## **How to Test Your Own Web Server against Mysql Vulnerabilities**

(Adapted from : https://www.instructables.com/id/How-to-test-your-own-web-server-against-mysql-vuln/)

Every programmer should take care of the security of their code and servers. This exercise demonstrates how a MySQL injection works and what you can do to prevent your own code from get used for other purposes that you expect it. After reading this, you will understand what a MySQL injection is, how it works technically and what you can do to protect your code.

### Step 1: Set Up a Database

To test your code, you will need a **MySQL** database and a running. This can be done using xampp on your PC/Hard drive or using MySQL Workbench.

### Step 2: Example Database

For our test case, we use a very basic database with two users. Passwords are protected using the [md5](http://php.net/md5) algorithm. Run this MySQL code on your database to create the tables:

### SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

### SET AUTOCOMMIT = 0;

### START TRANSACTION;

### SET time\_zone = "+00:00";

### CREATE TABLE IF NOT EXISTS user2 (

### id bigint(15) NOT NULL,

### email varchar(255) NOT NULL,

### password varchar(32) NOT NULL

### ) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO\_INCREMENT=3 ;

### INSERT INTO user2 (id, email, password) VALUES

### (1, 'admin@mysite.com', 'b655e3f4ae881514c4896b9cd707e4d2'),

### (2, 'guest@mysite.com', '5d41402abc4b2a76b9719d911017c592');

### ALTER TABLE user2

### ADD PRIMARY KEY (id);

### ALTER TABLE user2

### MODIFY id bigint(15) NOT NULL AUTO\_INCREMENT,AUTO\_INCREMENT=3;COMMIT;

### Step 3: A Short Excursion About Password Security

Since the computing power all over the world is growing very fast, **md5** is no longer a strong hashing algorithm. The used hashes in the database above are generated using the following passwords:

**b655e3f4ae881514c4896b9cd707e4d2** = top secret password that will never be cracked that fast  
**5d41402abc4b2a76b9719d911017c592** = hello

**TRY IT YOURSELF!**

Since there are things like rainbow tables, even google itself is a good password cracking machine. When you google "***5d41402abc4b2a76b9719d911017c592***", the first result is the clear text password. When you google "**b655e3f4ae881514c4896b9cd707e4d2**" you will get no result since the input string is just too long to get cracked using the brute force method. **So: Use long passwords that don't appear in any dictionary!**

You can use websites like :

<https://md5.gromweb.com/> to **reverse** a hash

<https://www.md5hashgenerator.com/> to **generate** a hash

## Step 4: Example Code

For testing reasons, we will use this little script. It opens up a database connection, creates a form and tries to find a user in the database matching the credentials entered (email and password). Just **save** the script in a **folder** and call the file "**mysql-injection.php**".

**Note:** You may have to update the database name if your database has a different name.

<?php

$connection = mysqli\_connect("localhost","root","**password**");

mysqli\_select\_db($connection,"**mysecureapplication**");

if (!empty($\_POST['email']) && !empty($\_POST['password']))

{

$query = 'SELECT \* FROM `user` WHERE `email` = \''.$\_POST['email'].'\' AND `password` = \''.md5($\_POST['password']).'\'';

echo 'Used query: ',$query,'<br /><br />';

$result = mysqli\_query($connection,$query);

$matches = mysqli\_num\_rows($result);

if($row=mysqli\_fetch\_array($result))

{

$user = mysqli\_fetch\_assoc($result);

echo 'Hello '.$row['email'].'!';

}

else

{

echo 'Invalid credentials!';

}

}

?>

<form method="post" action="mysql-injection.php">

<p>Log in to get privileges!</p>

Mail: <input type="text" name="email" value="<?=!empty($\_POST['email']) ? $\_POST['email'] : ''?>" /><br />

Password: <input type="password" name="password" value="<?=!empty($\_POST['password']) ? $\_POST['password'] : ''?>" /><br />

<br />

<input type="submit" name="submit" />

</form>

## Step 5: Testing Your Script (Note: If you are doing this in XAMPP run it from XAMPP!!)

A screenshot of a computer

Description automatically generatedTo get started, run your script in VS code, make sure you have the necessary plugins.

Click the blue PHPServer button :

(Or right click PHPServer : Serve project)

This should automatically open a browser for you.

In most cases the URL is [http://localhost:3000/mysql-injection.php](http://localhost:3000/mysql-injection.php )

We now can test the login with the following credentials:

|  |  |
| --- | --- |
| **Username** | **Password** |
| admin@mysite.com | secret password that will never be cracked that fast |
| guest@mysite.com | hello |

As you can see in the screenshots, the script returns a "Hello <email>" to proof that your login credentials were correct. Use any other combinations to get an "Invalid credentials!" error. This is a working user authentication. But it’s very dangerous. Check out the next steps to see, why.

## Step 6: What’s happening behind the scenes..

Since the script outputs the database query that is used to select the correct user, you can see what is going on behind the scenes. When you login as admin the following query:

**'SELECT \* FROM `user` WHERE `email` = \''.$\_POST['email'].'\' AND `password` = \''.md5($\_POST['password']).'\''**

is computed to:

**'SELECT \* FROM `user` WHERE `email` = \'admin@mysite.com\' AND `password` = \''.md5('top secret password that will never be cracked that fast').'\''**

and then to:

'SELECT \* FROM `user` WHERE `email` = \'admin@mysite.com\' AND `password` = \'b655e3f4ae881514c4896b9cd707e4d2\''

and what is sent to the MySQL database is:

SELECT \* FROM `user` WHERE `email` = 'admin@mysite.com' AND `password` = 'b655e3f4ae881514c4896b9cd707e4d2'

**So,** it just selects a user by its email address and password. In this case, this is insecure because of the e-mail validation. The variable ***$\_POST['email']***is not secured before it is attached to the query, so we can inject a lot of evil things here. The ***$\_POST['password']*** variable is passed to md5() before its concatenated to the database query, so this is "accidentally safe" since md5() only returns numbers and characters and nothing that could crash our query.

## Step 7: Hack It!

As we learned in the previous step, ***$\_POST['email']*** is our backdoor. So, when you enter the following email, you can login as any user you want.

For the user with id 1:

Mail : **test@test.ie'** **OR `id` = 1 OR 1 = '**

Password : **anything you want**

For The user with id 2:

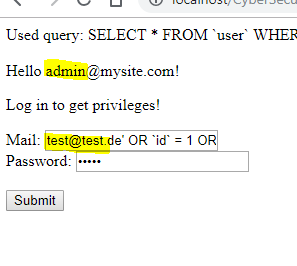
Mail : **test@test.de' OR `id` = 2 OR 1 = '**

Password : **anything you want**

As you can see, the used MySQL query is the following:

**SELECT \* FROM `user` WHERE `email` = 'test@test.de' OR `id` = 1 OR 1 = '' AND `password` = '098f6bcd4621d373cade4e832627b4f6'**

Since there are no brackets in the query, we can stop the string by just passing a ' in the email. After the ' we can modify the query itself. So we just add another "OR" condition that relates to the user id. This makes the email and the password obsolete since MySQL stops checking the condition as soon as one "OR" condition is true. Therefore, if there is a user with the id "1" in our database, we are logged in.



## Step 8: Protected Your Code

To make this code secure, just escape the users input. This can be done, for example, by using the method [mysqli\_real\_escape\_string](http://php.net/manual/de/function.mysql-real-escape-string.php). Another function may be [addslashes](http://php.net/manual/de/function.addslashes.php). So a "**secure**" solution would be using:

**mysqli\_real\_escape\_string($connection,$\_POST['email'])**

instead of

**$\_POST['email']**

So this is the final php code:

**$query = 'SELECT \* FROM `user` WHERE `email` = \''.mysqli\_real\_escape\_string($connection,$\_POST['email']).'\' AND `password` = \''.md5($\_POST['password']).'\'';**

And the computed query is:

**SELECT \* FROM `user` WHERE `email` = 'test@test.de\' OR `id` = 1 OR 1 = \'' AND `password` = '098f6bcd4621d373cade4e832627b4f6'**

So as you can see, the single quotes in the string are now escaped and no longer ending the mysql string. This causes the query to search for the following email address:

**test@test.de\' OR `id` = 1 OR 1 = \'**

And of course, this will fail, since there is no matching email in the database.