A HYBRID TECHNIQUE FOR SQL INJECTION ATTACKS DETECTION AND PREVENTION

Jalal Omer Atoum and Amer Jibril Qaralleh

Princess Sumaya University for Technology, Amman, Jordan

ABSTRACT

SQL injection is a type of attacks used to gain, manipulate, or delete information in any data-driven system whether this system is online or offline and whether this system is a web or non-web-based. It is distinguished by the multiplicity of its performing methods, so defense techniques could not detect or prevent such attacks. The main objective of this paper is to create a reliable and accurate hybrid technique that secure systems from being exploited by SQL injection attacks. This hybrid technique combines static and runtime SQL queries analysis to create a defense strategy that can detect and prevent various types of SQL injection attacks. To evaluate this suggested technique, a large set of SQL queries have been executed through a simulation that had been developed. The results indicate that the suggested technique is reliable and more effective in capturing more SQL injection types compared to other SQL injection detection methods.

KEYWORDS

Database SQL Injection Atttacks, Static Analysis, Runtime Analysis, Three Tier Architecture.

1. INTRODUCTION

SQL injection attacks (SQLIAs) are very effective system attacks that can be used to gain or manipulate data in data-driven systems, which is a common problem for web applications that are published on the internet. Furthrmore, SQLIAs are simple to be learned and simple to be executed; so they can be executed by unexperienced hackers [16].

There have been many researches that have developed various methods to detect and prevent SQLIAs. Each of these methods covers an objective or set of objectives related to this type of attacks, but there is no method that can cover the whole system from being attacked by SQL injections [6].

The risk of SQLIAs is that when they are performed through the victim back end system, they will be runing with the same priviliges that the system have on the database, that means if the system has a power user or administrator permisions then the injection code could be executed with a disaster effects on the victim machine [2].

Section two presents the aspects related to the different types of SQLIAs and describes the vulnerabilities that are used to perform the SQLIAs. Section three prsentes different previous solutions to deal with the SQLIAs detection and prevention. Section four presents the suggested hybrid technique. Section five presents a description of the simulation that has been developed to

DOI : 10.5121/ijdms.2014.6102

evaluate the reliability and accuracy of the suggested hybrid technique. Finally, section six prsents the conclusion and future works.

2. SQL INJECTION ATTACKS

There is no solution that ensures all vulnerabilities in a system will be covered and controlled completely 100%. There are vulnerabilities in which SQLIAs attackers preferred to use in order to breach the systems data, those vulnerabilities are either Software or Hardware elements such as (Servers, Web-Services, Operating Systems, Applications, Database Engines, etc.). If these elements are not continuasly updated with the latest patches and security updates, then they will be more vulnerable to be attacked, and then they might not be able to reject such attacks.

Monitoring, logging, validation, intrusion detection, and other operations are very useful in system architectures to increase the security of the database. If the system is not applying a strong input validation technique to check every database input to the system it will create a significant problem, because the input parameters are the first gate to the attacker that could be used to inject malicious code with this input [3].

The developers should be curious about the error reporting, they should not enable client error reporting service, because it may lead to an important information of the code or the database of the system.

The first mechanism to handle the security of a database is to ensure that their access is well controlled, by assigning the access rights to the appropriate users or objects [8]. Hence, if the first defense line is not handled as well as required then the database will be vulnerable to different type of attacks.

It is important to secure the data especially the sensitive data, so even though the database is secured from being hacked, sensitive data should be encrypted in the database or through the network [13].

In advanced SQLIAs attackers prefer to use the database core tables that contain sensitive information about the whole database system. Table 1 shows some of the common useful database system tables that are preferred to be used in the SQLIAs.

Table 1. Database system's tables for different Database environment

MS SQL Server	MS Access Server	Oracle
sysobjects syscolumns	MSysACEs MsysObjects	SYS.USER_OBJECTS SYS.TAB
	MsysQueries	SYS.USER_TABLES
	MSysRelationships	SYS.USER_VIEWS
		SYS.ALL_TABLES
		SYS.USER_TAB_COLUMNS
		SYS.USER_CONSTRAINTS
		SYS.USER_TRIGGERS
		SYS.USER_CATALOG

SQLIAs target database engines that are connected with data-driven systems. Hence, once users are connected to database to get answers for their requests, the system submits these answers as SQL queries to the database management system (DBMS) in the database server. After that, the database server returns the related information (answers) to the system. Finally, the system renders the resulted data as visual information to the requester (user).

The attacker can exploit the flow of data between the user, the system, and the database to gain or manipulate the data by sending queries loaded (injected) by malicious scripts, inline SQL queries, or commands that will be executed by the database engine and applied to the system database [7].

The intents of the SQL injection attacks could be categorized as; Determining database information, Data Gathering, Database Manipulation, Code Injection, Function Call Injection, or Buffer Overflows. For more information on these SQL injection attacks please refere to [4].

The most effective gateways that are used to perform different types of SQLIAs are: browser variables, user inputs, and injection HTTP header [4].

3. BACKGROUND

This section presents the litrature review that is relevant to SQLIAs and describes the common researches and techniques that have been done in order to detect and prevent SQLIAs.

SQLIAs detection and prevention techniques have followed various aspects in order to come up with an appropriate solution so as to prevent SQLIAs from being applied to different types of databases. Some of these aspects are:

- **Static Analysis**: Static analysis is a principle that depends on finding the weaknesses and malicious codes in the system source code prior to reaching the execution stage [10, 12]. Generally, this principle has been one of the most widely used to detect or prevent SQLIAs.

- **Runtime Analysis**: It is a technique which has been used to detect a specific type of attacks that should be identefied in advance without the need of modifing the development lifecycle nor the need of the source code of the system. Such a technique depends on tracking the events of the system through its execution process and detects if there is any of attack that is happing while execution [7] .

- **Static and Runtime Analysis**: In this type of analysis, different researches had choosen to combine the two aforementioned techniques to create a more effective and reliable solution to obtain a higher quality with a faster development and testing processes [1].

4. SUGGESTED HYBRID TECHNIQUE

This section focuses on the main idea of the suggested hybrid technique for detecting and preventing SQLIAs.

4.1 Normal Data Exchanging Strategy

There are many architectures to manage and to organize any data-driven systems, but the most common architecture that has been used is the three-tier architecture that depends on dividing the system into three tiers [15] as follows:

1. Presentation Tier (a Web browser or rendering engine).
2. Logic Tier (a server code, such as C#, ASP, .NET, PHP, JSP, etc …).
3. Storage Tier (a database such as Microsoft SQL Server, MySQL, Oracle, etc.).

Figure 1 summerizes the steps of exchanging data among the three-tier system architecture.

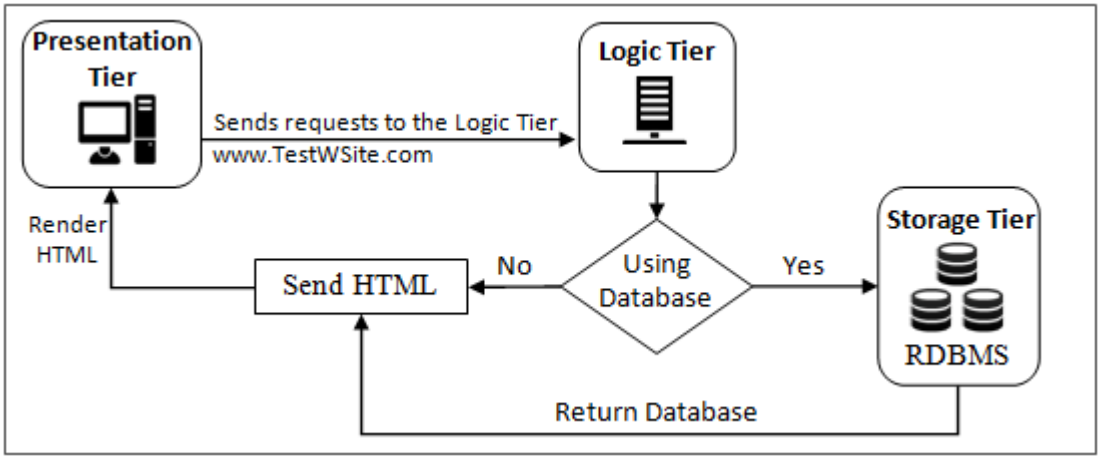

Figure 1. Three-Tier Architecture Data Exchanging

Figure 2 describes the normal mode to link the logged on users to systems that have the database instances and to determine the accessible instances.

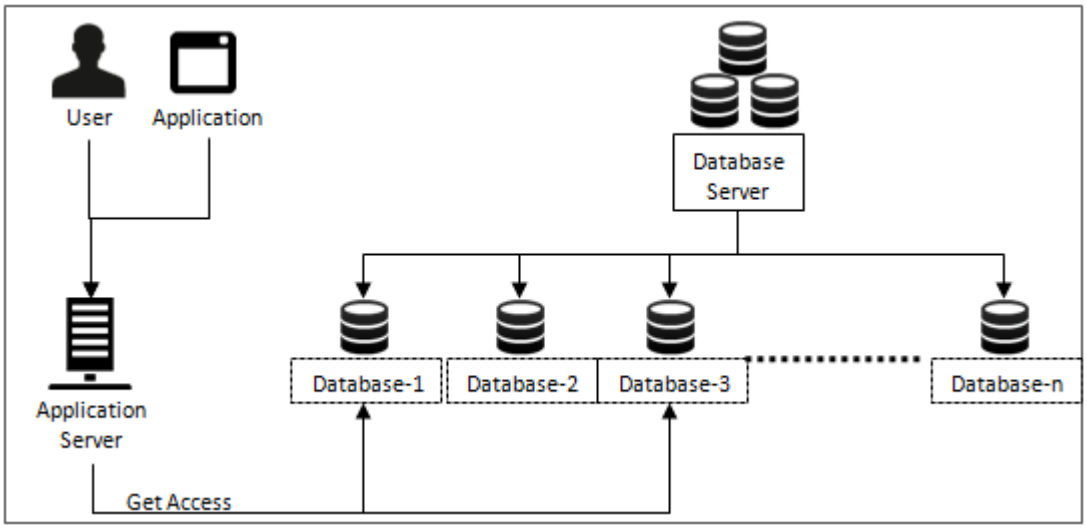

Figure 2. Accessing Database in Normal Mode

4.2 Suggested Approach Strategy

The suggested approach is a runtime detection and prevention methodology that follows the same steps as the normal approach to exchange the queries between the architecture parties (Presentation-Logic-Storage), however, it provides an extra defense line on the Data-Tier to ensure that this side will not execute any abnormal codes that incase affect the system partially or completely or it affects the hosted operating system and devices.

This approach is based on providing security controlling methodology on the database server side to ensures that all requested SQL queries from an inside or an outside the system are executed

securely without any database fabrication or hacking. Figure 3 illustrates the process flow diagram of the suggested approach stages from getting user or application access to the execution of the queries that have been delivered to the database.

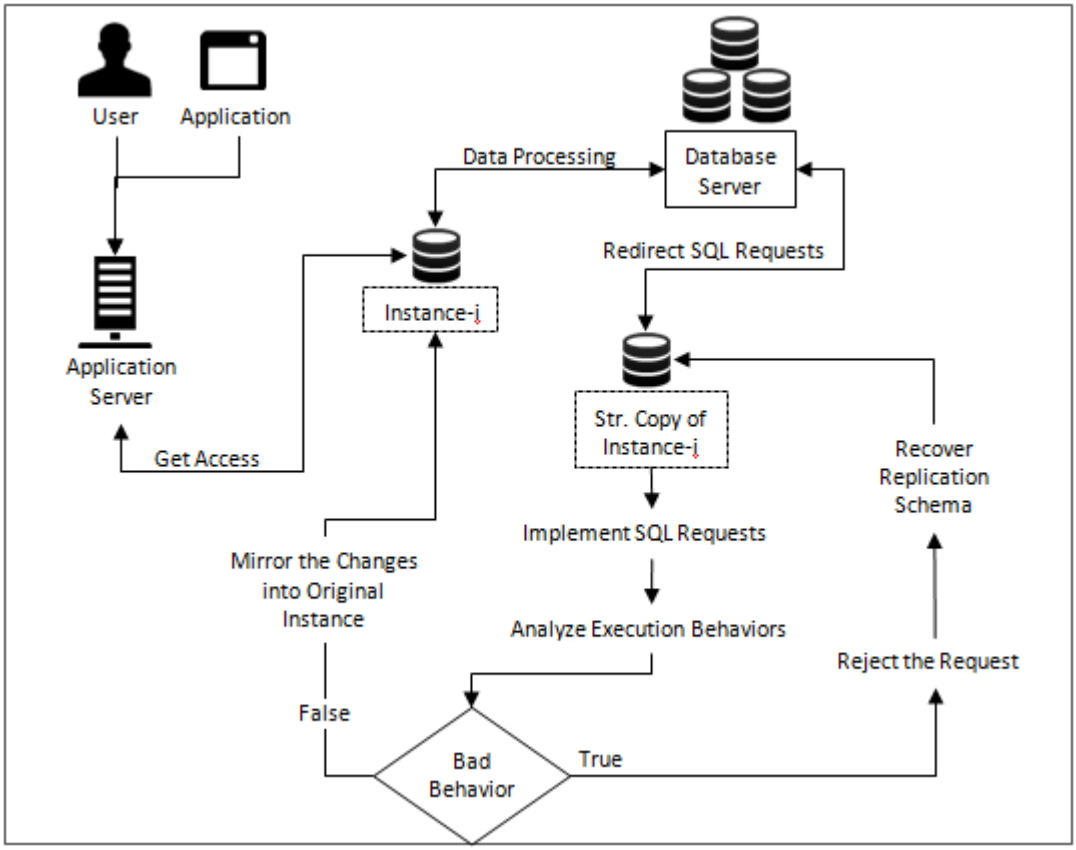

Figure 3. Process Flow Diagram for Suggested Approach

4.3 Suggested Approach Stages

The suggested approach is based on different stages to reject any malicious query from being passed through the database engine before its execution process, and those stages could be listed as follows:

• **Replicate system databases:** For each database to be secured from SQLIAs, there should be a new replication database and it should contain a small amount of sample data.

• **Creating "*database_Behaviors*" database:** The suggested approach should have a separate database called "***database_Behaviors***" that contains all system database queries and their expected behaviors that have resulted from SQL queries execution in normal cases. This database is placed in the replicated instances.

• **Redirect SQL queries:** Any SQL query assigned to be executed in the target database will be initally delayed and replicated by the database engine then this replicated query is sent to the

virtual database (Schema Replicated database). Hence, the original SQL query will be not executed yet in this stage and it will be delayed to a later stage.

• **Simple SQL syntax checking:** All SQL queries that are passing through the replicated database should also pass through multiple check processes before they move to the next step namely, "The execution process". The following list presents the checks processes that the SQL queries should pass through:

- Encoding analysis: Before continuing to any next step the received SQL queries should be analyzed to determine the character encoding that has been used to write these queries. There are many techniques that can be used to do this analysis process such as "Automatic Identification of Language and Encoding" [11].
- Simple White-Box validation: The query should go through simple syntax validation and filtering for specific SQL reserved words especially those that use (EXECUTE, SHELL commands).
- Parameters replacement: Any parameter that has been found in the SQL query should be replaced by an indexing parameter names. Such as (@par_1, @par_2 … @par_n).

• **Virtual execution:** After the SQL syntax checking process, the SQL query will be executed on the replicated database "Virtual Database" in which it is a process that is running simultaneously with the execution process, it monitors and traces the behaviors of the SQL query.

• **SQLIA Detection:** This stage is the most important stage in the suggested technique, its purpose is to detect whether the received SQL query is valid and expected query or not. The idea here is to catch the objects that have been affected by the current SQL query whatever the type of such objects and create a list of these objects to use them in the next step of this stage.

The resulted list of affected objects will be compared with the "***database_Behaviors"***. If there is a query that handles all of the listed objects with the same type of behavior that is detected from the previous step then this behavior query will be added to a new list (Expected Queries). Any resulted behavior that is detected as a suspicious should be rejected and deleted from the actual database instance execution queue, otherwise the query will be transferred to the actual database instance for being executed.

5. EVALUATION AND DISCUSSION

As described before, the proposed hybrid technique combines static and runtime analysis approaches to create a new solution to detect and prevent the SQLIAs. This suggested hybrid technique will be installed and integrated with the database engines in the database server.

An application using VB .Net has been developed to simulate the work of the suggested approach. The simulation application has been used to evaluate the performance and accuracy of the detection and prevention processes in this approach. Using this application, two hundred and fifty (250) SQL queries that cover all different types of SQLIA have been tested.

The results that had been obtained from simulating this hybrid technique of those 250 queries prove that this hybrid technique could cover all known SQLIA gateways, and prevents any type of SQLIAs.

Table 2 gives a comparison of well-known set of SQLIA detection and prevention techniques along with our suggested hybrid technique in terms of their capability of detection and

preventions, namely: Tautology, Built-In Functions, Logically Incorrect Queries, Union Query, Stored Procedure, Piggy-Backed Queries, Inference, Alternate Encoding, and the Direct Attack.

Table 1: Various Schemes and SQL Injection Attacks

SCHEMES	Tautology	Built-In Functions	Logically Incorrect Queries	Union Query	Stored Procedure	Piggy-Backed Queries	Inference	Alternate Encod.	Direct Attack
Suggested Technique	✓	✓	✓	✓	✓	✓	✓	✓	✓
AMNESIA	✓	×	✓	✓	×	✓	✓	✓	×
SQLrand	✓	×	×	✓	×	✓	✓	×	×
SQLDOM	✓	×	✓	✓	×	✓	✓	✓	×
WebSSARI	✓	×	✓	✓	✓	✓	✓	✓	×
SQLGuard	✓	×	✓	✓	×	✓	✓	✓	×
CANDID	✓	×	×	×	×	×	×	×	×
SQLIPA	✓	×	×	×	×	×	×	×	×
SQLCHECK	✓	×	✓	✓	×	✓	✓	✓	×
DIWeDa	×	×	×	×	×	×	✓	×	×
Automated approaches	✓	×	✓	✓	×	✓	✓	×	×

This table has been originally presented by [5] except for the first row of our suggested hybrid technique, the third column of Built-In Functions, and the last column of direct attacks. From this table, it can be concluded that our hybrid technique covers all types of SQLIA and it is the only technique that prevents the direct attack type; that means it can detect and prevent any type of SQLIA even if this attack is applied into the database directly. In other words, this hybrid technique can detect and prevent SQLIAs that are performed through the system or through a direct SQL query to the database. Finally, the suggested hybrid technique is the only one that can detect and prevent SQLIAs that are using Built-In functions to perform such attacks.

6. CONCLUSION AND FUTURE WORK

This paper has presented a novel hybrid technique that detects and prevents all types of SQLIAs in different system categories regardless of the system development language or the database engine.

The suggested hybrid technique is done in two main phases: runtime analysis, and static analysis. The first phase is a dynamic/runtime analysis method that depends on applying tracking methods to trace and monitor the execution processes of all received queries. The result of affected objects of this monitoring will be compared with a prepared set of expected changes that the developer had created before, and the result of this comparison process will decide if there is an existence of any type of SQLIA and if so they will be forwarded to the next phase. The next phase is a static analysis phase that is performing a string comparison between the received SQL queries and previous expected SQL queries to prevent any query that is described as a suspicious query.

Furthermore, the simulation showed that the suggested hybrid technique can detect and prevent all types of SQLIAs.

The future plan is to enhance this technique by decreasing the time delay that the database recovery takes after the SQLIA is detected.

REFERENCES

[1] Graham, B., Leroux, P. N., and Landry, T. "Using Static and Runtime Analysis to Improve Developer Productivity and Product Quality," white paper, QNX Software Systems, April 2008.

[2] Guimarães, B. D., "Advanced SQL Injection to Operating System Full Control," Black Hat Europe, white paper, April 2009.

[3] Halde, J., "SQL Injection Analysis, Detection and Prevention," MSc Thesis, Department of Computer Science, San Jose State University, San Jose, CA, USA, 2008.

[4] Halfond, W. G., Viegas, J. and Orso, A., "A Classification of SQL Injection Attacks and Countermeasures", In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, 2006..

[5] Kindy, D. A., and Pathan, A. S., "A Detailed Survey on Various Aspects of SQL Injection: Vulnerabilities, Innovative Attacks, and Remedies," International Journal of Communication Networks & Information Security, Aug. 2013, Vol. 5 Issue 2, pp 80-92.

[6] Majumder, J., and Saha, G., "Analysis of SQL Injection Attack," Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (PRINT) : 2231–5292, Vol.- II, Issue-1.

[7] Mishra, R. and Bhattacharjya, A., "A Study on Deterrence Methods from SQLIA," VSRD International Journal of CS & IT, vol. I, no. 8, pp. 608-617, 2011.

[8] Murray, M., "Database Security: What Students Need to Know," Journal of Information Technology Education, Innovations in Practice (9), pp. 61-77.

[9] Rani, D. R., Kumar, B. S., Rao, L. R., Jagadish, V. T., and Pradeep, M., "Web Security by Preventing SQL Injection Using Encryption in Stored Procedures," (IJCSIT) International Journal of Computer Science and Information Technologies, vol. 3, no. 2, 0975-9646, pp. 3689-3692, 2012.

[10] Roy, S., A. K. Singh and Sairam, A. S., "Analyzing SQL Meta Characters and Preventing SQL Injection Attacks Using Meta Filter," 2011 International Conference on Information and Electronics Engineering, IPCSIT vol.6 (2011) © (2011) IACSIT Press, Singapore, pp 167-170.

[11] Russell, G., Lapalme, G., Plamondon, P, "Automatic Identification of Language and Encoding". Rapport Scientifique. Laboratoire de Recherche Appliquée en Linguistique In-formatique (RALI), Université de Montréal, Canada, 7-2003 (2003).

[12] Shanmughaneethi, V., and Swamynathan, S., "Detection of SQL Injection Attack in Web Applications using Web Services," IOSR Journal of Computer Engineering (IOSRJCE), vol. 1, no. 5, pp. 13-20, 2012.

[13] Shaul, J., and Ingram, A., Practical Oracle Security, Rockland: Syngress Publishing, Rockland, MA: Syngress Pub., c2007.

[14] Spett, K., "SQL Injection: Are your web applications vulnerable," Technical report, SPI Dynamics, Inc., 2005. available at URL http:// www.spidynamics.com/papers/sql injectionwhitepaper.pdf.

[15] Srivastava, S., and Tripathi, R., "Attacks Due to SQL Injection & Their Prevention Method for Web-Application" (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012,pp. 3615-3618.

[16] Williams, J., "OWASP Top 10 Project," OWASP Community, The Open Web Application Security Project , 2013. http://www.owasp.org.