



INJECTION and PREVENTION of SQL INJECTION ATTACKS on WEB APPLICATIONS

¹Bharti Nagpal, ²Naresh Chauhan, ³Nanhay Singh

^{1,3}Computer Engg. AIAC&R, Indraprastha University Delhi, India

²Computer Engg. YMCAUST, YMCA University of Science & Technology, Faridabad, India

Abstract: With the expansion of internet, the number of users is increasing at an unimaginable rate. The same could be said about the web applications. A large number of web applications are user driven and requires some sort of storage of data in the SQL database. The security of these databases has emerged as one of the primary concern for both the users and the developers. For the web applications using SQL database as their back-end server, we are going to present some of the vulnerabilities and methods to eliminate them by using PHP and HTML code.

Keyword: SQL Injection Attack, Web Applications, SQL Injection Vulnerabilities, Hash, SQL query.

I. INTRODUCTION

With the advancement in the technology, the need to provide large number of online services are growing. The internet is buzzing with user and huge number of online applications with the need to store and retrieve information securely. With the increase in user driven applications like social networking, gaming, etc. the need to secure the SQL web server is the most important aspect to provide uninterrupted service. But the vulnerability of SQL server has been exposed and exploited by the hackers. A large number of attacks on the internet try to exploit this weakness.

SQL Injection attack possess serious threat to the confidentiality of the data stored at different online platforms like banks, e-business, social network, etc. SQL Attack does not damage much to the operating system or the application, but the compromise and the extraction of the information is sure a threat to large number of organizations.

This paper proposes some simple tweaking at the developer level to help mitigate or to prevent SQL injection attacks.

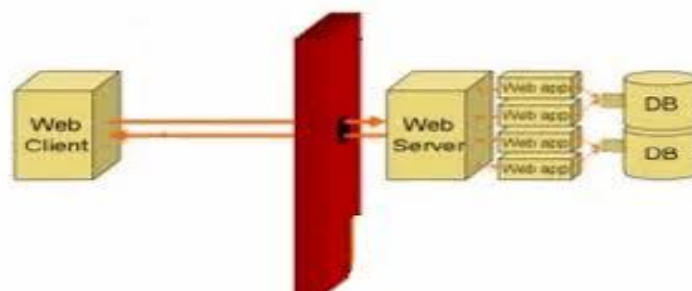
II. WEB APPLICATION ARCHITECTURE

A typical web based application is a program in action on a web browser. A typical web application involves three important aspects.

- 1) The first aspect of the web application is the end page that a user gets to input data or to see data or some output when he performs some action. Typically, this is the Graphical User interface which is presented in the web browser and this GUI is present in HTML, Javascript, Flash, etc.
- 2) The second aspect is the intermediate layer which separates our application from the SQL server or the database layer. All the data is stored in the database, whenever the user asks for some data, the intermediate layer relay the message and gets the asked information and send it to the presentation layer. The intermediate layer is coded in PHP, JSP, ASP, etc.
- 3) All the processed input is stored and messaged at the third layer. The database layer is responsible for the authentication of the user and denial of hacker from the system.

Web application architecture[1] can be shown in fig.1.

Figure 1 Web Application Architecture



III. IMPLEMENTATION

There are variety of ways by which SQL injection and its prevention can be implemented in PHP web applications. Some of the famous methods are mentioned below.

(i) Injection based on 1=1 query.

The special thing about the 1=1 input in a SQL query is that it is always true and will provide the complete database if the command gets executed. For example:

If we enter username = 'simon'

password = ' or 1=1 ; --

The query that goes to MySQL will look like this:

```
SELECT fname, lname, address FROM member WHERE username = 'simon' AND
password = '' or 1=1 ; -- '
```

The above query tells MySQL to find all the rows with a username equal to "simon" and password equal to an empty string OR "1=1".

This can be represented logically as shown below:

(username = "simon" and password = "") || (1 = 1)

Now 1=1 is always going to be true, so this is equal to:

(false) || (true)

which means that ALL the records in the table will be returned. Our login processor will log me on with someone else's credentials.

(a) Prevention for Injection based on 1=1 query.

The solution of the above mentioned problem can be done by using parametrized query. The SQL engine checks each parameter to ensure that it is correct for its column and is treated literally. It can be shown below in fig. 2.

Figure 2 An example of parametrized query

```
$wrd = $dbh->prepare("INSERT INTO member (firstname,lastname,address)
VALUES (:fnam, :lnam, :add)");
$wrd->bindParam(':fnam', $txtNam);
$wrd->bindParam(':lnam', $txtAdd);
$wrd->bindParam(':add', $txtCit);
$wrd->execute();
```

ii) Injection based on ""="" query.

The execution of the above query while validating the user may prove harmful because like the 1=1 input, the above query will return all the rows of the database.

(a) Prevention for Injection based on ""="" query

The smart way to tackle this problem is to count the number of rows returned by the database. If after an input command, the number of rows returned is more than 1, then the user should be redirected to the same page otherwise he should be allowed to go to another page. It can be done using PHP as shown below in fig. 3.

Figure 3 PHP code to count number of rows

```
$sql="SELECT * FROM member WHERE username='$name' and password='$password'";
$result=mysql_query($sql);
$count=mysql_num_rows($result);
if ($count==1)
{
echo "succesfull login:";
}
else
echo "Sorry, your credentials are not valid, Please try again.";
?>
```

iii) Attack using mysql keyword

An attacker could empty out a table by executing a DELETE statement. For example-

SELECT * FROM customers WHERE username = ''; DELETE FROM customers WHERE 1 or username = ''

If we run this query, then the injected DELETE statement would completely empty our "customers" table.

(a) Prevention for attacks using mysql keyword

PHP has a special function to prevent these attacks. Use the function `mysql_real_escape_string`. This function takes a string which will be used in MySQL query and return the same string with all SQL Injection attempts safely escaped. Basically, it will replace those troublesome quotes (") a user might enter with a MySQL-safe substitute, an escaped quote \. It can be done using Procedural Style[2] as shown below in fig. 4.

Figure 4 Procedural style of mysql keyword

```
string mysql_real_escape_string ( mysqli $link , string $escapestr )
```

This function is used to create a legal SQL string that we can use in an SQL statement. The given string is encoded to an escaped SQL string into account the current character set of the connection.

IV. ENCRYPTION OF PASSWORD

The prominence of hacking has led to ever so need to encrypt data and the huge use of online web applications has also influenced its use. Password, still remain one of the most abused data item in the hacking world. The use of hashing algorithm has risen significantly in the past few years. Likewise the processor speed has exposed their vulnerability to hackers, increasing the number of steps in computation will severely impact the speed. In the past few years, MD5(message digest) and SHA(secure hash algorithm) has been broken but combination of both hashing techniques are widely used. The algorithm of MD5[3] can be shown below using fig.5.

Figure 5 Algorithm of MD5

Step 1. Append Padding Bits
Step 2. Append Length
Step 3. Initialize MD Buffer
Step 4. Process Message in 16-Word Blocks
Step 5. Output

The algorithm of SHA[4] can be shown below using fig. 6.

Figure 6 Algorithm of SHA

Step 1: initialize some variables
Step 2: pick a string
Step 3: break it into characters
Step 4: convert characters to ASCII codes
Step 5: convert numbers into binary
Step 6: add 1 at the end
Step 6 append '0's' at the end
Step 7: chunk the message
Step 8: break the 'chunk' into 'words'
Step 9: extend into 80 words

SHA is similar to MD5 algorithm. These two algorithms are vulnerable to break in isolation. The new method will combine the effect of SHA and MD5. The password to be encrypted will first be hashed using MD5 and then the generated hash will be used as an input to SHA algorithm which will generate another hash value and hence increasing the security. Even if two passwords are same they will be hashed differently as one algorithm will generate distinct hash value for the password and the other algorithm will generate the different hash value. It can be shown below using fig.7.

Figure 7 Concatenation of SHA and MD5

<code>\$password=sha(md5(\$password));</code>

V. RESULTS

The php code which is vulnerable to SQL injection is shown below in fig.8.

Figure 8 PHP code to attempt SQL injection

<pre><?php mysql_connect("localhost","root",""); mysql_select_db("simple_login"); // Check connection if (mysqli_connect_errno()) { echo "Failed to connect to MySQL: " . mysqli_connect_error(); } echo "database connected"; \$name=\$_POST['username']; \$password=\$_POST['password']; \$sql="SELECT * FROM member WHERE username='\$name' and password='\$password'"; if(\$sql==1) { echo "successful login"; } else echo "Sorry, your credentials are not valid, Please try again."; ?></pre>

When the proposed methods were implemented in the above code shown in figure8, the php code will look like as follows which can be shown below using fig. 9.

Figure 9 PHP code to prevent SQL injection

```

<?php
mysql_connect("localhost","root","");
mysql_select_db("simple_login");
// Check connection
if (mysqli_connect_errno())
{
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
echo "database connected";
$name=$_POST['urmm'];
$password=$_POST['passwr'];
// protection from sql injection
$password=sha(md5($password)); // Encrypted Password
$sql="SELECT * FROM member WHERE username='$name' and password='$password'";
$result=mysqli_query($sql);
$count=mysqli_num_rows($result);
if ($count==1)
{
    echo "succesful login:";
}
else
    echo "Sorry, your credentials are not valid, Please try again.";
?>

```

After the execution of the above code, the result shows that the attempt to SQL injection using `1=1` or `'='` were tried to done. But the code was effective in defeating the hacker in his attempt to bypass the security measures. The hash value of the password was stored in the database.

VI. CONCLUSION

Many of the techniques have problems in handling attacks that take advantage of poorly-coded stored procedures and cannot handle attacks that disguise themselves using alternate encodings. In this paper, we proposed different methods by which SQL injection attack can be prevented in php web applications

References

- [1] <https://www.google.co.in/search?sa=G&q=web+application+server&tbm=isch&tbs=simg:CAQSZxplCxCo1NgEGgQICAgLDA sQsIynCBo8CjoIAhIU2AzXDPMLqQzSDNAMhg3NFYgNkAkaIMwWsBx8DSHFTKFXP04Ad1DsheZFFRPQ4BronAwMzU V3DAsQjq7-CBoKCggIARIEQIaUmAw&ei=yX7eU5-uLoy-uATh-YL4Cg&ved=0CBkQwg4oAA>
- [2] <http://php.net/manual/en/mysqli.real-escape-string.php>
- [3] <http://www.freesoft.org/CIE/RFC/1321/3.htm>
- [4] http://en.wikipedia.org/wiki/Secure_Hash_Algorithm
- [5] The Open Web Application Security Project, "OWASP TOP 10 Project", <http://www.owasp.org/>
- [6] Ramakanth Dorai, Vinod Kannan, SQL Injection-Database Attack Revolution and Prevention Journal of International Commercial Law and Technology Vol6, issue4, 2011.
- [7] Xin Wang, Luhua Wang, Gengyu Wei, Dongmei Zhang, Yixian Yang, Hidden Web Crawling For SQL Injection Detection, IEEE, 2010.
- [8] Justin Clarke, SQL injection attacks and Defense, 2nd ed, Syngress publication, 2012, isbn-13:978-1597494243
- [9] 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, 1 2009.

Acknowledgments

We would like to thank reviewers for reviewing this paper.