**Cookies & Sessions**

**Intro**

The HTTP is a stateless technology, meaning that each HTML page is an unrelated entity. HTTP has no method of tracking users or retaining variables as a person traverses a site. Without the server being able to track a user, there can be no shopping carts etc… Cookies & sessions can overcome this.

The key difference between cookies & sessions is that cookies store data in the users web browser and sessions store data on the server itself.

**Cookies**

Cookies are a way for a server to store information on the user’s machine. Think of a cookie as being like a name tag: you tell the server your name, and it gives you a name tag.

Setting cookies

The most important thing to understand about cookies is that they must be sent from the server to the client prior to any other information.

Cookies are set via the setcookie(name, value) function. You can continue to send more cookies to the browser with subsequent uses of the setcookie() function.

Accessing cookies

To retrieve a value from a cookie, you only need to refer to the $\_COOKIE superglobal, using the appropriate cookie name as the key. $\_COOKIE[‘name’];

Setting cookie parameters

Look at p382 for the other optional arguments in the setcookie() function.

Deleting cookies

A cookie will automatically expire when the users browser is closed or when the expiration date/time is met (optional argument).

To delete a cookie:

1. Setcookie(name); i.e. don’t set the value in the function

e.g.

setcookie(‘name’,’ali’); //create a cookie with ‘name’ as the key

setcookie(‘name’); //delete cookie using the key name, ‘name’.

\*See examples of cookies on p368 – 386 – **very good example, use this!**\*

**Using sessions**

|  |  |
| --- | --- |
| **Session advantages** | **Cookie advantages** |
| * More secure * More data can be stored * They can be used without cookies | * Easier to program * Require less of the server * They can be made to last far longer |

The premise of a session is that data is stored on the server, not in the web browser, and a session identifier is used to locate a particular user record (the session data). This session identifier is normally stored in the users’ web browser via a cookie, but the data itself always remains on the server.

Setting session variables

Session\_start() - this function tells PHP to either begin a new session, or access an existing one. This function is best placed at the top of a script and must be called before anything is sent to the web browser.

Once the session has been started, values can be registered to the session using the normal array syntax, using the $\_SESSION superglobal:

$\_SESSION[‘key’] = value;

Accessing session variables

Once a session has been started and variables have been registered to it, you can create other scripts that will access these variables. To do so, each script must first enable sessions, using session\_start();

Use - $\_SESSION[‘key’] to refer to a session variable.

Session\_start();

If(isset($\_SESSION[‘user\_id’])) {

Echo “Hello {$\_SESSION[‘first\_name’]}”;

}

\*The first time the session\_start() function is used, the function will attempt to send a cookie with a name of PHPSESSID & a 32 hexadecimal letter value. When session\_start() is used in another script, the function will give the current script access to the previously started session, or create a new session if it cannot. If the current session ID cannot be found, and a new session ID is generated, none of the data stored under the old session ID will be available. Due to this, the first debugging test should be to see if a new session ID is being created on each page:

Echo session\_id(); //put this on each page with a session to see if ID is the same

\*Session variables are available as soon as you’ve established them, unlike cookies\*

Deleting session variables

To delete an individual session variable:

Unset($\_SESSION[‘var’]);

To delete every session variable, you shouldn’t use unset(), instead, reset the $\_SESSION array:

$\_SESSION = array();

Finally, to remove all of the session data from the server, call: session\_destroy();

\*Note that prior to using any of these methods, the page must begin with session\_start();\*

With sessions, 3 kinds of information exist:

1. The session identifier (stored in a cookie by default)
2. Session data (stored in a text file on the server)
3. $\_SESSION array (how a script accesses the text file)

Just removing the cookie doesn’t remove the data file and vice versa. Clearing the $\_SESSION array would erase the data from the text file, but the file itself would still exist, as would the cookie. Make sure to follow these steps:

$\_SESSION = array(); //clear the array

Session\_destroy(); //destroy the session

Setcookie(‘PHPSESSID’); //destroy the cookie

\*see examples of sessions on p388 - 395\*

**Improving session security**

Because important information is normally stored in a session (you should never store sensitive data in a cookie), security becomes more of an issue.

With sessions there are two things to pay attention to:

1. The session ID (reference point to the session data)
2. Session data (stored on the server)

A malicious person is far more likely to hack into a session through the session ID than the data on the server.

One method of preventing hijacking is to store some sort of user identifier in the session, and then repeatedly double check the value. The HTTP\_USER\_AGENT – a combination of the browser and operating system being used, is a likely candidate for this purpose. This adds a layer of security in that one person could only hijack another users session if they are both running the exact same browser and OS.

$\_SESSION[‘agent’] = md5($\_SERVER[‘HTTP\_USER\_AGENT’]);

The md5() function returns a 32-character hexadecimal string (called a hash), based upon a value. no two strings will have the same md5() result.

If(!isset($\_SESSION[‘agent’]) OR ($\_SESSION[‘agent’] != md5($\_SERVER[‘HTTP\_USER\_AGENT’]) )) {

//code

} \*full example on p397-398\*

By default, a server stores every session file for every site within the same temp directory, meaning any site could theoretically read any other sites session data. If you are using a server shared with other domains, changing the session.save\_path from its default setting will be more secure. For example, it’d be better if you stored your sites session data in a dedicated directory particular to your site.

Preventing session fixation

Another specific kind of session attack is known as session fixation. This approach is the opposite of session hijacking. Instead of a malicious user finding another users session ID, the malicious user instead creates their own session ID (perhaps by logging in legitimately), and then gets the legit user to access that site using that session. The hope is that the legit user would then do something that would unknowingly benefit the malicious user.

You can help protect against these types of attacks by changing the session ID after the user logs in. The session\_regenerate\_id() does just that, providing a new session ID to refer to the current session data. You can use this function on sites for which security is paramount (e-commerce or online banking…), or in situations when it’d be particularly bad if certain users (i.e., administrators) had their sessions manipulated.

**Test**

**Cookies**

1. What is the key difference between cookies & sessions
2. How do you set cookies?
3. How do you access cookies?
4. How do you delete cookies?
5. Imitate the example on p368-386 (Make sure you do this!)

**Sessions**

1. How do you set session variables?
2. How do you access session variables?
3. What does session\_id() do & when should you use it?
4. How do you delete an individual session?
5. How do you delete every session variable?
6. What 3 kinds of information exist in a session?
7. Change the example in cookies, Q5, to use sessions instead of cookies
8. What can you do to improve the security of using sessions? Give example.
9. What does session\_regenerate\_id() do & when would you use it?