

Forum:[Hack Forums](#)

Thread: [PHP registration script.](#)

VipVince - January 16, 2014, 6:40 am

I am by no means an expert in PHP, however I am doing my best to learn and become better at it. I decided to make a login system, this way I will get my hands dirty working with sessions, security concepts, and other web app fundamentals that can benefit me. I know plenty of login scripts have been made before, but I just wanted to make one that I can use myself in the future. This script is currently giving me two issues however:

1. Is not writing values to the database
2. The email check is not working, I can register the same email twice.

It seems the more I look at it, the less I can see.

Perhaps some of you guys can fix it up.

PHP Code

```
<?php
<!DOCTYPE html>
<html>

<head>
<title>Registration page</title>
</head>

<body>

<?php
$dbhost    = "localhost";
$dbname    = "membersdb";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
{
    die("Error: All fields must be filled in!");
}

else
{

$username = htmlentities($_POST['username']);
$password = $_POST['password'];
$password2 = $_POST['password2'];
```

```
$email= htmlentities($_POST['email']);

if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}

if (strlen($username)>15) {
die("Username character limit exceeded!");
}

if (strlen($username)<6) {
die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
die("Email character limit exceeded!");
}

//secure the password
$salt = '-45dfeHK/___yu349@-/klF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
$hashedpass2 = sha1($salt.$password2);

if($hashedpass !== $hashedpass2)
{
die("Error: Passwords do not match!");
}

$stmt = $conn->prepare("SELECT username FROM members WHERE username = :name");
$stmt->bindParam(':name', $username);
$stmt->execute();

if($stmt->rowCount() > 0){
die("Username: $username already exists!");
}

$stmt = $conn->prepare("SELECT email FROM members WHERE email = :email");
$stmt->bindParam(':email', $email);
$stmt->execute();

if($stmt->rowCount() > 0){
die("Email: $email already exists in our database!");
}

else {

//get data ready for db insert
$sql =
```

```

"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
//prepare statement
$q = $conn->prepare($sql);
//execute statement, with variable data to placeholders
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, 'email'=>$email));

echo "Thank you for registering! <a href='login2.php'>Click here to login</a>";
    }
}

?>

</body>
</html>

```

Thanks in advance. VV.

Idle cares - January 16, 2014, 6:52 am

From the php.net's pdo rowCount() page:

Quote:For most databases, PDOStatement::rowCount() does not return the number of rows affected by a SELECT statement. Instead, use PDO::query() to issue a SELECT COUNT(*) statement with the same predicates as your intended SELECT statement, then use PDOStatement::fetchColumn() to retrieve the number of rows that will be returned. Your application can then perform the correct action.

So you can't use the function in the way you're attempting to.
What you can do, as they've explained is something like:

PHP Code

```

<?php
$sql = "SELECT COUNT(*) FROM fruit WHERE calories > 100";
if ($res = $conn->query($sql)) {

    /* Check the number of rows that match the SELECT statement */
    if ($res->fetchColumn() > 0) {

        /* Issue the real SELECT statement and work with the results */
        $sql = "SELECT name FROM fruit WHERE calories > 100";
        foreach ($conn->query($sql) as $row) {
            print "Name: " . $row['NAME'] . "\n";
        }
    }
    /* No rows matched -- do something else */
    else {
        print "No rows matched the query.";
    }
}

$res = null;
$conn = null;
?>

```

Edit: just put the code directly from the [rowCount\(\)](#) page, it was better than my own >.<

La Manifesto - January 16, 2014, 6:53 am

You can use SQL's built in COUNT to check the count.

PHP Code

```
<?php
$stmt = $conn->prepare(
"SELECT COUNT(`username`) FROM members WHERE `username` = :name");
$stmt->execute(array(':name' => $username));

if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: $username already exists!");
}
```

And do the same for email. Also why the random else before the INSERT query?

The Alchemist16 - January 16, 2014, 7:47 am

This should work :

PHP Code

```
<?php
<!DOCTYPE html>
<html>

<head>
<title>Registration page</title>
</head>

<body>

<?php
$dbhost    = "localhost";
$dbname    = "membersdb";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']) || empty($_POST['email']))
{
    die("Error: All fields must be filled in!");
}

$username = $_POST['username'];
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email= $_POST['email'];

if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
    die("Error: E-mail address is not valid!");
}
```

```

if($password != $password){
    die("The two passwords entered do not match");
}
if (strlen($username)>15) {
    die("Username character limit exceeded!");
}

if (strlen($username)<6) {
    die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
    die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
    die("Email character limit exceeded!");
}

//secure the password
$salt = '-45dfeHK/___yu349@-/kIF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);

$stmt = $conn->prepare("SELECT * FROM members WHERE username = :name");
$stmt->execute(array('name', $username));

if($stmt->rowCount() > 0){
    die("Username: ".htmlspecialchars($username)." already exists!");
}

$stmt = $conn->prepare("SELECT * FROM members WHERE email = :email");
$stmt->execute(array('email', $email));

if($stmt->rowCount() > 0){
    die("Email: ".htmlspecialchars($email)." already exists in our database!");
}

//get data ready for db insert
$sql =
"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
//prepare statement
$q = $conn->prepare($sql);
//execute statement, with variable data to placeholders
$q->execute(array('username' => $username, 'password' => $hashedpass, 'email' => $email));

echo "Thank you for registering! <a href='login2.php'>Click here to login</a>";

?>

</body>
</html>

```

I dont know why nothing is getting inserted. Your table should be like this :

Code CREATE TABLE IF NOT EXISTS members (id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, email VARCHAR(30), username VARCHAR(30), password CHAR(40))
Havent really tested them. But they SHOULD work.

Idle cares - January 16, 2014, 7:59 am

(Jan 16, 2014 08:47 AM)The Alchemist16 Wrote: [\[CITE\]](#) PHP Code

```
<?php  
code blah
```

I dont know why nothing is getting inserted. Your table should be like this :
Code CREATE TABLE IF NOT EXISTS members (id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, email VARCHAR(30), username VARCHAR(30), password CHAR(40))
Havent really tested them. But they SHOULD work.
Password field should be larger. By quite a bit :) OP should also take a look at [password_hash](#)
Quote:PASSWORD_DEFAULT - Use the bcrypt algorithm (default as of PHP 5.5.0). Note that this constant is designed to change over time as new and stronger algorithms are added to PHP. For that reason, the length of the result from using this identifier can change over time. Therefore, it is recommended to store the result in a database column that can expand beyond 60 characters (255 characters would be a good choice).

I got ahead of myself in my first post and forgot about your second issue.
Try out this code, and it will spit out any errors.

PHP Code

```
<?php  
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);  
try  
{  
    //get data ready for db insert  
    $sql =  
"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";  
    //prepare statement  
    $q = $conn->prepare($sql);  
    //execute statement, with variable data to placeholders  
    $q->execute(array(':username' => $username, ':password' => $hashedpass, 'email' =>  
$email));  
} catch (PDOException $e)  
{  
    print_r($e);  
}
```

VipVince - January 16, 2014, 8:04 am

Made some changes, however for some reason email is still able to be registered twice nor data being saved to db ^_^. It tells me it has though with the "thanks for registering" bla bla.

PHP Code

```
<?php  
<!DOCTYPE html>  
<html>  
  
<head>
```

```
<title>Registration page</title>
</head>
```

```
<body>
```

```
<?php
```

```
$dbhost  = "localhost";
$dbname  = "membersdb";
$dbuser  = "root";
$dbpass  = "";
```

```
// database connection
```

```
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);
```

```
if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
```

```
{
    die("Error: All fields must be filled in!");
}
```

```
else
```

```
{
```

```
$username = htmlentities($_POST['username']);
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email= htmlentities($_POST['email']);
```

```
if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}
```

```
if (strlen($username)>15) {
die("Username character limit exceeded!");
}
```

```
if (strlen($username)<6) {
die("Username must contain more than 5 characters!");
}
```

```
if (strlen($password)<6) {
die("Password must contain more than 5 characters!");
}
```

```
if (strlen($email)>30) {
die("Email character limit exceeded!");
}
```

```
//secure the password
```

```
$salt = '-45dfeHK/___yu349@-/kIF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
$hashedpass2 = sha1($salt.$password2);
```

```

if($hashedpass !== $hashedpass2)
{
die("Error: Passwords do not match!");
}

$stmt = $conn->prepare(
"SELECT COUNT(`username`) FROM members WHERE `username` = :name");
$stmt->execute(array(':name' => $username));

if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
die("Username: ".htmlentities($username)." already exists in our database!");
}

$stmtz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");
$stmtz->execute(array(':email' => $email));

if ($stmtz->fetch(PDO::FETCH_COLUMN) >= 1)
{
die("Email: ".htmlentities($email)." already exists in our database!");
}

//get data ready for db insert
$sql =
"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
//prepare statement
$q = $conn->prepare($sql);
//execute statement, with variable data to placeholders
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, 'email'=>$email));

echo "Thank you for registering! <a href='login2.php'>Click here to login</a>";

}

?>

</body>
</html>

```

Thanks for the contributions guys.

Edit: records are now being saved fine.

Only issue remaining is the email check.

Edit, solved. All working fine.

Thanks again, going to do some more reading and further implementations :).

La Manifesto - January 16, 2014, 8:08 am

Any way that we could see your table?

VipVince - January 16, 2014, 8:20 am

(Jan 16, 2014 09:08 AM)Adherence Wrote: [\[CITE\]](#) Any way that we could see your table? Sure.

[img]http://#/ikXq8Y.jpg[/img]

I have a very simple login script now, that allows the user to login after registering. Its still in the basic stages and insecure, I want to have the theory working 100% before adding other features like sessions etc.

When I attempt to login with a registered username and password, it is telling me that it has been unsuccessful. However, when I go into my database and change the hashed password to plain text, and attempt the login again it is successful. So I think the hashing is the issue. Here is the four scripts, perhaps someone can see the issue.

Register form.

PHP Code

```
<?php
<form action="checkregister.php" method="POST">
    Username<br>
    <input type="text" name="username" /><br>
    Password<br>
    <input type="password" name="password" /><br>
    Re-Enter Password<br>
    <input type="password" name="password2" /><br>
    Email<br>
    <input type="text" name="email" /><br>
    <input type="submit" value="Register" />
</form>
```

checkregister.php

PHP Code

```
<?php
<!DOCTYPE html>
<html>

<head>
<title>Registration page</title>
</head>

<body>

<?php
$dbhost    = "localhost";
$dbname    = "mymembers";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
{
    die("Error: All fields must be filled in!");
}
```

```

    }

else
{

$username = htmlentities($_POST['username']);
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email= htmlentities($_POST['email']);

if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}

if (strlen($username)>15) {
die("Username character limit exceeded!");
}

if (strlen($username)<6) {
die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
die("Email character limit exceeded!");
}

//secure the password
$salt = '-45dfeHK/___yu349@-/klF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
$hashedpass2 = sha1($salt.$password2);

if($hashedpass !== $hashedpass2)
{
    die("Error: Passwords do not match!");
}

//check if the username is already registered.
$stmt = $conn->prepare(
"SELECT COUNT(`username`) FROM members WHERE `username` = :name");
$stmt->execute(array(':name' => $username));

if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: ".htmlentities($username)." already exists in our database!");
}

//check if the email is already registered.
$stmtz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");

```

```

$stmz->execute(array(':email' => $email));

if ($stmz->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Email: ".htmlentities($email)." already exists in our database!");
}

//get data ready for db insert
$sql =
"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
//prepare statement
$q = $conn->prepare($sql);
//execute statement, with variable data to placeholders
$q->execute(array(':username'=>$username,':password'=>$hashedpass, ':email'=>$email));

echo "Thank you for registering ".htmlentities($username).
"! <a href='login.php'>Click here to login</a>";

}

?>

</body>
</html>

```

loginform
PHP Code

```

<?php
<form action="checklogin.php" method="POST">
    Username<br>
    <input type="text" name="username" /><br>
    Password<br>
    <input type="password" name="password" /><br>
    <input type="submit" value="Login" />
</form>

```

checklogin.php
PHP Code

```

<?php

$dbhost    = "localhost";
$dbname    = "mymembers";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

$username = $_POST['username'];
$password = $_POST['password'];

if($username == "") {
    die("You must enter a username!");
}

```

```

    }
    if($password == "") {
        die("You must enter a password!");
    }
    // query
    $result = $conn->prepare(
"SELECT * FROM members WHERE username= :user AND password= :pass");
    $result->bindParam(':user', $username);
    $result->bindParam(':pass', $password);
    $result->execute();
    $rows = $result->fetch(PDO::FETCH_NUM);
    if($rows > 0) {
        echo "You are now logged in!";
    }
    else{
        echo "Login unsuccessful!";
    }

    ?>

```

Mafugsa Bob - January 16, 2014, 11:06 am

PHP Code

```

<?php
$dbhost = "localhost";
$dbname = "membersdb";
$dbuser = "root";
$dbpass = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname", $dbuser, $dbpass);

/*

```

If you give the submit <input> a name attribute you can simply do like I have done..

```

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']) || empty($_POST['email']))
{
    die("Error: All fields must be filled in!");
}
*/

if (!isset($_POST['submit']) || empty($_POST['submit']))
{
    die("Error: All field must be filled in!");
}
else
{
    /*
    $username = htmlentities($_POST['username']);
    $password = $_POST['password'];
    $password2 = $_POST['password2'];

```

```
$email    = htmlentities($_POST['email']);
```

Don't run htmlentities on data you've been submitted, run it when you output it instead.

```
*/
```

```
$username = $_POST['username'];  
$password = $_POST['password'];  
$password2 = $_POST['password2'];  
$email    = $_POST['email'];
```

```
if (filter_var($email, FILTER_VALIDATE_EMAIL) === false)  
{  
    die("Error: E-mail address is not valid!");  
}
```

```
if (strlen($username) > 15)  
{  
    die("Username character limit exceeded!");  
}
```

```
if (strlen($username) < 6)  
{  
    die("Username must contain more than 5 characters!");  
}
```

```
if (strlen($password) < 6)  
{  
    die("Password must contain more than 5 characters!");  
}
```

```
/*  
if (strlen($email) > 30)  
{  
    die("Email character limit exceeded!");  
}
```

your earlier filter_var should take care of the length (which is WAAAY more than just 30 (254 according to the standard))

```
*/
```

```
/*  
//secure the password  
$salt    = '-45dfeHK/___yu349@-/klF21-1_V4JkUP/4';  
$hashedpass = sha1($salt . $password);  
$hashedpass2 = sha1($salt . $password2);
```

More like UNSECURE the password (SHA1 shouldn't be used, the salt should always be random)
! Upgrade to PHP >= 5.5!

```
*/
```

```
$hashedpass = password_hash($password, PASSWORD_BCRYPT, [ "cost" => 12 ]);
```

```
/*
```

```

if ($hashedpass !== $hashedpass2)
{
    die("Error: Passwords do not match!");
}

$hashpass will always be different, password_hash will generate a random salt, blah blah
*/

if (!password_verify($password2, $hashpass))
{
    die("Error: Password do not match!");
}

$stmt = $conn->prepare("SELECT username FROM members WHERE username = :name");
$stmt->bindParam(':name', $username);
$stmt->execute();

if ($stmt->rowCount() > 0)
{
    die("Username: $username already exists!");
}

$stmt = $conn->prepare("SELECT email FROM members WHERE email = :email");
$stmt->bindParam(':email', $email);
$stmt->execute();

if ($stmt->rowCount() > 0)
{
    die("Email: $email already exists in our database!");
}

else
{
    //get data ready for db insert
    $sql =
"INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
    //prepare statement
    $q = $conn->prepare($sql);
    //execute statement, with variable data to placeholders
    $q->execute(array(
        ':username' => $username,
        ':password' => $hashedpass,
        ':email' => $email
    ));

    echo "Thank you for registering! <a href='login2.php'>Click here to login</a>";
}
}

?>

<!DOCTYPE html>

```

```
<html>

<head>
<title>Registration page</title>
</head>

<body>
  <form method="POST">
    <input type="text" name="username"/><br />
    <input type="password" name="password"/><br />
    <input type="submit" name="submit"/>
  </form>
</body>
</html>
```

I fixed it up a little and commented out some parts.

Can we see your database structure? Also, I would make the username and email field UNIQUE fields, that way there's no chance you'll have duplicates.

La Manifesto - January 16, 2014, 11:15 am

(Jan 16, 2014 09:20 AM)VipVince Wrote: [\[CITE\]](#) Sure.

[img]http://#/ikXq8Y.jpg[/img]

I have a very simple login script now, that allows the user to login after registering. Its still in the basic stages and insecure, I want to have the theory working 100% before adding other features like sessions etc.

When I attempt to login with a registered username and password, it is telling me that it has been unsuccessful. However, when I go into my database and change the hashed password to plain text, and attempt the login again it is successful. So I think the hashing is the issue. Here is the four scripts, perhaps someone can see the issue.

Register form.

PHP Code

```
<?php
<form action="checkregister.php" method="POST">
  Username<br>
  <input type="text" name="username" /><br>
  Password<br>
  <input type="password" name="password" /><br>
  Re-Enter Password<br>
  <input type="password" name="password2" /><br>
  Email<br>
  <input type="text" name="email" /><br>
  <input type="submit" value="Register" />
</form>
```

checkregister.php

PHP Code

```
<?php
<!DOCTYPE html>
<html>
```

```
<head>
<title>Registration page</title>
</head>

<body>

<?php
$dbhost = "localhost";
$dbname = "mymembers";
$dbuser = "root";
$dbpass = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
{
    die("Error: All fields must be filled in!");
}

else
{

$username = htmlentities($_POST['username']);
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email= htmlentities($_POST['email']);

if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}

if (strlen($username)>15) {
    die("Username character limit exceeded!");
}

if (strlen($username)<6) {
    die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
    die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
    die("Email character limit exceeded!");
}

//secure the password
$salt = '-45dfeHK/___yu349@-/kIF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
```

```

$hashedpass2 = sha1($salt.$password2);

if($hashedpass !== $hashedpass2)
{
    die("Error: Passwords do not match!");
}

//check if the username is already registered.
$stmt = $conn->prepare(
    "SELECT COUNT(`username`) FROM members WHERE `username` = :name");
$stmt->execute(array(':name' => $username));

if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: ".htmlentities($username)." already exists in our database!");
}

//check if the email is already registered.
$stmtz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");
$stmtz->execute(array(':email' => $email));

if ($stmtz->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Email: ".htmlentities($email)." already exists in our database!");
}

//get data ready for db insert
$sql =
    "INSERT INTO members (username, password, email) VALUES (:username, :password, :email)";
//prepare statement
$q = $conn->prepare($sql);
//execute statement, with variable data to placeholders
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, 'email'=>$email));

echo "Thank you for registering ".htmlentities($username).
"! <a href='login.php'>Click here to login</a>";

}

?>

</body>
</html>

loginform
PHP Code
<?php
<form action="checklogin.php" method="POST">
    Username<br>
    <input type="text" name="username" /><br>
    Password<br>
    <input type="password" name="password" /><br>
    <input type="submit" value="Login" />

```

</form>

checklogin.php

PHP Code

<?php

```
$dbhost = "localhost";
$dbname = "mymembers";
$dbuser = "root";
$dbpass = "";
```

```
// database connection
```

```
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);
```

```
$username = $_POST['username'];
```

```
$password = $_POST['password'];
```

```
if($username == "") {
    die("You must enter a username!");
}
```

```
if($password == "") {
    die("You must enter a password!");
}
```

```
// query
```

```
$result = $conn->prepare(
```

```
"SELECT * FROM members WHERE username= :user AND password= :pass");
```

```
$result->bindParam(':user', $username);
```

```
$result->bindParam(':pass', $password);
```

```
$result->execute();
```

```
$rows = $result->fetch(PDO::FETCH_NUM);
```

```
if($rows > 0) {
    echo "You are now logged in!";
}
```

```
else{
    echo "Login unsuccessful!";
}
```

```
?>
```

Try changing the \$row > 0 to count(\$rows) > 0.

Idle cares - January 16, 2014, 12:00 pm

Erm, you're not hashing the user entered password and trying to compare it to a hashed password, this will never work, never.

OP go install php 5.5+ and look at password_hash() and password_verify(). They automate the whole process and mean that you're not using already broken hashing algorithms and make sure you're not playing around with salts when you don't need to.

The creators of php have also said that they will continue to update the functions to use newer, more secure hashing methods when they come out.

The only reasons you wouldn't want to use them are:

1. You want to learn correct hashing/salt theory type stuff

2. You think that using something mainstream will make your code insecure if someone finds a vulnerability in the functions.
3. your hosting/OS doesn't support 5.5+

Which:

1. kudos, but make damn sure you're not using your tests in production code till you really know what you're doing.
2. You're more likely to code in vulnerabilities, but skepticism is good.
3. I believe there are methods to manually install the functions.

However, if you're still determined to use the code you have here, add this line of code to checklogin.php.

PHP Code

```
<?php
//hash the password to compare to the hashed db value
$salt = '-45dfeHK/___yu349@-/kIF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);

//and
$result->bindParam(':pass', $hashedpass);
```

Edit:

To then fix the issue Adherence has responded to, again you just need to use the same thing you did to make sure the registering member didn't have a duplicate email

PHP Code

```
<?php
$result = $conn->prepare(
"SELECT COUNT(*) FROM members WHERE username= :user AND password= :pass");
$result->bindParam(':user', $username);
$result->bindParam(':pass', $hashedpass);
$result->execute();
$rows = $result->fetchColumn();
if ($rows > 0)
{
    echo "You are now logged in!";
}
else
{
    echo "Login unsuccessful!";
}
```

FETCH_NUM sets the return to a numeric array, meaning that it would look something like

PHP Code

```
<?php
array(0 => 1, 1 => "my@email.com", 2 => "dave", 3 => "dsfglvjhsdf809u3546tlkj")
fetchColumn() returns the value of the very first column queried, and as we have changed that to a
count() it will be the number of rows where the user and pass match.
```

VipVince - January 16, 2014, 12:37 pm

Thanks for all the constructive and informative posts.

I definitely have learned a thing or two, I initially just wanted to get the whole concept working,

which I now have. Now I will take the advice given and make my current code better, and improved following the advice given. I guess the best way of learning is through trial and error, fixing and improving.

Mafugsa Bob - January 16, 2014, 1:52 pm

(Jan 16, 2014 01:37 PM)VipVince Wrote: [\[CITE\]](#) Thanks for all the constructive and informative posts.

I definitely have learned a thing or two, I initially just wanted to get the whole concept working, which I now have. Now I will take the advice given and make my current code better, and improved following the advice given. I guess the best way of learning is through trial and error, fixing and improving.

Post whatever code you're using now so we can judge you and tell you how bad you are.

Spoiler ([Click to View](#))

obviously I was joking about the last part

VipVince - January 16, 2014, 2:13 pm

(Jan 16, 2014 02:52 PM)Mafugsa Bob Wrote: [\[CITE\]](#) Post whatever code you're using now so we can judge you and tell you how bad you are.

Spoiler ([Click to View](#))

obviously I was joking about the last part

I stopped for today, my concentration after several hours is not the same and my eyes don't interpret the code the same way. Will continue on this in the morning, and will post the code for examination!

So... - January 16, 2014, 5:33 pm

If you are planning on continuing and further expanding your knowledge into a professional based environment, I would highly recommend learning a PHP Framework, such as Codeigniter

Only downside is you won't learn the entry level concepts such as database's and securing forms/user inputs, all this is already built in, you do however dive straight into the more complicated areas

VipVince - January 17, 2014, 5:51 am

(Jan 16, 2014 06:33 PM)So... Wrote: [\[CITE\]](#) If you are planning on continuing and further expanding your knowledge into a professional based environment, I would highly recommend learning a PHP Framework, such as Codeigniter

Only downside is you won't learn the entry level concepts such as database's and securing forms/user inputs, all this is already built in, you do however dive straight into the more complicated areas

Sounds promising. I will check that out.

Offtopic: just wondering what IDE you guys use?

Idle cares - January 17, 2014, 6:13 am

I use Netbeans, it's free and pretty good imo. Has code hints, supports php docs/docblocks, syntax formatting, syntax error alerts, and the holy grail of ftp upload on save >.<

Most competitive IDEs have all those features (and many more which I haven't mentioned) I use netbeans because it's free and was recommended to me by my mentor. You'll probably find there's a lot of personal preference when it comes to finding the right IDE.

DogEatsDog - January 17, 2014, 7:20 am

(Jan 17, 2014 07:13 AM)Idle cares Wrote: [\[CITE\]](#) I use Netbeans, it's free and pretty good imo. Has code hints, supports php docs/docblocks, syntax formatting, syntax error alerts, and the holy grail of ftp upload on save >.<

Most competitive IDEs have all those features (and many more which I haven't mentioned) I use netbeans because it's free and was recommended to me by my mentor. You'll probably find there's a lot of personal preference when it comes to finding the right IDE.

FTP as in upload to live environment, that's kind of a risky life you're living.

I use PHPStorm by JetBrains. A solid IDE which has everything you'll ever need. Such as built-in command line commands for frameworks, such as Laravel's Artisan.

You can check all of its features here <http://www.jetbrains.com/phpstorm/whatsnew/index.html> .

It is not free but I am sure you'll easily find a way around that.

Idle cares - January 17, 2014, 7:31 am

(Jan 17, 2014 08:20 AM)DogEatsDog Wrote: [\[CITE\]](#) FTP as in upload to live environment, that's kind of a risky life you're living.

What's life without a little risk ??

I only use it on my dev servers ^^

OP, I actually made a thread, with the same question you asked, a couple months ago and the discussion might give you some ideas. <https://hackforums.net/showthread.php?tid=3625559>

VipVince - January 25, 2014, 6:55 am

Haven't had much time to add more features to it as I have been busy, however I am free today and going to get to work on it. Need to change the password hashing method, possibly bcrypt. This is the code so far, nothing special but I just want a secure and solid registration and login system. Will work on the html/css after its finished to make it look visually better.

PHP Code

```
<?php
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Registration page</title>
```

```
</head>
```

```
<body>
```

```
<form method="POST">
```

```
Username:<input type="text" name="username"/><br />
```

```
Password:<input type="password" name="password"/><br />
```

```
Re-Enter Password:<input type="password" name="password2"/><br />
```

```
Email:<input type="text" name="email"/><br />
```

```
<input type="submit" name="submit" value="Register" />
```

```
</form>
```

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

```
<?php
```

```
if (isset($_POST['submit'])) {
```

```
    $dbhost    = "localhost";
    $dbname    = "vipvincemembers";
    $dbuser    = "root";
    $dbpass    = "";
```

```
    // database connection
```

```
    $conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);
```

```
    if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']) || empty($_POST['email']))
    {
        die("Error: All fields must be filled in!");
    }
```

```
else
{
```

```
    $username = $_POST['username'];
    $password = $_POST['password'];
    $password2 = $_POST['password2'];
    $email = $_POST['email'];
    $userIp = ($_SERVER["REMOTE_ADDR"]);
    $joinDate = date("Y-m-d H:i:s");
```

```
    if(!filter_var($email, FILTER_VALIDATE_EMAIL))
    {
        die("Error: E-mail address is not valid!");
    }
```

```
    if (strlen($username)>15) {
        die("Username character limit exceeded!");
    }
```

```
    if (strlen($username)<6) {
        die("Username must contain more than 5 characters!");
    }
```

```
    if (strlen($password)<6) {
        die("Password must contain more than 5 characters!");
    }
```

```
    if (strlen($email)>30) {
        die("Email character limit exceeded!");
    }
```

```
    //secure the password
```

```

$salt = '-45dfeHK/___yu349@-/kIF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
$hashedpass2 = sha1($salt.$password2);

if($hashedpass !== $hashedpass2)
{
    die("Error: Passwords do not match!");
}

//check if the username is already registered. we use prepare() to get the data ready for assignment
$stmt = $conn->prepare(
    "SELECT COUNT(`username`) FROM members WHERE `username` = :name");

//notice we use :name which is a placeholder for the username variable value, we execute the command
and.
$stmt->execute(array(':name' => $username));

//we use pdo's fetch to see if its equal to 1 which means it exists in the db
if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: ".htmlentities($username). " already exists in our database!");
}

//check if the email is already registered. Again we start with prepare.
$stmtz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");

//execute() again assigning the placeholder with the variable data
$stmtz->execute(array(':email' => $email));

if ($stmtz->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Email: ".htmlentities($email). " already exists in our database!");
}

//get data ready for db insert, notice the insert into command in caps.
$sql =
    "INSERT INTO members (username, password, email, ip, joindate) VALUES (:username, :password
    , :email, :ip, :joindate)";

//prepare statement before inserting.
$q = $conn->prepare($sql);

//execute statement, with variable data into placeholders
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, ':email'=>$email, ':ip'=>
    $userIp,
    ':joindate'=>$joinDate));

echo "<b>Thank you for registering ".htmlentities($username).
    "! <a href='login.php'>Click here to login</a></b>";

}
}
?>

```

Mafugsa Bob - January 25, 2014, 7:02 am

Your formatting is incredibly inconsistent, pl0x fix.

VipVince - January 25, 2014, 7:04 am

(Jan 25, 2014 08:02 AM)Mafugsa Bob Wrote: [\[CITE\]](#) Your formatting is incredibly inconsistent, pl0x fix.

I am not sure if its when I paste it to hackforums or my editor, but I will get onto it.

Mafugsa Bob - January 25, 2014, 7:11 am

(Jan 25, 2014 08:04 AM)VipVince Wrote: [\[CITE\]](#) I am not sure if its when I paste it to hackforums or my editor, but I will get onto it.

<http://phpformatter.com>

I use it all the time when copying code from HF.

Idle cares - January 25, 2014, 8:02 am

I think you posted somewhere that you use netbeans, press alt + shift + F and it'll format it for you.

The script looks functional, but I really do recommend using password_hash() for hasing passwords (or crypt() if you're not using php 5.5+) as it's a hell of a lot more secure than that single salt you're using.

For example, if someone compromises your server, finds this file with your single salt in it, and your database:

Because you're using the same salt for every member, if any members are using the same password their hashes will be exactly the same. People are quite dim when it comes to passwords and like to use really common ones, so if there are a load of matching hashes chances are they're all using a simple common password which the hacker can then bruteforce/dictionary attack.

Also, using the same hash means that if a hacker does find your hash they will be able to create a rainbow table for your entire database and be able to crack the passwords much easier.

But, if you use something like password_hash() it will generate a 22 character random salt for each password you hash, meaning that no two hashes would ever be the same (well collisions might be possible but very, very, very rare) and even though you would be storing the salts in the database the hacker would need to create a rainbow table/attack for each individual password making it somewhat infeasible.

For example:

PHP Code

```
<?php  
echo password_hash("idleCares", PASSWORD_DEFAULT);
```

Returns:

```
$2y$10$uIRHatiJ6wZiqE3VN3/QfuLFUqkC1Br29QajLSUve4WOz8Ear7xhm
```

If we break that string down:

\$2y = the hashing algorithm (bcrypt)

\$10\$ = the computational cost of the hash (how many iterations of the bcrypt algorithm to use)

uIRHatiJ6wZiqE3VN3/Qfu = random 22 character salt

LFUqkC1Br29QajLSUve4WOz8Ear7xhm = the hashed password

Once you've created this nicely secure hash, it's very simple to check against:

PHP Code

```
<?php
if (password_verify('idleCares',
'$2y$10$uIRHatiJ6wZiqE3VN3/QfuLFUqkC1Br29QajLSUve4WOz8Ear7xhm'))
{
    echo "matching password";
}
else
{
    echo "passwords do not match";
}
```

VipVince - January 25, 2014, 8:03 am

New code:

PHP Code

```
<?php
<!DOCTYPE html>
<html>

<head>
<title>Registration page</title>
</head>

<body>
<form method="POST">
Username:<input type="text" name="username"/><br />
Password:<input type="password" name="password"/><br />
Re-Enter Password:<input type="password" name="password2"/><br />
Email:<input type="text" name="email"/><br />
<input type="submit" name="submit" value="Register" />
</form>
</body>
</html>

<?php

if (isset($_POST['submit'])) {

$dbhost    = "localhost";
$dbname    = "vipvincemembers";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
{
    die("Error: All fields must be filled in!");
}
```

```

else
{

$username = $_POST['username'];
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email= $_POST['email'];
$userIp = ($_SERVER["REMOTE_ADDR"]);
$joinDate = date("Y-m-d H:i:s");

if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}

if (strlen($username)>15) {
    die("Username character limit exceeded!");
}

if (strlen($username)<6) {
    die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
    die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
    die("Email character limit exceeded!");
}

//secure the password
$salt = '-45dfeHK/___yu349@-/klF21-1_V4JkUP/4';
$hashedpass = sha1($salt.$password);
$hashedpass2 = sha1($salt.$password2);

if($hashedpass !== $hashedpass2)
{
    die("Error: Passwords do not match!");
}

//check if the username is already registered. we use prepare() to get the data ready for assignment
$stmt = $conn->prepare(
    "SELECT COUNT(`username`) FROM members WHERE `username` = :name");

//notice we use :name which is a placeholder for the username variable value, we execute the comm
and.
$stmt->execute(array(':name' => $username));

//we use pdo's fetch to see if its equal to 1 which means it exists in the db
if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: ".htmlentities($username)." already exists in our database!");
}

```

```
}
```

```
//check if the email is already registered. Again we start with prepare.
```

```
$stmz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");
```

```
//execute() again assigning the placeholder with the variable data
```

```
$stmz->execute(array(':email' => $email));
```

```
if ($stmz->fetch(PDO::FETCH_COLUMN) >= 1)
```

```
{  
    die("Email: ".htmlentities($email)." already exists in our database!");  
}
```

```
//get data ready for db insert, notice the insert into command in caps.
```

```
$sql =
```

```
"INSERT INTO members (username, password, email, ip, joindate) VALUES (:username, :password  
, :email, :ip, :joindate)";
```

```
//prepare statement before inserting.
```

```
$q = $conn->prepare($sql);
```

```
//execute statement, with variable data into placeholders
```

```
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, ':email'=>$email, ':ip'=>  
$userIp,  
':joindate'=>$joinDate));
```

```
echo "<b>Thank you for registering ".htmlentities($username).
```

```
"! <a href='login.php'>Click here to login</a></b>";
```

```
}
```

```
}
```

```
?>
```

Any feedback negative or positive would be appreciated, I am aware of the password hashing being insecure, will be fixing that.

Idle cares - January 25, 2014, 8:20 am

(Jan 25, 2014 09:03 AM)VipVince Wrote: [\[CITE\]](#) Any feedback negative or positive would be appreciated, I am aware of the password hashing being insecure, will be fixing that.

Damnit lol. Just as I finish adding all that info.

VipVince - January 25, 2014, 8:30 am

(Jan 25, 2014 09:20 AM)Idle cares Wrote: [\[CITE\]](#) Damnit lol. Just as I finish adding all that info.

Haha, that had me tripping there for a minute.

Totally agree with what you said about the same salt being present within the script, and how that could work very much against me. Going to write up a password function now with password_hash(). Thanks for the informative posts!

Haxalot - January 25, 2014, 9:36 am

(Jan 25, 2014 09:03 AM)VipVince Wrote: [\[CITE\]](#) New code:

PHP Code

```
<?php
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Registration page</title>
```

```
</head>
```

```
<body>
```

```
<form method="POST">
```

```
Username:<input type="text" name="username"/><br />
```

```
Password:<input type="password" name="password"/><br />
```

```
Re-Enter Password:<input type="password" name="password2"/><br />
```

```
Email:<input type="text" name="email"/><br />
```

```
<input type="submit" name="submit" value="Register" />
```

```
</form>
```

```
</body>
```

```
</html>
```

```
<?php
```

```
if (isset($_POST['submit'])) {
```

```
    $dbhost    = "localhost";
```

```
    $dbname    = "vipvincemembers";
```

```
    $dbuser    = "root";
```

```
    $dbpass    = "";
```

```
    // database connection
```

```
    $conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);
```

```
    if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']) || empty($_POST['email']))
```

```
    {
        die("Error: All fields must be filled in!");
    }
```

```
else
```

```
{
```

```
    $username = $_POST['username'];
```

```
    $password = $_POST['password'];
```

```
    $password2 = $_POST['password2'];
```

```
    $email= $_POST['email'];
```

```
    $userIp = ($_SERVER["REMOTE_ADDR"]);
```

```
    $joinDate = date("Y-m-d H:i:s");
```

```
    if(!filter_var($email, FILTER_VALIDATE_EMAIL))
```

```
    {
        die("Error: E-mail address is not valid!");
    }
```

```
    if (strlen($username)>15) {
```

```
        die("Username character limit exceeded!");
```

```
    }
```

```
if (strlen($username)<6) {  
die("Username must contain more than 5 characters!");  
}
```

```
if (strlen($password)<6) {  
die("Password must contain more than 5 characters!");  
}
```

```
if (strlen($email)>30) {  
die("Email character limit exceeded!");  
}
```

```
//secure the password  
$salt = '-45dfeHK/___yu349@-/klF21-1_V4JkUP/4';  
$hashedpass = sha1($salt.$password);  
$hashedpass2 = sha1($salt.$password2);
```

```
if($hashedpass !== $hashedpass2)  
{  
die("Error: Passwords do not match!");  
}
```

```
//check if the username is already registered. we use prepare() to get the data ready for assignment  
$stmt = $conn->prepare(  
"SELECT COUNT(`username`) FROM members WHERE `username` = :name");
```

```
//notice we use :name which is a placeholder for the username variable value, we execute the comm  
and.  
$stmt->execute(array(':name' => $username));
```

```
//we use pdos fetch to see if its equal to 1 which means it exists in the db  
if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)  
{  
die("Username: ".htmlentities($username)." already exists in our database!");  
}
```

```
//check if the email is already registered. Again we start with prepare.  
$stmz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");
```

```
//execute() again assigning the placeholder with the variable data  
$stmz->execute(array(':email' => $email));
```

```
if ($stmz->fetch(PDO::FETCH_COLUMN) >= 1)  
{  
die("Email: ".htmlentities($email)." already exists in our database!");  
}
```

```
//get data ready for db insert, notice the insert into command in caps.  
$sql =
```

```
"INSERT INTO members (username, password, email, ip, joindate) VALUES (:username, :password  
, :email, :ip, :joindate)";
```

```
//prepare statement before inserting.
```

```
$q = $conn->prepare($sql);
```

```
//execute statement, with variable data into placeholders
```

```
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, ':email'=>$email, ':ip'=>$userIp, ':joindate'=>$joinDate));
```

```
echo "<b>Thank you for registering ".htmlentities($username).
```

```
"! <a href='login.php'>Click here to login</a></b>";
```

```
}
```

```
}
```

```
?>
```

Any feedback negative or positive would be appreciated, I am aware of the password hashing being insecure, will be fixing that.

I've got a few improvements. I'll start at the top of the script and work my way down.

First things first, you should be specifying the character set in your HTML code:

Code <!DOCTYPE html>

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8" />
```

```
<title>Registration page</title>
```

```
</head>
```

Connection details are usually stored in a separate file (so that it can be globally included into other files), preferably outside of the public document root.

You don't need your ELSE statement after your IF expression where you check to ensure all fields aren't empty. You're killing the page if any fields are empty.

You've then made a number of pointless variable assignments from the \$_POST associative array. You usually validate the data first, and then reassign the POST data so that you can tell the difference between validated variables and potentially tainted ones.

You're limiting email length to 30 characters long. Emails can be much longer than that.

Like you've already said, your password hashing is poor.

It would be more optimal for you to firstly compare the two input passwords, and then hash only one of them if they validate.

In your parametrised query, you're COUNTing the username attribute; it would be more optimal to count an indexed attribute.

You haven't closed the body and html tags.

Perhaps the next step for you would be to start encapsulating the functionality of input validation into a class?

VipVince - January 25, 2014, 10:00 am

Thank you will check them points out once I get this resolved. Basically I am getting login unsuccessful when I log in with the right credentials.

```
Register.php
PHP Code
<?php
<!DOCTYPE html>
<html>

<head>
<title>Registration page</title>
</head>

<body>
<form method="POST">
Username:<input type="text" name="username"/><br />
Password:<input type="password" name="password"/><br />
Re-Enter Password:<input type="password" name="password2"/><br />
Email:<input type="text" name="email"/><br />
<input type="submit" name="submit" value="Register" />
</form>
</body>
</html>

<?php
require '/password.php';

if (isset($_POST['submit'])) {

$dbhost    = "localhost";
$dbname    = "vipdb";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

if (empty($_POST['username']) || empty($_POST['password']) || empty($_POST['password2']
)) || empty($_POST['email'] ))
{
    die("Error: All fields must be filled in!");
}

else
{
```

```

$username = $_POST['username'];
$password = $_POST['password'];
$password2 = $_POST['password2'];
$email = $_POST['email'];
$userIp = ($_SERVER["REMOTE_ADDR"]);
$joinDate = date("Y-m-d H:i:s");

//secure the password
$hashedpass = password_hash($password, PASSWORD_DEFAULT);
$hashedpass2 = password_hash($password2, PASSWORD_DEFAULT);

//compare the plain text passwords
if($password !== $password2)
{
    die("Error: Passwords do not match!");
}

//check if email is valid
if(!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    die("Error: E-mail address is not valid!");
}

if (strlen($username)>15) {
    die("Username character limit exceeded!");
}

if (strlen($username)<6) {
    die("Username must contain more than 5 characters!");
}

if (strlen($password)<6) {
    die("Password must contain more than 5 characters!");
}

if (strlen($email)>30) {
    die("Email character limit exceeded!");
}

//check if the username is already registered. we use prepare() to get the data ready for assignment
$stmt = $conn->prepare(
    "SELECT COUNT(`username`) FROM members WHERE `username` = :name");

//notice we use :name which is a placeholder for the username variable value, we execute the comm
and.
$stmt->execute(array(':name' => $username));

//we use pdo's fetch to see if its equal to 1 which means it exists in the db
if ($stmt->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Username: ".htmlentities($username)." already exists in our database!");
}

```

```

//check if the email is already registered. Again we start with prepare.
$stmtz = $conn->prepare("SELECT COUNT(`email`) FROM members WHERE `email` = :email");

//execute() again assigning the placeholder with the variable data
$stmtz->execute(array(':email' => $email));

if ($stmtz->fetch(PDO::FETCH_COLUMN) >= 1)
{
    die("Email: ".htmlentities($email)." already exists in our database!");
}

//get data ready for db insert, notice the insert into command in caps.
$sql =
"INSERT INTO members (username, password, email, ip, joindate) VALUES (:username, :password
, :email, :ip, :joindate)";

//prepare statement before inserting.
$q = $conn->prepare($sql);

//execute statement, with variable data into placeholders
$q->execute(array(':username'=>$username, ':password'=>$hashedpass, ':email'=>$email, ':ip'=>
$userIp,
':joindate'=>$joinDate));

echo "<b>Thank you for registering ".htmlentities($username).
"! <a href='login.php'>Click here to login</a></b>";

}
}
?>
login.php
PHP Code
<?php
<!DOCTYPE html>
<html>

<head>
<title>Login page</title>
</head>
<h1>Please Log In</h1>
<body>
<form method="POST">
Username:<input type="text" name="username"/><br />
Password:<input type="password" name="password"/><br />
<input type="submit" name="submit" value="Login" />
</form>
</body>
</html>

<?php
session_start();
require '/password.php';

```

```

if (isset($_POST['submit'])) {

$dbhost    = "localhost";
$dbname    = "vipdb";
$dbuser    = "root";
$dbpass    = "";

// database connection
$conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

$username = $_POST['username'];
$password = $_POST['password'];
$password = password_hash($password, PASSWORD_DEFAULT);

if($username == "") {
die("You must enter a username!");
}
if($password == "") {
die("You must enter a password!");
}
// query
$result = $conn->prepare("SELECT password FROM members WHERE username = :user");
$result->execute(array(':user' => $username));
$row = $result->fetch(PDO::FETCH_ASSOC);
if (password_verify($password, $row['password']))
{
    $_SESSION['username'] = $username;
    header("location:home.php");
}
else
{
    echo "Login unsuccessful!";
}
}

?>

```

The registration works fine, just the login always gives me unsuccessful.

Hoping to get this resolved.

Haxalot - January 25, 2014, 11:02 am

You're incorrectly implementing the [password_verify\(\)](#) function. You should be passing the input password (in plain text) as the first parameter, and then the hash as the second. Try deleting the following line and tell me if it works:

PHP Code

```
<?php
```

```
$password = password_hash($password, PASSWORD_DEFAULT);
```

VipVince - January 25, 2014, 11:09 am

(Jan 25, 2014 12:02 PM)Haxalot Wrote: [\[CITE\]](#) You're incorrectly implementing the [password_verify\(\)](#) function. You should be passing the input password (in plain text) as the first parameter, and then the hash as the second. Try deleting the following line and tell me if it works:

PHP Code

```
<?php
$password = password_hash($password, PASSWORD_DEFAULT);
```

Took your advice, but still login unsuccessful.

Script without that line:

PHP Code

```
<?php
<!DOCTYPE html>
<html>

<head>
<title>Login page</title>
</head>
<h1>Please Log In</h1>
<body>
<form method="POST">
Username:<input type="text" name="username"/><br />
Password:<input type="password" name="password"/><br />
<input type="submit" name="submit" value="Login" />
</form>
</body>
</html>

<?php
session_start();
require '/password.php';

if (isset($_POST['submit'])) {

    $dbhost    = "localhost";
    $dbname    = "vipdb";
    $dbuser    = "root";
    $dbpass    = "";

    // database connection
    $conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

    $username = $_POST['username'];
    $password = $_POST['password'];

    if($username == "") {
        die("You must enter a username!");
    }
    if($password == "") {
        die("You must enter a password!");
    }
    // query
    $result = $conn->prepare("SELECT password FROM members WHERE username = :user");
    $result->execute(array(':user' => $username));
    $row = $result->fetch(PDO::FETCH_ASSOC);
    if (password_verify($password, $row['password']))
```

```

{
    $_SESSION['username'] = $username;
    header("location:home.php");
}
else
{
    echo "Login unsuccessful!";
}
}
}

?>

```

Haxalot - January 25, 2014, 11:15 am

(Jan 25, 2014 12:09 PM)VipVince Wrote: [\[CITE\]](#) Took your advice, but still login unsuccessful.

Script without that line:

PHP Code

```

<?php
<!DOCTYPE html>
<html>

<head>
<title>Login page</title>
</head>
<h1>Please Log In</h1>
<body>
<form method="POST">
Username:<input type="text" name="username"/><br />
Password:<input type="password" name="password"/><br />
<input type="submit" name="submit" value="Login" />
</form>
</body>
</html>

<?php
session_start();
require '/password.php';

if (isset($_POST['submit'])) {

    $dbhost    = "localhost";
    $dbname    = "vipdb";
    $dbuser    = "root";
    $dbpass    = "";

    // database connection
    $conn = new PDO("mysql:host=$dbhost;dbname=$dbname",$dbuser,$dbpass);

    $username = $_POST['username'];
    $password = $_POST['password'];

    if($username == "") {
        die("You must enter a username!");
    }
}

```

```

}
if($password == "") {
die("You must enter a password!");
}
// query
$result = $conn->prepare("SELECT password FROM members WHERE username = :user");
$result->execute(array(':user' => $username));
$row = $result->fetch(PDO::FETCH_ASSOC);
if (password_verify($password, $row['password']))
{
    $_SESSION['username'] = $username;
    header("location:home.php");
}
else
{
    echo "Login unsuccessful!";
}
}

?>

```

Upon registration, did everything get correctly inserted into the database (i.e. it actually has the user entered data, and that the password field doesn't truncate values - it should be 60 characters long). Also, have you tried performing a `var_dump()` upon the `$row` variable to see what's coming out?

VipVince - January 25, 2014, 11:20 am

Yep, the issue must be in the login.php script.

Here I just registered a test account, the password is "testing" hashed.

[img]http://#/f1pGil.jpg[/img]

So it inserted fine etc, then when I go over to login.php and enter the credentials, it is telling me unsuccessful. So I assume the issue is in my login script.

[img]http://#/bAusQj.jpg[/img]

And `var_dump` result:

PHP Code

```
<?php
```

```
array(1) { ["password"]=> string(50)
```

```
"$2y$10$8s2Z1ZHWXWfASoJshzdVseMumdCzcZcbszkrAQDMjR1" } Login unsuccessful!
```

Anxuein - January 25, 2014, 11:23 am

Check the hash you're getting on the login script and compare it to what is in the database.

PHP Code

```
<?php
```

```
if (isset($_POST['submit'])) { print_r($_POST); }
```

Haxalot - January 25, 2014, 11:25 am

Like I said above, can you provide a `var_dump()` of the `$row` variable so that we can see what's

being retrieved from the database? Can I also see your table structure (the fields and what types they're set to).

VipVince - January 25, 2014, 11:26 am

Result from login query:

PHP Code

```
<?php
```

```
Array ( [username] => testaccount [password] => testing [submit] => Login ) array(1) { ["password"] => string(50) "$2y$10$8s2Z1ZHWXWfASoJshzdVseMumdCzcZcbszkrAQDMjR1" }
```

Login unsuccessful!

Database value:

PHP Code

```
<?php
```

```
$2y$10$8s2Z1ZHWXWfASoJshzdVseMumdCzcZcbszkrAQDMjR1
```

Strange.

(Jan 25, 2014 12:25 PM)Haxalot Wrote: [\[CITE\]](#) Like I said above, can you provide a var_dump() of the \$row variable so that we can see what's being retrieved from the database? Can I also see your table structure (the fields and what types they're set to).

I posted the var_dump just above mate, also here is my table structure.

PHP Code

```
<?php
```

```
CREATE TABLE IF NOT EXISTS members (
```

```
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, email VARCHAR(30),
```

```
username VARCHAR(30), password CHAR(50), ip CHAR(50), joindate CHAR(50))
```

Haxalot - January 25, 2014, 11:36 am

(Jan 25, 2014 12:15 PM)Haxalot Wrote: [\[CITE\]](#) Upon registration, did everything get correctly inserted into the database (i.e. it actually has the user entered data, and that the password field doesn't truncate values - it should be 60 characters long). Also, have you tried performing a var_dump() upon the \$row variable to see what's coming out?

There's your answer (in bold above). Your database is truncating the inserted password hash (from 60 characters to 50), and so you're only returning the first 50 characters of the hash from your database. Thus, when you try to compare the plain text password to the hash returned from your database, it'll never validate.

What you need to do is to update the password attribute or your members table to 60 characters:

Code ALTER TABLE members MODIFY COLUMN password CHAR(60);

VipVince - January 25, 2014, 11:46 am

(Jan 25, 2014 12:36 PM)Haxalot Wrote: [\[CITE\]](#) There's your answer (in bold above). Your database is truncating the inserted password hash (from 60 characters to 50), and so you're only returning the first 50 characters of the hash from your database. Thus, when you try to compare the plain text password to the hash returned from your database, it'll never validate.

What you need to do is to update the password attribute or your members table to 60 characters:

Code ALTER TABLE members MODIFY COLUMN password CHAR(60);

And there was me skimming down every line of code in the login file lol, well spotted man. Altered it and its now working perfect. Thank you.

Anybody free for some assistance, its a private matter however, and will have to send via pm.