ISSN:0975-9646

Shubbham Srivastava / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (1) , 2012, 3225-3228

A Survey On: Attacks due to SQL injection and their prevention method for web application

Shubham Srivastava

Department of Computer Science & Engineering
Teerthankar mahaveer, University
Moradabad (INDIA)

Abstract— In this paper we present a detailed review on various types of SQL injection attacks and prevention technique for web application. Here we are presenting our findings from deep survey on SQL injection attack. This paper is consist of following five section:[1] Introduction, [2]Types of Sql Injection, [3] Related work,

[4] Conclusion, And [5] References.

Keywords— SQL injection, database security, authentication, web system architecture

I. INTRODUCTION

The World Wide Web has been developed with very rapid progress in recent years. Web applications use the database at the backend for storing data and SQL (Structured Query Language) for insertion and retrieval of data. There are some malicious code that can be attach to the SQL called SQL Injection.

SQL injection attacks are possible because web application code is not secured during application development. One of the best ways to secure applications is by limiting access to those authorized to access the application.

SQL injection is a hacking method that is based on the security vulnerabilities of web application.

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 [9], [5]. 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) [10].

A. What is SQL Injection?

SQL Injection is one of the many web attack mechanisms used by hackers to steal data from organizations. It is perhaps one of the most common application layer attack techniques used today. It is the type of attack that takes advantage of improper coding of our web applications that allows hacker to inject SQL commands into say a login form to allow them to gain access to the data held within our database.

In essence, SQL Injection arises because the fields available for user input allow SQL statements to pass through and query the database directly.

SQL Injection is the hacking technique which attempts to pass SQL commands (statements) through a web application for execution by the backend database. If not sanitized properly, web applications may result in SQL Injection attacks that allow hackers to view information from the database and/or even wipe it out.

B. Impact of SQL Injection

Once an attacker realizes that a system is vulnerable to SQL Injection, he is able to inject SQL Query / Commands through an input form field. This is equivalent to handing the attacker our database and allowing him to execute any SQL command including DROP TABLE to the database.

An attacker may execute arbitrary SQL statements on the vulnerable system. This may compromise the integrity of our database and/or expose sensitive information. Depending on the back-end database in use, SQL injection vulnerabilities lead to varying levels of data/system access for the attacker.

C. SQL Injection in 3-Tier web system

Three –tier is a client- server architecture in which the user interface, functional process logic, Data storage and access are developed and maintained as independent modules , most often on separate plateforms .

Three-tier architecture has the following three tiers:

1) Presentation tier (front end)

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

2) Application tier (Middle tier)

The logic tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing.

3) Data tier (Backend)

This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

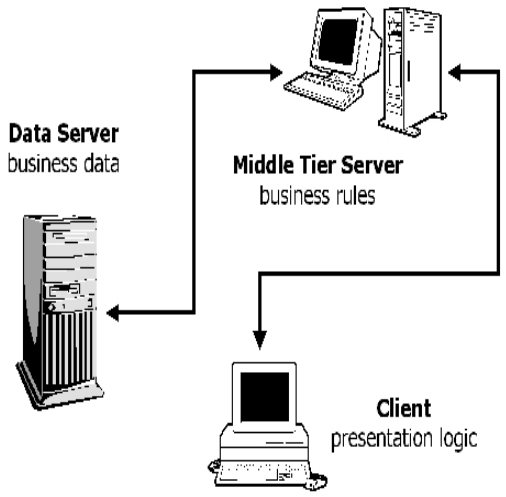

fig.1(a)

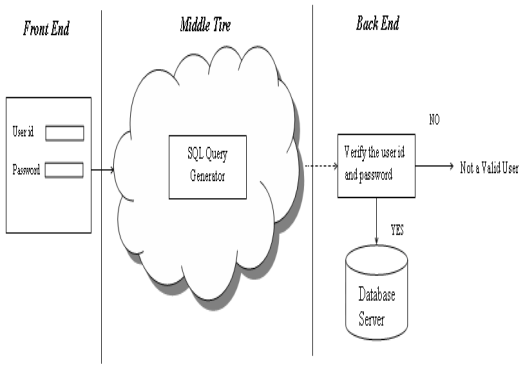

fig.1(b)

BASIC MODEL FOR WEB APPLICATION

Here three tier architecture for web-system is shown in the figure, In which front end is user interface from where user sends input SQL queries to database(back end) and store or reteieve data from database.

SQL injection attack can be detected by analyzing the SQL query before reaching to database i.e. back end.

II. TYPES OF SQLIA

Tautology attacks- The basic goal of this attack is to inject code into one or more conditional statements so that they always evaluate to true. Bypassing authentication page and fetching data is the most common example of this kind of attack. In this type of injection, the attacker exploits an inject-able field contained in the WHERE clause of a query. He transforms this conditional query into a tautology.and hence causes all the rows in the database table targeted by the query to be returned.

Logically incorrect query attacks: This type of attack lets an attacker gather important information about the type and structure of the back-end database in a Web application. The attack is considered to be a preliminary, information gathering step for subsequent attacks.

UNION Attacks: Here, an attacker exploits a vulnerable parameter to alter the data set returned by a given query. Using this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer.

Attackers do this by injecting a statement of the form:

UNION SELECT <rest of injected query>. Because the attacker is in complete control of the second/injected query, he can use that query to retrieve information from any desired table in the database. The result of this attack is that the database returns a dataset that is the union of the results of the original/first query and the results of the injected/second query.

Piggybacked Query : In this attack type, an attacker tries to inject additional queries along with the original query, which are said to "piggy-back" onto the original query. As a result, the database receives multiple SQL queries for execution. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. This type of attack can be extremely harmful. If successful, attackers can insert and execute virtually any type of SQL command, including stored procedures, into the additional queries and have them executed along with the original query.

Stored Procedure: Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes.

III. RELATED WORK

Many existing techniques, such as filtering, information-flow anal- ysis, penetration testing, and defensive coding, can detect and pre- vent a subset of the vulnerabilities that lead to SQLIAs. In this section, we list the most relevant techniques-

Ali et al.'s Scheme - [3] 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

William G.J.Halfond et al.'s Scheme- [2]- This approach works by combining static analysis and runtime monitoring. In its static part, technique uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, technique monitors the dynamically generated queries at runtime and checks them for compliance with the statically-generated model. Queries that violate the model represent potential SQLIAs and are thus pre- vented from executing on the database and reported.

SAFELI - 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 [12] 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 [13], 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 [14], 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 - [15] 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.

SQL-IDS Approach - Kemalis and Tzouramanis in [16] suggest using a novel specification-based methodology for the detection of exploitations ofSQL 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.

AMNESIA - In [17], Junjin proposes AMNESIA approach for tracing SQL input flow and generating attack input, JCrasher for generating test cases, and SQLInjection Gen 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 [18] 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.

SQLIA Prevention Using Stored Procedures - Stored procedures are subroutines in the database which the applications can make call to [19]. 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. [20] proposed a combination of static analysis and runtime monitoring to fortify the security of potential vulnerabilities.

Parse Tree Validation Approach - Buehrer et al. [21] 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 [11], 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.

A brief overview of various Scheme:

SNo.	*Author*	*Description*
1	Ali et al.'s	Adopts the hash value approach .The user name and password hash values are created and calculated at runtime
2	Halfond et al.'s	This approach works by combining static analysis and runtime monitoring. Technique uses program analysis to automatically build a model and monitors the dynamically generated queries at runtime and checks them for compliance with the statically-generated model.
4	Ruse et al.'s	The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically.
5	Buehrer et al.	Buehrer et al. adopt the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement.
6	Bisht et al.	This framework dynamically extracts the query structures from every SQL query location which are intended by the developer

Tabular representation of Detction and Prevention Scheme:

Sno.	*Scheme*	*Detction*	*Prevention*
1	AMNESIA	Yes	Yes
2	SQLrand	Yes	Yes
3	SQLdom	Yes	Yes
4	SQLGuard	Yes	No
5	SQLIPA	YES	No
6	CANDID	YES	No
7	SQL-IDS	YES	YES

IV. CONCLUSIONS

SQL injection attacks are one of the largest classes of security problems most existing technique either require developers to manually specify the interfaces to an application or, if automated, are often inadequate when applied to modern, complex web applications.

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.We have also presented a summary of various detction and prevention schemes.

V. REFERENCES

[1]- Indrani Balasundaram, E.Ramaraj "An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption(PSQLIA-HBE"(ISSN 1450-216X Vol.53 No.3 (2011),pp.359-368)

[2]-William G.J.Halfond and Alessandro Orso "AMNESIA:Analysis and Monitoring for Neutralizing SQL-Injection Attacks"

[3]- Shaukat Ali, Azhar Rauf, Huma Javed "SQLIPA:An authentication mechanism Against SQL Injection"

[4]- M.Mutuprasanna, Ke Wei,, Suuraj *Kothari'* Eliminating SQL Injection Attacks - A Transparent Defense Mechanism

[5]- William G.J.Halfond ,JeremyViegas, Alessandro Orso "A Classification of SQL injection Attacks And Countermeasures"

[6]- Romil Rawat , Chandrapal Singh Dangi,Jagdish Patil" Safe Guard Anomalies against SQL Injection Attacks" (IJCA(0975-8887)Volume 22-No.2,May2011)

[7]- Indrani Balasundaram, Dr.E.Ramaraj "An Approach to Detection of SQL Injection Attacks in DatabaseUsing Web Services"(IJCSNS ,VOL. 11 No.1,January 2011

[8]- Debasish Das,Utpal Sharma & D.K. Bhattacharyya "An Approach to Detect and Prevent SQL Injection Attack Based on Dynamic Query Matching"

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

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

[11] 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

[12] 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).

[13] 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)

[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 SQLinjection defense mechanisms," Proc. Of ICITST 2009, vol., no., pp.1-8, 9-12 Nov. 2009

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

[17] 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.

[18] 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.

[19] 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

[20] 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.

[21] 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.