Week 11 Lab – Managing State and SQL Injection

PHP Sessions http://php.net/manual/en/book.session.php

This lab will not be assessed by your tutor but you may find the exercises useful for Part 3 of your assignment.

Aims:

- To be able to manage state information by passing information from one PHP page to another, using **sessions**.
- To become aware of how to provide redirection from one PHP script to another in PHP.
- To understand how SQL injection works and how to avoid it.

Getting Started:

Create a new folder '**lab11**' under the unit folder on the mercury server in your */htdocs* folder.
Save today's work in this folder.

All delivered web pages should conform to HTML5 and must be validated.
You could also create and link an external stylesheet, to the pages, and this should be valid CSS3.

Task 1: Up and down counter using PHP sessions

The overall task is to create a simple web application that displays an integer and contains three links to update the integer. One link increments the integer by 1, another link decrements the integer by 1, and the last link sets it to 0.

Step 1:

Create a file **`number.php`** that starts up a session, creates a session variable if it does not exists and displays it on the web page.

```
  <?php
    session_start();                      // start the session
    if (!isset ($_SESSION["number"])) {   // check if session variable exists
      $_SESSION["number"] = 0;            // create and set the session variable
    }
    $num = $_SESSION["number"];           // copy the value to a variable
  ?>

<!DOCTYPE html>
<html lang="en" >
      <head>
      <meta charset="utf-8" />
      <meta name="description" content="Playing with PHP Sessions" />
      <title>PHP Sessions Lab</title>
</head>
<body>
  <h1>Creating Web Applications - PHP Sessions Lab</h1>
  <?php
      echo "<p>The number is", $num , "</p>";         // displays the number
  ?>
  <p><a href="numberup.php">Up &#x2191;</a></p><!-- links to updating pages -->
  <p><a href="numberdown.php">Down &#x2193;</a></p>
  <p><a href="numberreset.php">Reset</a></p>
</body>
</html>
```

Test in the browser, and check that the `number.php` page, as delivered to the browser, is valid HTML5.

Step 2:

Create a file **`numberup.php`** that increments the session variable by 1.
This page does not contain any HTML and redirects to **`number.php`** after update.

```
<?php
  (1)                                       // start the session
  (2)                                   // copy the value to a variable
  $num++;                              // increment the value
  $_SESSION["number"] = $num;          // update the session variable
  header("location:number.php");       // redirect to number.php
?>
```

Fill in the blanks.

Step 3:

Create a file **`numberdown.php`** that decrements the session variable by 1.
This page does not contain any HTML and redirects to **`number.php`** after update.

```
<?php
  (3)                             // start the session
  (4)                             // copy the value to a variable
  (5)                             // decrement the value
  (6)                             // update the session variable
  (7)                             // redirect to number.php
?>
```

Fill in the blanks

Step 4:

Create a file **`numberreset.php`** that clears out all session variables and redirects to **`number.php`** after reset.
This page does not contain any HTML and redirects to **`number.php`** after update.

```
<?php
  session_start();                         // start the session
  (8)                                     // unset all session variables
  (9)                                     // destroy all session data
  (10)                                     // redirect to number.php
?>
```

Fill in the blanks. See Lecture notes

Test in the browser, and check that the page is valid HTML5.

Note:

Using `header("location:URL")` function, provides redirection. It is needed as a php script on a server cannot ***directly*** pass control to another php script on the server. So it does it ***indirectly*** by passing a http header message back to the client, then the client makes a request to the URL of the php script on the server.

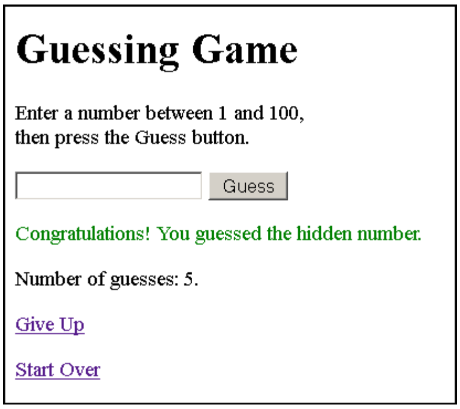
Task 2: Creating a simple "Guessing Game"

The overall task is to create a simple web application that generates and uses **sessions** to store a random number between 0 and 100.

Step 1:

Create a file **`guessinggame.php`** that will be the main page for the game.
In this page, a session variable is randomly set with a value between 1 and 100, if it does not already exist.
A user can input a guess, and when the value is returned back to this same page, the page displays whether the number input is higher or lower than the generated number; congratulates them if they correctly guess the number; and displays the number of times the user has guessed.
(Always checking that the input data is "in-range" and is numeric).

The page also has:
a `'Give Up'` link to `giveup.php`, and
a `'Start Over'` link to `startover.php`.



Guessing Game

Enter a number between 1 and 100, then press the Guess button.

Congratulations! You guessed the hidden number.

Number of guesses: 5.

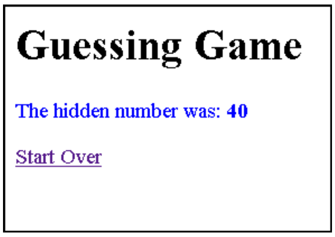
Give Up

Start Over

Hint:
The PHP's `rand()` function can be used to generate a random integer.
The `rand()` function accepts two arguments: the first argument specifies the minimum integer to generate; and the second argument specifies the maximum integer to generate.
For example, the statement
`$randNum = rand(10,20)` generates a random integer between `10` and `20` and assigns the number to the `$randNum` variable.

Step 2:

Create a file **`giveup.php`** that displays the random number generated for the current game. The value of the random number is accessed from the session variable.



Guessing Game

The hidden number was: **40**

Start Over

Step 3:

Create a file **`startover.php`** that has no html, it simply destroys the session then **redirects** the user back to `guessinggame.php`

Step 4:

Test in the browser, and check that the pages `guessinggame.php` and `giveup.php` are valid HTML5.

Step 5: *Optional*

Combine `guessinggame.php` and `giveup.php` into one HTML page.
*(At present you can 'cheat' by 'giving up', then going **back** in your browser and then 'guessing' the answer!)*

Hint: Change the 'Give Up' link in `guessinggame.php` to a self call with a 'flag' in the Query String.
e.g. `guessinggame.php?giveup=yes` Then place an 'if' control around the form code, so that the form is not displayed, and add an 'else' that contains the code from 'giveup' to show the hidden number.

Task 3: VIP member Registration System

The overall task is to create a Web application that manages a VIP Member's details using a MySQL database table created and accessed through PHP scripts.

The following fields are to be created for the '**vipmembers**' table:

member_id	INT AUTO_INCREMENT PRIMARY KEY
fname	VARCHAR(40)
lname	VARCHAR(40)
gender	VARCHAR(1)
email	VARCHAR(40)
phone	VARCHAR(20)

Step 1: Create the VIP Member Home Page vip_member.html

Create a simple "Home Page" document **vip_member.html** that contains links to:

- "Add New Member Form", **member_add_form.html,**
- "Display All Members Page" **member_display.php,** and
- "Search Member Page" **member_search.php**.

Step 2: Create the add member form member_add_form.html

Create a document **member_add_form.html** that contains a form with method **post** and action to **member_add.php** The form will need inputs for all the data for a new member, except member_id, the "Membership Number".

Hint: You could adapt any of the previous forms you have created in labs for this purpose.

Step 3: Create the database table and a form to insert records member_add.php

Create the document **member_add.php** to create a connection to the mysql server and your database, create the 'vipmembers' table if it does not exist, and insert the details from "Add New Member Form" into the table.

Hint: You will need to execute two queries in this script: the first to create the table if does not exists and the second to insert the data.

Step 4: Retrieve and display records from the database table member_display.php

Create a document "Display all Members Page" **member_display.php** that selects only member_id, fname, lname from the database table and displays them neatly in a html table.

Step 5: Create a Search Page member_search.php

Create a document "Search Member Page" **member_search.php** with a form to enable a search for a member(s) based on the last name in the database table, and then display the member_id, fname, lname, email details of all members that matched the last name in a html table.

Hint: This page will have a form with an action that calls itself (a self call) as demonstrated in Lecture 9.

Step 6: Test and Improve Security

Save **`member_search.php`** as **`member_login.php`** and change it to create a "Member Log-in" page with a form to enable a user to input a member's id (member_id) and lastname (lname), that will connect to the database and then display the `member_id, fname, lname, email` details of all members ***that matched the selection***, in a html table. Again this page will have a form action that calls itself (a self call)

Example A: Initially use a query string something like that shown below. Note that here we do not have quotes around the variables **`'$memId'`** or **`'$lname'`**

```
$query = "select member_id, fname, lname, email from $sql_table "
       ."where member_id = $memId and lname= $lname order by lname, 
fname";
```

Test the page with known data, to make sure it is working. Then consider the security issues and solutions discussed in Lecture 10 and Lecture 11, let us try to insert SQL into the inputs in the log-in page.

Example B: Firstly try some SQL injection testing on phpMyAdmin:

```
select * from vipmembers where member_id =1 or 1=1 -- and lname=anyname;
```

Example C: Then try using SQL injection through your member_login.php form:

eg. Input a member id of **`1 or 1=1 --`** and any value for the lastname.

This injected SQL breaks the script and now shows all the values in the table.

Even worse, if our error message shows the "something wrong with the query string …" then we have exposed our database variables to the world! A hacker could then try something like:
eg. Member id of **`1 OR member_id LIKE '%'--`** and any value for the lastname.

Example D: How can we fix this?
For a start, change the query string to include quotes:

```
$query = "select member_id, fname, lname, email from $sql_table "
  ."where member_id = '$memId' and lname= '$lname' order by lname, 
fname";
```

Test the form again:
eg. Member id of **`1 or 1=1 --`** and any value for the lastname. What happens?

Add some regular expression checking to ensure that only expected values are ;permitted.

Task 4: More on SQL Injection

Optional Task

SQL injection means injecting parts of an SQL statement through a web interface such as a “login” or “forgotten password” page.

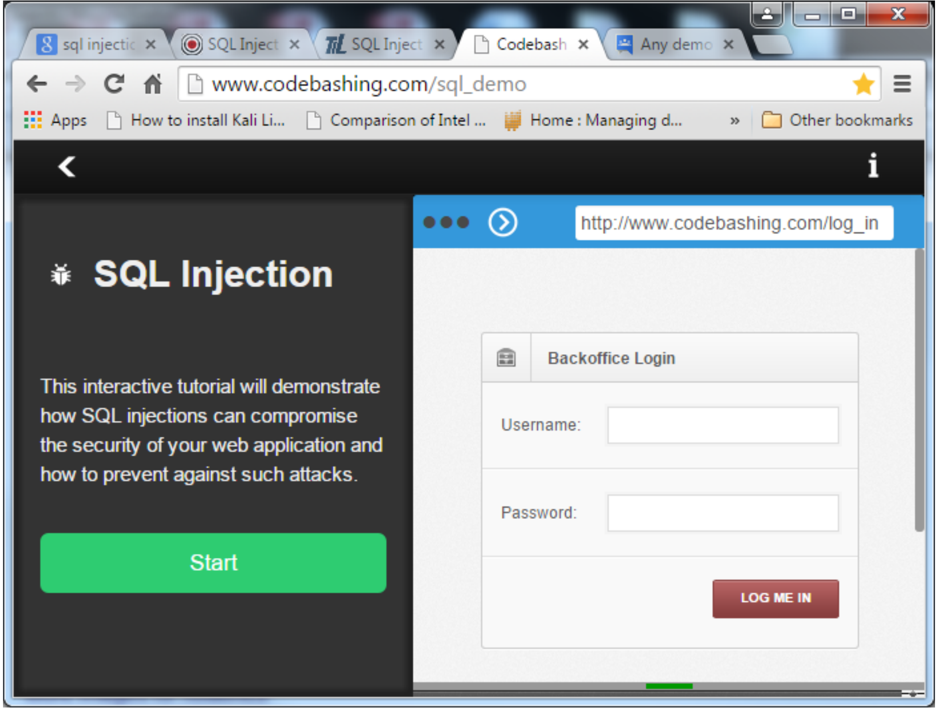
SQL injection allows you to execute arbitrary and sometimes malicious SQL commands on the back-end database server.

Note that you need a database for SQL injection to be effective. SQL injection is unlikely to achieve anything if there is not a database attached to the web page, although some web sites use “no-sql” databases which store data in the browser DOM and access the data using AJAX (Javascript) function calls.

There are many types of SQLi attacks:

- Logic attacks (injecting “true” into WHERE conditions);
- Stacked queries (Appending your own SQL statements to an existing hard-coded SQL string;
- Passthrough attacks (using SQL to pass operating system commands (Linux or DOS/Windows) to the operating system.
- Timing attacks (detecting errors by measuring response time of the web server).

Let’s examine some simple logic attacks.
Open a web browser and go to:
http://www.codebashing.com/sql_demo
You will be presented with a simple log-in page (Figure 1).



This interactive tutorial will demonstrate how SQL injections can compromise the security of your web application and how to prevent against such attacks.

Figure 1. Codebashing Demo site (initial load).

If the page does not look like this, Press F5 to reload the page. Expand the web browser sufficiently so that you can see the Live Log panel (at the bottom).

Click on Start and follow the instructions. The site will give you three forms of feedback – The web server response (main panel), an “escaped” version of the SQL statement received by the database server (green) and the MySQL error message (red). The green text is visible only for this demo site – the web developer wrote scripts specifically to display this text to you to help you understand what’s going on at the back end. The red text comes from the database server, and if the web site is improperly configures could be visible to the web site user by mistake. In a production system the user should never get to see the DBMS error messages. Can you say why?

Follow the instructions until you complete part 5. You will have by-passed the second term in the WHERE condition by entering `' or 1=1)#`

Press F5 to re-set the application.

This is not the only way to break in. Try entering this into the Username field:
' or 1=1 or '

Reset and try this input (Password field only):
x' or '1'='1

Anything that evaluates to TRUE will work provided you respect the SQL command syntax.
The error messages from the RDBMS (database server) are a valuable source of information to the attacker. If exposed, they reveal the type of database, and information about the database schema (including table and column names). Knowing the type of RDBMS allows the attacker to determine the correct syntax to use (' vs ", comment character, stacked or nested queries …).

Try this input (Login field):
' AND password LIKE 'a
This will not log us in, but it will tell us indirectly if the table has a column called password. If so, there will be no error message from the DB; if wrong it will send an error message.

Try
' AND pass LIKE 's
That gives an error.

You can also prepend a tablename to the column name to see if you've guessed it too.
Another technique is to progressively brute force the LIKE statement ('a 'b 'c … 'aa 'ab 'ac 'ad…) and measure the time taken for the web page to respond. This "Timing attack" can be used to brute-force user names, passwords and even hashed passwords.

There are various ways of preventing SQL injection, and they are all the responsibility of the web developer.
1. Escape characters that are used in SQL commands. Escaping them allows them to be stored in a database without being executed. Escaping will also reduce the risk of XSS (Cross-site Scripting) attacks and buffer-overflow attacks.

2. Filter input characters as demonstrated in the PHP lecture. Don't accept characters that form parts of SQL commands at all.

3. Validate input – only accept input that makes sense. e.g. don't accept non-numeric characters in number fields (Javascript and php treat all input as strings, so use functions like isnumeric() to check.

4. Use stored procedures at the RDBMS. These are programmer-written functions which accept user input as input parameters. Because the user input is not used to build an SQL statement it can never be executed.

This task is not assessed but you should use appropriate input sanitizing techniques in Assignment 3.