Lab 4: Cookies, Sessions and Email

Due: at the start of your next lab period.

Objectives

Upon successful completion of this lab, you will have demonstrated the ability to:

- Use PHP Apache module to send an email
- Use PHP to create, recover and modify a cookie
- Use PHP to create and use a session variable
- Use PHP to initiate and end sessions

Reference

- See lecture notes - Blackboard slides and Moodle readings

Required Materials

- None

Lab 4 Preparation

- Read the assigned sections on Email, Cookies and Sessions - see topic on Moodle

Part I - Cookies

Cookies are used by web applications to store information on local hosts through their web browsers. Cookies are associated with **the web site that created them** (as well as with a directory on that server), and **only** that web site (and directory and its subdirectories) may access them. Thus the **server-localhost** combination is part of what defines a cookie, and only this host and server share that cookie. Thus, when you return to the website that originally stored the cookie on your local machine, it is automatically sent the cookie. This enables web server programs to remember your personal preferences, credentials, etc.

Notes:

- cookies are passed through both the request and response headers, so you can't "create" a cookie after you've sent a header - it's too late since the header's already been sent!
- if you write a program that creates a cookie, and in the same program try to display that cookie using $_COOKIE, you won't see it because the cookie has not been sent and returned yet (i.e. you have to run the program again so that it receives the cookie in the response header then loads it into $_COOKIE.) This often confuses students because they forget about the cookie needing to be returned from the client to the server.

1. Write a PHP program that prompts the user for the name of a cookie and the value of a cookie. When the user presses Submit, store the name/value in a cookie for that browser. See the assigned reading on creating cookies.
2. Open a browser, and display all the cookies it has (this is done in different ways

for different browsers - but all of them will allow you to see what cookies have been stored). You should see your cookie.
3. Run your program again and create a different cookie. Check to make sure it's been created and stored by your browser.
4. Modify your program so that below the form it displays all cookies stored on that client created by your program (note that this will not display cookies you are currently creating, only those that have already been stored in a prior program run).
5. Add some more cookies and make sure they appear in the list.
6. Modify your program so that it prints the following message: "Welcome back - you visited this page *N* times" where N is the number of times this browser has visited the page. Test your program by going to the page several times. Hint: store the number of visits in the cookie.
7. For fun (if you have time), trade cookies with the person sitting next to you by visiting his/her web page. Look at the cookies generated by your page versus those generated by the other person's page.

Part II - Sessions

We'd like to keep track of "sessions" people start from a particular browser to a particular web site. This is often used so that the server can remember if the user has already logged in - i.e. you don't want them to login every time they access a web page/program on your site. We can do this through session variables. By default, PHP's session-handling mechanism generates a unique ID for each session started, and uses a single cookie, usually called `PHPSESSID`, to store a session id on the browser (it may go under another name, and you can retrieve this name using the session_id() function). It then creates a file on the server that represents this session where it stores values unique to this session with a particular client browser (user agent). To determine if a client has a session running, the session ID stored on the client is sent back to the server through the cookie, is compared to the session IDs (files) stored on the server. If the session ID exists on the server, then there is a "session" running. For example, if two people were using the add/view pages from your assignment at the same time, they would have different session IDs, and these IDs would be stored on their local hosts as the `PHPSESSID` cookie. The next time the same browser accesses your pages, the server is automatically sent the `PHPSESSID` as one of cookies (as long as it hasn't expired), and can compare the session ID of the cookie to the file created to represent the session. If a file with that session ID exists, then we know we have an active "session".

Your program may also create additional session variables associated with this session ID, which are also stored in the same file **on the sever** (but not the client!), and are accessible through the `$_SESSION` superglobal. While a programmer could write code to create custom cookies and store other information on the browser, it would be unwise to do so, and unnecessary. It is easier, and more secure, to use session variables. For example, you should never store login credentials in a cookie since a session variable, stored on the server, is much more secure. Typically, things like the contents of a shopping cart, or other personal information, would be stored in session variables, while non-personal information, like the preferences for a website (how items are displayed on a page, for example), would be stored in a cookie.

PHP has a set of functions that make it easy to create and manage session variables.

Note: like cookies, session IDs are passed through the header, so you can't call session_start( ) or set_cookie( ) after after you've sent a header - it's too late since the header's already been sent! However, since session variables are stored on the server, they may be created at any time.

1. Create a web page named **protectedstuff.php** that prints "You are logged in!"
2. Create a new table in your DB called users:

```
create table users (
username varchar(100) not null,
password blob not null,
role enum('user','admin') not null,
```

```
passwordHint varchar(100),
primary key (username)
);
```

3. Create some entries in the table. For the username, enter only email addresses (many web applications use email addresses for account ID); for the password, create only passwords of 8 or more characters that consist of upper and lowercase letters, digits, and all the special characters across the top row of the keyboard (but none of the others, especially the < and > which can be used to launch xss attacks!).
4. Write a program called **login.php** that displays a login web page prompting the user for username and password. Use a password input element (it hides your password). You do not have to validate the input!
5. On submit, compare both the username and password to each username/password set in the DB. If the **username and password match** one set, then login the user (i.e. set a session variable, such as $_SESSION['username']) and display the **protectedstuff.php** page. Note that we can check in this and other programs when they are called later to see if this session variable exists - if it does, we know this user is logged in. If the username and password **do not match** any in the DB, reprint the login form with the fields blank and an error message saying "Invalid username or password". Note that we want our error message to be vague because we don't want people using our login to check for valid user names. Telling someone the password is invalid, means the username isn't, or vice versa!
6. Revise **protectedstuff.php** so that it prompts for a login the first time a user goes to this page. The next time the user goes to this page (in the same session), it should **not** prompt for a login, but should only display the page. To do this, you need to create a session variable when a user successfully logs in (for example, you might store the user's username). When the visitor returns to any pages included in this session, start a session and check for the existence of that session variable. If it exists, there is a session running and you can safely display the page. If it doesn't exist, redirect to the login page. See the reading for an example of this.
7. Session variables may be used to manage logins across multiple pages on the same site in the same way. For each page you want to include in the session, you must start a session at the top of the page, then check for the existence of the session variable created to represent this active session. Create a second program called **page2.php** that displays the session variable you used if a session is active, or redirects to the login page if the session variable doesn't exist. Test it to make sure it works.
8. Close all instances of the browser and reopen it. Are you still logged in?
9. Add a logout link to your **protectedStuff.php** page. When clicked, it will delete the session (if there is one) - that is, it will log you out, which means the next time you try to access the page you will be required to login again. Besides unsetting `$_SESSION` and calling `session_destroy( )`, you have to render the cookie inactive on the browser by setting its expiry time in the past so it won't be sent anymore. For example,

   setcookie("PHPSESSID", "", time() - 61200,"/");

   time( ) gives the system time on the server, and subtracting 61200 (60 * 60 * 17, which takes you to the date line) should account for differences in time zone etc. on the browser. After doing all this, your program should return the user to the login page. Test your link to make sure it works. Login and logout several times, and visit your two pages to make sure it works.

Part III - email

Modify your previous program so that it has a **Forgot your password?** link. When the user clicks this link the program calls itself, generating a page that prompts the user for her email address. When the user submits this form, it checks to see if the user is in the DB table, and if she is, sends an email to her **with the username and the password hint** for that user and returns to the login page. If the email address isn't in

the DB, **don't** send an email, just return to the login page. See the assigned reading on email for an explanation on how to send email.

Note: some POP3 and IMAP servers may refuse your message or mark it as spam, especially if you don't include a "subject", "from" or "reply-to" address. So include these things.

Submission

Demonstrate Part III to my trusty lab assistant or me. To prepare, you should

- have at least two users in the DB with valid email address and password hints
- ssh to the command line of your server so that we can see the usernames/passwords in your DB table

Available from: Wednesday, 6 July 2016, 05:10 PM
