Fragmented SQL Injection Attacks – The Solution

Ziyahan Albeniz - Thu, 29 Nov 2018 -

In this blog post, we discuss the research on Fragmented SQL Injection where the hackers control two entry points in the same context in order to bypass the authentication form. Our security researcher looks at the importance of single quotes in the SQL injection attacks and the solution, Prepared Statements, also known as Parameterized Queries.

likely to suggest putting a single quote into a parameter in the application. Then, if they received an error, they could infer the presence of an SQL injection vulnerability. Don't be surprised if you come across someone defining SQL injection as Single Quote Injection.

of single quotes in SQL injection attacks.

Single Quotes in SQL Injections

In a system (command interpreter, file system or database management system, for example), characters that have special meanings are called metacharacters. For instance, in the SQL query context, single and double quotes are used as string delimiters. They are used both at the beginning and the end of a string. This is why when a single or double quote is injected into a query, the query breaks and throws an error. Here's an example of where the quotes are placed in the query.

```
SELECT * FROM users WHERE user_name='USER_INPUT'
```

So, when a single quote is injected into the entry point above, the query interpreter will either complain about invalid syntax or report that it can't find the quote's pair at end of the string.

```
Code:
$username = "'";
$query = "SELECT * FROM users WHERE username='".$username."'"

Result:
SELECT * FROM users WHERE username='''
```

The system will throw an error for the single quote left unpaired at the end of the query. This is only valid for the

```
$query = "SELECT * FROM users WHERE id= " . $user_input;
```

In the example above, in order to perform an SQL injection, you have to input a numeric value, and the following values will then be evaluated as part of the SQL command.

The error returned due to the injection of a single quote may signify that the input from the user was not filtered or sanitized in any way, and that the input contains characters that have special meaning on the database.

Let's take a look at an instance where the single quote is blacklisted or escaped from the command.

```
$username ="' or 1=1 --";
$password ="qwerty123456";
// . . .
$query = "SELECT * FROM users WHERE username='".$username."' AND password='".$password."'";

select * from users where username='\' or 1=1 -- ' or password='qwerty123456';
```

As you see in this example, because the single quote (‘) is escaped with a backslash, the payload does not work as intended by the hacker.

Fragmented SQL Injection

Fragmented SQL Injection (not a term used by its inventor Rodolfo) takes place when two input points are used fragmented payloads to circumvent blacklists and character limits with this method.

We saw in the examples above that a single quote was injected and then escaped with a backslash (\). In a Fragmented SQL injection, if you use the backslash in the first field, and another SQL command that will return 'true' in the second field, you'll be able to bypass the form. Here's a demonstration of what happens in the background:

```
username: \
password: or 1 #

$query = select * from users where username='".$username."' and password='".$password."'";

select * from users where username='\' or password=' or 1 # ';
```

The backslash neutralizes the following single quote. So the value for the *username* column will end with the single quote that comes right after *password=* (the end of the gray text). Doing so will eliminate the required password field from the command. Due to the or 1 command, the condition will always return 'true'. The # (hash) will ignore the rest of the function, and you'll be able to bypass the login control and login form.

The Inconvenient Solution to SQL Injection Attacks

Please note that the blog post we referenced in this article suggests using the htmlentities() function in PHP to filter inputs, as a way to prevent the attack we described above. If you set the *ENT_QUOTES* flag, HTML encoding will convert single quotes, double quotes, and tag opening and closing signs, to their corresponding

However, this not the ideal solution, because there are situations where single or double quotes are not required to fulfill an SQL injection attack. In addition to that, some old school techniques like GBK Encoding can be used to bypass preventions like the addslashes() function in PHP and this weakens the overall prevention mechanism.

Prepared Statements are the Ideal Way to Prevent SQL Injection Attacks

At Netsparker, we believe that the correct and proper solution to prevent SQL Injection attacks is to use Prepared Statements, otherwise known as Parameterized Queries.

Parameterized Queries allow you to separate the structure of the SQL query from its values. All remaining methods to prevent SQL injection attacks may be bypassed in the near future with neat tricks such as that of **Chris Shiflett**, and are therefore not reliable.

Implementation of Parameterized Query in PHP and .NET

In PHP, you can use the Parameterized Query technique as illustrated:

```
$stmt = $dbh->prepare("UPDATE users SET email=:new_email WHERE id=:user_id");
$stmt->bindParam(':new_email', $email);
$stmt->bindParam(':user_id', $id);
```

```
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId", System.Data.SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

Conclusion

Developers still use blacklists to prevent the SQL Injection vulnerability. They do this either manually or using functions designed for this purpose (e.g. addslashes). However, we encounter new tactics in information security every day that attempt to bypass these blacklists. Ultimately, the best way to prevent injection based flaws like SQL Injections is to use a Prepared Statement. This is the only effective way developers can teach the system not to evaluate user controlled parameters as part of the query structure.

Read more about SQL injections in our **SQL injection cheat sheet**.

Related Articles