Failles PHP à éviter

Date de dernière mise à jour : 27/06/2007 à 19:36

La sécurisation de scripts PHP est une étape importante à prendre en compte lors de toute conception de site ou de script, simple ou complexe.

PHP permet déjà si on le configure correctement d'éliminer pas mal de problèmes de sécurité (on parlera à ce sujet notamment des **register_globals**). Nous traiterons ensuite quelques exemples pratiques couramment employés dans certains sites web et qu'il ne faut surtout pas faire. Enfin nous parlerons de quelques techniques de dissimulation (Url Rewriting, Force download), pour dissimuler des URL et des fichiers.

Les register_globals :

PHP est enfin configuré avec le paramètre **register_globals** à off par défaut. Vous pouvez désactiver les register_globals via le fichier **php.ini**, recherchez la ligne **register_globals** et attribuez-lui la valeur **Off**. Vous pouvez également passer par un fichier **.htaccess**, vous devrez dans ce cas rajouter la ligne suivante : **php_flag register_globals off**. Ceci désactivera les register_globals, qui sont une vraie plaie pour la sécurité d'un script.

Nous allons maintenant présenter quelques exemples à ne pas faire et comment corriger les failles potentielles :

La faille include :

Prenons un exemple simple de cette faille : La plupart des sites ayant cette faille utilisent des url du style **monsite.com/index.php?page=xxxxx** ou xxxxx est une page qui sera inclue comme ceci :

```
<?phpif(isset($_GET['page'])
){include $_GET['page'].'.php'
;}else{include 'accueil.php'
;}
?>
```

Maintenant il suffit que le pirate place une page intitulée par exemple **page** sur son serveur. Il se rend ensuite sur le site concerné par la faille, et rentre l'url suivante : **monsite.com/index.php?page=http://autre_site.com/page**. La page n'est pas exécutée par PHP sur le site distant, elle sera donc exécutée sur le site possédant la faille. Vous pouvez donc vous imaginer des conséquences que cela peut engendrer... Pour contrer cette faille nous devons simplement utiliser la fonction **file_exists()** (attention en PHP5 ceci n'est plus forcément valable, assurez-vous d'avoir la directive **allow_url_fopen** à **OFF** dans votre fichier **php.ini**. Voici le code corrigé :

```
<?phpif(isset($_GET['page']) AND file_exists($_GET['page'].'.php')
){include $_GET['page'].'.php'
;}else{include 'accueil.php'
;}
?>
```

Attention maintenant à la faille dans votre site, c'est à dire qu'un visiteur inclue une page de votre site pour laquelle il n'a pas les droits d'accès en remontant d'un répertoire par exemple... Vous pouvez utiliser pour contrer cela les Expressions régulières en enlevant les caractères spéciaux : le slash, le point et l'antislash... qui pourraient prendre place dans l'url.

La faille du code HTML et des formulaires :

Il est courant maintenant de demander aux internautes tout ce qui vous passe par la tête, et d'afficher ensuite ces informations. Un internaute mal intentionné pourrait alors y mettre du code HTML pour défacer votre site (rien à craindre pour les donnés, juste pour la présentation).

Voici un exemple de formulaire courant :

```
&amp;quot;page.php&amp;quot;
 method=&amp;quot;post&amp;quot;
>
&amp;quot;message&amp;quot;
>
 type=&amp;quot;submit&amp;quot;
 value=&amp;quot;Envoyer&amp;quot; />
```

Voici maintenant un exemple de code PHP qui récupère la valeur du message tapé et l'affiche sur la page Web :

```
<?phpif(isset($_POST['message'])
){echo $_POST['message']
; //On aurait très bien pu stocker le message, cela serait revenu au même
}
?>
```

Maintenant imaginions que l'internaute rentre du code html quelconque, il fait ce qu'il veut avec la présentation de votre site, et peut même utiliser du javascript pour rediriger l'internaute n'importe où. Nous allons utiliser la fonction **htmlentities()** pour corriger la faille :

```
<?phpif(isset($_POST['message'])
){echo htmlentities($_POST['message'])
; //Les caractères HTML sont désormais affichés et non interprétés
}
?>
```

Les failles de type **SQL injection** :

Les failles de type SQL injection consistent à exécuter des requêtes arbitraires sur une base de données. En général ces failles sont utilisées grâce à un formulaire. Imaginions que nous ayons une table **membres** et que notre requête pour savoir si un membre est bien identifié est la suivante :

```
mysql_query(&amp;quot;SELECT login,passe FROM membres WHERE login='&amp;quot;.$_POST['login'].&amp;quot;' AND
passe='&amp;quot;.$_POST['passe'].&amp;quot;'&amp;quot;
);
```

login et passe sont deux champs d'un formulaire. Imaginions maintenant que nous rentriions cette valeur dans le champ login : **' OR 1=1"); #**. La requête exécutée par PHP devient la suivante :

```
mysql_query(&amp;quot;SELECT login,passe FROM membres WHERE login='' OR 1=1&amp;quot;
);
```

Comme la condition **OR 1=1** est toujours vraie, la requête est exécutée et le membre peut se connecter sans aucun pseudo.

Pour remédier à cette faille, nous allons échapper les caractères spéciaux en utilisant la fonction mysql_real_escape_string (cela est aussi valable pour une insertion dans une base de données) :

Les magic_quotes si elles sont activées vont automatiquement échapper les caractères, voilà pourquoi on doit toujours regarder si cette fonction est active avant d'échapper quoi que ce soit. Voici le code corrigé :

```
<?php
/*** Si des slashes ont été ajoutés via les magic quotes, on les enlève*/
if(
get_magic_quotes_gpc
()===
1
){
$_POST
[
'login'
] =
stripslashes
(
$_POST
[
'login'
]);
$_POST
[
'passe'
] =
stripslashes
(
$_POST
[
'passe'
]);}
/*** On protège les chaînes*/
$_POST
[
'login'
] =
mysql_real_escape_string
(
$_POST
[
'login'
]);
$_POST
[
'passe'
] =
mysql_real_escape_string
(
$_POST
[
'passe'
```

```
]);
/*** On effectue la requête*/
mysql_query
(
"SELECT login,passe FROM membres WHERE login='"
.
$_POST
[
'login'
].
"' AND passe='"
.
$_POST
[
'passe'
].
"'"
);
?>
```

Les failles utilisant l'**upload** :

Ce genre de faille est de plus en plus courant. Pour y remédier, vous pouvez consulter cet article : Upload sécurisé en PHP.