PHP Piscine

Day 04

Staff 42 piscine@42.fr

Summary:

This document is the day04's subject for the PHP Piscine.

Contents

Chapter 1

Foreword

You must have finished the exercises from the prior days and watched video of Day 04, in particular the section about the security questions of the algorithms of hash.

Chapter 2

General Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- Only the work submitted on the repository will be accounted for during peer-2-peer correction.
- As when you did C Piscine, your exercises will be corrected by your peers AND/OR by Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Using a forbidden function is considered cheating. Cheaters get **-42**, and this grade is non-negotiable.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We **will not** take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- You cannot leave any additional file in your repository than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called **Google / the Internet / http://www.php.net / ....**
- Think of discussing on the Forum. The solution to your problem is probably there already. Otherwise you will start the conversation.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject …
- By Odin, by Thor ! Use your brain !!!

Chapter 3

Exercise 00 : Session

Turn-in directory : `ex00/`
Files to turn in: `index.php`
Allowed functions: `session_start()`

Create a page named **index.php** that contains a form allowing you to create and modify the username and password.

- The form will be named **index.php** and use the **GET** method.
- The username field will be named **login**.
- The password field will be named **passwd**.
- The submit button will be named **submit** and have a value of **"OK"** (if you do not receive this value in the submit field, do not modify the values stored in this session).
- Every tag in the form will have to be on one and only one line (see example).
- The fields already filled beforehand will have to be pre-filled.

Example of use, we start by testing that we are receiving the session cookie from the first access (there are a few lines missing in the headers on purpose for clarity and some bits of HTML in the form to let you work in peace):

```
$> curl -v -c cook.txt 'http://localhost:8080/d04/ex00/index.php'
> GET /ex00/index.php HTTP/1.1
>
< HTTP/1.1 200 OK
* Added cookie PHPSESSID="mfpmngdi4qjbfli03nfgrevu17" for domain localhost,
path /, expire 0
< Set-Cookie: PHPSESSID=mfpmngdi4qjbfli03nfgrevu17; path=/
<
<html><body>
<form ...>
   Username: <input ... name="login" value="" />
   <br />
   Password: <... />
  <input type="submit" ... />
</form>
</body></html>
$>
```

Then submit the form and watch the result.

```
$> curl -v -b cook.txt 'http://localhost:8080/d04/ex00/index.php?login=sb&
passwd=beeone&submit=OK'
> GET /ex00/index.php?login=sb&passwd=beeone&submit=OK HTTP/1.1
> Cookie: PHPSESSID=mfpmngdi4qjbfli03nfgrevu17
>
< HTTP/1.1 200 OK
<
<html><body>
<form ...>
   Username: <input ... name="login" value="sb" />
   <br />
   Password: <... value="beeone" />
  <input type="submit" ... />
</form>
</body></html>
$>
```

Then we reload the page without passing the values in the url to check that they are still present.

```
$> curl -v -b cook.txt 'http://localhost:8080/d04/ex00/index.php'
> GET /ex00/index.php HTTP/1.1
> Cookie: PHPSESSID=mfpmngdi4qjbfli03nfgrevu17
>
< HTTP/1.1 200 OK
<
<html><body>
<form ...>
   Username: <input ... name="login" value="sb" />
   <br />
   Password: <... value="beeone" />
  <input type="submit" ... />
</form>
</body></html>
$>
```

Finally, we remove the cookie from the request and we can see that the form is blank again and that PHP will send us a new cookie for the session.

```
$> curl -v 'http://localhost:8080/d04/ex00/index.php'
> GET /ex00/index.php HTTP/1.1
>
< HTTP/1.1 200 OK
< Set-Cookie: PHPSESSID=r7u3bnggoe91negv7aqnjkm0q6; path=/
<
<html><body>
<form ...>
   Username: <input ... name="login" value="" />
   <br />
   Password: <... value="" />
  <input type="submit" ... />
</form>
</body></html>
$>
```

Chapter 4

Exercise 01 : Create account

Turn-in directory : `ex01/`
Files to turn in: `index.html, create.php`
Allowed functions: `hash(), file_get_contents(), file_put_contents(), serialize(), unserialize(), file_exists(), mkdir()`

Create a page named **index.html** that contains a form allowing you to create an account with a username and a password. A valid account consists of a username and a non blank password (not an empty string). If the password is empty, return "**ERROR\n**". In case of success, return "**OK\n**".

- The form will be called **create.php** and use the **POST** method.
- The username field will be called **login**.
- The password field will be named **passwd**.
- The submit button will be named **submit** and have a value of "**OK**" (if you do not receive this value in the submit field, do not create an account and return"**ERROR\n**".
- Every tag in the form will have to be one and only one line (see example).
- You must store the accounts created in **/private/passwd**. That file must be a serialized array, see example.
- Every account must be an array's element, and be an array itself with a cell called **login** containing the username and another called "**passwd**" containing the hash of the password.
- You cannot store the "plain" passwords they must be "hashed" (watch elearning video).
- Choose carefully you hash algorithm (watch elearning video).
- If the username submitted already exists in the array you must return "**ERROR\n**".

Example:

```
$> rm /path/to/mamp/apache2/htdocs/d04/private/passwd
$> curl -d login=toto1 -d passwd=titi1 -d submit=OK 'http://localhost:8080/d04/ex01/create.php'
OK
$> more /path/to/mamp/apache2/htdocs/d04/private/passwd
a:1:{i:0;a:2:{s:5:"login";s:5:"toto1";s:6:"passwd";s:128:"2bdd45b3c828273786937ac1b4ca7908a431019e8b93c9fd337317f92fac80dace29802bedc33d9259c8b55d1572cb8a6c1df8579cdaa02256099ed52a905d38";}}
$> curl -d login=toto1 -d passwd=titi1 -d submit=OK 'http://localhost:8080/d04/ex01/create.php'
ERROR
$> curl -d login=toto2 -d passwd= -d submit=OK 'http://localhost:8080/d04/ex01/create.php'
ERROR
$>
```

Chapter 5

Exercise 02 : Modif account

Turn-in directory : `ex02/`
Files to turn in: `index.html, modif.php`
Allowed functions: `hash(), file_get_contents(), file_put_contents(), serialize(), unserialize()`

Create a page named **index.html** that will contain a form allowing you to modify the password associated with an account. The user will need to submit their username, their current password (that we will call "old password" to avoid any ambiguity) and their new password. We will modify the password, of course, only if the username matches an existing account and if the old password is matching the one stored in our files. If the username doesn't exist, or if the old password is wrong, or if the new password isn't valid, **modif.php** will have to return **"ERROR\n"** without any more details, otherwise **"OK\n"**.

- The form will be called **modif.php** and use the **POST** method.
- The username field will be called **login**.
- The "old password" password field will be named **oldpw**.
- The "new password" password field will be named **newpw**.
- The submit button will be named **submit** and have a value **"OK"** (if you do not receive this value in the submit field, do not modify the account and return **"ERROR\n"**).
- Every tag in the form will have to be one and only one line (see example).
- You will have to use and modify the **/private/passwd** file that contains the account created in ex01.
- You cannot store the "plain" passwords they must be "hashed" (watch elearning video).
- Choose carefully your hash algorithm (watch elearning video).

Example:

```
$> rm /path/to/mamp/apache2/htdocs/d04/private/passwd
$> curl -d login=x -d passwd=21 -d submit=OK 'http://localhost:8080/d04/ex01/
create.php'
OK
$> curl -d login=x -d oldpw=21 -d newpw=42 -d submit=OK 'http://localhost:8080/
d04/ex02/modif.php'
OK
$> more /path/to/mamp/apache2/htdocs/d04/private/passwd
a:1:{i:0;a:2:{s:5:"login";s:1:"x";s:6:"passwd";s:128:"
fee32be7c00e73eab97a39549d79af73aec87b6fa22a0b
  56867a4975fe82344cd9776c6d6dff419e0f2e415
  c492340bb8329bbfac0c872934df66466c2e0e5d3";}}
$> curl -d login=x -d oldpw=21 -d newpw=42 -d submit=OK 'http://localhost:8080/
d04/ex02/modif.php'
ERROR
$> curl -d login=x -d oldpw=42 -d newpw= -d submit=OK 'http://localhost:8080/
d04/ex02/modif.php'
ERROR
$>
```

Chapter 6

Exercise 03 : Auth

Turn-in directory : ex03/
Files to turn in: auth.php, login.php, logout.php, whoami.php
Allowed functions: hash(), file_get_contents(), file_put_contents(), serialize(), unserialize(), session_start()

Create a file named **auth.php** that contains the following function:

```
function auth($login, $passwd);
```

This function will have to return **TRUE** if the "login/passwd" combination does match an account in **/private/passwd** and **FALSE** if it doesn't. The **$passwd** variable contains the password in plain, so prior to hashing.

Create as well a file named **login.php** taking as arguments in the url: a **login** variable and a **passwd** variable. This page will start the session, validate the combination "login/passwd" and store in the session a variable **loggued_on_user** that contains:

- Either the login of the user that submitted a correct "login/passwd" combo.
- Or an empty string otherwise.

If the login/passwd combo is correct the page must display **"OK\n"**, and **"ERROR\n"** otherwise. That file will have to use **auth.php** via an include.

```
  $> rm /path/to/mamp/apache2/d04/htdocs/private/passwd
  $> curl -d login=toto -d passwd=titi -d submit=OK 'http://localhost:8080/d04/
ex01/create.php'
  OK
  $> curl 'http://localhost:8080/d04/ex03/login.php?login=toto&passwd=titi'
  OK
  $>
```

Create also a file named **logout.php** that won't take any arguments but will use the session cookie to remove that last one. That page will display nothing.

Create also a file named **whoami.php** that won't take any arguments but will use the session cookie to display the login contained in the **loggued_on_user** session variable followed by a newline. If this variable does not exist or contains an empty string, only display **"ERROR\n"**.

```
$> rm /path/to/mamp/apache2/htdocs/d04/private/passwd
$> curl -d login=toto -d passwd=titi -d submit=OK 'http://localhost:8080/d04/
ex01/create.php'
OK
$> curl -c cook.txt 'http://localhost:8080/d04/ex03/login.php?login=toto&passwd
=titi'
OK
$> curl -b cook.txt 'http://localhost:8080/d04/ex03/whoami.php'
toto
$> curl -b cook.txt 'http://localhost:8080/d04/ex03/logout.php'
$> curl -b cook.txt 'http://localhost:8080/d04/ex03/whoami.php'
ERROR
$>
```

Chapter 7

Exercise 04 : 42chat

Turn-in directory : `ex04/`

Files to turn in: `index.html, create.html, modif.html, auth.php, create.php, modif.php, login.php, logout.php, speak.php, chat.php`

Allowed functions: `session_start(), header(), hash(), file_get_contents(), file_put_contents(), serialize(), unserialize(), fopen(), flock(), fclose(), time(), date(), date_default_timezone_set(), file_exists(), mkdir()`

The goal of this exercise is to create a multi-user chat. You will have to start by taking your files from the previous exercises and make the following app:

- The welcome page **index.html** offers a form to connect to the chat (the form has an action **login.php**, with the **POST** method, update **login.php** accordingly) but also a link to create an account (**create.html**) and one to modify the password (**modif.html**)
- **create.html** is the ex01's form. It's called **create.php** according to the same rules.
- **modif.html** is the ex02's form. It's called **modif.php** according to the same rules.
- **create.php** and **modif.php** must display "**OK\n**" in case of success AND redirect the user towards the welcome page with a "**Location :**" header. The user will then have to log-in.
- In case of success, **login.php** will have to display an iframe, horizontally split, the top part will have to measure 550px and contain **chat.php**, the lower part will measure 50px and contain **speak.php**.
- **speak.php** will allow a user to post a message on the chat. You must check that the user is logged in. It's a form that calls itself with simply the msg field passed using POST. The identity of the user is taken from the session. **speak.php** will have to fill a serialized array in a **/private/chat** file. Each element of this array is an array containing : "login", "time", "msg". The "login" field is of course the login of the user that posted the message. The time field is the date/hour of when the message was sent, to the timestamp format. The msg field is of course the message.

- Be careful! You must lock the file using flock during its use (reading/writing) so that you cannot corrupt in case of a simultaneous writing by 2 users, or from partial reading during the writing by any other process.
- **chat.php** will allow you to display the content of the file **/private/char** while formatting it. See example (careful with the timezone, http://php.net/manual/en/timezones.php)

Example:

```
$> curl -d login=user1 -d passwd=pass1 -d submit=OK 'http://localhost:8080/d04/
ex04/create.php'
OK
$> curl -d login=user2 -d passwd=pass2 -d submit=OK 'http://localhost:8080/d04/
ex04/create.php'
OK
$> curl -c user1.txt -d login=user1 -d passwd=pass1 'http://localhost:8080/d04/
ex04/login.php'
...
  <iframe name="chat" src="chat.php" width="100%" height="550px"></iframe>
  <iframe name="speak" src="speak.php" width="100%" height="50px"></iframe>
...
$> curl -b user1.txt -d submit=OK -d msg=Bonjours 'http://localhost:8080/d04/
ex04/speak.php'
...
$> curl -b user1.txt -c user1.txt 'http://localhost:8080/d04/ex04/logout.php'
$> curl -b user1.txt -d submit=OK -d msg=Bonjours 'http://localhost:8080/d04/
ex04/speak.php'
ERROR
$> curl -c user2.txt -d login=user2 -d passwd=pass2 'http://localhost:8080/d04/
ex04/login.php'
...
$> curl -b user2.txt -d submit=OK -d msg=Hello 'http://localhost:8080/d04/ex04/
speak.php'
...
$> more /path/to/mamp/apache2/htdocs/d04/private/chat
a:2:{i:0;a:3:{s:5:"login";s:5:"user1";s:4:"time";i:1364287362;s:3:"msg";s:8:"
Bonjours";}i:1;a:3:{s:5:
"login";s:5:"user2";s:4:"time";i:1364287423;s:3:"msg";s:5:"Hello";}}
$> curl -b user2.txt 'http://localhost:8080/d04/ex04/chat.php'
[09:42] <b>user1</b>: Bonjours<br />
[09:43] <b>user2</b>: Hello<br />
$>
```

Free bonus:

Once you have done all that, you will notice that the iframe **chat.php** won't reload by itself when you are using **speak.php**. Here is a bit of javascript to all to the head of your **speak.php**:

```
<script langage="javascript">top.frames['chat'].location = 'chat.php';</script>
```

Careful: Do update your **/private/chat** file BEFORE returning your page html code to avoid the navigator to reload **chat.php** too early. Good luck ;-)