CS 85:
PHP PROGRAMMING

Module 10: Cookies & Sessions

Santa Monica College
Computer Science &
Information Systems Dept.

When the HTTP protocol was initially developed, there was no consideration taken for tracking an individual user visits to a webpage. Meaning every user that visited the website was identical as far the web server was concern. This is known as a stateless design. Over time there became a need to provide different users a different experience or to track different users individually on website. For website that require a login or a portal to access specific information depending on the user. It is these site that require statefull design. This module will cover the different techniques in PHP to track individual users *state* information when accessing a website. This module will cover how to use Cookies and Sessions to store user information.

Cookies

Cookies are small pieces of data stored as text on the client's computer. Normally cookies are used only to store small amounts of data, including user preferences, time and more. Even though cookies are not harmful some people do not permit cookies due to concerns about their privacy. There are two types of cookies: Temporary and Persistent. Temporary Cookies remain on the system only for the life of the browser. Once the browser window closes the cookies are removed. Persistent cookies remain on the system in a text files for a specified period of time. Once cookies are saved on a computer, each time the user visit the website the saved cookies are sent from the client to the server. The server will then interpret the cookies information and provide a customized webpage or information. For example, a web server could receive user information in the cookies, then greet them with a welcome message or access to their user information.

PHP allows easy setting and retrieving of cookies.

Setting a cookie

Setting a cookie is extremely easy with *setcookie()*.

Syntax

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

where `name` is the cookie name, `value` is the data to be contained in the cookie, `expire` the time after which the cookie should expire in seconds, it could be one minutes or 1 week. If expiresis set to 0, or omitted, the cookie will expire at the end of the session (when the browser closes). The argument `path` is the path on the server which can use the cookie, `domain` can be used to set permissions for subdomains and secure if set true only transmits the cookie if a secure connection is present.

Since all cookies are sent by the server along with HTTP headers you need to set any cookie at the start of a page **before** any other code. You will normally only need to use the name, value and expire arguments. If expire not set the cookie will expire when the client closes the browser.

Examples

```
setcookie("wikibooks", "user", time()+3600);
```

The above code will set a cookie having the name wikibooks, value user and will expire an hour after it is set.

```
setcookie("test", "PHP-Hypertext-Preprocessor", time()+60,
"/location", 1);
```

Here the setcookie function is being called with four arguments (setcookie has 1 more optional argument, not used here). In the above code, the first argument is the cookie name, the second argument is the cookie contents and the third argument is the time after which the cookie should expire in seconds (*time()* returns current time in seconds, there time()+60 is one minute from now). The path, or location, element may be omitted, but it does allow you to easily set cookies for all pages within a directory, although using this is not generally recommended.

You should note that since cookies are sent with the HTTP headers the code has to be at the top of the page (Yes, even above the DOCTYPE declaration). Any other place will generate an error.

Retrieving cookie data

If a server has set a cookie on the user's computer, the user's browser sends it to the server each time a page loads. The name of each cookie sent by your server is stored in the superglobal array _*COOKIE*. So in the above example the cookie would be retrieved by calling **$_COOKIE['test']**. To access data in the cookie we use *explode()*. explode() turns a string into an array with a certain delimiter present in the string. That is why we used those dashes(-hyphens) in the cookie contents. So to retrieve and print out the full form of PHP from the cookie we use the code:

```
//retrieve contents of cookie
$array = explode("-", $_COOKIE['test']);

//display the content
print("PHP stands for " . $array[0] . $array[1] . $array[2]);
```

Deleting Cookies

Cookies are not deleted, they automatically expire. Temporary cookies are removed when the browser is called while persistent cookies are deleted when the time assigned expires. To force a cookies to get deleted, subtract any number of seconds from the expire augment.

```
setcookie("Name", "", time()-3600);
setcookie("Birthdate", "", time()-3600);
setcookie("Hometown", "", time()-3600);
```

Where are cookies used?

Cookies can be often used for:

- user preferences
- inventories
- quiz or poll results
- user authentication
- remembering data over a longer period

Security Warning!!!

You should **never** store unencrypted passwords in cookies as cookies can be easily read by other users.

You should **never** store critical data in cookies as cookies can be easily removed or modified by other users.

Sample Code

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer). We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30),
"/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if (!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30),
"/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Check if Cookies are Enabled

```
<!DOCTYPE html>
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Sessions

Sessions allow the PHP script to store data on the web server that can be later used, even between requests to different PHP pages. Every session has a different identifier, which is sent to the client's browser as a cookie or as a $_GET variable. Sessions end when the user closes the browser, or when the web server deletes the session information, or when the programmer explicitly destroys the session. In PHP it's usually called **PHPSESSID**. Sessions are very useful to protect the data that the user wouldn't be able to read or write, especially when the PHP developer doesn't want to give out information in the cookies as they are easily readable. Sessions can be controlled by the $_SESSION superglobal array. Data stored in this array is persistent throughout the session. It is a simple array. Sessions are much easier to use than cookies, which helps PHP developers a lot. Mostly, sessions are used for user logins, shopping carts and other additions needed to keep browsing smooth. PHP script can easily control the session's cookie which is being sent and control the whole session data. Sessions are always stored in a unique filename, either in a temporary folder or in a specific folder, if a script instructs to do so.

Using Sessions

At the top of each PHP script that will be part of the current session there must be the function `session_start()`. It must be before the first output (echo or others) or it will result in an error "Headers already sent out".

```
session_start();
```

This function will do these actions:

1. It will check the `_COOKIE` or `_GET` data, if it is given
2. If the session file doesn't exist in the `session.save_path` location, it will:
   1. Generate a new Unique Identifier
   2. Create a new file based on that Identifier
   3. Send a cookie to the client's browser
3. If it does exist, the PHP script will attempt to store the file's data into `_SESSION` variable for further use

Now, you can simply set variables in two different ways, the default method:

```
$_SESSION['example'] = "Test";
```

Or the deprecated method:

```
$example="Test";
session_register($example);
```

Both of the above statements will register the session variable `$_SESSION['example']` as "Test". The deprecated method should not be used; it is only listed because you can still see it in scripts written by programmers that don't know the new one. The default method is preferred.

Example #1 Register a variable with $_SESSION

```
<?php

session_start();
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}

?>
```

Example #2 Unregister a variable with $_SESSION

```
<?php
session_start();
unset($_SESSION['count']);
?>
```

Ending a Session

When user clicks "Logout", or "Sign Off", you would usually want to destroy all the login data so nobody could have access to it anymore. The session file will be simply deleted as well as the cookie to be unset by:

```
session_destroy();
```

Passing the Session ID

There are two methods to propagate a session id:

- Cookies
- URL parameter

The session module supports both methods. Cookies are optimal, but because they are not always available, we also provide an alternative way. The second method embeds the session id directly into URLs.

PHP is capable of transforming links transparently. If the run-time option *session.use_trans_sid* is enabled, relative URIs will be changed to contain the session id automatically.

Note:
The arg_separator.output php.ini directive allows to customize the argument separator. For full XHTML conformance, specify &amp; there.

Alternatively, you can use the constant **SID** which is defined if the session started. If the client did not send an appropriate session cookie, it has the form *session_name=session_id*. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using **SID**.

Example #1 Counting the number of hits of a single user

```
<?php
session_start();


   if (empty($_SESSION['count'])) {
        $_SESSION['count'] = 1;
    } else {
        $_SESSION['count']++;
    }


?>

<p>
    Hello visitor, you have seen this page <?php echo
$_SESSION['count']; ?> times.
</p>

<p>To continue, <a href="nextpage.php?<?php echo
htmlspecialchars(SID); ?>">click here</a>.</p>
```

The htmlspecialchars() may be used when printing the **SID** in order to prevent XSS related attacks.

Example #2 Session Across Pages

Defining session before everything, No output should be before that:

```php
<?php

session_start(); //First line in .php file

?>
```

Set your session inside a page and then you have access in that page. For example, this is script1.php

```php
<?php
//This is script1.php and then we will use session that defined
from this page:
session_start();
$_SESSION['email']='email@example.com';

?>
```

Using and Getting session in script2.php

```php
<?php
//This script will use the session:

session_start();
if($_SESSION['email']){
    echo 'Your Email Is Here!  :) ';
}

?>
```

Example #3: Session with Cookies

It is possible that the second script does not share access to the session cookies. In this case you will set a session cookie path on the user sysyem to store the cookie using function session_set_cookie_params().

```php
<?php
session_set_cookie_params( $lifetime, '/shared/path/to/files/'
);
session_start();
$_SESSION['myvar']='myvalue';
?>
```

Access stored cookie:

```php
<?php
session_set_cookie_params( $lifetime, '/shared/path/to/files/'
);
session_start();
echo("1");
if(isset($_SESSION['myvar']))
{
    echo("2");
    if($_SESSION['myvar'] == 'myvalue')
    {
        echo("3");
        exit;
    }
}
?>
```

Using Session Data of Other Types

Simple data such as integers, strings, and arrays can easily be stored in the $_SESSION superglobal array and be passed from page to page. But problems occur when trying to store the state of an object by assignment. Object state can be stored in a session by using the serialize() function. serialize() will write the objects data into an array which then can be stored in a $_SESSION supergloblal. unserialize() can be used to restore the state of an object before trying to access the object in a page that is part of the current session. If objects are to be used across multiple page accesses during a session, the object definition must be defined before calling unserialize(). Other issues may arise when serializing and unserializing objects.

Avoiding Session Fixation

Session fixation describes an attack vector in which a malicious third-party sets (i.e. *fixes*) the session identifier (SID) of a user, and is thus able to access that user's session. In the base-level implementation of sessions, as described above, this is a very real vulnerability, and every PHP program that uses sessions for anything at all sensitive should take steps to remedy it. The following, in order of how widely applicable they are, are the measures to take to prevent session fixation:

1. Do not use GET or POST variables to store the session ID (under most PHP configurations, a cookie is used to store the SID, and so the programmer doesn't need to do anything to implement this);
2. Regenerate the SID on each user request (using session_regenerate_id() at the *beginning* of the session);
3. Use session time-outs: for each user request, store the current timestamp, and on the next request check that the timeout interval has not passed;
4. Provide a logout function;

5. Check the 'browser fingerprint' on each request. This is a hash, stored in a `$_SESSION` variable, comprising some combination of the user-agent header, client IP address, a salt value, and/or other information. See below for more discussion of the details of this; it is thought by some to be nothing more than 'security through obscurity'.
6. Check referrer: this does not work for all systems, but if you know that users of your site *must* be coming from some known domain you can discard sessions tied to users from elsewhere. It relies on the user agent providing the referrer header, which should not be assumed.

This example code addresses all of the above points save the referrer check.

```php
<?php

$timeout = 3 * 60; // 3 minutes
$fingerprint = md5('SECRET-SALT'.$_SERVER['HTTP_USER_AGENT']);
session_start();

if ( (isset($_SESSION['last_active']) && (time() >
($_SESSION['last_active']+$timeout)))
    || (isset($_SESSION['fingerprint']) &&
_SESSION['fingerprint']!=$fingerprint)
    || isset($_GET['logout']) ) {
    do_logout();
}
session_regenerate_id();
$_SESSION['last_active'] = time();
$_SESSION['fingerprint'] = $fingerprint;

?>
```

The `do_logout()` function destroys the session data and unsets the session cookie.

Session Configuration Options

PHP sessions are easy to control and can be made even more secure or less secure with small factors. Here are runtime options that can be easily changed using php_ini() function:

Name	Default	Changeable
`session.save_path`	`"/tmp"`	`PHP_INI_ALL`
`session.name`	`"PHPSESSID"`	`PHP_INI_ALL`
`session.save_handler`	`"files"`	`PHP_INI_ALL`
`session.auto_start`	`"0"`	`PHP_INI_ALL`
`session.gc_probability`	`"1"`	`PHP_INI_ALL`

session.gc_divisor	"100"	PHP_INI_ALL
session.gc_maxlifetime	"1440"	PHP_INI_ALL
session.serialize_handler	"php"	PHP_INI_ALL
session.cookie_lifetime	"0"	PHP_INI_ALL
session.cookie_path	"/"	PHP_INI_ALL
session.cookie_domain	""	PHP_INI_ALL
session.cookie_secure	""	PHP_INI_ALL
session.use_cookies	"1"	PHP_INI_ALL
session.use_only_cookies	"0"	PHP_INI_ALL
session.referer_check	""	PHP_INI_ALL
session.entropy_file	""	PHP_INI_ALL
session.entropy_length	"0"	PHP_INI_ALL
session.cache_limiter	"nocache"	PHP_INI_ALL
session.cache_expire	"180"	PHP_INI_ALL
session.use_trans_sid	"0"	PHP_INI_SYSTEM/ PHP_INI_PERDIR
session.bug_compat_42	"1"	PHP_INI_ALL
session.bug_compat_warn	"1"	PHP_INI_ALL
session.hash_function	"0"	PHP_INI_ALL
session.hash_bits_per_char acter	"4"	PHP_INI_ALL
url_rewriter.tags	"a=href,area=href,f rame=src,input=src, form=fakeentry"	PHP_INI_ALL

A simple example of this use would be this code:

```
<?php

//Setting The Session Saving path to "sessions",
//'''must be protected from reading'''
session_save_path("sessions"); // This function is an
alternative to ini_set("session.save_path","sessions");

//Session Cookie's Lifetime ( not effective, but use! )
ini_set("session.cookie_lifetime",time()+60*60*24*500);
```

```
//Change the Session Name from PHPSESSID to SessionID
session_name("SessionID");

//Start The session
session_start();

//Set a session cookie ( Required for some browsers,
//as settings that had been done before are not very effective
setcookie(session_name(), session_id(), time()+3600*24*365,
"/");

?>
```

This example simply sets the cookie for the next year.