International Journal on Recent and Innovation Trends in Computing and Communication ISSN 2321 – 8169
Volume: 1 Issue: 4 293 – 296

SQL INJECTION ATTACKS AND PREVENTION TECHNIQUES

Sampada Gadgil[1], Sanoop Pillai[2], Sushant Poojary[3]
[1]*Asst.professor, Information Technology,* [2,3]*Student,Information Technology*
SIES GST Graduate School of Technology
Nerul ,Navi Mumbai
sampada_gadgil@rediffmail.com,pillaisanoop@gmail.com,sushant.poojary123@gmail.com

Abstract: SQL injection attacks are a serious security threat to Web applications. They allow attackers to obtain unrestricted access to the databases underlying the applications and to the potentially sensitive information these database contain. Various researchers have proposed various methods to address the SQL injection problem. To address this problem, we present an extensive review of the various types of SQL injection attacks. For each type of attack, we provide descriptions and examples of how attacks of that type could be performed. We also present a methodology to prevent the SQL injection attacks.

Keywords: SQL Injection, Attack Intent, Blind Injection, Prevention, Attack Intent, Prepared Statement, Stored procedure

*****

I. INTRODUCTION

A common break-in strategy is to try to access sensitive information from a database Although current database systems have little vulnerability, the Computer Security applying this query to the desired database. Such an approach to gaining access to private be over four million dollars. Additionally, recent research by the "Imperva Application .By first generating a query that will cause the database parser to malfunction, followed by from the internet, dealing with SQL injection has become more important than ever.

In recent years, widespread adoption of the internet has resulted in to rapid advancement in information technologies. The internet is used by the general population for the information, in a way that allows the information owners quick access while blocking break-in attempts from unauthorized users. Institute discovered that every year about 50% of databases experience at least one purposes such as financial transactions, educational endeavours, and countless other activities. The use of the internet for accomplishing important tasks, such as transferring a balance from a bank account, always comes with a security risk. Today's web sites strive to keep their users' data confidential and after years of doing secure business online, these companies have become experts in information security. The database systems behind these secure websites store non-critical data along with sensitive security breach. The loss of revenue associated with such breaches has been estimated to be over four million dollars. Additionally, recent research by the "Imperva Application Defence Centre" concluded that at least 92% of web applications are susceptible to "malicious attack" (Ke Wei, M. Mprasanna, Suraj Kothari, 2007).

Simply put, the end goals of a SQL injection attack is to gain private (maybe confidential) data, perform a create or change on data which is not meant to be changed, or altogether purge the data, data object or dataset. Just a couple of months back, i.e. March 27th 2011 to be precise, Romanian Hackers by the name of "Tin Kode" and "Ne0h" attacked MySQL.com and Sun.com. They did this with a SQL injection attack, to gather table names, col names and email addresses stored in one of the tables

II. WORKING OF SQL INJECTION

SQL injection can be used using variousmethods. In this tutorial we will explain to the basic concepts behind the SQL injection. Suppose you are on a shopping site and you have selected of showing all the accessories that costless then and its URL is like

http://www.shoppingsite.com/products.php? val=100

To test this website for SQL injection try appending your SQL injection commands in the Val parameter 'OR '1'='1
http://www.shoppingsite.com/products.php?
val=100'OR'1'='1

If the above injection works and shows the list of all the accessories then the website is vulnerable type of SQL injection. This means that at the backend the script executed as shown:

SELECT * FROM Products WHERE OR '1'='1' ORDER BY Product description

As the condition 1=1 so this will give you list of all the products.

• How this SQL injection Attack is launched. Suppose a website uses the following logging into admin panel

http://www.website.com/cms/login.php?username=saini
&password=go

Now if the above website is vulnerableinjection as mentioned in the above example then by entering any username and password in the can login

http://www.website.com/ms/login.php?username=dnt&password=dnt'OR'1'='1

So you will just login without valid username and password to the admin panel of a website.

III. ATTACK INTENT

Identifying injectable parameters:
The attacker wants to probe a Web application to discover which parameters and user input fields are vulnerable to SQL injection Attacks

Performing database finger-printing:
The attacker wants to discover the type and version of database that a Web application is using. Certain types of databases respond differently to different queries and attacks, and this information can be used to "fingerprint" the database. Knowing the type andversion of the database used by a Web application allows an attacker to craft database specific attacks.

Determining database schema:
To correctly extract data from a database, the attacker often needs to know database schema information, such as table names, column names, and column data types. Attacks with this intent are created to collect or infer this kind of information. These types of attacks employ techniques that will extract data values from the database

IV. ATTACKS

4.1 Tautologies

Tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages of this technique are to bypass authentication pages and extract data. If the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

Example: In this example attack, an attacker submits " ' or 1=1 - -"
The Query for Login mode is:

SELECT * FROM user info WHERE
loginID='' or 1=1 - - AND pass1=''

The code injected in the conditional (OR 1=1)transforms the entire WHERE clause into a tautology the query evaluates to true for each row in the table and returns all of them. In our example, the returned set evaluates to a not null value, which causes the application to conclude that the user authentication was successful. Therefore, the application would invoke method user_main.aspx and to access the application.

4.2 Union Query

In union-query attacks , Attackers do this by injecting a statement of the form: UNION

SELECT <rest of injected query> because the attackers completely control the seond/injected query they can use that query to retrieve information from a specified table. The result of this attack is that the database returnsa dataset that is the union of the results of the original first query and the results of the injected second query.

Example: An attacker could inject the text "' UNION SELECT pass1 from user_info where LoginID='secret - -" into the login field, which produces the following query:

SELECT pass1 FROM user_info WHERE loginID=''

UNION SELECT pass1 from user_info where
LoginID='secret' -- AND pass1=''

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "user_info" table. In this case, the database would return column "pass1" for account "secret". The database takes the results of these two queries, unions them, and returns them to the application.

In many applications, the effect of this operation is that the value for "pass1" is displayed along with the account information.

4.3 Blind Injection

Web applications commonly use SQL queries with client-supplied input in the WHERE clause to retrieve data from a database. By adding additional conditions to the SQL statement and evaluating the web application's output, you can determine whether or not the application is vulnerable to SQL injection.

For instance, many companies allow Internet access to archives of their press releases.A URL for accessing the company's fifth press release might look like this:

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5

The SQL statement the web application would use to retrieve the press release might look like this (client-supplied input is underlined):

SELECT title, description, releaseDate, body FROM pressReleases WHERE pressReleaseID = 5

The database server responds by returning the data for the fifth press release. The web application will then format the press release data into an HTML page and send the response to the client.

To determine if the application is vulnerable to SQL injection, try injecting an extra true condition into the WHERE clause. For example, if you request this URL . . .

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND 1=1
. . . and if the database server executes the following query . . .

SELECT title, description, releaseDate, body FROM pressReleases WHERE pressReleaseID = 5 AND 1=1

. . . and if this query also returns the same press release, then the application is susceptible to SQL injection

Suppose you have a Web-based application which stores usernames alongside other session information. Given a session identifier such as a cookie you want to retrieve the current username and then use it in turn to retrieve some user information. You might therefore have code for an "Update User Profile" screen somewhat similar to the following:

execute immediate 'SELECT username FROM sessiontable WHERE session='"||sessionid||"' into username;

execute immediate 'SELECT ssn FROM users WHERE username='"||username||"' into ssn;

This will be injectable if the attacker had earlier on the "Create Account" screen created a username such as: XXX' OR username='JANE

Which creates the query:
SELECT ssn FROM users WHERE username='XXX' OR username='JANE'

If the user XXX does not exist, the attacker has successfully retrieved Jane's social security number.

The attacker can create malicious database objects such as a function called as part of an API, or a maliciously named table by using double quotation marks to introduce dangerous constructs.

For example, an attacker can create a table using a table name such as "tab') or 1=1--", which can be exploited later in a second order SQL injection attack.

V. PREVENTION TECHNIQUES

To prevent SQL injection, we are primarily concerned with the double quote, single quote, and backslash characters. Without encoding, the double and single quotes will be interpreted as string delimiters and backslashes can be used to subvert any encoding that only escapes string delimiters. The ability to inject string delimiters into a SQL statement is one of the primary ways of executing a SQL injection attack.

5.1 Prepared Statement

The Java API's PreparedStatement interface is superior to the Statement interface because it provides methods that take care of the encoding for you. By properly composing the query (with question marks in place of the input) and then using "setString(...)" or other methods, the encoding is taken care of for you.However, if you compose the PreparedStatement by concatenating strings that might contain user input, you bypass the encoding and leave your software open to attack.

The following is an example of passing two parameters "loginID" and "password" safely to a query:

```
PreparedStatement pstmt=conn.prepareStatement
("SELECT * FROM users WHERE loginID=? AND password=?");
pstmt.setString(1, ""+loginID);
pstmt.setString(2, ""+password);
resultset = pstmt.executeQuery();
```

5.2 Stored Procedures

SQL Injection is caused chiefly by the use of dynamic SQL queries. There is, however, nothing to stop the stored procedure being composed of dynamic SQL statements. Although only typed parameters will be accepted by a call to a stored procedure, there is no saying how the parameters will be used once inside the procedure. Therefore, it is important to ensure that the stored procedures do not themselves undo the benefits that accrue from their use. In Java, The CallableStatement interface in java.sql is employed to call stored procedures. The following is an example corresponding to the one presented above, this time using the java.sql.CallableStatement interface:

```
CallableStatement cstmt = conn.prepareCall("{CALL check_user(?,?,?)}");
cstmt.setString(1, ""+loginID);
cstmt.setString(2, ""+password);

cstmt.registerOutParameter(3, java.sql.Types.TINYINT);
cstmt.executeQuery();
```

In .NET, the System.Data.SqlClient namespace provides means for using parameterized
queries and stored procedures.

5.3 Patch your SQL server regularly

Before we get into the coding part of the advice how to prevent an SQL injection, we need to start with the fundamental issues. SQLinjections might be a frequent programming error but they aren't the only way for a hacker to break into. If your underlying software – i.e. the database

and the operating system have vulnerabilities, then your efforts to secure your code become obsolete. This is why you should always patch your system, especially your SQL server.

5.4 Use the principle of least privilege

The principle of least privilege is a security cornerstone and it applies to SQL injections as well. For instance, when you grant a user access only to the tables he or she needs rather to the whole database; this drastically reduces the damage potential.

5.5 Disable shells

Many databases offer shell access which essentially is what an attacker needs. This is why you need to close this door. Consult your DB's documentation about how to disable shell access for your particular database.

5.6 Test your code

Finally, the last step to ensure your code is SQL injections proofed is to test it. There are automated tools you can use to do this and one of the most universal is the SQL Inject Me Firefox extension. This tool has many options and many tests the best is if you have the time to run all of them.

VI. CONCLUSION

In this paper, we have presented a survey of current techniques of SQL injection as well as a solution methodology for preventing the attacks. To perform this evaluation, we first identified the various types of SQL Injection attacks .We also studied the different mechanisms through which SQL Injection Attacks can be introduced into an application and identified the techniques that are able to handle the mechanisms. Many of the techniques have problems handling attacks that take advantage of poorly coded stored procedures and SQL queries cannot handle attacks. This difference could be explained by the fact that prevention-focused techniques try to incorporate defensive coding best practices into their attack prevention mechanisms.

VII. FUTURE WORK

Future work should focus on evaluating the techniques precision and effectiveness in practice.Empirical evaluations will be performed which allow comparing the performance of the different technique swhen they are subjected to real-world attacks and legitimate inputs. As well as precision, accuracy and add on will be incorporated for the prevention of much complex attacks such as Linked Server, Internal Network Attack etc.

VIII. ACKNOWLEDGEMENT

We would like to thank our college SIES Graduate school of Technology for their constant encouragement. We would also like to thank the faculties for their guidance. The Library staff for providing us with the resources that they had regarding our topic.

IX. REFERENCES

[1] MeiJunjin: "An approach for SQL injection vulnerability detection". 2009 Sixth International Conference on Information Technology: New Generations.

[2] Joao Antunes,Nuno Neves,Miguel Correia, Paulo Verissimo and Rui Neves has suggested the attack injection methodology in their paper named "Using Attack Injection to Discover New Vulnerabilities"

[3] http://www.sqlsecurity.com/

[4] http://www.developerdrive.com/2011/10/how-to-prevent-a-sql-injection-attack/

[5] SQL Injection Cheat Sheet http://ferruh. mavituna.com/sql-injection-cheatsheetoku/

[6] Basics And Working of SQL Injection Attacks http://www.pctipstricks.net/hacking/basicworking-sql-injection/