Volume 2, Issue 7, July 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA

Mayank Namdev *, Fehreen Hasan, Gaurav Shrivastav
Department Of Computer Science and Engineering,
R.K.D.F. Institute of Science and Technology,
Bhopal (M.P.) India

ABSTRACT: —This paper gives an overview to the SQL Injection attacks (SQLIA) and methods to prevent them; we will discuss all the proposed models to block SQL Injections. We also describes the technique to prevent injections attacks occurring due to dynamic SQL statements in database stored procedures, which are often used in e-commerce applications. As we know that SQL injection attack can be easily prevented by applying more secure scheme in login phase. To address this problem, we studied and present here an overview of the different types of attacks with descriptions and examples of how attacks of that type could be performed and their detection & prevention schemes. This paper also contains strengths and weaknesses of various SQL injection attacks. At last we also proposed the scheme to handle the SQLIA and strong enough to prevent them.

Keywords: — Cybercrime, SQL injection, hash function, encryption algorithm.

INTRODUCTION

The use of internet increases day by day. As the number of computer users increases, the number reported cases of cybercrime increases. While, on the other side, the organization increases the use of office automation software & services, that helps them to maintain the confidential information with less efforts. Therefore, in this scenario it is not wrong to say that Information will be the single most important business asset today and achieving a high level of information security can be viewed as imperative in order to maintain a competitive edge. SQL Injection Attacks (SQLIA's) are one of the most severe threats to web application security. They are frequently employed by malicious users for a variety of reasons like financial fraud, theft of confidential data, website defacement, sabotage, etc. The number of SQLIA's reported in the past few years has been showing a steadily increasing trend and so is the scale of the attacks. It is, therefore, of paramount importance to prevent such types of attacks, and SQLIA prevention has become one of the most active topics of research in the industry and academia. There has been significant progress in the field and a number of models have been proposed and developed to counter SQLIA's, but none have been able to guarantee an absolute level of security in web applications, mainly due to the diversity and scope of SQLIA's. One common programming practice in today's times to avoid SQLIA's is to use database stored procedures instead of direct SQL statements to interact with underlying databases in a web application, since these are known to use parameterized queries and hence are not prone to the basic types of SLQIA's. However, there are vulnerabilities in this scheme too, most notably when dynamic SQL statements are used in the stored procedures, to fetch the database objects during runtime. Our work is cantered on this particular type of vulnerability in stored procedures and we develop a scheme for detection of SQLIA in scenarios where dynamic SQL statements are used. This paper is organised as follows: Section I show the introduction of web attack and how SQLIA is vulnerable cause of attack, section II show briefing about SQLIA, section III show types of SQLIA and in IV section show the various methods used for detecting and preventing SQLIA and in V section our proposed work and in last section conclusion is present.

WHAT is SQL INJECTION ATTACK?

SQL Injection is a type of web application security vulnerability in which an attacker is able to submit a database SQL command, which is executed by a web application, exposing the back-end database. SQL Injection attacks can occur when a web application utilizes user-supplied data without proper validation or encoding as part of a command or query. The specially crafted user data tricks the application into executing unintended commands or changing data. SQL Injection allows an attacker to create, read, update, alter, or delete data stored in the back-end database. In its most common form, SQL Injection allows attackers to access sensitive information such as social security numbers, credit card number or other financial data. According to Veracode's State of Software Security Report SQL Injection is one of the most prevalent types of web application security vulnerability.

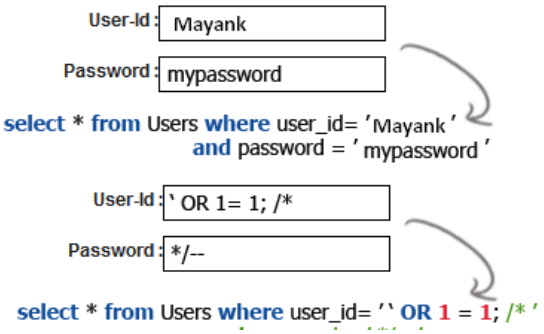

Figure 1: SQL Injection

Key Concepts of SQL Injection

- SQL injection is a software vulnerability that occurs when data entered by users is sent to the SQL interpreter as a part of an SQL query
- Attackers provide specially crafted input data to the SQL interpreter and trick the interpreter to execute unintended commands
- Attackers utilize this vulnerability by providing specially crafted input data to the SQL interpreter in such a manner that the interpreter is not able to distinguish between the intended commands and the attacker's specially crafted data. The interpreter is tricked into executing unintended commands
- SQL injection exploits security vulnerabilities at the database layer. By exploiting the SQL injection flaw, attackers can create, read, modify, or delete sensitive data

TYPES of ATTACK

There are different methods of attacks that depending on the goal of attacker are performed together or sequentially. For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query. Now the following classification of SQLIAs in accordance to [1][9] be presented.

Tautologies: This type of attack injects SQL tokens to the conditional query statement to be evaluated always true. This type of attack used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause. "SELECT * FROM employee WHERE userid = '112' and password ='aaa' OR '1 '='1 III as the tautology statement (1=1) has been added to the query statement so it is always true.

Illegal/Logically Incorrect Queries: when a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker makes a type mismatch error by injecting the following text into the pin input field:

1) Original URL: http://www.arch.polimi.itleventil?id nav=8864
2) SQL Injection: http://www.arch.polimLitieventil? id_nav=8864'
3) Error message showed: SELECT name FROM Employee WHERE id =8864\'

From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker can organize more strict attacks.

Union Query: By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application. Suppose for our examples that the query executed from the server is the following:
SELECT Name, Phone FROM Users WHERE Id=$id
By injecting the following Id value: $id= I UNION ALL SELECT credit Card Number, 1 FROM Credit Car Table We will have the following query:
SELECT Name, Phone FROM Users WHERE Id= 1 UNION ALL SELECT creditCardNumber, 1 FROM Credit Car Table
This will join the result of the original query with all the credit card users.
Piggy-backed Queries: In this type of attack, intruders exploit database by the query delimiter, such as ";", to append extra query to the original query. With a successful attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate. So attacker can inject any SQL command to the database. In the following example, attacker inject " 0; drop table user" into the pin input field instead of logical value. Then the application would produce the query:

SELECT info FROM users WHERE login='doe' AND pin=O; drop table users Because of ";" character, database accepts both queries and executes them. The second query is illegitimate and can drop users table from the database. It is noticeable that some databases do not need special separation character in multiple distinct queries, so for detecting this type of attack, scanning for a special character is not impressive solution.

Stored Procedure: Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. As stored procedure could be coded by programmer, so, this part is as inject able as web application forms. Depend on specific stored procedure on the database there are different ways to attack. In the following example, attacker exploits parameterized stored procedure.

CREATE PROCEDURE DBO .is Authenticated @user Name varchar2, @pass varchar2, @pin int
AS EXEC ("SELECT accounts FROM users WHERE login='" +@user Name+ If' and
pass='" +@password+ " ' and pin=" +@pin);

GO For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder

input "SHUTDOWN; - -" for username or password. Then the stored procedure generates the following query: SELECT accounts FROM users WHERE login= 'doe' AND pass=' '; SHUTDOWN; -- AND pin = after that, this type of attack works as piggy-back attack.

The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down. So, it is considerable that stored procedures are as vulnerable as web application code.

Inference: By this type of attack, intruders change the behaviour of a database or application. There are two well known attack techniques that are based on inference: blind injection and timing attacks. Blind Injection: Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field:
SELECT accounts FROM users WHERE login= 'doe' and 1 =0 -- AND pass = AND pin=O
SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=O

If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submits the first query and receives an error message because of "1 =0 ". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection.

Timing Attacks: A timing attack lets an attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. This technique uses an if-then statement for injecting queries. W AITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in the following query:

declare @ varchar(8000) select @ = db_nameO if (ascii(substring(@, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:5'

Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.

RELATED WORK

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

William G.J.Halfond et al.'s Scheme- [1]- proposed an approach that works by combining static analysis and runtime monitoring of database queries. In its static part, technique uses program analysis to automatically build a model of the legitimate queries that will be generated by the application. While in the dynamic part, the technique monitors the dynamically runtime generated queries and checks them for acceptability with the statically-generated model. A query that doesn't match with the model represent potential SQLIAs and are hence prevented from executing on the database and reported.

SAFELI – [2] This research deals with the Static Analysis Framework in order to detect SQL Injection Vulnerabilities. This framework aims to identifying the SQL Injection attacks during the compile-time. The two main advantages of this static analysis tool are: first, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. If we consider the White-box we found the Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. While on the other hand, the Hybrid-Constraint Solver implements the methods to 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 [3] proposed an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities. The authors implement their research work using four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. On the basis of 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 [4], 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.

Ali et al.'s Scheme - [5] 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

Roichman and Gudes's Scheme – [6] 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.

SQLIA Prevention Using Stored Procedures - Stored procedures are subroutines in the database which the applications can make call to [7]. 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 SQL Checker for input identification.

PROPOSED APPROACH

After studying the previous work by the researchers, we proposed a new scheme on SQLIA which is based on verification of information. This a new hybrid algorithm of prevention of SQL injection attacks. We combine the previous two approaches and then applying the hash code with encryption for more security purpose. By this mechanism we can easily prevent the SQLIA and safe the web application from any intrusion.

I. Hash function algorithm [10]

SQL Injection Protector for Authentication (SQLIPA) uses for preventing database against SQL injection. In the proposed approach there is a need for two extra columns in database. The first one is for the hash values of user name and other one for the hash values of password. The hash values are calculated for user name and password when a user account is created for the first time and stores it in the User table. Whenever user wants to login to database his/her identity is checked using user name and password and its hash values. These hash values are calculated at runtime using store procedure when user wants to login into the database.

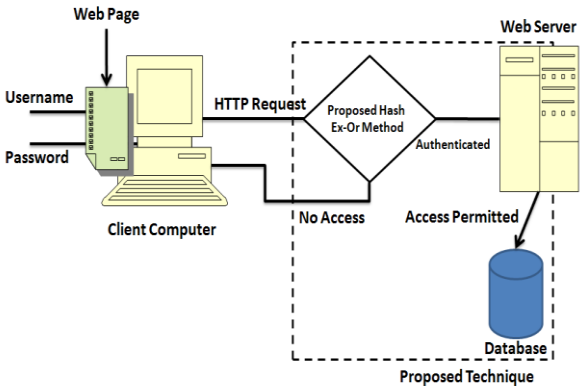

Figure 2 : System design for SQLIA Detection & Prevention

In the proposed technique first the hash values of user name and password are calculated at runtime and checked with stored hash values in the database table. During the authentication the original query will change into a new modified query with hash parameters.

Hence, if a user tries the injection to the query, and our proposed methodology is working with SQL query, it will automatically detect the injections as the potentially harmful content and rejects the values. Therefore, it cannot bypass the authentication process. The advantage of the proposed technique is that the hackers do not know about the hash values of user name and password. So, it is not possible for the hacker to bypass the authentication process through the general SQL injection techniques. The SQL injection attacks can only be done on codes which are entered through user entry form but the hash values are calculated at run time at backend before creating SELECT query to the underlying database therefore the hacker cannot calculate the hash values as it dynamic at Runtime.

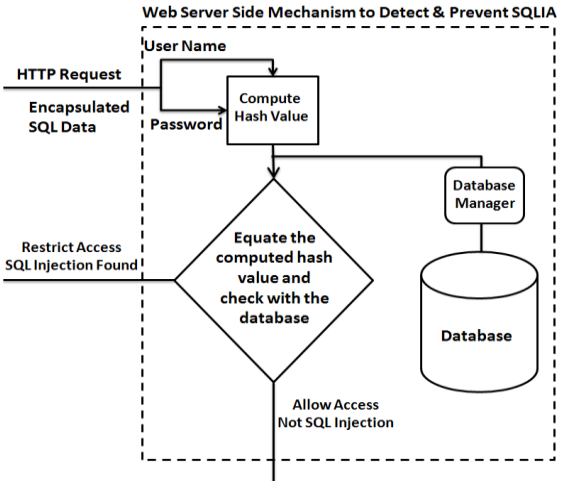

Figure 3 : Proposed Hash Scheme for Detecting SQLIA & Prevent Them

In addition to simplify the proposed technique is that, we can apply the Ex-or function to the above decided two fields, so that instead of having two additional fields in the database. We kept only field in the database. Using this modification, we can achieve the security in an efficient way.

Consider a scenario, where a user is authenticated by the secure login mechanism and login the system. Now, if this authenticated user make any intrusion into the system. How can we defend it?

Hence to prevent after-login attacks we have taken the help of data encryption. As we saw in the previous section that it is possible to collect the highly confidential information by using **union** operator we find out an alternative way to store all these confidential information's. In our database, instead of directly storing all confidential information's, we store them in encrypted format with a secure and confidential encryption-key. Now even if the dispatcher user can able to see the atm_pin by using union operation, he cannot able to decrypt it without knowing the exact encryption method and encryption- key. So he cannot able to do any damage with that encrypted atm_pin.

CONCLUSION

During the study of several researches based on the SQL injection Prevention & Attacks, we found that

certain cases are there, where these approaches are not found to be effective. Hence, these approaches become un-useful cannot able to detect the injections to prevent them. In addition, the attackers can access the database directly in an illegal way. Therefore, we are at the proposed a new approach that is completely based on the hash method of using the SQL queries in the web-based environment, which is much secure and provide the prevention from the attackers SQL. But, our proposed strategy requires the alterations in the design of existing schema database and a new guideline for the database user before writing any new database. Through these guidelines, we expect the effective outcomes in SQL injections Preventions. In addition, many researchers can contribute to this SQL Injections Attacks Preventions Mechanisms without any extra effort to the database like alteration in schema design, adding new tables, adding new attributes etc.

REFERNCES

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

[2]. X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A StaticAnalysis ramework for Detecting SQL Injection Vulnerabilities, OMPSAC 2007, pp.87-96, 24-27 July 2007.

[3]. 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).

[4]. 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)

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

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

[7]. 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

[8]. G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), 2005. [II] F.Monticelli., PhD SQLPrevent thesis. University of British Columbia (UBC) Vancouver, Canada.2008.

[9]. C. Gould, Z. Su, and P. Devanbu. JOBC Checker:A Static Analysis Tool for SQLlJOBC Applications. In Proceedings of the 26th International Conference on Software Engineering (lCSE 04) Formal Demos, pp 697-698, 2004.

[10]. Shaukat Ali, Azhar Rauf and Huma Javed , SQLIPA: An Authentication Mechanism Against SQL Injection. European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4 (2009), pp 604-611.

