



# Checklist sécurité pour les applications web en php

## dossier

Pour plus de sécurité, adoptez les réflexes CASES !

Petite check-list contenant les vulnérabilités les plus communes des applications web et les solutions et bonnes pratiques applicables en PHP.

### Configurations obsolètes dans le php.ini

Configurations dangereuses ou obsolètes, qui existent encore dans php5 pour des raisons de compatibilité, mais qui sont désactivées par défaut.

- register\_globals
- magic\_quotes\_gpc
- safe\_mode

### Validation des entrées

Conseils:

- pendant le développement la configuration "error\_reporting=E\_ALL" (sans enlever les "E\_NOTICE") permet de reconnaître les variables non initialisées;
- l'utilisation de l'égalité de contenu et de type « === » permet d'éviter des erreurs lors de la comparaison avec des booléens. L'expression « (0 == false) » est vraie alors que « (0 === false) » est fausse ;
- le casting peut être plus rapide que le test (comparer la fonction is\_numeric à une conversion de type (int) \$valeur) ;
- méfiez-vous de la variable super globale \$\_REQUEST, car elle peut contenir des valeurs de plusieurs sources différentes ;
- vérifiez l'existence de différentes entrées à l'aide de « isset » ;
- vu que l'on ne peut pas faire confiance aux navigateurs passez \$\_FILES['file']['name'] par la fonction « basename » lors d'un upload de fichiers ; (imaginez \$\_FILES['file']['name']='../../etc/passwd')
- vérifiez le contenu des fichiers uploadés. Les images devraient être vérifiées avec la fonction getimagesize(), qui renvoie « faux » quand le fichier n'est pas de ce type. Pensez aussi à l'extension fileinfo ;
- ne pas accepter des objets sérialisés en entrée.

En général il est conseillé d'utiliser le concept de liste blanche pour de nombreux cas de validation.

Pour avoir une couche de protection supplémentaire ou pour protéger une application déjà existante pensez aux ajouts « PHP Input Filter » et « PHPIDS » ou « mod\_security » si vous utilisez Apache

### Injection SQL

Utiliser des requêtes SQL brutes en concaténant les variables à la chaîne de la requête est une mauvaise pratique qui peut facilement mener à l'injection de SQL non désiré.

Conseils :

- Utiliser des requêtes paramétrées (prepared statements). En PHP la manière la plus facile de traiter l'échappement des variables est d'utiliser une couche d'abstraction comme PDO.
- Les requêtes paramétrées sont accessibles aussi pour certaines bases de données par des fonctions de base de PHP (pgsql par exemple, voir pg\_prepare).
- Si la concaténation s'avère impossible à éviter il est impératif d'échapper les variables à l'aide de fonctions du type « (real\_)escape\_string ». Pour mysql, par exemple, renseignez-vous sur la fonction « mysql\_real\_escape\_string ».
- Évitez à tout prix les techniques automatiques du type « magic\_quotes » et/ou génériques « add\_slashes ». Chaque base de données réagira de manière différente à l'échappement.

## Cross-site scripting (XSS)

Consiste en l'introduction de javascript en entrée qui sera alors exécuté lors de l'affichage. Il s'agit d'une vulnérabilité très commune et dangereuse, bien qu'elle soit facile à éviter dans beaucoup de cas.

- transformer tous les caractères des variables affichées en entités html. En PHP il suffit donc souvent de passer toutes les variables affichées par la fonction « html\_entities ». Certains « templating systems » comme « flexy » font un échappement automatique;
- s'il faut passer du HTML dans les variables, il est possible d'utiliser la fonction strip\_tags pour filtrer les tags permis. Cette fonction est néanmoins dangereuse à cause de l'injection de javascript dans les arguments permis; exemple:  ou <a href="javascript:alert('XSS');">lien</a>. Il faut donc en général créer sa propre fonction de filtre en addition à strip\_tags ;
- une bonne manière de filtrer le html est la classe « html purifier » (<http://htmlpurifier.org/>, licence LGPL).

En général, il faut écrire une fonction générique d'affichage, qui devra être impérativement utilisée lors de tout affichage.

## Injection de code

Injection de code malicieux qui sera exécuté par l'application.

- l'utilisation de variables dans les instructions « include » ou « require » est à proscrire. S'il est impossible de l'éviter, il est important d'utiliser des filtres par liste blanche ;
- éviter à tout prix ou traiter avec grand respect l'instruction « eval » ;
- dans php.ini mettre « allow\_url\_fopen » à « off » si la fonction n'est pas nécessaire.
- éviter les variables dans la partie « remplacement » des fonctions du type « preg\_replace ».

## Injection d'instructions

Certaines fonctions permettent d'exécuter des instructions système.

- évitez les variables dans les instructions du type shell\_exec, exec, system, passthru, popen;
- s'il est impossible de les éviter, utilisez impérativement les fonctions escapeshellarg() et escapeshellcmd();
- utilisez des listes blanches ainsi que la fonction « basename » sur les noms de fichiers.

## Sécurité de la session

En volant l'identifiant de la session, il est possible d'utiliser la session d'autres utilisateurs. Voir « firesheep » par exemple.

- évitez l'identifiant de session dans l'URL car un autre site pourrait voler l'identifiant en vérifiant le « referer », voire fixer l'identifiant de votre session (imaginez un lien vers un site du type `lien?PHPSESSID=123`). Mettre donc `use_only_cookies` à « 1 » ou « On » dans `php.ini` ;
- utiliser uniquement des communications SSL (TLS), sinon le cookie de session peut être intercepté sur des réseaux hostiles. Obliger le cookie de session à être communiqué uniquement en SSL : `session.cookie_secure=1` ;
- définir un moment d'expiration pour les sessions.

## Cross-Site Request Forgery (XSRF)

Consiste en l'utilisation de la session ouverte dans un autre tab du navigateur par une page web malicieuse. Le plus souvent les parties vulnérables seront les formulaires, qui pourraient être remplis par un site malveillant. Dans ce cas il est facile d'éviter ce genre d'attaque en créant un identifiant unique pour le formulaire et contrôlant cet identifiant lors de la soumission. Une bonne pratique consiste à utiliser un nonce créé par une fonction du type « `uniqid` » ou pseudoaléatoire haché avec un code secret en utilisant la fonction « `hash_hmac` », que l'on met dans un champ « `hidden` » du formulaire ainsi qu'en session, et qui sera vérifié lors de la soumission.

Pour protéger les fonctionnalités en dehors des formulaires, il faudra faire une analyse au cas par cas.

## Bonnes pratiques pour le stockage des mots de passe utilisateurs dans la base de données

Les mots de passe doivent être stockés de manière hachée dans la base de données. Pour cela il ne faut surtout pas utiliser des algorithmes du type « `md5` » ou « `sha1` » pour lesquels il existe des « `rainbow tables` », qui permettent l'obtention rapide du mot de passe à partir du hash.

De plus, pour justement éviter l'utilisation de « `rainbow tables` », et pour masquer les hashes qui seraient identiques car issus du même mot de passe, il est important d'utiliser un « `salt` ». Ce « `sel` » sera déterminé de manière aléatoire et concaténé au mot de passe avant hachage. On stockera ensuite le sel ainsi que le hash dans la base de données.

Il est important aussi de rajouter un « `salt` » constant secret qui serait stocké ailleurs que dans la base, par exemple dans un fichier de configuration.

L'utilisation d'un « `salt` » concaténé peut avantageusement être remplacée par une fonction de hachage « `hash_hmac` », mais nous recommandons l'utilisation de la fonction « `crypt` » qui permet d'ajouter un facteur de coût au calcul quand on utilise « `CRYPT_BLOWFISH` », « `CRYPT_SHA256` » ou « `CRYPT_SHA512` » (voir « `key stretching` »).

Au final le calcul de hash aura la forme suivante :

```
$hash=crypt(crypt('mot de passe secret', '$2a$'. $scout.'$sel.secret.config.1234$'), '$2a$'. $scout.'$sel.aleatoire.DB.45678$');
```

La variable `$scout` devra être ajustée à un nombre tel que la durée de l'opération soit viable pour votre application mais prohibitive pour une attaque brute-force (1/100 secondes par exemple).

## Utilisation de contremesures

En général une application web sera scannée à l'aide d'un outil automatique par un attaquant avant compromission. De ce fait, il est possible de parsemer son application de pièges à ours où autres « `tar`

pits ». Un exemple pratique appelé « weblabyrinth » peut être téléchargé sur <http://www.mayhemlabs.com/content/new-tool-weblabyrinth>. Mais des scripts inutiles contenant des fonctions « sleep » ou autres fausses authentifications peuvent déjà faire l'affaire.

## Webroot != Approot

Avoir ses bibliothèques et autres scripts dans des endroits inaccessibles directement depuis Internet est d'un avantage certain. En cas de vulnérabilités de certaines bibliothèques elles ne peuvent pas être exploitées directement.

## Déni de service

Il est impossible de se défendre de manière absolue contre les attaques de type « déni de service ». Il est néanmoins possible d'essayer d'optimiser son application pour qu'elle ait un meilleur répondant. Bien que cela dépasse le cadre de ce document voici quelques concepts que vous pourrez approfondir :

- n'utiliser « foreach » que pour les tableaux associatifs et évitez en général toute itération non bornée sur les données venant des bases de données ;
- utilisez un moteur de cache de code compilé (memcache, eaccelerator, Zend Platform,...) ;
- certains outils de débogage peuvent vous indiquer les goulots d'étranglements (xdebug, yslow, firebug, ...)
- compressez votre javascript ;
- contrôlez quelles sont les requêtes lentes (mysql\_slow\_query) ;
- éventuellement comprimer la communication entre le serveur et le client (uniquement si le poids des pages est conséquent et le CPU du serveur assez puissant) ;
- si vous utilisez Apache pensez au module mod\_evasion.

## Frameworks

Beaucoup de frameworks sont écrits par des développeurs chevronnés et profitent de leur expérience étendue, non seulement en termes de sécurité, mais aussi de bonnes pratiques de programmation. Ils incluent donc souvent beaucoup des techniques exposées ci-dessus et permettent le développement rapide et plus sûr d'applications web. Exemples : Zend, Symfony,...

## Conseils pour php.ini

Les paramètres suivants dans php.ini doivent être ajustés aux besoins de votre application tout en essayant de les garder le plus bas possible. En général il est avantageux de régler ces paramètres à la volée avec « ini\_set » dans les scripts qui auraient besoin de plus de ressources que le gros de l'application :

- max\_execution\_time
- memory\_limit
- post\_max\_size
- upload\_max\_filesize

Vérifier cette valeur en environnement de production :

- `display_errors = Off`

## Conclusion

Ce guide très court n'est qu'un survol des problèmes potentiels de sécurité les plus communs et n'est évidemment pas exhaustif. Vous pouvez contacter CASES sur [www.cases.lu](http://www.cases.lu) pour des commentaires ou questions.

Retrouvez les dossiers, fiches thématiques alertes et actualités sur:

[www.cases.lu](http://www.cases.lu)