PREVENTING HACKING ATTEMPTS

Prevent hacking attempts

- It is dangerous to pass user input unchecked to MySQL
  - Example

```
$user = $_POST[‘user’];
$pass = $_POST[‘pass’];
$query = “SELECT * FROM users WHERE
user=‘$user’ AND pass = ‘$pass’”;
```

Prevent hacking attempts

Username	Password	Query String
fredsmith	mypass	SELECT * FROM users WHERE user=‘fredsmith’ AND pass=‘mypass’
admin’ #		SELECT * FROM users WHERE user=‘admin’ #’ AND pass=‘’
abc’ OR 1=1 #		SELECT * FROM users WHERE user=‘abc’ OR 1=1 #’ AND pass=‘’

Case1: Normal case
Case2: In MySQL, # is the start of the comment, so user only needs to input user name and will be able to access the database without having to input password.
Case3: There even no need to know a username in order to access the system.

Preventing hacking attempts

- A more dangerous case which is to delete the user out from the system
  - Example

```
$user = $_POST[‘user’];
$pass = $_POST[‘pass’];
$query = “DELETE FROM users WHERE
user=‘$user’ AND pass=‘$pass’”;
```

Preventing hacking attempts

Username	Password	Query String
fredsmith	mypass	DELETE FROM users WHERE user='fredsmith' AND pass='mypass'
abc' OR 1=1 #		DELETE FROM users WHERE user='abc' OR 1=1 #' AND pass=''

Case1: Is a normal case that you intended to delete that user
Case2: Is a dangerous case that 1=1 is always true and will delete all the users in the table

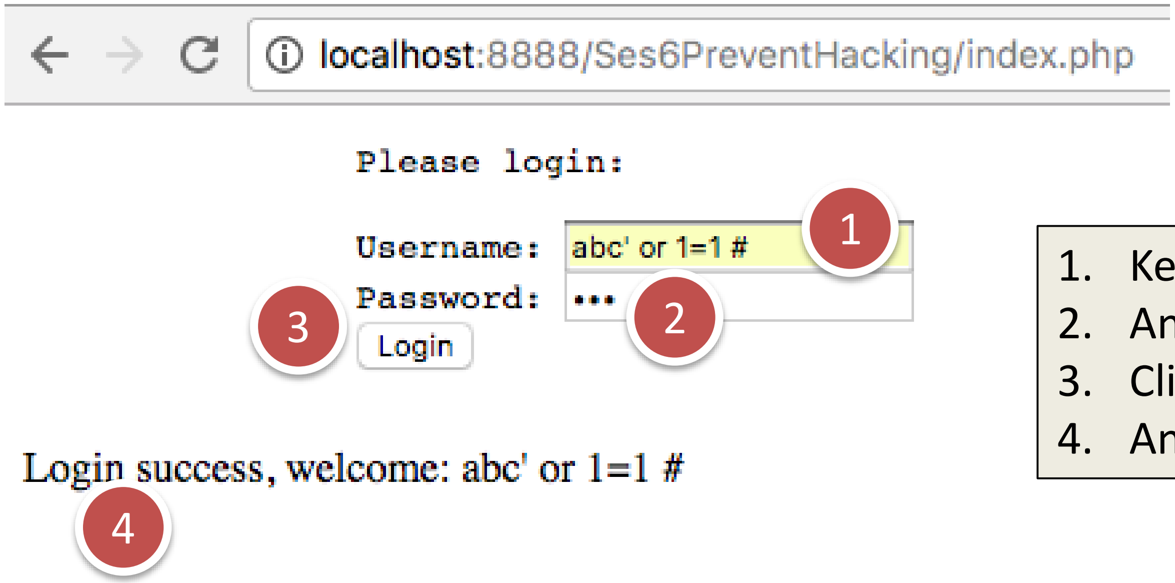
Activity: SQL Injection

```
create table user(
username varchar(50) primary key,
password varchar(50) not null
);
-- pass should be encrypted, but we use plain text here
insert into user values('admin', 'pass'),
                       ('user1', 'pass1');
-- test these queries
select * from user where username = 'admin' #' and password='abc';
select * from user where username = 'abc' or 1=1 #' and password='abc';
```

Activity: SQL Injection

```
<form action="index.php" method="post">
    <pre>
        Please login: <br/>
        Username: <input type="text" name="username"/>
        Password: <input type="password" name="password"/>
        <input type="submit" value="Login"/>
    </pre>
</form>
<?php
require_once './login.php';
$conn = new mysqli($host, $user, $password, $database, $port);
if($conn->error) die("Connection failed: " . $conn->error);
if(isset($_POST['username']) && isset($_POST['password'])){
    $username = $_POST['username'];
    $password = $_POST['password'];
    //Try to check the connection
    $query = "select * from user where username = '$username'"
            . " and password='$password'";
    $result = $conn->query($query);
    if($row = mysqli_fetch_assoc($result)){
        echo "Login success, welcome: $username";
    }else{
        echo "Login failed, try again!";
    }
}
?>
```

Activity: SQL Injection


localhost:8888/Ses6PreventHacking/index.php

Please login:

Username: abc' or 1=1 #

Password: •••

Login

Login success, welcome: abc' or 1=1 #

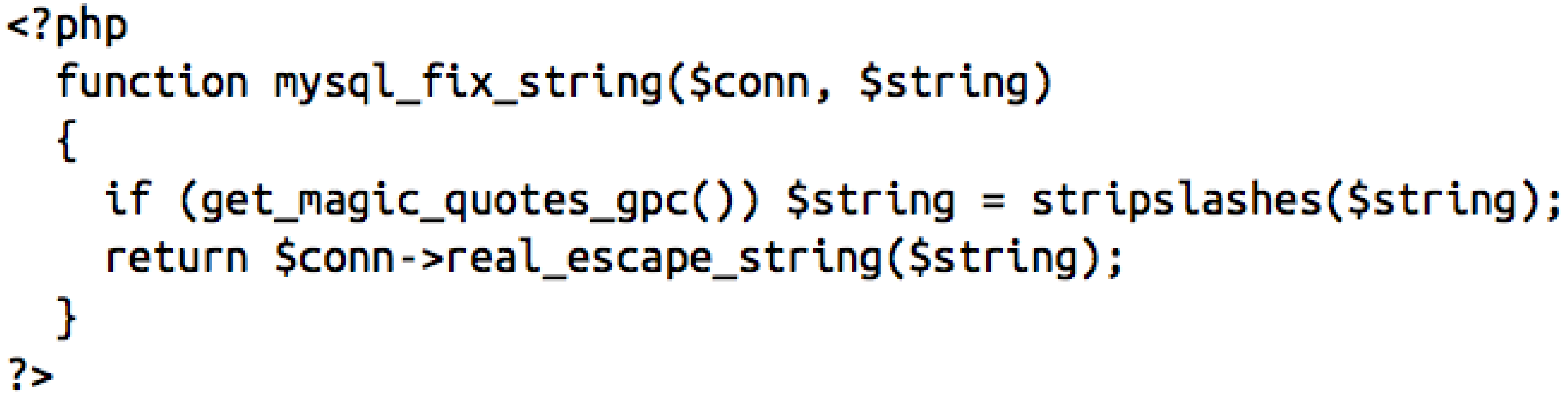
1. Key in abc’ or 1=1 #
2. Anything for the pass
3. Click login
4. And you are logged in successfully

Steps You Can Take

- The first thing is not to rely on PHP’s built-in magic quotes
  - Which escape any characters such as single and double quotes by prefacing them with a backslash (\)
  - Because this feature can be turned off; many programmers do so in order to put their own security code in place
  - So there is no guarantee that this hasn’t happened on the server you are working on.
  - In fact, the feature is deprecated in PHP5.3.0 and was removed in PHP 6.0.0

Steps You Can Take

- To make sure you can check if the magic quote is supported using this method
  - get_magic_quotes_gpc()
- Or you should use the real_escape_string method for all calls to MySQL. Example



```
<?php
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Sample code

```
<?php
require_once './login.php';

function mysql_fix_string($conn, $string) {
    if (get_magic_quotes_gpc()) {
        $string = stripcslashes($string);
        return $conn->real_escape_string($string);
    }
}

$conn = new mysqli($host, $user, $pass, $database, $port);
if ($conn->error)
    die($conn->error);
$user = mysql_fix_string($conn, $_POST['user']);
$pass = mysql_fix_string($conn, $_POST['pass']);
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
//Do some procesing here
?>
```

Using Placeholders

- Prepared statements with placeholders
  - provide a method by which only data is transferred to the database
  - doesn’t allow user-submitted data to be interpreted as MySQL statements
  - The prepared statement can use parameters with the ‘?’ as a placeholder for the data
  - Example:

```
$stm = $conn->prepare(‘INSERT INTO classics VALUES(?,
?, ?, ?, ?)’);
```

Binding the parameters

- Before executing the prepared statement, we need to bind parameters to it
- Example
  - $stm->bind_param('sssss', $isbn, $author, $title, $category, $year);
  - The first parameter is a string representing the type of each of the arguments in turn
  - E.g., five s to represent five strings
- Data type character can be
  - i: The da is an integer
  - d: The data is a double
  - s: The data is a string
  - b: The data is a BLOB (and will be sent in packets)
- After binding we can execute the statement
  - $stm->execute();

Activity: Try with the add book

Preventing HTML Injection

- Another type of injection you need to concern
  - Not for the safety of your websites
  - But for the users’ privacy and protection
  - It is cross-site scripting (XSS)
- This occurs when you allow HTML (or JavaScript code) to be input by a user and then displayed back by your website
  - One common place is *comment form*.
- What frequently happen is that user can write code
  - To steal cookies from your site’s users
  - Or may launch an attack to download a Trojan onto a user’s computer

Preventing HTML Injection

- Preventing this is as simple as calling to the *htmlentities* function
  - This strips out all HTML markup codes and replaces them with a form that displays the characters, but does not allow a browser to act on them
  - Example
    - <script src=‘http://x.com/hack.js’></script>
    - <script>hack();</script>
  - If we load these through html entities, it would display
    - &lt;script src='http://x.com/hack.js'&gt; &lt;/script&gt; &lt;script&gt;hack();&lt;/script&gt;

Function for preventing both SQL and XSS injection attacks

```
function mysql_fix_string($conn, $string){
    if(get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
}
function mysql_entities_fix_string($conn, $string){
    return htmlentities(mysql_fix_string($conn, $string));
}
```

register_globals: An old solution hangs on

- Before security concern
  - Default behavior was to assign the $_POST and $_GET arrays directly to PHP variables
  - E.g., $name = $_POST['name']; is not necessary because variable $name is given that value automatically
- With security concern
  - Before PHP 4.2.0, this seemed a very useful idea that saved a lot of extra code writing
  - Now, this practice has been discontinued and the features is disabled by default
  - You should disable register_globals on production web server
- Why this security concern?
  - E.g., if your program do use the variable $override and you forgot to initialize it (e.g., with $override = 0)
  - Users can exploit this by entering http://server.com?override=1. To assign your variable with 1, even you don't want to get that value from user
  - So you should always initialize variables that you use

Questions

1. How do you connect to a MySQL database using `mysqli`?
2. How do you submit a query to MySQL using `mysqli`?
3. How can you retrieve a string containing an error message when a `mysqli` error occurs?
4. How can you determine the number of rows returned by a `mysqli` query?
5. How can you retrieve a particular row of data from a set of `mysqli` results?
6. Which `mysqli` method can be used to properly escape user input to prevent code injection?
7. What negative effects can happen if you do not close the objects created by `mysqli` methods?

References

Nixon, R., 2014. *Learning PHP, MySQL & JavaScript*. 4th ed. Oreilly.