Understanding Basic SQL Injection

SQL injection (also known as SQLI) is a code injection technique that occurs if the user-defined input data is not correctly filtered or sanitized of the ‘string literal escape characters’ embedded in SQL.

Basically SQLI is a way of injecting and executing arbitrary SQL statements. The whole idea is to make the application execute our arbitrary code which was not intended. In this tutorial we’ll be looking on how a basic SQL code injection can cause the application to mess up its authentication login and which would eventually lead to data access. So what’s the waiting then let’s get started.

Authentication Bypass (SQL injection)

Most of the authentication scripts you’ll find on the web are not secured and despite

this vulnerability first appeared in 1990’s there are still many applications vulnerable to this attack.

How SQL injection works?

This attack simply exploits bad filtering or sanitizing mechanism in the database layer of an application, this vulnerability gives the room to attackers to basically alter arbitrary SQL code to be executed.

For example you have a basic SQL statement as follows:-

Code:

```
SELECT * FROM Users where Name = ‘UserInput’;
```

Now if the page is vulnerable to this kind of attacks then an attacker have the room to alter anything to this SQL statement.
For Example the attacker can simple add

Code:

```
‘ or ‘1’ = ‘1
```

Which would result in :-

Code:

```
SELECT * FROM Users where Name =  '' or '1' = '1'
```

Now if you know some basic SQL you can simply point out that this means that now the application will be forced to get all the users in the table as the statement now includes a or condition i.e '1' = '1' which in any case will always be true.

Demonstration

To demonstrate a basic SQL authentication bypass attack I have created a set of some php scripts.

defines.php

Code:

```
<?php
$tableName = "badlogin";
$dbName               = "sqlnjection";
$sqlServer = "localhost";
$sqlUser = "root";
$sqlPass = "";
?>
```

functions.php

Code:

```
<?php
require "defines.php";

function checkTable()
{
        global $tableName;
        $query = "SELECT * from $tableName";
        $result = mysql_query($query) or die(mysql_error());
        if($result == FALSE) // Table is not created till
        createTable();
```

```
}
function createTable()
{
        global $tableName;
        $query = "CREATE TABLE $tableName(login char(50),pass char(50))";
        $result = mysql_query($query);
        $query = "INSERT INTO $tableName(login,pass)
values('admin','UnCrACkAbLe')";
        $result = mysql_query($query);
}
function checkCredentials($login,$pass)
{
        global $tableName;
        $query = "SELECT * FROM $tableName WHERE login='$login' AND
pass='$pass';";
        //             echo "<br/>$query<br/>";
        $result = mysql_query($query) or die(mysql_error());
        $rowsnum = mysql_num_rows($result);
        if($rowsnum > 0)
        {
                congrats();
        }
}
function congrats()
{
        echo"<p class='warning'>Congratulations You just completed the
Challenge...</p>";
        echo"<script type='text/javascript'>alert('Mission
Completed');</script>";
        // The redirection and Points award code should go here
}
?>
```

sqlInjection.php

Code:

```
<?php
```

```
require "defines.php";
require "functions.php";
?>
<html>
<head>
	<title>Bad Login</title>
	<link href='style.css' type='text/css' rel='stylesheet'/>
</head>
<body>
	<h1>Welcome to bad Login Please Enter your Credentals</h1>
	<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
	<table align="center" class="login">
		<tr>
			<td>Login</td>
			<td> : <input type="text" name="login"/></td>
		</tr>
		<tr>
			<td>Password</td>
			<td> : <input type="text" name="email"/></td>
		</tr>
		<tr>
			<td><input type="submit" name="go"/></td>
		</tr>
	</table>
	</form>
<?php
if( isset($_POST['login']) && isset($_POST['email']) &&
	isset($_POST['go']) )
{
	mysql_connect("$sqlServer","$sqlUser","$sqlPass") or mysql_error();
	mysql_select_db("$dbName") or mysql_error();
	//checking if table exists
	checkTable(); // checks if the table exists and create new if not
found
	checkCredentials($_POST['login'],$_POST['email']);
}
?>
```

```
</body>
</html>
```

Constructing the Attack String :-

Our main purpose for this scenario is to check is to exploit the vulnerability in the above set of pages and gain access to ‘admin’ account , We can simply do that by the following ways :-

1. Giving “admin'#” as a username to the application , This means that after writing admin as a username we used ‘#’ to comment any other code after that.
2. Giving “admin” as username and “ ‘ or ‘1’ = ‘1 “ as password. This would trick the application to use an ‘or always true’ condition which would eventually result in authentication bypass.

Source: http://www.go4expert.com/articles/understanding-basic-sql-injection-t26236/