Databases and the Web – Exercise 6

Outcomes:

After you work through this exercise you should:

- Know how to use HTTP Basic Authentication.
- Use HTTP redirection with PHP pages.
- Investigate simple hashing and/or encryption strategies.

Debugging:

Because PHP is *server-side* you **must** check your code in the browser through the web server "http://..." *not* by loading the page as "file://"! It is a good idea to:

i. Make small changes to code.
ii. Save the file and/or FTP it to the web server.
iii. Refresh the page in the browser.
iv. Fix any reported syntax errors.
v. Test the changes.

Repeat *ad nauseum*. (Get used to using Alt-Tab to cycle through windows ☺)

Task 1: Try .htaccess on Linux

1. Start SUSE Linux ;-)
2. Create a subfolder within **`public_html`**, *e.g.* called "**`secure`**" and inside that create at least one HTML and one PHP page, *e.g.* by copying a file from a previous exercise.
    - Verify it works – browse to your equivalent of "http://localhost/~linux/secure/" can you see & open the files?
3. Within "secure" create a **`.htaccess`** file pointing to a "**`users`**" file *outside* the web root.
    - Verify it works – browse to your equivalent of "http://localhost/~linux/secure/" can you still see & open the files? (It probably should return an error message from the server...)
4. Use the command-line utility "**`htpasswd`**" to add a user to the **`users`** file.
    - This is *usually* part of every standard Apache installation...
5. Verify that opening "http://localhost/linux/secure/" prompts you for the password and thereafter you can see the files.
    - If you close the browser and reopen it, must you login again?
    - What's maintaining the logged-in state? Is it a client-side mechanism like a cookie or is it server-side?
    - Use a tool like Firebug to see what data are being exchanged.

Databases and the Web – Exercise 6

Unfortunately HTTP Basic Auth uses folders in your web root (by default), tends to add the authentication information to too many requests and (.htaccess) adds overhead to the web server, so building authentication into your application is generally more efficient.

Continue your login/logout exercise from last week to experiment with redirection...

Task 2: Add redirection to the secure page

Modify **`secure.php`** so that instead of showing "public" information it redirects the user to the **`login.php`** page.

Can you move the relevant code to a **`require`** file?

Examples for last week's Task 3 & this week's Task 2 can be found here:

- `http://staffnet.kingston.ac.uk/~ku13043/WebDB/ex/week06/sessions3-content.php` (source: `http://staffnet.kingston.ac.uk/~ku13043/pp.php?s=WebDB/ex/week06/sessions3-login.php`)
- `http://staffnet.kingston.ac.uk/~ku13043/WebDB/ex/week06/sessions3-login.php` (source: `http://staffnet.kingston.ac.uk/~ku13043/pp.php?s=WebDB/ex/week06/sessions3-content.php` and `http://staffnet.kingston.ac.uk/~ku13043/pp.php?s=WebDB/ex/week06/session3.inc.php`)
- `http://staffnet.kingston.ac.uk/~ku13043/WebDB/ex/week06/sessions3-logout.php` (source: `http://staffnet.kingston.ac.uk/~ku13043/pp.php?s=WebDB/ex/week06/sessions3-logout.php`)

Task 3: Secure the password?

Consider how you might secure the login password – after all, it's not that secure to *send* the password in plain text, nor is it good practice to *store* the password in plain text.

E.g. Blackboard's way of "securing" passwords is to apply an MD5 hash function to the password on the *client* (using JavaScript: http://lms.kingston.ac.uk/javascript/md5.js) and compare that *on the server* to the hashed password. There are hash functions in MySQL and PHP… look them up and implement one in your **`login.php`** page.

If you're using logins, build this into your group project & discuss the pro's & con's in the report.