Auteur : Spl3en
Sujet : Tutorial – Apprentissage PHP
Fait le : 31/03/10

Le but de ce tutorial sera de vous apprendre en partie le merveilleux langage qu'est le PHP.
La seule différence qui fera de ce tutorial sur le PHP un tutorial un peu spécial, c'est qu'il aura pour but de vous **éveiller sur le côté sécuritaire des applications PHP, autant en terme d'attaque que de défense.**
Ainsi, vous n'apprendrez pas forcément à construire un site dynamique via ce tuto, mais plutôt à construire quelque chose de sécurisé / être en capacité de coder quelque chose d'exploitable.

Attention, j'insiste, mais notez bien que ce tutorial n'a pas du tout pour but de vous apprendre tout ce que PHP peut vous offrir ... !
Il ne servira que de "petit manuel" de ce qui aurait tendance à intéresser un éveillé à la sécurité informatique...
Ainsi, vous devriez devenir prêt à "comprendre" la majorité des types de failles web courantes.

Si vous voulez plus d'informations sur le PHP, ou si vous avez un doute durant le tuto, une magnifique documentation en français est disponible en ligne.
Je sauterai donc volonterairement des notions qui ne nous serons pas utiles pour notre tutorial.

Avant de commencer, vérifiez que votre installation de PHP marche ;
Si vous n'avez pas installé de serveur PHP, je vous invite à suivre ce tutorial qui vous permettra de l'installer en moins d'une minute.

Sommaire

Voici comment se construira le tutorial :

- **I) Les variables et les types**
  - a) Introduction sur les variables
  - b) Entier
  - c) Chaîne de caractère
    -> Point d'approfondissement personnel : Les Arrays

- **II) Les branchements conditionnels**
  - a) If, else
  - b) Les opérateurs logiques
  - c) Les boucles : while, for

- **III) Fonctions**
  - a) Les fonctions "officielles"
  - b) Créer sa fonction

- **IV) Entrée et sortie**
  - **i) Entrée**
    - a) GET / POST, quesako ?
    - b) Lecture d'un fichier
    - c) Un cas un peu spécial : les Cookies
    - d) Introduction aux abus via les Cookies
  - **ii) Sortie**
    - a) Affichage sur la page
    - b) Introduction à la faille XSS
    - c) Ecriture dans un fichier
    - d) Introduction au log poisoning

- **V) L'inclusion, a.k.a la conception d'un script plus avancé**
  - a) L'inclusion de fichier
  - b) Introduction à la faille include

Allez ... Prenez votre courage à deux mains, et on commence ! :)

I) Les variables et les types

a) Introduction sur les variables

Il faut tout d'abord savoir qu'est ce qu'on entend par "variable".
Une variable est une sorte de "**boîte**", qui vous permettra de stocker une valeur quelconque.
Lors d'un script, vous devrez donc jouer avec différentes variables afin de produire **un résultat** ; Nous verrons dans la suite du tutorial comment les utiliser.

Dans un code en PHP, les variables sont facilement reconnaissables :
Elles commencent toutes par un "**$**".

Voyons maintenant comment remplir ces "boîtes" :

b) Les entiers

Prenons un exemple concret :
On veut résoudre l'équation mathématiques "x = 1+1".
(pas facile n'est ce pas ?)
Comment va-t'on exprimer cela en PHP ?
Facile !

```php
<?php

$somme = 1+1;

?>
```

Et voilà, $somme contiendra le résultat ... C'est à dire 2 !
Vous remarquerez le "<?php ?>" en début et fin de script...
Il est important, il permet de délimiter un script php ! Sans eux, rien ne marche !

Il faut savoir que rien n'empêche de faire ce genre de calcul entre variables ...

Exemple :

```php
<?php

$a = 10;
$b = 5;
$somme = $a + $b; // $somme = 15
$division = $a / $b; // $division = 2
$multiplication = $a * $b; // $multiplication = 50
$soustraction = $a - $b; // $soustraction = 5

?>
```

Plutôt simple, n'est ce pas ?
Notez que j'utilise des commentaires avec la syntaxe "//". En effet, tout ce qui est situé après ce signe ne sera pas pris en compte ... Vous pouvez donc écrire ce que vous voulez dans les commentaires !

Aussi, si vous souhaitez "afficher" vos résultats pour vérifier, vous pouvez utiliser la commande "echo" : par exemple, "echo $a" affichera "10", la valeur de $a.
Nous reviendrons plus en détail sur l'affichage à la suite du tutorial, pas d'inquiétude.

c) Chaîne de caractère

Maintenant, comment faire pour par exemple mettre un prénom dans une variable ?
Encore une fois, c'est tout simple :)

```php
<?php

$prenom = "Spl3en";

?>
```

Notez bien qu'il y a des guillemets autour du nom ;
C'est ce qui permet de repérer une chaine de caractères :)

Maintenant, imaginons le cas suivant :

```php
<?php

$prenom = "Spl3en";
$activite = "Mange des cacahouètes";

?>
```

J'ai envie de mettre ces deux chaînes dans une seule et unique ... Comment je fais ? :(
Il existe en PHP un petit outil qui permet de **concaténer** deux chaînes : Le point "." .
Par exemple :

```php
<?php

$prenom = "Spl3en";
$activite = "Mange des cacahouètes";

$final = $prenom . " " . $activite;

?>
```

Ma variable $final contiendra la chaîne "Spl3en Mange des cacahouètes" ainsi :) !
=> "Spl3en" + " " (espace) + "Mange des cacahouètes"

II) Les branchements conditionnels

a) If, else

Bon, c'est bien beau tout ça, mais dans l'histoire ça me sert pas à grand chose de faire des calculs comme ça en PHP !

Rassurez vous, ça n'est pas fini :P
Il existe comme dans tout type de langage de programmation ce qu'on appelle des "branchements conditionnels".

Qu'est ce que c'est ?
C'est ce qui va permettre dans votre script de prendre une direction ou une autre, selon la valeur de la variable testée.

Voilà sa syntaxe :

```php
<?php

$a = 3;
if ($a == 3) {
   // $a est bien égal à 3 !
   // Les accolades servent à "délimiter" l'espace où la conditio
n est vraie
   // Donc tout le code à partir d'ici sera exécuté si $a = 3 !
}

?>
```

Ok, cool ... Mais si $a n'est pas égal à 3 alors ?

On a juste à dire "sinon je fais ça", avec **"else"**, comme cela :

```php
<?php

$a = 3;
if ($a == 3)
{
   // $a est bien égal à 3 !
}
else
{
   // Cas où $a est différent de 3
}

?>
```

Maintenant, on pourrait imaginer un petit script qui détermine si la somme de deux nombre est égal à 10, et dans le cas contraire voir si la somme de deux autres nombre est égal à 5 !

```php
<?php
$a = 2;
$b = 8;
$c = 3;

if ( $a + $b == 10)
{
  // $a + $b est égal à 10 ici ! Cool :)
}
else if ( $a + $c == 5)
{
  // $a + $c = 5 ici !
}
else
{
   // Afficher un message d'erreur par exemple
}
?>
```

En résumant en français parlé, ça donnerait :
"a est égal à 2, b est égal à 8, et c est égal à 3.
si (a + b est égal à 10) alors
// On fait quelque chose ici
Sinon, si (a + c est égal à 5), on fait quelque chose d'autre.
Sinon, afficher un message d'erreur"

Pas super utile ? Pensez vous donc !
Voici un petit exemple concret.
Imaginez que vous voulez faire un petit script d'identification :

```php
<?php

if ( $password == "HelloWorld" )
{
    // Le mot de passe est correct !
    // -> Acceder au panel admin
}
else
{
   // Le mot de passe est incorrect :
   // Afficher message d'erreur
}

?>
```

Ce n'est pas vraiment sécurisé, mais dans l'esprit c'est pas plus dur que ça !

b) Les opérateurs logiques

Je ne sais pas si vous avez remarqué, mais dans mon "if", j'utilise un opérateur :
"==", qui veut dire "est-ce que les deux variables d'un côté et d'un autre sont bien égales ?"
Attention ! A ne pas confondre avec le "=" simple, qui sert à mettre une valeur dans une autre.

Il existe un petit paquet d'opérateur logiques :

== : est égal
!= : différent
>, < : supérieur, inférieur
>=, <= : supérieur ou égal, inférieur ou égal

Vous pouvez donc faire aussi :

```
<?php

$a = 3;
$b = 2;
if ($a > $b) {
   // faire quelque chose si a est supérieur à b ici
}
else if ($a != b) {
   // faire quelque chose si a est différent de b ici
}

?>
```

Et ce n'est pas tout !
Il existe deux autres opérateurs très utiles :
&& = "et"
|| = "ou"

Qu'est ce que c'est que ça encore ?

Imaginons que nous voulons réunir **deux** conditions pour réaliser une action ...
Nous avons besoin de l'opérateur "et" && !

Exemple :

```php
<?php

if ($login == "Spl3en" && $password == "HelloWorld")
{
   // Le login est bien Spl3en, et le password est bien "HelloWor
ld" ici !
}

?>
```

Concernant l'opérateur "ou", c'est le même topo :

```php
<?php

if ($login == "Spl3en" || $login == "admin")
{
   // Le login est bien "Spl3en" ou "admin", on accède au panel d
'amin
}

?>
```

Vous comprendrez que c'est très utile de bien utiliser ces opérateurs :)

c) Les boucles : while, for

Bon, ça gère déja pas mal de savoir faire tout ça !
Mais voilà, imaginez si vous avez envie de faire une tâche répétitive, du genre vérifier 10 fois si un nombre est égal à 10 ?
Au lieu d'écrire 10 fois la même condition, on peut utiliser une ... boucle !

```php
<?php

$a = 10;

while ($a != 0)
{
   $a --; // Cette écriture revient à la même chose que dire : $a
= $a - 1;
}

?>
```

Suivez bien ce qui a été fait ici :

On dit :

a est égal à 10.
Tant que a est différent de 0, je fais ce qu'il y a à l'intérieur des accolades, à savoir : Je décrémente $a ($a = $a - 1). Donc $a = 9 après un premier tour de boucle.
Une fois arrivé à la parenthèse fermante, je remonte à la condition : $a = 9, donc c'est toujours bon ! Puis c'est reparti...

Ainsi de suite, jusqu'à ce que $a = 0.
Dans ce cas, la condition n'est pas vérifiée, donc on ne rentre pas dans les accolades, on passe à la suite du script.

Pour "**for**", c'est presque pareil, en plus "simple" :

```php
<?php

for ($i=0; $i<10; $i++)
{
   // faire quelque chose
}

?>
```

Ce code veut dire :
- on met $i = 0 au premier tour de la boucle
- tant que $i est inférieur à 10, on fait ce qui est à l'intérieur des accolades
- une fois arrivé à la fin, on incrémente en faisant $i++ ($i = $i + 1), puis on revérifie la condition -> ainsi de suite jusqu'à ce qu'elle ne soit pas vérifiée.

Vous voilà donc prêt à coder vos propres fonctions, maintenant que vous savez tout ça !... Ce qui sera l'objet de notre prochain point :)

III) Les fonctions

a) Les fonctions "officielles"

Il faut savoir d'abord ce qu'est une fonction :
Dans le cas de PHP, ça sera une "boîte" où vous pouvez rentrer plusieurs paramètres en entrée ; La fonction vous sortira un résultat en sortie selon ce que vous voulez faire.
Pour que cela soit plus clair, voyons la fonction "str_replace", qui sert à remplacer dans une chaîne de caractère un bout de chaine par une autre :
Documentation

```php
<?php

$string = "Bonjour le monde";
$nouvelle_string = str_replace("le monde", "PHP", $string);

// $nouvelle_string sera égal à "Bonjour PHP" !

?>
```

Il existe en PHP des tonnes de fonctions à votre disposition !
Vous pouvez un listing de toutes les fonctions ici :
http://fr.php.net/manual/fr/funcref.php

Notez que celles qui risquent de plus vous interessez sont dans :

http://www.php.net/manual/fr/ref.strings.php > Fonctions sur strings
http://fr.php.net/manual/fr/ref.filesystem.php > Fonctions sur fichier.

Faîtes l'effort d'en lire le maximum possible, vous verrez que plus vous en connaîtrez, plus vous vous simplifierez la vie : **Elles sont là pour ça !**

b) Créer sa fonction

Mais voilà qu'il y a une fonction que ne fais pas exactement ce que vous voulez :(!
Il y a une solution : Nous allons créer notre propre fonction, rien qu'à nous !

Voilà comme ça se passe, syntaxiquement :

```php
<?php

function nom_de_la_fonction (paramètre_1, paramètre_2, ...)
{

    // Ici, on manipule ce qu'on veut faire

    return $quelque_chose;
    // Ici, on "renvoie" la sortie de la fonction,
    // c'est à dire le résultat qu'on compte donner à cette fonct
ion !
}

?>
```

Par exemple, si on a envie de faire une fonction qui fait la moyenne de deux nombres :

```php
<?php

function moyenne ($a, $b)
{
    $c = ($a + $b) / 2;

    return $c;
}

// Dans notre script, on pourra donc l'utiliser de cette manière
:

$c = moyenne (2, 4);

// $c sera égal à 3, la moyenne entre 2 et 4 !

?>
```

Rien ne vous empêche de créer des fonctions, à partir de celles officielles ... C'est même conseillé :)

IV) Entrée / Sortie

i) Entrée

a) GET / POST, quesako ?

Vous avez peut être déja croisé ses mots, parmi d'autres...
Et vous les utiliser terriblement souvent, sans le savoir !
En effet, regardez l'URL de ce topic :

http://www.site.fr/showthread.php?tid=1234

Remarquez qu'après le "showthread.php" qui indique le script php qui est éxecuté, un petit "tid=1234" ...
Qu'est ce que c'est que ça ?

C'est en fait une variable que le script PHP peut récupérer, via un tableau appelé "**GET**".
Ainsi, quand le script PHP showthread sera lancé, on aura $_GET['tid'] qui sera égal à 1234 !
Absolument tout ce qui passe par l'URL sera enregistré dans ce tableau "$_GET['nom_de_la_variable']" ... Bien utile :) !

Imaginons l'exemple suivant :
On a dans l'url : http://site.fr/script.php?a=2&b=4

Et le script suivant :

```php
<?php

function moyenne ($a, $b)
{
    $c = ($a + $b) / 2;
    return $c;
}

$c = moyenne ($_GET['a'], $_GET['b']);

// $c sera la moyenne des deux nombres passés dans l'url !

?>
```

De cette manière, on peut récupérer très simplement des données via l'URL ! :)

Maintenant, **POST** !
Sans que vous le sachiez, vous utilisez cette méthode de faire transiter des informations à chaque fois que vous ... POSTez sur un forum !
En effet, les données ne passent pas par l'URL, mais directement au script PHP.

Désavantages :
Comme les données ne passent pas par l'URL, si quelqu'un souhaite obtenir le même résultat que vous par exemple, il sera contraint de retaper dans le champ la même chose que vous.

Avantages :
C'est quand même plus sécurisé ... Imaginez une identification où vous pourriez voir votre login/mot de passe en clair dans l'URL ! Ca serait plutôt mauvais...

Vous devez donc bien faire attention, et **utiliser au bon moment** l'un ou l'autre.

Voilà comment cela s'utilise :

Le fichier HTML formulaire.html : (pas détaillé, vous êtes censé connaître un peu de HTML déjà)

```
<form action="script.php" method="post">
   <input type="text" name="search" />
   <input type="submit" />
</form>
```

Le script php script.php :

```
<?php

if (isset($_POST['search'])) // Si la variable existe bien
{
    echo $_POST['search']; // On l'affiche à l'écran
}

?>
```

Ainsi, notre script "affichera" le texte envoyé dans le formulaire via **POST** :)

Euuuh... Et c'est quoi « isset » ?

La fonction isset ('*is set?*') permet de vérifier qu'une variables est bien définie.

En effet, lorsque vous tentez de manipuler une variable qui n'est pas définie, le script vous renverra une erreur. (imaginez le cas où l'utilisateur ne soit pas très doué, et aie oublié d'envoyer quelque chose dans le champ de recherche...)

b) Lecture d'un fichier

Il existe en PHP une fonction très simplissime pour lire un fichier, qui est "**file_get_contents**".
Il y en a bien d'autres qui permettent la lecture, mais celle-ci est vraiment passe partout ;
Elle s'utilise ainsi :

```php
<?php

$contenu = file_get_contents("fichier.txt");
// $contenu sera égal à une chaine de caractère du contenu de tou
t le fichier texte !
// Par exemple, s'il y avait "bonjour" dans fichier.txt, $contenu
sera égal à "bonjour"...
// Simple non ? ;)

?>
```

Maintenant, on va aller encore plus loin, et vous faire découvrir le merveilleux côté de cette fonction...
Elle marche aussi pour les URL !
Ainsi, rien ne vous empêche de récupérer le code source d'une page comme vous le souhaitez ! :)

Exemple :

```php
<?php

$contenu = file_get_contents("http://google.fr");
// $contenu sera égal au code source HTML de http://google.fr,
// exactement comme vous le recevez avec votre navigateur...
// Magique n'est-ce pas ? :)

?>
```

Je vous laisse imaginez le nombre d'applications que cette fonctionnalité permet d'exploiter...
Récupérer les dernières news d'une autre page, traiter des informations comme vous le souhaitez (link checker/grabber), etc !

Et pour les accrocs du phising, il est également possible de coder un phishing dynamique, qui change d'apparence lorsque la page phishée change :)
Evidemment, ça serait vraiment petit de limiter ses fonctionnalités à du phishing, tant le nombre d'applications est important...!

c) Un cas un peu spécial : les Cookies

Il reste un ultime point d'entrée qu'on peut simplement utiliser, ces fameux « cookies » dont vous avez sûrement déjà entendu parler.

Qu'est ce que c'est, concrètement ?

Rien à voir avec les petits gâteaux au chocolat (hélas!), ce sont de petits fichiers, chargés de garder en mémoire certains paramètres ;
On peut y stocker des variables quelconque, comme l'heure de connexion, la résolution de l'écran, le thème utilisé, etc... Bref, vous êtes libre d'y mettre ce que vous voulez.

L'intérêt ?

Les cookies, s'ils n'ont pas été effacés, sont conservés sur l'ordinateur.
Ainsi, d'un jour à l'autre, vous pouvez garder des informations du côté du client, sans pour autant encombrer votre base de données d'une tonne de variables à enregistrer.
C'est principalement dans ce but que les cookies existent :
Sauvegarder des données côté client, pour éviter au serveur d'amasser des données inutiles.

Et comment ça marche ?

Tout simplement comme ceci :

```php
<?php

setcookie('nom_cookie', 'contenu_cookie', (time() + $temps_sec));

?>
```

Suite à cette ligne, un cookie sera écrit avec le nom et la variable entré dans les deux premiers paramètres.
Et le troisième ?
Il s'agit du **temps de validité** du cookie, exprimé en secondes.

Une fois ce temps dépassé, le cookie sera détruit.

Bon, certes... Et comment je récupère une valeur du cookie moi, une fois que c'est écrit ?

C'est d'une simplicité déconcertante !
La valeur de votre cookie 'nom_cookie' sera contenu dans :

```php
<?php

$_COOKIE['nom_cookie'];

?>
```

Ni plus, ni moins !
Vous pouvez donc utiliser cette variable comme une autre :)

d) Introduction aux abus de privilèges via les Cookies

Quoi ? Abus de privilèges ? Mais où ça ?
Certes, le titre est bien aguicheur pour ce que c'est :P

Imaginez le cas où vous voudriez sécuriser un pannel d'administration.
Au préalable, quand vous vous inscrivez pour la première fois, vous écrivez un cookie 'is_admin', égal à 0 si vous êtes un membre, et égal à 1 si vous l'êtes.
Ainsi, lors de vos prochaines connexions au panel d'administration, inutile de vous identifier, puisque votre cookie servira de « patte blanche » pour le script PHP :

Ainsi, pour vérifier qu'un admin tente de se connecter, vous faîtes :

```php
<?php

if ($_COOKIE['is_admin']) {

  // Ton cookie prouve bien que tu es l'administrateur, tu peux r
entrer.

}
else {

  // Erreur, tu n'es pas admin !

}

?>
```

Certes, ça fonctionne à merveille.
Mais cela vous est il traversé l'esprit qu'à **n'importe quel moment**, le client peut modifier son cookie 'is_admin', et passer sa valeur à 1, rendant l'accès au panel d'administration totalement ouvert ?

Et oui, il s'agit là d'une modification du cookie que vous n'aviez pas du tout prévu, enfermé dans l'idée que ce dernier obéirait à la façon dont vous l'avez crée.
En moralité : méfiez vous des cookies comme vous vous méfierez d'autres variables reçues en POST ou en GET !

Ils sont autant exposés que ces derniers aux diverses failles que vous rencontrerez au cours de votre apprentissage, voir même plus souvent que les autres : Le webmaster oublie fréquemment ou même ignore que l'utilisateur peut les modifier.

Pour votre information, il existe beaucoup de plugin sous votre navigateur qui vous permet de modifier vos Cookies.
Sous Firefox, on peut citer parmi d'autres WebDeveloppeur, ou le fameux HTTP Live Headers si on ne souhaite modifier sa valeur qu'à l'envoi.

ii) Sortie

a) Affichage sur la page

Voilà le moment tant attendu d'apprendre à afficher nos résultats sur notre page, afin de vérifier nos informations !

Comme déja dit précédemment, il est tout à fait possible d'utiliser la commande "echo", utilisable pour les string et les entiers :

```php
<?php

$string = "HelloWorld";
echo $string;

?>
```

Notre script affichera "HelloWorld" ... Jusque là, tout va bien.

De la même manière, il existe pour les array la fonction print_r($array), qui affichera un peu spécialement certes, mais affichera le contenu de votre tableau.
Pour avoir un rendu détaillé de tout ce que contient une variable, vous pouvez utiliser var_dump($var), pour tout type de variable :)

b) Introduction à la faille XSS

Faille ixèssèss ? Quoi, oulà mais qu'est ce que ça vient faire là ça ?!

Comme vous le savez peut être, une page HTML peut appeler des scripts extérieurs pour étendre les fonctionnalités de la page.
Par exemple, faire une petite animation lorsque vous passez le curseur sur un menu, etc ...
En général, on utilise du Javascript pour cela.

Maintenant, essayez de scripter le code suivant :

```
<?php

echo '<script>alert("Bonjour PHP!")</script>';

?>
```

Lancez le ; Oh surprise! une petite fenêtre avec notre texte apparaît :)
Maintenant, réfléchissez...

Voici le script suivant :

```
<?php

if ( isset( $_GET['prenom'] ) )
{
    echo "Votre nom est : " . $_GET['prenom'];
}

?>
```

Vous faîtes donc passer la variable "prenom" en url, du genre :
http://site.fr/script.php?prenom=Spl3en
Ce qui affichera :

Votre nom est : Spl3en

Où est-ce que je veux en venir ?
Imaginons qu'on contourne l'utilisation du script et qu'on aille sur :
http://site.fr/script.php?prenom=Spl3en<script>alert(0)</script>

Surprise ! Une alerte javascript s'éxecute ... Et oui, à aucun moment, vous n'aviez prévu que l'utilisateur injecte du Javascript dans son prénom !

C'est une faille grave : L'attaquant peut très bien **récupérer vos cookies** via cette technique en une fraction de secondes, si vous allez sur une URL piégée...
Voir pire ! Dans certains cas, il est même possible d'installer un keylogger limité au site, lancer des exploits via Javascript visant la victime, et d'autres crasses du même genre.

C'est pourquoi il est important d'utiliser la fonction "**htmlentities**" quand vous faîtes un echo d'une variable donnée par l'utilisateur.
Retenez qu'il ne faut jamais faire confiance à des données reçues de l'utilisateur ;
A un moment où l'autre, il peut toujours y insérer des données dont vous n'aviez pas prévu les conséquences.

Voici le code qui évite la XSS :

```php
<?php

if ( isset( $_GET['prenom'] ) )
{
    $secure_prenom = htmlentities($_GET['prenom']);
    echo "Votre nom est : " . $secure_prenom;
}

?>
```

Vos "<" et ">" seront encodé en **"<" et ">"**, quelque chose d'**innofensif**, rendant l'éxecution impossible du javascript, mais l'affichage reste *exactement* le même :) !

On aura donc pour http://site.fr/script.php?prenom=Spl3en<script>alert(0)</script>

Votre nom est : Spl3en<script>alert(0)</script>

En réalité, si vous regardez le code source, vous verrez :

Votre nom est : Spl3en<script>alert(0)</script>

On voit clairement ici nos balises encodées et n'exécutant dorénavant plus rien... Objectif atteint ;) !

c) Ecriture dans un fichier

Décidemment, PHP sait tout faire vous vous direz... Vous avez bien raison, écrire des fichiers n'est pas un problème pour lui non plus :)
Rappelez vous, je vous ai parlé de la fonction "file_get_contents" tout à l'heure, pour **lire** dans un fichier, n'est ce pas ?

Et bien, j'ai l'honneur de vous présenter sa soeur, la fonction "file_put_contents", qui permet d'écrire dans un fichier !

Voilà en résumé de sa manière de fonctionner :

```php
<?php

$fichier = "fichier.txt";
$contenu = "HelloWorld";

file_put_contents($fichier, $contenu);

?>
```

Et voilà, de cette manière vous avez tout simplement réussi à écrire dans fichier.txt un petit message qui est "HelloWorld" ! :)
Pour plus de détail concernant cette fonction, je vous invite à lire la documentation à ce sujet ;)

C'est aussi simple que cela !

d) Introduction au log poisoning

Attention, cette partie du tutorial vous demandera plus d'attention pour la comprendre !
Log poisining ... Ca n'a pas l'air très gentil ça, rien que vu le nom, pas vrai? En effet, c'est une attaque terriblement vicieuse et puissance !
De cette manière, vous pouvez, selon le cas, créer une "XSS persistant" (une XSS qui n'a pas besoin d'être passée dans l'URL ou en POST car elle est "incrustée" dans le site), voir même ... De prendre le contrôle du serveur combiné à d'autres failles ! Gravissime, vous l'aurez compris...

Voyons la bête :
Imaginons le script qui permet d'écrire dans un fichier le nom passé en paramètre par l'utilisateur :

```php
<?php

if ( isset( $_GET['prenom'] ) ) {
    file_put_contents("fichier.txt", $_GET['prenom']);
}

?>
```

Vous venez donc d'écrire le prénom passé tel quel en paramètre...
Hé mais !
Quel andouille ce webmaster, il n'a pas utilisé htmlentities comme on lui avait appris, l'utilisateur peut injecter du Javascript dans la variable ! :(
Mais bon, peut être que vous vous direz que ce n'est pas si grave que cela, dans un fichier.txt, ça ne fait de mal à personne... Pensez vous ?

Voyons maintenant un script qui permettrait de lire et d'afficher le prénom du fichier :

```php
<?php

    $fichier = file_get_contents("fichier.txt");
    echo $fichier;

?>
```

Vous l'avez deviné : $fichier contiendra le contenu du fichier "fichier.txt"...
Et que se passe-t'il si le nom écrit auparavant contient du javascript ? Boum, une potentielle XSS ! :(

Imaginez les dégâts : A **chaque fois** que quelqu'un appellera le fichier qui s'occupe de "lire" le fichier.txt, il exécutera votre XSS...
Un carnage !

Ce type de XSS est appelé "**persistant**", car elle reste "inscrustée" dans le site, enregistrée quelque part (fichier ou base de données)...
Rappelez vous en, ce sont les XSS les plus intéressantes à exploiter pour les hackers.

On retrouve très fréquemment ce type de XSS dans les livres d'or, dans les système de news, dans les chatbox, ...

Pourtant, ça n'est pas compliqué à s'en protéger : un simple htmlentities permet d'enrayer la plupart des risques :) !

C'est tout ce que ça fait le log poisoning ?

Et bien non !
Imaginez si vous vous amusez à injecter du code PHP dans votre variable ...
Et que vous arriviez à l'éxecuter ?
Ainsi, vous auriez un contrôle non plus au niveau du client, mais du serveur !
Mais ceci est une autre histoire ...

V) L'inclusion, a.k.a la conception d'un script plus avancé

a) L'inclusion de fichier

Bon, maintenant que vous savez faire le B.A.-BA du PHP, peut être avez vous envie de faire un gros script ?
Imaginez dans le cas d'un site que vous ayiez fait dans un fichier différent les fonctions pour traiter des chaînes de caractères, un fichier pour sauvegarder/charger des données, un autre pour afficher...

Oulà, mais comment faire pour joindre les bouts avec tout ça ?
Mettre dans un seul et unique fichier ? Et si les autres fichiers ont besoin de ces fonctions, je dois faire plein de copier/coller ?

Et bien non ! Il suffit d'utiliser la fonction "include" :
Imaginez vos fichiers :

fonctions_traiter.php

```php
<?php
function traiter ($a, $b)
{
    $c = ($a + $b) / 2;
    return $c;
}
?>
```

fonctions_fichier.php

```php
<?php
function save ($fichier, $data)
{
    file_put_contents($fichier, $data);
}

function load ($fichier)
{
    return file_get_contents($fichier);
}
?>
```

fonctions_afficher.php

```php
<?php
function afficher ($data)
{
    echo htmlentities($data);
}
?>
```

Rien ne vous empêche, dans un fichier script.php, de faire :

```php
<?php

include("fonctions_traiter.php");
include("fonctions_fichier.php");
include("fonctions_afficher.php");

$data = traiter(2, 6);
save("fichier.txt", $data);

$data = load("fichier.txt");
afficher($data); // Affichera "4"

?>
```

Ainsi, grâce à la fonction "include", vous pouvez retrouver toutes vos fonctions sans souci !
C'est la magie de l'inclusion :)

Concrètement, ça donne quoi ?
Et bien, imaginez le cas où vous voudriez faire un site ...
Plus besoin de faire d'affreux copier coller pour inclure un menu par exemple, il suffit de le coder dans un script, puis de faire include("menu.php") !
Idem pour les headers, ou pour tout autre bout de page répétitif :)

b) Introduction à la faille include

Une des failles web des plus dévastatrices : en théorie, elle permet d'inclure n'importe quel fichier situé sur votre serveur.
Il existe un cas (RFI) où il est possible d'inclure un fichier sur un serveur distant, mais seule la LFI, càd l'inclusion de fichier sur le serveur local ne sera introduite ici :

Voici un exemple de script faillible, pour servir d'explication :

```php
<?php

if (isset ($_GET['page']))
{
    $script_a_ouvrir = $_GET['page'] . ".php";
    include ($script_a_ouvrir);
}

?>
```

A la base, ce n'est pas si bête : On passe par l'URL le script à ouvrir.
Par exemple, on aurait pu faire http://site.fr/index.php?page=news , ce qui aurait eu comme effet d'include "news.php", et donc d'éxecuter le script à l'intérieur.
C'est simple et génial à la fois !

Seulement voilà ... Que se passe-t'il si un petit plaisantin s'amuse à changer à sa guise cette variable ?
Il y a un petit souci ... Car on va lui autoriser d'inclure absolument n'importe quel fichier.

Par exemple, s'il met : http://site.fr/index.php?page=../../../..../passwd%00 , et que le serveur tourne sur unix, le fichier "/etc/passwd" contenant les identifiants et quelques fois mots de passe des utilisateurs du serveur distant sera affiché ! Une catastrophe !

Imaginez que ce même plaisantin aie réussi à uploader sur votre machine un script en PHP dans fichier contenant intialement une image, il ne suffira plus qu'à l'inclure, et adieu votre serveur adoré :(...

Plusieurs solutions s'ouvrent alors à vous pour vous protéger : Par exemple, faire une liste blanche :

```php
<?php

if (isset ($_GET['page']))
{
    if ($_GET['page'] == "news")
        include ("news.php");

    else if ($_GET['page'] == "search")
        include ("search.php");

    //...
}
?>
```

C'est un peu long et fastidieux à entretenir, mais aucun problème de ce côté au moins pour la faille include au moins ;) !

~----------------~

Voilà, ce tutorial est maintenant terminé ...
Dorénavant, je vous conseille de vous penchez plus en profondeur sur :

- les XSS,
- la LFI/RFI
- Log Poisoning
- CSRF, pouvant, entre autre, être mise en oeuvre via une XSS
- Faille Upload
- SQL Injection
- et bien d'autres encore ;) !

Une fois tout cela maîtrisé, vous aurez un point de vue plus ouvert pour apprendre d'autres failles web plus poussées ...
Bon courage, et n'hésitez pas à poser vos questions par mail.