# Sécurité d'un site PHP

*Lundi 7 juin 2010*

*Jean-Ferdinand Susini*

Cnam
CONSERVATOIRE NATIONAL
DES ARTS ET METIERS

# Les impératifs de sécurité

- Un programme php  est une application distribuée qui pose très tôt le problème de la sécurité. On peut la définir par trois propriétés non fonctionnelles :
  - L'application manipule-t-elle des données fiables (intégrité)
  - L'application interagit-elle avec les bons interlocuteurs (authentification)
  - Le secret des données échangées est-il préservé (confidentialité)

# Les impératifs de sécurité

- Un programme php  est une application distribuée qui pose très tôt le problème de la sécurité. On peut la définir par trois propriétés non fonctionnelles :
  - L'application manipule-t-elle des données fiables (intégrité)
    ➡ des données échangées, des programmes exécutés
  - L'application interagit-elle avec les bons interlocuteurs (authentification)
  - Le secret des données échangées est-il préservé (confidentialité)

# Les impératifs de sécurité

- Un programme php  est une application distribuée qui pose très tôt le problème de la sécurité. On peut la définir par trois propriétés non fonctionnelles :
  - L'application manipule-t-elle des données fiables (intégrité)
    ➡ des données échangées, des programmes exécutés
  - L'application interagit-elle avec les bons interlocuteurs (authentification)
    ➡ de qui par rapport à qui ? (pb du rejeu)
  - Le secret des données échangées est-il préservé (confidentialité)

# Les impératifs de sécurité

- Un programme php  est une application distribuée qui pose très tôt le problème de la sécurité. On peut la définir par trois propriétés non fonctionnelles :
  - L'application manipule-t-elle des données fiables (intégrité)
    ➡ des données échangées, des programmes exécutés
  - L'application interagit-elle avec les bons interlocuteurs (authentification)
    ➡ de qui par rapport à qui ? (pb du rejeu)
  - Le secret des données échangées est-il préservé (confidentialité)
    ➡ par rapport à qui ?

# Les impératifs de sécurité

- Un équilibre à trouver :
  - entre le “coût” du développement de programmes ayant des propriétés de sécurité satisfaisantes et la valeur intrinsèque des données ainsi que la maintenance de l’application ou des serveurs
  - entre la “souplesse” d’utilisation et les contraintes induites sur l’utilisation de l’application des mesures de sécurité
- Intégrer le plus tôt possible ces contraintes dans la conception de l’application pour en appréhender les “coûts”

# Les attaques

- Considérer que l'application distribuée s'exécute dans un monde hostile :
  - Empêcher les déroulement correcte de l'application par brouillage des données
  - Désorganiser les échanges et le fonctionnement de l'application
  - Usurper l'identité d'un interlocuteur pour utiliser l'application pour soi et/ou récupérer des données confidentielles
  - Prendre le contrôle partiel ou complet du système plus ou moins discrètement

# Mesures de sécurité

- Angle d'attaque du problème de la sécurité :
  - Fiabiliser les données et le logiciel
  - Fiabiliser les échanges et en particulier le traitement des formulaires
  - Sécuriser l'accès au bases de données
  - Gestion sécurisée des sessions
  - Gestion des serveurs partagés

# Register Globals

- Cette directive permet d'appliquer la valeur d'un paramètre de requête directement dans une variable globale de même nom. Cela évite le recourt aux super globales $_GET ou $_POST
- L'emploie de cette directive est largement déconseillé. Bien que très pratique elle présente un haut risque d'interférence aisément exploitable par un attaquant
- Cette faille met en évidence la nécessité pour le programmeur de toujours explicitement initialiser toutes ses variables avant de les utiliser, même si une initialisation est supposée

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables

➡ Concentré dans un point de passage unique maîtrisé

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables

➡ Concentré dans un point de passage unique maîtrisé

  - un point d'entée unique

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables

➡ Concentré dans un point de passage unique maîtrisé

  - un point d'entée unique
  - un module unique appliqué systématiquement à tous les points d'entré

# Vérifier et filtrer les données

- Vérification du type des données par des programmes génériques extrayant les données des variables super globales
- Initialisation des variables au démarrage du script
- Configuration soignée de la gestion des erreurs (php.ini) :
  - error_reporting : E_ALL pour détecter les variables non initialisées
  - display_errors : a utiliser systématiquement en phase de développement et masquer en production
  - log_errors/error_log: pour avoir des retours en production

# Vulnérabilité XSS

- Les “Cross-site scripting” exploitent des vulnérabilité mettant en œuvre des informations partagées par des sites différents : des sites affichant des données externes
- Si les données affichées ne sont pas correctement filtrées elle peuvent contenir des codes malicieux qui s’exécuteront sur le client
- Contre-mesures : utilisation des fonctions de filtrage (ne pas réinventer la roue) : htmlentities, htmlspecialchars,…
- objectif : filtrer les donner pour empêcher l’interprétation de contenu html ou javascript.

# Construire de fausses requêtes

- Les dangers de l'utilisation de la méthode GET :
- Le requête engendré par la balise :
- `<img src="http://monsitephp.fr/action.php?task=delete&num=100" />`
- ne peut être distingué du côté du serveur de la requête engendrée par l'utilisation du formulaire :
- 
```
<form action="/action.php">
<select name="task">
  <option value="delete">delete</option>
  <option value="view">view</option></select>
<input type="text" name="num"/>
</form>
```

# Vulnérabilités CSRF

- Consiste à forger de fausses requêtes à partir d'URL authentifiées et de pousser le client à exécuter des actions sans le savoir.
- Pour fonctionner, il faut que l'authentification ait été réalisée et que l'action du formulaire ne demande pas d'authentification ou de confirmation
- On forge alors la requête en dissimulant un formulaire ou en utilisant une balise image contrôlé par exemple par du JavaScript

# un formulaire, un traitement

- Il faut s'assurer qu'une demande de traitement est bien issue d'un formulaire géré par l'application
- Utilisation de mécanisme cryptographique pour "identifier les formulaires utilisés avec un numéro unique
  - `$token = uniqid(rand(),true);`
- et ajouter un champ invisible au formulaire :
  - `<input type="hidden" name="token" value="<?php echo $token; ?>" />`
- ne pas oublier de stocker (dans une session, dans une base de données ou dans un fichier le tocken)

# Injection SQL

- Lorsque l'on construit une requête SQL directement à partir des champs d'une requête POST ou GET, on fournit à un attaquant la possibilité de pratiquer de l'injection de code malicieux
- 
```
<?php
$sql = "INSERT INTO users
(reg_username,reg_password,reg_email)
 VALUES ('{$_POST['reg_username']}',
'$reg_password', '{$_POST['reg_email']}')";
?>
```

- par exemple en entrant dans le champ utilisateur :
- `bad_guy', 'mypass', ''), ('good_guy`
- utilisation de filtres : mysql_real_escape_string, ...

# Le vol de session

- start_session démarre une session mais c'est à votre code PHP de vérifier si la session existe déjà ou pas. Un attaquant malveillant peut fixer un identifiant de session si vous ne le vérifiez pas
- Rajout d'information qui assurent que l'on reçoit bien des requêtes de la même session
- Utilisation d'identificateurs non prédictibles
- Utilisation de l'interactions utilisateur pour éveiller l'attention sur quelque chose de suspect