Should I mysql_real_escape_string all the cookies I get from the user to avoid mysql injection in php?

Asked 16 years, 3 months ago Modified 8 years, 3 months ago

Viewed 700 times Part of PHP Collective

2

When a user goes to my site, my script checks for 2 cookies which store the user id + part of the password, to automatically log them in.

It's possible to edit the contents of cookies via a cookie editor, so I guess it's possible to add some malicious content to a written cookie?

Should I add `mysql_real_escape_string` (or something else) to all my cookie calls or is there some kind of built in procedure that will not allow this to happen?

PHP php validation

Share

Improve this question

Follow

edited Aug 26, 2016 at 15:31

hlh3406

1,400 ● 5 ● 31 ● 47

asked Sep 18, 2008 at 6:34

user15063

9 Answers

Sorted by: Highest score (default)

8

What you *really* need to do is not send these cookie values that are hackable in the first place. Instead, why not hash the username and password and a (secret) salt and set that as the cookie value? i.e.:

```
define('COOKIE_SALT', 'secretblahblahlkdsfklj');
$cookie_value = sha1($username.$password.COOKIE_SALT);
```

Then you know the cookie value is always going to be a 40-character hexidecimal string, and can compare the value the user sends back with whatever's in the database to decide whether they're valid or not:

```
if ($user_cookie_value ==
sha1($username_from_db.$password_drom_db.COOKIE_SALT))
  # valid
} else {
  #not valid
}
```

`mysql_real_escape_string` makes an additional hit to the database, BTW (a lot of people don't realize it requires a DB connection and queries MySQL).

The best way to do what you want if you can't change your app and insist on using hackable cookie values is to use prepared statements with bound parameters.

Share Improve this answer

Follow

answered Sep 18, 2008 at 6:57

joelhardi

11.2k ● 3 ● 35 ● 38

If I combine the username + pass into a single hashed string, how will I know what to compare it to in the database? If I have several hundred thousand users, Im not going to convert all the usernames + passwords into a hash, just to compare it with every single login attempt. – user15063 Sep 18, 2008 at 7:04

Well, you can set a cookie that is just their username so you will know which username to look up and compare against. Then you validate whatever value they send thru whatever username-validator function you have, look them up in the DB, and then do the comparison in my second example. – joelhardi Sep 18, 2008 at 7:11

Good point about the connecting/querying the database. I had forgotten about this. – J D OConal Sep 18, 2008 at 7:12

This isn't replacing the user supplying their username and password when they first log in (and you set some kind of session cookie), btw. It's just about not setting cookie values that say "hack me". – joelhardi Sep 18, 2008 at 7:16

JD, yeah, there used to be mysql_escape_string that didn't hit the database, then they added mysql_real_escape_string. Maybe they need to call it mysql_really_real_escape_string so people remember it uses the DB?!?! Just a little joke at the expense of PHP's ridiculous function names there. :) – joelhardi Sep 18, 2008 at 7:25

The point of mysql_real_escape_string isn't to protect against injection attacks, it's to ensure your data is

3

accurately stored in the database. Thus, it should be called on ANY string going into the database, regardless of its source.

You should, however, *also* be using parameterized queries (via mysqli or PDO) to protect yourself from SQL injection. Otherwise you risk ending up like little Bobby Tables' school.

Share Improve this answer

Follow

answered Sep 18, 2008 at 7:03

Nathan Strong

2,400 ● 13 ● 17

1

I only use mysql_real_escape_string before inserting variables into an SQL statement. You'll just get yourself confused if some of your variables are *already* escaped, and then you escape them again. It's a classic bug you see in newbies' blog webapps:

> When someone writes an apostrophe it keeps on adding slashes ruining the blog\\\\\\\'s pages.

The value of a variable isn't dangerous by itself: it's only when you put it into a string or something similar that you start straying into dangerous waters.

Of course though, never trust anything that comes from the client-side.

Share Improve this answer

answered Sep 18, 2008 at 6:42

Follow

nickf

546k ● 198 ● 658 ● 725

Well, I do run the info in the cookie through a SELECT statement, to check if the login data is valid. – user15063 Sep 18, 2008 at 6:44

ah well then yes - mysql_real_escape_string is the way to go. :) – nickf Sep 18, 2008 at 6:49

1

Prepared statements and parameter binding is always a good way to go.

PEAR::MDB2 supports prepared statements, for example:

```
$db = MDB2::factory( $dsn );

$types = array( 'integer', 'text' );
$sth = $db->prepare( "INSERT INTO table (ID,Text) (?,?
if( PEAR::isError( $sth ) ) die( $sth->getMessage() );

$data = array( 5, 'some text' );
$result = $sth->execute( $data );
$sth->free();
if( PEAR::isError( $result ) ) die( $result->getMessag
```

This will only allow proper data and pre-set amount of variables to get into database.

You of course should validate data before getting this far, but preparing statements is the final validation that should be done.

Share Improve this answer

Follow

answered Sep 18, 2008 at 9:07

myyra

0

You should mysql_real_escape_string ***anything*** that could be potentially harmful. Never trust any type of input that can be altered by the user.

Share Improve this answer

Follow

answered Sep 18, 2008 at 6:36

Jeremy Privett

4,455 ● 2 ● 33 ● 35

0

I agree with you. It is possible to modify the cookies and send in malicious data.

I believe that it is good practice to filter the values you get from the cookies before you use them. As a rule of thumb I do filter any other input that may be tampered with.

Share Improve this answer

Follow

answered Sep 18, 2008 at 6:38

Marcel

6,290 ● 15 ● 49 ● 52

0

Yegor, you can store the hash when a user account is created/updated, then whenever a login is initiated, you hash the data posted to the server and compare against what was stored in the database for that one username.

(Off the top of my head in loose php - treat as pseudo code):

```
$usernameFromPostDbsafe = LimitToAlphaNumUnderscore($u
$result = Query("SELECT hash FROM userTable WHERE
username='$usernameFromPostDbsafe' LIMIT 1;");
$hashFromDb = $result['hash'];
if( (sha1($usernameFromPost.$passwordFromPost.SALT)) =
    //Auth Success
}else{
   //Auth Failure
}
```

After a successful authentication, you could store the hash in $_SESSION or in a database table of cached authenticated username/hashes. Then send the hash back to the browser (in a cookie for instance) so subsequent page loads send the hash back to the server to be compared against the hash held in your chosen session storage.

Share Improve this answer Follow

edited Sep 18, 2008 at 7:34

answered Sep 18, 2008 at 7:25

micahwittman

12.5k ● 2 ● 34 ● 37

0

mysql_real_escape_string is so passé... These days you should really use parameter binding instead.

I'll elaborate by mentionning that i was referring to prepared statements and provide a link to an article that demonstrates that sometimes mysl_real_escape_string isn't sufficient enough: http://www.webappsec.org/projects/articles/091007.txt

Share Improve this answer Follow

edited Sep 18, 2008 at 11:08

answered Sep 18, 2008 at 6:43

timvw

346 ● 1 ● 4 ● 9

elaborate? You mean like intval for numeric strings, and so on? – user15063 Sep 18, 2008 at 6:45

-1

I would recommend using htmlentities($input, ENT_QUOTES) instead of mysql_real_escape_string as this will also prevent any accidental outputting of actual HTML code. Of course, you could use mysql_real_escape_string and htmlentities, but why would you?

Share Improve this answer Follow

answered Sep 18, 2008 at 6:49

J D OConal

624 ● 4 ● 14