SQL INJECTION Attacks in Web Application

Mihir Gandhi, JwalantBaria

Abstract: Databases are the first target of the attackers in Web Application Once your ID and PASSWORD are out there may be several misuse of it. These paper discuss about Advance SQL Injection (ASQLIA) first of all it identifies which type of attacks according to that prevention measures are suggested .Some New features are added to it Web Crawling ,Web Services and Advance SQL Injection (ASQLA)which will emphases more Security of Web Application. In short enhancing database security with the aspect of web developer is main aim of my paper.

Keywords— Cybercrime, hash function, encryption algorithm.SQL Injection, Tautology, SQLIA, Blind injection, piggy backing, PSIAW.

I. INTRODUCTION

Since web applications have become one of the most important communication channels between service providers and clients, more script kiddies and sophisticated hackers target victims either for fun, commercial reasons or personal gain. The increasing frequency and complexity of web based attacks has raised awareness of web application administrators of the need to effectively protect their web applications. The OWASP 2010 report places Injection Attacks, including SQLIAs, as the most likely and damaging . SQLIAs are caused by attackers inserting a malicious SQL query into the web application to manipulate data, or even to gain access to the back-end database[1]. 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.

II. 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.

Manuscript received on January, 2013.

Mihir Gandhi, ME Student of Information Technology Parul Institute of Engg. & Tech. At.Baroda, Gujrat, India

JwalantBaria, ME Computer Science and Engineering Parul Institute of Engg. & Tech. At.Baroda, Gujrat, India.

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.[2].

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

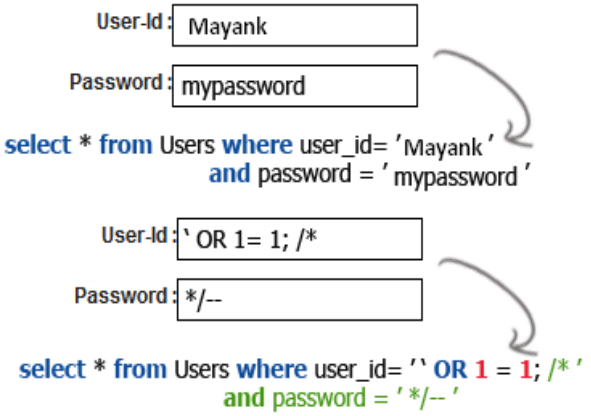


Figure : SQL Injection

- Attackers provide specially crafted input data to the SQL interpreter and trick the interpreter to execute unintended commands [2]
- 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 [2].

Types Of Sql Injection 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 be presented[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 tothequery statement so it is always true.

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 producesyntax 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:[3]

1) Original URL:http://www.arch.polimi.it/eventi/?id_nav=886
2) SQL Injection: http://www.arch.polimi.it/eventi/?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[3].

Union Query*:* By this technique, attackers join injected query to the safe query by the word UNION and then canget 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.

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' ANDpin=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.

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[3].

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[2].

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'

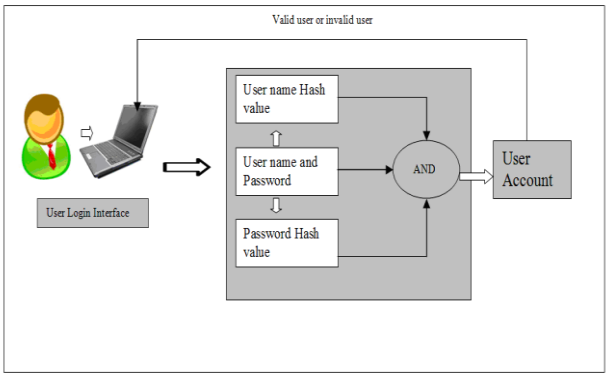
Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.[2]

Proposed Technique: Our research work proposes the technique, **Preventing SQL Injection Attack in Web Application (PSIAW)[1]**. In a Login Table, two columns are created by DBA. One for username and other is used for password. Our 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 5.1 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 5.2 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[1].

III. 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.[1]



Architecture of Preventing SQL Injection Attack in Web Application.

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[1].

IV. 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 behaviour of the program for the benefit of the hacker. In our research work, we have presented a technique for protecting authentication against SQL Injection. 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 can not access the database.[1]

Future Work: This technique can only protect authentication mechanism. Rest of the SQL Injection techniques can't be prevented using this technique. So, in future, we will try to improve the technique by making it efficient for other types of SQL Injection Attacks also. Then, this technique will be able to prevent SQL Injection Attack completely.

V. REFERENCES

[1] By1Prasant Singh Yadav, 2 Dr pankajYadav, 3Dr. K.P.Yadav "A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications",IJRREST: International Journal of Research Review in Engineering Science and Technology ,Volume-1 Issue-1, June 2012.

[2] By MayankNamdev *, FehreenHasan, GauravShrivastav "Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA"Volume 2, Issue 7, July 2012.

[3] By AtefehTajpour ,Suhaimi Ibrahim, Mohammad SharifiWeb Application Security by SQL Injection DetectionTools.IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012