

MySQL Tutorials

PHP, MySQL and Authentication 101

Contributed by Havard Lindset

2002-07-07

[[Send Me Similar Content When Posted](#)]

[[Add Developer Shed Headlines To Your Site](#)]



[DISCUSS](#) [NEWS](#) [SEND](#) [PRINT](#) [PDF](#)

advertisement

[Access Your PC from Anywhere - Free Download](#)

Article Index:

Authentication is required by any online admin system. In this article, Havard introduces us to two types of PHP/MySQL authentication: dialogs and forms. When you're making a dynamic site in PHP, you might want to restrict an area from normal users and grant access only to a set of trusted users. You wouldn't want to make the admin area open for everybody now, would you?

To restrict an area you need to create some sort of authentication method, and how you do this is one of the questions I've seen get asked a lot. Throughout this article I intend to show you a few different approaches to authentication with PHP and MySQL. We will be making use of a MySQL database to store the usernames and passwords of our authenticated users. Firstly we will have to set up the database and its respective tables. Run this set of commands through the MySQL console application:

```
CREATE DATABASE mydatabase;  
USE mydatabase;
```

```
CREATE TABLE users {  
  userId SMALLINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,  
  userName VARCHAR(30) NOT NULL,  
  userPass VARCHAR(32) NOT NULL,  
  PRIMARY KEY (userId),  
  UNIQUE KEY username (username)  
}
```

The code above creates a database containing a table named users. We made the userName column a unique key to prevent having two users with the same username. Let's insert a user into the database, so we have something to authenticate against:

```
INSERT INTO users (userName, userPass) VALUES ('testUser', MD5('testPass'));
```

You may want to change the values for the username and password. The MD5() function is a built-in MySQL function, which calculates a 128 bit checksum for the provided string. The returned string is 32 characters long, hence we used VARCHAR(32) for the userPass column. We will be using this table through the whole article.

Now that we've created the database, table and a user, we can continue.

You should have PHP version 4.1.0 or above. If you have an earlier version you'll have to rewrite some of the code. This is because I'm using super global arrays such as `$_SESSION` and `$_SERVER`, which were introduced in PHP version 4.1.0.

HTTP Authentication

If PHP is installed as an Apache module, then you can use PHP's HTTP Authentication hook to pop up a username/password authentication window in the browser. This is done by sending some special parameters in the `header()` function. When the user has filled in both the username and password fields, the values can be accessed within a PHP script using the variables `$PHP_AUTH_USER` and `$PHP_AUTH_PW`.

Remember that this type of authentication only works when PHP is installed as an apache-module, which means that if you are using the CGI version, you can skim through this part of the article as we'll be discussing authentication through forms on the next page.

Let's take a look at some sample code:

```
<?PHP

function displayLogin() {
header("WWW-Authenticate: Basic realm=\"My Website\"");
header("HTTP/1.0 401 Unauthorized");
echo "<h2>Authentication Failure</h2>";
echo "The username and password provided did not work. Please reload this page and try again.";
exit;
}

$db = mysql_connect('localhost','dbuser','dbpass') or die("Couldn't connect to the database.");
mysql_select_db('dbname') or die("Couldn't select the database");

if (!isset($PHP_AUTH_USER) || !isset($PHP_AUTH_PW)) {
// If username or password hasn't been set, display the login request.
displayLogin();
} else {
// Escape both the password and username string to prevent users from inserting bogus data.
$PHP_AUTH_USER = addslashes($PHP_AUTH_USER);
$PHP_AUTH_PW = md5($PHP_AUTH_PW);

// Check username and password against the database.
$result = mysql_query("SELECT count(id) FROM users WHERE password='$PHP_AUTH_PW'
AND username='$PHP_AUTH_USER'") or die("Couldn't query the user-database.");
$num = mysql_result($result, 0);

if (!$num) {
// If there were no matching users, show the login
displayLogin();
}
}

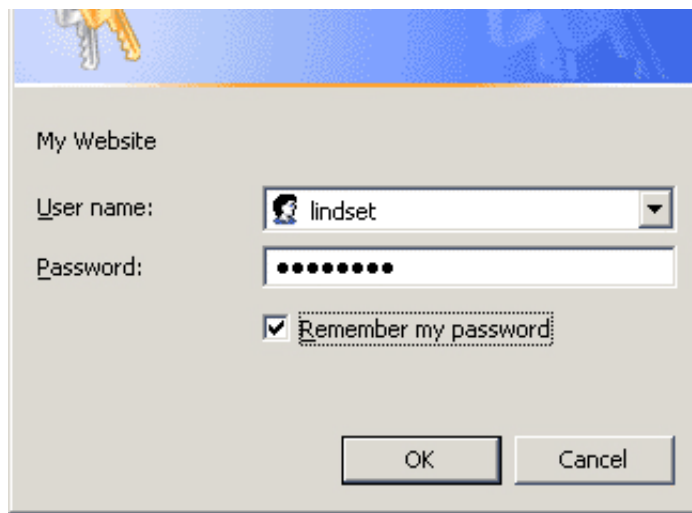
// All code/html below will only be displayed to authenticated users.

echo "Congratulations! You're now authenticated.";

?>
```

The code above produces a dialog authentication window, which looks like this:





Let's take a closer look at the different

parts of this example.

```
function displayLogin() {
header("WWW-Authenticate: Basic realm=\"My Website\"");
header("HTTP/1.0 401 Unauthorized");
echo "<h2>Authentication Failure</h2>";
echo "The username and password provided did not work. Please reload this page and try again.";
exit;
}
```

This function is called when either `$PHP_AUTH_USER` or `$PHP_AUTH_PW` isn't set, and when the MySQL query didn't return anything. The first header calls the browser's authentication window, while the second header tells the browser what type of error has occurred. Everything between the last header and "exit;" will be displayed to the user in case the authentication failed, or cancel was pressed in the authentication window.

The realm name must remain the same on all of your pages. If it doesn't, the browser will require authentication for all unvisited realms.

```
if (!isset($PHP_AUTH_USER) || !isset($PHP_AUTH_PW)) {
// If username or password hasn't been set, display the login request.
displayLogin();
} else {
// Escape both the password and username string to prevent users from inserting bogus data.
$PHP_AUTH_USER = addslashes($PHP_AUTH_USER);
$PHP_AUTH_PW = md5($PHP_AUTH_PW);

// Check username and password against the database.
$result = mysql_query("SELECT count(id) FROM users WHERE password='$PHP_AUTH_PW'
AND username='$PHP_AUTH_USER'") or die("Couldn't query the user-database.");
$num = mysql_result($result, 0);

if (!$num) {
// If there were no matching users, show the login
displayLogin();
}
}
```

In this code we check if `$PHP_AUTH_USER` or `$PHP_AUTH_PW` hasn't been set. If they haven't been set, then we call the `displayLogin()` function. If both the username and password have been set, we authenticate them against our database. By the way, we're now using the built-in `md5` function in PHP to create a `md5` checksum, instead of using the MySQL function.

If the user wasn't found in the database, we call the `displayLogin()` function.

We use the addslashes() function to escape the variables that are used in the MySQL query. By doing this, we prevent the user from entering bogus data, which in the worst case could cause havoc on your database.

All code below the if construct will only be displayed to authenticated users.

Place the code above in a .php file, and include it in every page you want authentication on. This way you only have to edit one file in case you need to make some changes to the authentication code.

What about logging out?

If you'd like to make a logout function, you can use some PHP code like this:

```
if ($_REQUEST['logout'] == true) {  
    // To logout a user, you can just use the displayLogin() function and resend the authentication  
    headers.  
    displayLogin();  
}
```

By calling the displayLogin() function when the user is already logged in, we cause the browser to display the authentication window, and clear any previous successful authentication. This works on most browsers. To log out with the code above you can add ? logout=true to the URL.

The only problem I can see with this type of authentication is that it's not available in the CGI version of PHP. Although most servers run PHP as a module, some don't, and that would mean trouble for your authentication script. Continue reading to learn another approach. If you would like a more aesthetic approach to authentication, you may want to allow the user to log in using a HTML form. This is probably the most popular approach. We will be using sessions, so the user doesn't have to re-authenticate on every page that requires authentication.

Put the following code in a file called login.php:

```
<?PHP
```

```
$db = mysql_connect('localhost', 'dbuser', 'dbpass') or die("Couldn't connect to the database.");  
mysql_select_db('dbname') or die("Couldn't select the database");
```

```
// Add slashes to the username, and make a md5 checksum of the password.  
$_POST['user'] = addslashes($_POST['user']);  
$_POST['pass'] = md5($_POST['pass']);
```

```
$result = mysql_query("SELECT count(id) FROM users WHERE password='$_POST[pass]'  
AND username='$_POST[user]'" ) or die("Couldn't query the user-database.");  
$num = mysql_result($result, 0);
```

```
if (!$num) {
```

```
    // When the query didn't return anything,  
    // display the login form.
```

```
    echo "<h3>User Login</h3>  
    <form action='$_SERVER[PHP_SELF]' method='post'  
    Username: <input type='text' name='user'><br>  
    Password: <input type='password' name='pass'><br><br>  
    <input type='submit' value='Login'>  
    </form>";
```

```
} else {
```

```

// Start the login session
session_start();

// We've already added slashes and MD5'd the password
$_SESSION['user'] = $_POST['user'];
$_SESSION['pass'] = $_POST['pass'];

// All output text below this line will be displayed
// to the users that are authenticated. Since no text
// has been output yet, you could also use redirect
// the user to the next page using the header() function.
// header('Location: page2.php');

echo "<h1>Congratulations</h1>";
echo "You're now logged in. Try visiting <a href='page2.php'>Page 2</a>.";

}

?>

```

Let's take a closer look at some parts of the code:

```

$db = mysql_connect('localhost', 'dbuser', 'dbpass') or die("Couldn't connect to the database.");
mysql_select_db('dbname') or die("Couldn't select the database");

// Add slashes to the username, and make a md5 checksum of the password.
$_POST['user'] = addslashes($_POST['user']);
$_POST['pass'] = md5($_POST['pass']);

$result = mysql_query("SELECT count(id) FROM users WHERE password='$_POST[pass]'
AND username='$_POST[user]'") or die("Couldn't query the user-database.");
$num = mysql_result($result, 0);

if (!$num) {

// When the query didn't return anything,
// display the login form.

echo "<h3>User Login</h3>
<form action='$_SERVER[PHP_SELF]' method='post'>
Username: <input type='text' name='user'><br>
Password: <input type='password' name='pass'><br><br>
<input type='submit' value='Login'>
</form>";

```

This code connects to the database, and prepares the variables for the SQL query. After the data is prepared, we're querying the database for the information entered in the form. If the query doesn't return anything, we display the login form. Instead of hard coding the form, you could also make the form a .html file, and just include() it.

```

} else {

// Start the login session
session_start();

// We've already added slashes and MD5'd the password
$_SESSION['user'] = $_POST['user'];
$_SESSION['pass'] = $_POST['pass'];

// All output text below this line will be displayed

```

```
// to the users that are authenticated.
```

```
echo "<h1>Congratulations</h1>";  
echo "You're now logged in. Try visiting <a href='page2.php'>Page 2</a>.";  
  
}
```

This part gets executed when the information entered matched a user. We're starting a session through using the `session_start()` function, and then we're adding the session variables `$_SESSION['user']` and `$_SESSION['pass']`. Since we've already added the slashes, and made the password an MD5 checksum, we'll just add them as they are. By the way, since we're using sessions, the login information will be deleted when you exit your browser. You may implement a normal cookie here too, so that it stays on your machine until it either expires, or the user deletes it manually.

Since there hasn't been any output anything to the browser just yet, we can redirect the user using `header()` redirection instead of displaying text. Just replace the text with this:
`header('Location: page2.php');`

Now it's time to take a look at `page2.php`, which we linked to from `login.php`. Insert the following code into a file called `page2.php`:

```
<?PHP  
  
// Start the login session  
session_start();  
  
if (!$_SESSION['user'] || !$_SESSION['pass']) {  
  
    // What to do if the user hasn't logged in  
    // We'll just redirect them to the login page.  
    header('Location: login.php');  
    die();  
  
} else {  
  
    // If the session variables exist, check to see  
    // if the user has access.  
  
    $db = mysql_connect('localhost', 'dbuser', 'dbpass') or die("Couldn't connect to the database.");  
    mysql_select_db('dbname') or die("Couldn't select the database");  
  
    $result = mysql_query("SELECT count(id) FROM users WHERE password='$_SESSION[pass]'  
    AND username='$_SESSION[user]'") or die("Couldn't query the user-database.");  
    $num = mysql_result($result, 0);  
  
    if (!$num) {  
        // If the credentials didn't match,  
        // redirect the user to the login screen.  
        header('Location: login.php');  
        die();  
    }  
}  
  
// All output text below this line will be displayed  
// to the users that are authenticated.  
  
echo "<h1>Access Granted</h1>";  
echo "You see? It travelled over these two pages.<br><br>";  
echo "You are authenticated as " . $_SESSION['user'] . "<br>";  
echo "The MD5 checksum of your password is " . $_SESSION['pass'];
```

?>

As usual, we'll take a closer look at the code:

```
// Start the login session
session_start();

if (!$_SESSION['user'] || !$_SESSION['pass']) {

    // What to do if the user hasn't logged in
    // We'll just redirect them to the login page.
    header('Location: login.php');
    die();
}
```

In this snippet we're checking to see if the session variables have been set. If they haven't, then we redirect them to the login.php again. In case you're wondering why we're using die after the header(), it's for extra security. A hacker can for example make his own browser that ignores header redirects. Better safe than sorry.

```
} else {

    // If the session variables exist, check to see
    // if the user has access.

    $db = mysql_connect('localhost', 'dbuser', 'dbpass') or die("Couldn't connect to the database.");
    mysql_select_db('dbname') or die("Couldn't select the database.");

    $result = mysql_query("SELECT count(id) FROM users WHERE password=$_SESSION[pass]
    AND username=$_SESSION[user]") or die("Couldn't query the user-database.");
    $num = mysql_result($result, 0);

    if (!$num) {
        // If the credentials didn't match,
        // redirect the user to the login screen.
        header('Location: login.php');
        die();
    }
}
```

This code is almost exactly the same as login.php. We don't have to add slashes here because they were already added in login.php. Again, you can see we're using die() after the header() redirect.

```
// All output text below this line will be displayed
// to the users that are authenticated.

echo "<h1>Access Granted</h1>";
echo "You see? It travelled over these two pages.<br><br>";
echo "You are authenticated as " . $_SESSION['user'] . "<br>";
echo "The MD5 checksum of your password is " . $_SESSION['pass'];
```

This is just placeholder text. Feel free to replace it with whatever you want.

Try authenticating yourself, and see how the session transfers the login information between the pages.

Instead of copying the code in page2.php into all pages you want authentication on, you can name it auth.php, and include() it in all of the pages you want authentication on.

All you have to do to delete the session data, thus logging yourself out, is to make a PHP

script with this code:

```
<?PHP  
session_start();  
session_destroy();
```

```
echo "You have been successfully logged out.";
```

?> You should now be able to protect your pages using PHP/MySQL authentication. Once you get into it, you'll see for yourself how valuable this can be. You may also have learned some precautions to take when querying a database. Basic rule: Always prepare variables before using them in SQL queries.

Suggestions for further expansion:

- ▶ Multiple access levels
- ▶ "Lifetime" cookies that keeps the users logged in even when they close the browser
- ▶ Make an auth class using OOP (I'm working on one!)
- ▶ User management (add, edit, remove users)

I could have included all of this functionality in the tutorial, but if I did that, there wouldn't be any fun left for you. As always, feel free to ask questions or discuss this article in the forums.