PHP – Lab Five – Cookies and Sessions

Objectives

1. Create, read, update and destroy PHP cookies
2. Create, read, update and destroy session variables
3. Understand key differences between cookies and sessions
4. Use sessions to build a simple login system
5. Protect pages from unauthorized viewing with a session variable
6. Consider security issues with Cookies and Sessions

Introduction

Many web applications require the developer to track some information about their visitors. This may be for a login system or to maintain a shopping cart. Cookies and Sessions can be used to perform this function.

The ‘Statelessness’ Problem

The web is ‘statelessness’. That is after a web server has ‘served’ a page your connection to the server is closed and it no longer cares you exist. Knowing who is connected to a page is obviously useful for web applications. For example, it allows pages to be personalized and is essential for features such as shopping carts.

The ‘statelessness’ problem can be overcome by passing information from one page to another using a URL query string and the superglobal `$_GET`. This method is insecure and time consuming to implement. Two alternative approaches for maintaining state are cookies and sessions.

Cookies

Cookies work by storing data in the user’s browser when they visit a page. That is the data is stored on the user’s own machine, the web server writes data to the client computer via the browser. As this is done by the browser the user can change their browser settings to block cookies and stop a webserver writing data to it. Cookies will only therefore work on browsers that have cookies enabled. Most people when browsing the web will have cookies turned ‘on’. However, users can change their browser settings to disable cookies if they so wish.

Testing for Cookies

To define a cookie use `setcookie()`. You must ensure that this function is placed before any output from a script as cookies are set via the HTTP headers of a HTML document. Any output here includes whitespacing. If `setcookie()` appears after any output then an error will occur.

***Warning:* If you place `setcookie()` after any outputs from echo or print it will fail. Like other headers, cookies must be sent before any output from your script (this is a protocol restriction).**

This requires that you place calls to this function prior to any output, including `<html>` and `<head>` tags as well as any whitespace. If output exists prior to calling this function, `setcookie()` will fail and return FALSE. If `setcookie()` successfully runs, it will return TRUE. This does not indicate whether the user accepted the cookie.

The setcookie() function and Retrieving values with $_COOKIE

```
setcookie(string CookieName, string CookieValue, int
CookieExpireTime, path, domain, int secure, int httponly);
```

- CookieName: String naming the cookie.
- CookieValue: String for the value of the cookie. This value is stored on the client's computer. As such do not store sensitive information.
- CookieExpireTime: Integer representing the time the cookie expires in seconds. Thus `time()+60*60*24*10` will set the cookie to expire in 10 days. If not set, the cookie will expire when the browser closes (end of the session).
- Path: the directory under the webserver this cookie is used for. Default is the directory of the requested page.
- Doman: The domain name this cookie can be used under. Default is the domain of the requested page. The domain must have two '.' in it, so if you decide to specify you're top level domain, you must use ".mydomain.com".
- Secure: Boolean if set to '1' or true, indicates that the cookie should only be transmitted over a secure HTTPS connection.
- httponly: Boolean if set to '1' or true indicates cookies are only accessible via HTTP and not by scripting languages such as Javascript. Although not supported by all browser this does offer some security against XSS (cross site scripting) attacks.

The following code creates a cookie named ‘count' to hold a value. No expiration time is set so the cookie will expire when the browser is closed.

```
setcookie("count", 1);
```

The following code creates a cookie ‘count’. This takes it values from a variable `$count`. The expiration time is 600 seconds (10 minutes) from the current time.

```
setcookie("count", $count, time()+600);
```

To recall a cookies use `$_COOKIE['mycookie']`.

Use `isset()` in an `if` condition to check for the presence of a cookie.

```
if(isset($_COOKIE["count"])) {
..
}
```

Tip: When changing the value of an existing cookie use setcookie() to re-set/re-create the cookie with the same name.

Removing Cookies

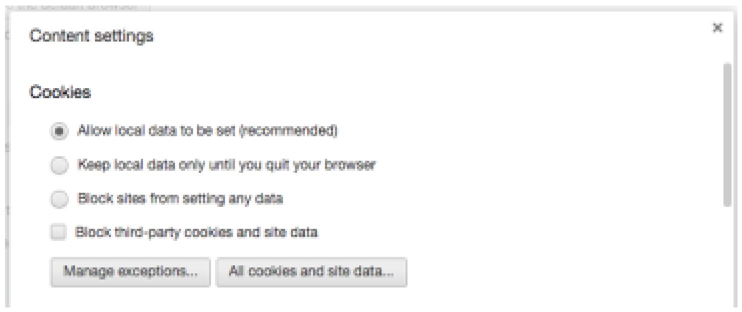
Cookies will expire on the date set via `setcookie()`. If you do not set an expiry date then when the browser is closed the cookies are removed. To remove a cookie via your code you need to re-set the cookie with a negative expiry date ie some time in the past.

```
setcookie("myname", '', time()-600);
```

Cookie Settings in Browsers

Whether cookies will work with your application depends on whether the browser will accept the cookie. The user can choose to reject cookies through settings in the browser.

For example in Google Chome under `chrome://settiings/content` cookies can be blocked by selecting the 'Block third-party cookies and site data' option.



Content settings

Cookies

● Allow local data to be set (recommended)

○ Keep local data only until you quit your browser

○ Block sites from setting any data

☐ Block third-party cookies and site data

Cookies and Security

Cookies are often targeted by hackers. As such they should not be used for storing sensitive information. Cookies can only store around 4K of data so are best used for tracking information, often that only has some meaning when reunited with data on a server.

If using cookies for any data that might be useful to a hacker then make sure that the cookie is associated with specific directory and domain, and if possible set the `https` and `httponly` flags to true ie:

```
setcookie('myCookie', 'myValue', 0, '/dir', 'domain.com', 1,
1);
```

Working with Sessions

Sessions can be used as an alternative to cookies. Whilst cookies are stored in the browser, the data held in a session variable is stored on the web server.

A session is available as long as the browser is opened.

Every page that makes use of sessions MUST begin with the following.

```
<?php
session_start();
?>
```

If no session exists this commands starts one, if one is already started this command recalls values in this session such that they can be displayed via the `$_SESSION` global array.

Session variables are set using the normal variable assignment method.

```
$_SESSION['user'] = $_POST['userID'];
```

To completely remove session variables it is recommended the `$_SESSION` is set to an empty array and the `PHPSESSID` removed by setting a negative timestamp (as with other cookies).

```
<?php
session_start();
unset($_SESSION['user']);
$_SESSION = array();
session_destroy();
setcookie('PHPSESSID', '', time()-3600);
?>
```

PHPSESSID and session_set_cookie_params()

Sessions actually use cookies. That is they store a session ID as a cookie on the browser. This cookie is named `PHPSESSID` and relates to a value stored on the server. As such the same security issues can arise as with cookies. Therefore before calling `session_start()` it is advisable to amend the way the `PHPSESSID` cookie is stored.

This is done with `session_set_cookie_params()`.

Like `setcookie()`, `session_set_cookie_params()` takes values for expiry time, path, domain, https and httponly.

```
session_set_cookie_params(0, '/', 'homepages.shu.ac.uk', 1,
1);
```

Setting https and httponly to 1/true should be done to protect the `PHPSESSID` from XSS attacks.

Viewing $_SESSION and $_COOKIE with print_r()

As both `$_SESSION` and `$_COOKIE` are both arrays, the array dump `print_r()` is useful for debugging purposes.

Task 1 – Working with Cookies

Open the files *preferences.php* and *setPreferences.php* in the *cookies* folder. To test these files you will need to view the pages through *homepages.shu.ac.uk*.

In this demonstration you will create a page that uses cookies to remember a user choice of colour and also counts how many times the user has visited the page.

In the *preferences.php* file note that three variables are set:

```
$favColour = 'red';
$colourCode = '#ff0000';
$value = 1;
```

There is also a switch/case to assign a new value to `$colourCode` based on `$favColour`.

A form in *preferences.php* submits a value to *setPreferences.php*.

Add the necessary code to *setpreferences.php* to create a cookie called `favCol` that will store the users chosen colour value and then redirect the user back to *preferences.php*.

Set the cookie to expire in one day.

In the *preferences.php* file check to see if the `favCol` cookies exists and if it does assign it to the `$favColour` variable.

In the *preferences.php* file create a cookie called `myCount` and have it count how many times the user has visited the page. Use the `myCount` cookie to update the `$value` variable that is displayed to the user.

The following code snippets might help.

```
<?php
$name = 'myFav';
$value = $_POST['favColour'];
$expires = time() + (60*60*24);// one day from now.
setcookie($name, $value, $expires);
header('Location: preferences.php');
?>
```

To check if a cookie exists use `isset()`.

```
if(isset($_COOKIE['myFav'])){
     $favColour = $_COOKIE['myFav'];
}
```

Remember to update a cookie it has to be recreated with `setcookie()` ie:

```
if(isset($_COOKIE['myCount'])){
     $value = $_COOKIE['myCount'] + 1;
     setcookie($name, $value, $expires);
}else{
     setcookie($name, $value, $expires);
 }
```

Task 2 - Creating a Simple Login with PHP Sessions

Open the files in the *sessions* folder.

In this demonstration you will build a simple login script using PHP Sessions and protect a page from unauthorized viewing. You'll need to test these files using https.

In the pages *index.php* and *blog.php* there is an include called *debugger.inc.php* that is designed to help you build the application. This does a `print_r` of the `$_SESSION` which you might find useful. Obviously this would not be included in production code.

As we are going to use sessions we'll start by creating a sessions include file that can be included into any page that will make use of sessions.

Open the file *includes/sessions.inc.php* and add the following:

```
session_set_cookie_params(0, '/~<yourID>',
'homepages.shu.ac.uk', 1, 1);
session_start();
```

Open the page *index.php* and *blog.php* pages to add this include at the very top of the page as follows:

```
<?php
include('includes/sessions.inc.php');
?>
```

We'll add a conditional include into *index.php* after the closure of the `<div id="logo">`.

```
if(isset($_SESSION['login'])){
     include('includes/session-logout.inc.php');
}else{
     include('includes/session-login.inc.php');
}
```

If the `$_SESSION['login']` exists then the 'logout' include is called if not the 'login' include is called. The two files include the HTML relevant to each situation. Open *session-login.inc.php* and you'll see it includes a form that posts a username and password to *logic/checklogin.php*.

Open the file *logic/checklogin.php* and add the following:

```
include('../includes/sessions.inc.php');
//check login / password combination
if ($_POST['username'] == "admin" && $_POST['password'] ==
"letmein"){
     $_SESSION['login'] = 1;
}
// redirect browser
header("Location: ".$_SERVER['HTTP_REFERER']);
```

This makes a very simple check of the username and password combination and if correct creates a session variable `$_SESSION['login']`.

The user is then redirected back to where they came from using `header`.

Warning: This should be made much more securer by storing the username / password combination in a database. We'll see how to connect to a database later in the module.

If the user is successfully logged in the *session-logout.inc.php* will now be included. This includes a hyperlink to logic/logout.php. Add the following to logic/logout.php

```
<?php
include('../includes/sessions.inc.php');
//logout code
if(isset($_COOKIE[session_name()])){
// match PHPSESSID settings
setcookie(session_name(), '', time()-3600, '/~<yourID>',
'homepages.shu.ac.uk', 1, 1);
// clear the Session cookie
}
$_SESSION = array();
// empty the array
session_destroy();
//destroy the session
header("location:../index.php");
//to redirect
exit();
?>
```

We should now protect *blog.php* from unauthorized access. Add an additional include to the top of *blog.php* as follows:

```
include('includes/sessions.inc.php');
include('logic/authorize.php');
```

In *logic/authorize.php* add the following:

```
if(!isset($_SESSION['login'])){
     header('Location: '.$_SERVER['HTTP_REFERER']);
     exit;
}
```

Finally we can improve the login by counting how many attempts have been made and blocking more than three incorrect attempts.

In *logic/checklogin.php* add an `else` clause to the logic that checks if a valid login is found. Inside the `else` clause add the following additional logic that will increment the `$_SESSION['count']` value.

```
if(!isset($_SESSION['count'])){
     // first fail
      $_SESSION['count'] = 1;
     }else{
     // 2nd or more fail
     $_SESSION['count']++;
}
```

Then in *includes/session-login.inc.php* add the following logic to display a message to the user. This should go before the `<form>`.

```
if($_SESSION['count'] > 0 && $_SESSION['count'] <= 3 ){
     echo "<p class=\"error\">Sorry incorrect</p>";
}
if($_SESSION['count'] > 3){
     echo "<p class=\"error\">Too many attempts</p>";
}else{
// don't forget to close this with } after the HTML form
```

You will then need to place the `<form>` inside an else clause so don't forget the closing curly brace after the `</form>`.