A TECHNIQUE FOR EXPLOITING DATABASE VULNERABILITIES OF WEB APPLICATION USING DETECTION TOOLS

Kirti Kakde[1]

Abstract

SQL injection attack is a form of attack that takes advantage of applications that generate SQL queries using user-supplied data without first checking or pre-processing it to verify that it is valid. The objective is to deceive the database system into running malicious code that will reveal sensitive information or otherwise compromise the server. By modifying the expected Web application parameters, an attacker can submit SQL queries and pass commands directly to the database. Although deployment of defensive coding or OS hardening energies security but they are not enough to stop SQLIAs. So this paper focuses on some tools and methodologies which can detect or prevent these attacks.

Keyword: SQL Injection Attacks, prevention, web application parameters, OS hardening.

Introduction

Web applications are often vulnerable to attacks, which can give attackers easy access to the application's underlying database. SQL injection attack occurs when a malicious user, through specifically crafted input, causes a web application to generate and send a query that functions differently than the programmer intended.

SQL Injection Attacks (SQLIAs) have known as one of the most common threats to the security of database-driven applications. So there is not enough assurance for confidentiality and integrity of this information.

SQLIA is a class of code injection attacks that take advantage of lack of user input validation. In fact, attackers can shape their illegitimate input as parts of final query string which operate by databases. Financial web applications or secret information systems could be the victims of this vulnerability because attackers by abusing this vulnerability can threat their authority, integrity and confidentiality but also attackers continue to bring some new ways to bypass these controls.

SQL injection is a type of attack which the attacker adds Structured Query Language code to input box of a web form to gain access or make changes to data. SQL injection vulnerability allows an attacker to flow commands directly to a web application's underlying database and destroy functionality or confidentiality.

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

[1] Asst. Professor , MCA Department, Y.M.T College of Management, Kharghar, kirtik@ymtcollegeofmanagement.org

utilizes usersupplied data without proper validation or encoding as part of a command or query.

User Id: [ ]

Password: [ ]

Select * from employee where user_id=’Mayank’ and password=’mypassword’

User Id: [‘ OR 1=1;/*]

Password: [*/--]

Fig:1 SQL Injections

Sql Injection Attack Process:

SQLIA is a hacking technique which the attacker adds SQL statements through a web application's input fields or hidden parameters to access to resources. Lack of input validation in web applications causes hacker to be successful. For the following examples we will assume that a web application receives a HTTP request from a client as input and generates a SQL statement as output for the back end database server. For example an administrator will be authenticated after typing: employee id=112 and password=admin. Figure1 describes a login by a malicious user exploiting SQL

Injection vulnerability: Basically it is structured in three phases: 1. an attacker sends the malicious HTTP request to the web application 2. creates the SQL statement 3. submits the SQL statement to the back end database

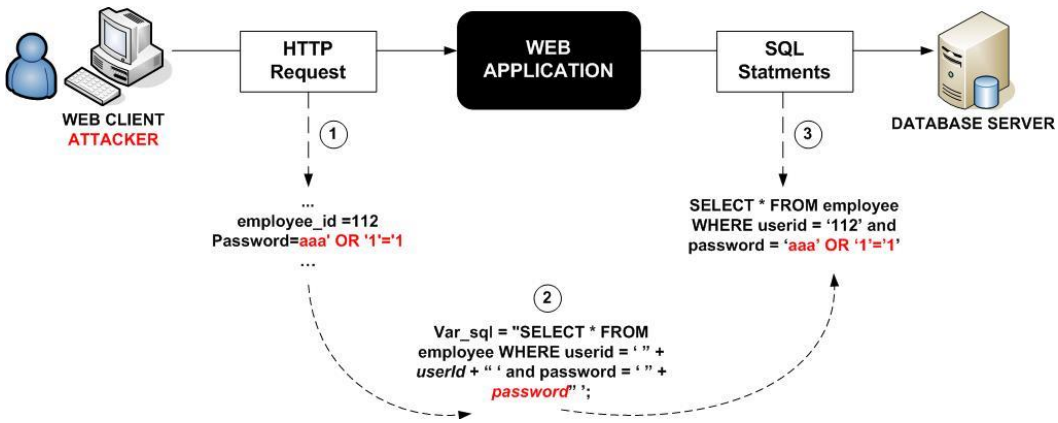

Figure 2: Example of a SQL Injection Attack

The above SQL statement is always true because of the Boolean tautology we appended (OR 1=1) so, we will access to the web application as an administrator without knowing the right password.

Main cause of SQL injection

Web application vulnerabilities are the main causes of any kind of attack. In this section, vulnerabilities that might exist naturally in web applications and can be exploited by SQL injection attacks will be presented:

1.Invalidated input: This is almost the most common vulnerability on performing a SQLIA. There are some parameters in web application, are used in SQL queries. If there is no checking for them so can be abused in SQL injection attacks. These parameters may contain SQL keywords, e.g. INSERT, UPDATE or SQL control characters such as quotation marks and semicolons.

2.Generous privileges: Normally in database the privileges are defined as the rules to state which database subject has access to which object and what operation are associated with user to be allowed to perform on the objects. Typical privileges include allowing execution of actions, e.g. SELECT, INSERT, UPDATE, DELETE, DROP, on certain objects. Web applications open database connections using the specific account for accessing the database. An attacker who bypasses authentication gains privileges equal to the accounts. The number of available attack methods and affected objects increases when more privileges are given to the account. The worst case happen If an account can connect to system that is associated with the system administrator because normally has all privileges.

3.Uncontrolled variable size: If variables allow storage of data be larger than expected consequently allow attackers to enter modified or faked SQL statements. Scripts that do not control variable length may even open the way for attacks, such as buffer overflow.

4.Error message: Error messages that are generated by the back-end database or other server-side programs may be returned to the client-side and presented in the web browser. These messages are not only useful during development for debugging purposes but also increase the risks to the application. Attackers can analyze these messages to gather information about database or script structure in order to construct their attack.

5.Dynamic SQL: SQL queries dynamically built by scripts or programs into a query string. Typically, one or more scripts and programs contribute and finally by combining user input such as name and password, make the WHERE clauses of the query statement. The problem is that query building components can also receive SQL keywords and control characters. It means attacker can make a completely different query than what was intended.

6.Client-side only control: If input validation is implemented in client-side scripts only, then security functions of those scripts can be overridden using cross-site scripting. Therefore, attackers can bypass input validation and send invalidated input to the server-side.

7.Stored procedures: They are statements which are stored in DBs. The main problem with using these procedures is that an attacker may be able to execute them and damage database as well as the operating system and even other network components. Usually attackers know system stored procedures that come with different and almost easily can execute them.

SQL INJECTION ATTACK TYPES:

For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query.

1.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'" As the tautology statement (1=1) has been added to the query statement so it is always true.

2.llegal/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: Error message showed:

SELECT name FROM employee WHERE userid =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.

3.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=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable which will join the result of the original query with all the credit card users.

4.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=0; 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.

5.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.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC("SELECT accounts FROM users WHERE login='" +@userName+ "' 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 wellknown attack techniques that are based on inference: blindinjection and timing attacks.

6.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=0 SELECT accounts FROM users WHERE login='doe' and 1=1 -- AND pass= AND pin=0 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 submit 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

Prevention of SQL Injection Attacks

1. **SQL Block:** SQL Block is an open database connectivity (ODBC) driver that acts as an SQL injection protection feature. It blocks the execution and sends an alert to administrator, in case of any client application attempt to execute any disallowed SQL statements. It works as an ordinary ODBC data source and monitor every SQL statements being executed.
2. **Proposed Technique:** In this paper, the research work proposes the technique of Preventing SQL Injection Attack in Web Application. In a Login Table, two columns are created

by DBA. One for username and other is used for password. The methodology requires two more columns. One for the hash value of the username and other is used for hash value of password. The hash values of username and password are calculated and stored in Login Table when the user's account is first time created with the web application. Whenever user wants to login to database his/her identity is checked using username, password and hash values. These hash values are calculated at runtime using stored procedure when user wants to login into the database. If only username and password are used for authentication, and the attacker enters Username = „ OR 1=1 – – and Password = pwd; The query becomes like this SQL_Server = Select * from Login where Username = „' OR 1=1 – –" and Password = „pwd"; Figure 2.Query without using hash values But, using PSIAW approach, the query for authentication will become like this SQL_Server = Select * from Login where Hash_value_user = „hash_user" and Hash_value_pwd =„hash_pwd" and Username = „' OR 1=1 – –' and Password = „pwd"; Figure 3.Query using hash values .Thus using hash values for password and username, the hacker cannot bypass authentication as attacker does not know the hash values of username and password and hence access the database of the web application. Thus, web application is secured. The error messages generated by application should not show that any hash values are calculated at the back end and it"s getting matched with the entered one. This prevents the attacker from accessing database as he is not aware of any hash values used and does not know the hash values of username and password as hash values are calculated at runtime. Only two text boxes are provided at the interface for entering username and password, he will not be able to enter hash values from anywhere. Hence, the attacker will not be able to attack database and web application is secured. When user changes password, hash value of old password supplied as well as new password is calculated. Hash value of old password is matched with the stored hash value and new hash value is stored with the new password in the Login Table. Every time database is accessed, hash value of supplied parameter is calculated and matched with the stored one. Whenever it does not match it simple generates the message, username and password do not match. So the attacker does not get to know about the hash values concept.

Architecture

Architecture of Preventing SQL Injection Attack in Web Application (PSIAW) technique consists of three components: User Login Interface, SQL Query Component and User Account Table.

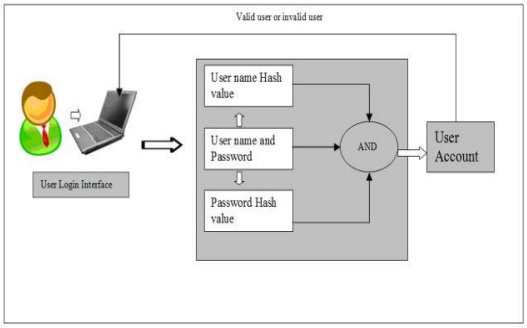

Fig.3. Architecture of Preventing SQL Injection Attack in Web Application.[2]

Here, user login interface is just the user entry form containing two columns for username and password. Main component of PSIAW is SQL Query Component. SQL

Query component is the component where hash value of username and password is calculated. These values are then combined with username and password using AND operator. Every time the user enters username and password, their hash values are calculated. The query formed is then sent to database. Subcomponents of SQL Query component are username hash value, username and password and password hash value. User account table is the component where username, password, hash values for user and passwords are stored here.

Conclusion

Most web applications employ a middleware technology designed to request from a relational database in SQL parlance. SQL Injection is a common technique hackers employ to attack these web based applications. These attacks reshape the SQL queries, thus altering the behavior of the program for the benefit of the hacker. In this research work a technique for protecting authentication against SQL Injection is presented. This technique presents the need for adding two additional columns in login table. These columns store hash values of username and password. When the user gets itself registered with a web application, it selects its username and password. At the same time, hash value of username and password is computed at the coding side and stored in the login table with username and password. When user logs in to the web application, hash value of username and password are matched at the backend and user is allowed to access the data. If SQL Injection attack string is entered for logging into the database, its hash value does not match with the hash values stored in the table and hence attacker cannot access the database.

References

[1] Atefeh Tajpour , Suhaimi Ibrahim, Mohammad Sharifi IJCSI International Journal of Computer Science www.IJCSI.org

[2] Prasant Singh Yadav, 2 Dr pankajYadav, 3Dr. K.P.Yadav “A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications”.

[3] Cyber Security by Nina Godbole

[4] SQL Injection analysis, Detection and Prevention by Jagdish Halde, San Jose State University