Lecture / Instructor: Leigh Cotnoir

PHP : GETTING INFO FROM USERS (B)

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input
- Cookies

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input
- Cookies
- Sessions

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links : $_GET
- Forms with user input : $_POST
- Cookies : $_COOKIE
- Sessions : $_SESSION

*These are some of the predefined PHP "superglobal" arrays.

How We Get Info From Web Users

In this lecture (part B), we will only address $_COOKIE and $_SESSION.

- Cookies : $_COOKIE
- Sessions : $_SESSION

How We Get Info From Web Users

Let's move on to discuss the next type of user input:

- URLs and links : $_GET
- Forms with user input : $_POST
- Cookies : $_COOKIE

How We Get Info From Web Users

- Cookies : $_COOKIE

Cookies are bits of data that get stored in the user's browser cache. The web application will set the cookie, and the data must then be accepted by and stored on the client computer's browser.

The $_POST and $_GET methods are useful in passing along info from one page to the next, but they will only last until the processed page has been refreshed. This is problematic for remembering variables across multiple pages. That is where cookies come in handy.

How We Get Info From Web Users

Cookies : $_COOKIE

To set a cookie, we first must use a function called "setcookie", which requires three parameters:

setcookie($*cookie_name*, $*value*, $*expiration_UnixTimestamp*);

- The cookie **name** can be set to anything that the programmer finds useful.
- The cookie **value** can also be set to anything that the programmer finds useful.
- The **expiration time** is how long the cookie will last, and this must be presented in the form of a UNIX timestamp. Because the UNIX timestamp format can be difficult for humans to read, we can use the "time()" function to get the current time on which to build the ultimate expiration time.

How We Get Info From Web Users

- Cookies : $_COOKIE

Let's set a cookie based on our "form_register_do.php" page. Remember, the following are the keys that were passed from the $_POST form:

```
Array
(
    [f_name] => Jane
    [l_name] => Doe
    [u_name] => jdoe
    [password] => dummy-password
    [sex] => f
    [bio] => Jane is an avid rock climber and really likes berry pies!
    [submit] => Submit
)
```

>> first name, last name, user name, password, sex, bio, submit

How We Get Info From Web Users

- Cookies : $_COOKIE

Remember, the following are the keys that were passed from the $_POST form to our "form_register_do.php" page:

>> f_name, l_name, u_name, password, sex, bio, submit

Now, let's set a cookie that identifies and stores the user name for three weeks.

```
setcookie('Registered User', $_POST['u_name'], time()+(60*60*24*21));
```

*The way to read the expiration is like this in plain english:
(current time) + (60 seconds*60 minutes*24 hours* 21 days)

How We Get Info From Web Users

- Cookies : $_COOKIE

Here's how we would place the command in our "form_register_do.php" page:

```
1  <?php setcookie('Registered_User', $_POST['u_name'], time()+(60*60*24*21)); ?>
2  <!DOCTYPE HTML>
3  <html>
4  <head>
```

Notice that the PHP block is introduced at the very beginning of the page, even before the DOCTYPE. This is necessary because cookies are passed as PHP "headers". We won't get into what a header is here, but it is extremely important that headers are sent without ANYTHING preceding them in the page, including white space.

How We Get Info From Web Users

- Cookies : $_COOKIE

Before we move on from our "form_register_do.php" page, we also need to place a link to a file that we will create in the next several steps. Place the link to "cookie.php" in the <body> of "form_register_do.php":

```
16  <p>Go to <a href="cookie.php">next page</a>
17  to read the cookie being set in this page.</p>
```

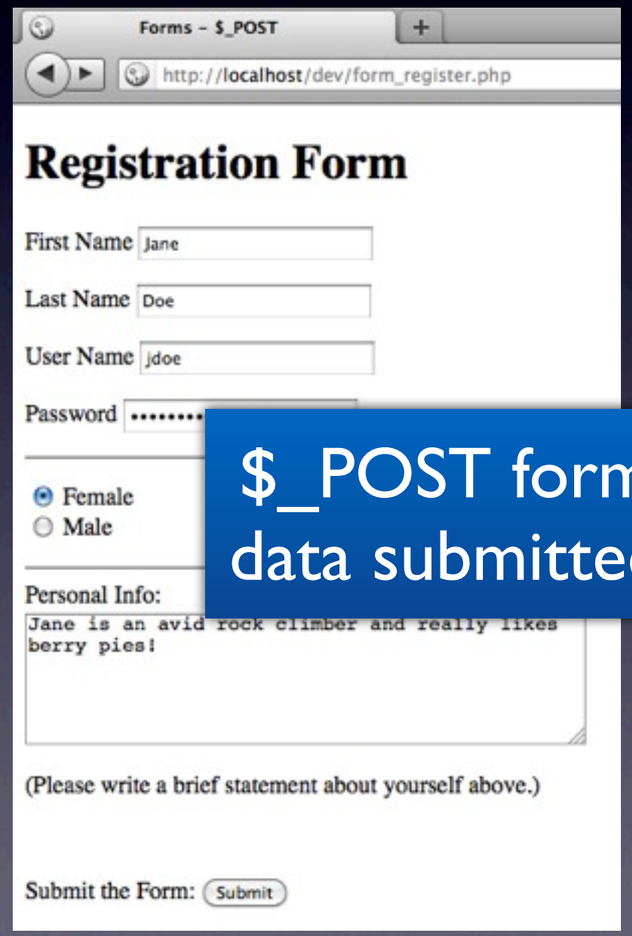
Now, let's make a new file called "cookie.php" that calls up the cookie we set in "form_register_do.php". Remember that we assigned the name key as "Registered_User":

```
10  <p>Welcome, user: <strong><?php echo $_COOKIE['Registered_User']; ?></strong>!</p>
```

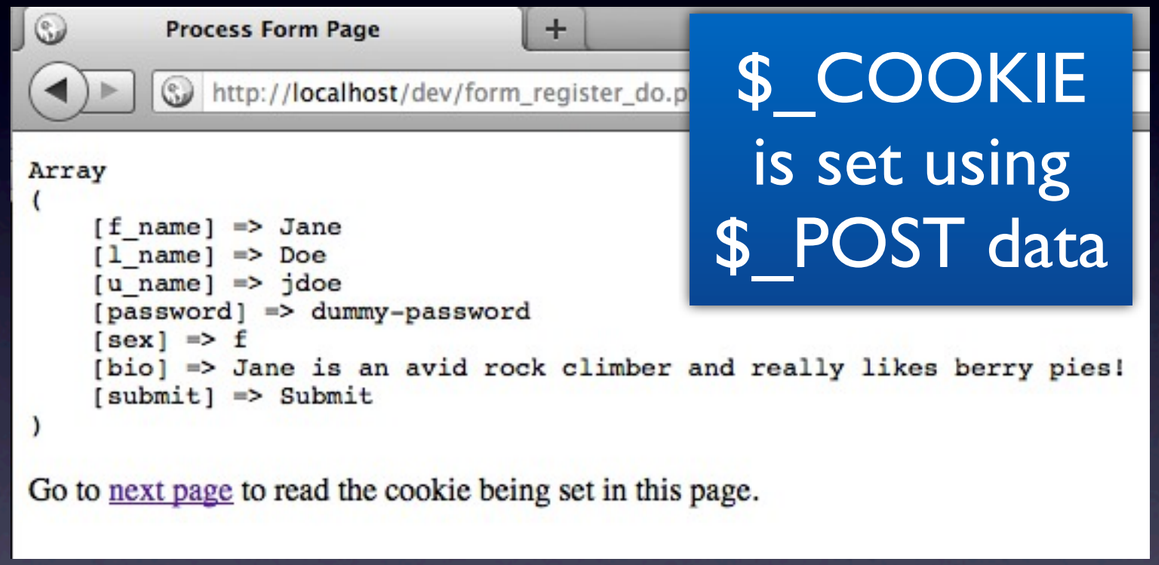
How We Get Info From Web Users

- Cookies : $_COOKIE

Open "form_register.php", fill out the form, submit it, and see the new results:



$_POST form data submitted



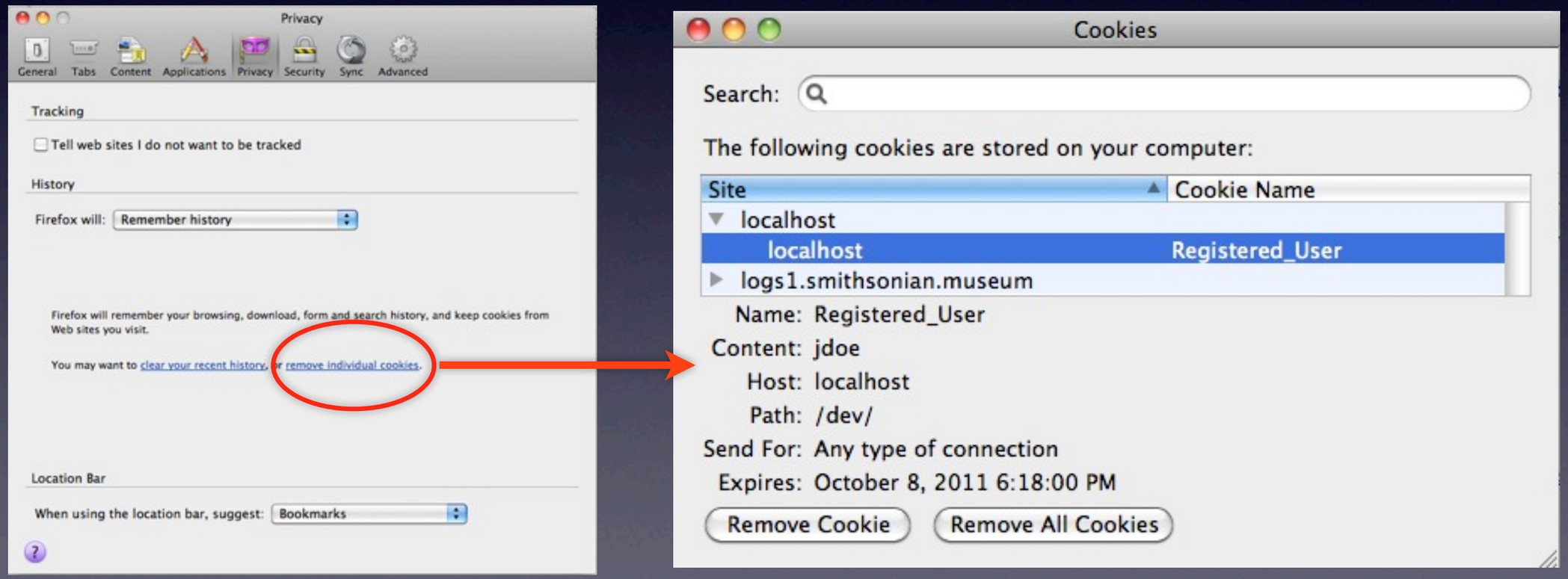
$_COOKIE is set using $_POST data

$_COOKIE name and value are stored in browser cache for possible use in all site pages.

How We Get Info From Web Users

- Cookies : $_COOKIE

If you look in your browser's cookie history, you will find the cookie. This example is set up on a local testing computer, so the host name is "localhost", although yours might be a fully qualified domain name.



How We Get Info From Web Users

- Cookies : $_COOKIE

To see how the cookie hangs onto the user name you set, make some more pages that call the cookie out as you did in the "cookie.php" page. As a reminder, this is how we did it:

```
10  <p>Welcome, user: <strong><?php echo $_COOKIE['Registered_User']; ?></strong>!</p>
```

Finally, make a page that has a logout link, which links to a new page called "cookie_logout_conf.php". This newest page will remove the cookie and give the user a logout confirmation. To remove a cookie, copy your previous "setcookie()" code and paste it before the DOCTYPE of the new page. All you have to do is change the "+" to a "-" symbol in the expiration parameter, which will essentially say that the cookie is set to expire 21 days prior to the current time. This will remove the cookie.

How We Get Info From Web Users

- Cookies : $_COOKIE

Here's a simple way to delete the same cookie on a logout page:

```
1  <?php setcookie('Registered_User', $_POST['u_name'], time()-(60*60*24*21)); ?>
2  <!DOCTYPE HTML>
```

If you check your browser's cookies after visiting this page, the old cookie for "jdoe" will be gone.

Changing the "+" to a "-" symbol subtracts time from the current time and immediately removes the cookie.

How We Get Info From Web Users

- Sessions : $_SESSION

Now, having seen how easy it is to read a cookie in plain text from your browser's cache, you might have realized that it is also a fairly insecure way to store user data for re-use.

There is another way that is a safer method . . . using sessions.

Session data is stored on the server in a file, and the session will place a cookie in your browser's cache that has a 32-character code to identify which server file is associated with the cookie. It is like a key to unlock the session data file on the server so that you can re-use the information over and over again until either 1) the user removes the cookie "key" or 2) until the server session files get cleaned up by a web administrator.

How We Get Info From Web Users

- Sessions : $_SESSION

Starting a session is not too different from setting a cookie in that you must use a function that passes headers. That means that you will need to place it before all other code in the file:

```
session_start();
```

After starting the session, you can pass info to the session array/file like this:

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

If the browser can connect to a current session with the server it will look for the matching server file. If a current session is not found, it will start a new session.

***If using the previous example files, you *MIGHT* need to remove the cookie headers prior to starting sessions, depending on your server's output buffer settings.

How We Get Info From Web Users

- Sessions : $_SESSION

Using the previous files, you can modify "form_register_do.php" like this:

```
1  <?php setcookie('Registered_User', $_POST['u_name'], time()+(60*60*24*21)); ?>
2  <?php session_start(); ?>
3  <?php $_SESSION['user'] = $_POST['u_name']; ?>
4  <!DOCTYPE HTML>
5  <html>
6  <head>
7  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8  <title>Process Form Page</title>
9  </head>
10
11 <body>
12 <p>Go to <a href="cookie.php">next COOKIE page</a>
13 to read the cookie being set in this page.</p>
14 <p>Go to <a href="session2.php">next SESSION page</a>
15 to read the session data being set in this page.</p>
```

How We Get Info From Web Users

- Sessions : $_SESSION

Next, create a new file called “session2.php”:

```
1  <?php session_start(); ?>
2  <!DOCTYPE HTML>
3  <html>
4  <head>
5  <meta http-equiv="Content-Type" conter
6  <title>Session 2</title>
7  </head>
8  <body>
9  <?php
10 $sessionName = $_SESSION['user'];
11 echo $sessionName;
12 ?>
13 </body>
14 </html>
```

How We Get Info From Web Users

- Sessions : $_SESSION

Now you can open "form_register.php" in a browser, fill out the form, submit it, and follow the "session page" link. It should print the form's User Name to screen from the session file.

Now if you look in your cookies, you will see a new cookie from your host with a cookie named "PHPSESSID". The "Content" contains a 32-character key that will be used to match the server's session file, which makes it much harder for hackers to hijack your session than to read normal cookies.