1]. On the client side, the main focus limited the length of string inputs that the software received, and used ASP.NET validation controls. The server side protection concentrated on validations that must be performed on the server by filtering functions, string detection functions, and type safe SQL parameters. The next level, database configuration, which focuses on how securely the database is constructed. For example ensuring user accounts run with the least amount of privileges necessary to complete the task. Web application deployment, in this case, is where special attention is given to browser error configuration, directory permissions, and other configuration issues like removing useless virtual directories. The final level is SQL injection vulnerability scanning, which involves checking one's website and/or web application for SQL Injection vulnerabilities on a constant basis [1].

The third mitigation technique analyzed was an automated approach for defending against injection attacks in stored procedures. In this method a static analysis is performed, then a SQL-graph representation is constructed, instrumentation is performed, and a runtime analysis is conducted. In the static analysis, a control flow is constructed, which is used to determine the SQL-graph. The SQL-graph representation is used to lower runtime scanning overhead. The instrumentation is used to isolate the user input from the original SQL statements. Then a runtime analysis compares the SQL statements with user inputs to a finite state automaton to ensure validity. If the input causes the statement not to conform, then it is flagged as a SQLIA [11].

The methodology proposed in this paper was titled the HCI-SQLIAM^2 approach and it is meant to be a prevention method as opposed to the other methods discussed. This method is composed of some of the techniques used by these other methods but differ slightly, in that these tactics are being applied from design all the way through to the implementation. The HCI-SQLIAM^2 method starts with a series of prototype evaluations, and then ends with an interface and/or software system that defends against SQLIAs, as opposed to an interface/software that has to be patched because no preventative measures were considered. Table 1 highlights the commonalities and differences in the approaches presented in this article.

VI. Conclusion & the Future of the Journey

Research has proven that SQLIAs are the leading threat for web applications [9]. Research has demonstrated that programmers and/or designers are to blame for the existence of these types of vulnerabilities in web applications. Furthermore, the common denominator among several researchers is that the only way to safely defend against SQLIAs is to change the practices of the application development team, both designers and programmers. It is evident that the defense and patching methods commonly used are not sufficient. Our work focuses on a methodology that is geared toward changing the tactics of developing web applications aimed specifically at SQLIAs. The method is called HCI-SQLIAM^2, and is easy to integrate into the commonly used development life cycle. The methodology is in its early stages of applying HCI techniques through the use of cognition and perception in combination with sketching to develop a user interface that contains the minimal number of possible SQLIA surface areas.

Table 1. A comparative analysis of the methods discussed.

Features:	Protective Shell Method:	Multi-level Prevention Method	Automated Protection Against Stored Procedure	HCI-SQLIAM^2
Character List Protection Layer:	X			
Assoication List Protection Layer:	X			
Command Structure Tree:	X			
Paper-Based Prototype Analyses				X
Computer-Based Prototype Analyses:				X
Fully-Functional Prototype Analysis:				X
Client Side Protection:		X		X
Server Side Protection:		X		X
Web Application Deployment Protection:		X		X
Database Configuration:		X		X
SQL Injection Vulnerability Scanning:		X		X
Static Analysis:			X	X
SQL–graph Representation:			X	X
Instrumentation:			X	X
Runtime Analysis:			X	X

Further research will be performed to fully develop this method. Also metrics need to be developed to determine proper security, usability, and user experience for each application. Along with the development of metrics, the creation of a standard would be ideal for the annotation of prototypes that will be developed in a later phase of this methodology. Once the method is sufficient for reducing the number of possible SQLIAs and is validated, further research will be conducted to determine its adaptability to other types of vulnerabilities.

ACKNOWLEDGMENT

The author would like to thank several people for multiple reasons in relation to their influence and support that contributed to the completion of this work. I would like to offer thanks and my sincere gratitude to Dr. Dave Dampier for providing this wonderful opportunity and funding to conduct research in the area of cyber security. Next I would like to thank Dr. Byron Williams, who helped inspire this topic as it combines HCI and Software Security. Also I would like to declare my utmost appreciation to Ms. Altricia Jordan for always being there to assist with editing. Last, I would like to thank Dr. Ed Allen for his guidance with the focus and scope of this work.

REFERENCES

[1] Y. Bing, W. Xin, "Multi-level Preventing SQL Injection Attacks," *Conference Anthology*, IEEE, 1-8 Jan. 2013, pp. 1-4.

[2] N. Davis, W. Humphrey, S. T. Redwine, G. Zibulski, "Processes for Producing Secure Software," *Security and Privacy*, IEEE, vol. 2, no. 3, May-June 2004, pp. 18-25.

[3] S. Greenberg, S. Carpendale, N. Marquardt, B. Buxton, *Sketching User Experiences: The Workbook*, Elsevier, 2011.

[4] L. Li, Q, Dong, D, Liu, L. Zhu, "The Application of Fuzzing in Web Software Security Vulnerabilities Test," *2013 International Conference on Information Technology, and Applications*, IEEE, 16-17 Nov. 2013, pp. 130-133.

[5] A. Stoica, S. Islam, "Teaching Information and Software Security Courses in Regular and Distance learning Programs," *2013 IEEE Global Engineering Education Conference (EDUCON)*, IEEE, 13-15 March 2013, pp. 44-50.

[6] L. Shan, D. Xiaorui, R. Hong, "An Adaptive Method Preventing Database from SQL Injection Attacks," *2010 3rd International Conference on Advance Computer Theory and Engineering (ICACTE)*, IEEE, 20-22 Aug. 2010, pp. 352-355.

[7] C. Sharma, S. C. Jain, "Analysis and Classification of SQL Injection Vulnerabilities and Attack on Web Applications," *2014 International Conference on Advances in Engineering and Technology Research (ICAETR)*, IEEE, 1-2 Aug. 2014, pp. 1-6.

[8] A. Sivaji, S. S. Tzuaan, "Website User Experience (UX) Testing Tool Development Using Open Source Software (OSS)," *2012 Southeast Asia Network of Ergonomics Societies Conference (SEANES)*, IEEE, 9-12 July 2012, pp. 1-6.

[9] The Open Web Application Security Project (OWASP), "OWASP Top 10- 2013: The Ten Most Critical Web Application Security Risks," www.osawp.org, 2 Jan. 2015.

[10] C. von Saucken, I. Michailidou, U. Lindemann, "Emotional Mental Model," *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, IEEE, 10-13 Dec. 2013, pp. 802-806.

[11] K. Wei, M. Muthuprasanna, S. Kothari, "Preventing SQL Injection Attacks in Stored Procedures," *2006 Australian Software Engineering Conference*, IEEE, 18-21 April 2006.

[12] J. Xie, H. R. Lipford, B. Chu, "Why do Programmers Make Security Errors?," *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 18-22 Sept. 2011, pp. 161-164.

[13] F. Zeng, L. Li, J. Li, X. Wang, "Vulnerability Testing of Software Using Extended EAI Model," *WRI World Congress on Software Engineering (WCSE)*, IEEE, vol. 4 19-21 May 2009, pp. 261-265.