

MYSQL AND SQL INJECTION

<http://www.tutorialspoint.com/mysql/mysql-sql-injection.htm>

Copyright © tutorialspoint.com

If you take user input through a webpage and insert it into a MySQL database, there's a chance that you have left yourself wide open for a security issue known as SQL Injection. This lesson will teach you how to help prevent this from happening and help you secure your scripts and MySQL statements.

Injection usually occurs when you ask a user for input, like their name and instead of a name they give you a MySQL statement that you will unknowingly run on your database.

Never trust user provided data, process this data only after validation; as a rule, this is done by pattern matching. In the example below, the username is restricted to alphanumerical chars plus underscore and to a length between 8 and 20 chars - modify these rules as needed.

```
if (preg_match("/^\w{8,20}$/", $_GET['username'], $matches))
{
    $result = mysql_query("SELECT * FROM users
                          WHERE username=$matches[0]");
}
else
{
    echo "username not accepted";
}
```

To demonstrate the problem, consider this excerpt:

```
// supposed input
$name = "Qadir'; DELETE FROM users;";
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

The function call is supposed to retrieve a record from the users table, where the name column matches the name specified by the user. Under normal circumstances, *namewouldonlycontainalphanumericcharactersandperhapsspaces, suchasthestringilia. Buthere, byappendinganentirelynewqueryto* name, the call to the database turns into disaster: the injected DELETE query removes all records from users.

Fortunately, if you use MySQL, the `mysql_query` function does not permit query stacking or executing multiple queries in a single function call. If you try to stack queries, the call fails.

However, other PHP database extensions, such as SQLite and PostgreSQL, happily perform stacked queries, executing all of the queries provided in one string and creating a serious security problem.

Preventing SQL Injection:

You can handle all escape characters smartly in scripting languages like PERL and PHP. The MySQL extension for PHP provides the function `mysql_real_escape_string` to escape input characters that are special to MySQL.

```
if (get_magic_quotes_gpc())
{
    $name = stripslashes($name);
}
$name = mysql_real_escape_string($name);
mysql_query("SELECT * FROM users WHERE name='{$name}'");
```

The LIKE Quandary:

To address the LIKE quandary, a custom escaping mechanism must convert user-supplied % and _ characters to literals. Use `addslashes`, a function that let's you specify a character range to escape.

```
$sub = addslashes(mysql_real_escape_string("%something_"), "%_");
```

```
// $sub == \%something\  
mysql_query("SELECT * FROM messages WHERE subject LIKE '{$sub}%'");  
Loading [MathJax]/jax/output/HTML-CSS/jax.js
```