

---

# Chapitre 12

## Sécurité

# La sécurité de PHP

---

- PHP : langage généraliste sur le serveur
  - Accès potentiellement à toutes des données du serveur
  - Ex: le fichier des mots de passe, des utilisateurs
  - Les bases de données avec des informations privées
  - Tous les pouvoir du shell (exec)
- Cas de trous de sécurité dans la presse
  - Perte pour les entreprises
    - de confiance
    - de finances (arrêt du serveur)
- PHP
  - Niveau de sécurité plus élevé que C car pas accès aux adresses mémoire
  - Dépend de l'utilisation qui en est faite
  - Beaucoup d'options pour limiter et contrôler
  - Connaitre les options qui fragilisent

# La sécurité en PHP

---

- PHP est fait pour le WEB
  - accessible à des centaines de million de personnes
  - accessible par des programmes
- Système 100% sûr : une illusion
  - Maitriser les risques potentiels
  - Equilibrer les risques et l'utilisabilité
- Sécurité = discrétion (ex: faille zéro Day)
  - Trou de sécurité : souvent un problème plus humain que technologique
- Sécurité : pas complexe, car la complexité est le risque de ne pas maitriser tout une chaine de traitement
  - "un système est aussi sûr que son maillon le plus faible"
- Contrôler toutes les entrées utilisateurs
  - Penser que toute entrée est possible, même (et surtout) par erreur

# Sécurité : les droits

---

- Dépend de l'installation
  - Utilisation en tant que CGI
  - Utilisation en tant que module du serveur (ex: apache)
- Faire attention aux droits
  - CGI : potentiellement tous les droits
  - Serveur : droit de ceux du serveur (groupe et user id)
  - Limiter les droits au strict nécessaire
  - Ne pas autoriser l'exécution en dehors de l'arborescence serveur
  - Ne pas autoriser les liens symboliques
  - ...

# Sécurité : configuration de PHP

---

- Comprendre la différence URL et chemin sur le serveur
- Vérifier le fichier php.ini
- Exemple :
  - open\_basedir : limite l'accès aux fichiers ayant un chemin spécifique sur le serveur
- La fonction htmlspecialchars() :
  - Permet d'inclure des données de l'extérieur sans perturber le HTML
  - Evite l'inclusion et ensuite l'exécution de JavaScript (XSS)
- XSS : cross-site scripting
  - type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page

# Exemple de trous de sécurité : XSS

---

```
<?php
$nom= $_GET['nom'];
?>
...
<h1>Hello</h1>
    <p>
        Bonjour <?=$nom?> !
    </p>
```

Si \$nom contient du Javascript il est exécuté !

Exemple : URL

xss.php?nom=<script>alert("SPAM");<script>

Le navigateur peut détecter et refuser d'exécuter.

Solution : **\$nom = htmlspecialchars(\$\_GET['nom']);**

# Exemple de trous de sécurité : les chemins

---

```
$username = $_POST['username'];  
$userfile = $_POST['userfile'];  
$homedir  = "/home/$username";  
  
unlink("$homedir/$userfile");  
  
echo "Ce fichier a été effacé !";
```

- Vérifier les données de l'utilisateur
  - Possible d'envoyer des chemins relatifs
  - Ex: username=../etc&userfile=passwd  
Resultat : **unlink("../home/../etc/passwd") !!!**
- Limiter les accès : ex. liste exhaustive de ce qui est possible
- Vérifier les chemins finaux calculés

# Trou de sécurité : l'octet nul en fin de chaîne

---

```
$file = $_GET['file']; // "../..etc/passwd\0"

if (file_exists('/home/wwwrun/'.$file.'.php')) {

    include '/home/wwwrun/'.$file.'.php';
    // le fichier /etc/passwd sera inclus !!
}
```

- Les routines de PHP sont écrites en C
- Appel du système d'exploitation écrit en C ou C++
- => convention des octets nuls en fin de chaîne
- file\_exists retournera true sachant que le fichier /home/wwwrun/../../etc/passwd existe
- Solution : vérifier le fichier obtenu, limiter les choix à une liste



# La sécurité des variables globales

---

```
if ($motdepasse == "monMotDePasse") {  
    $autorise = 1;  
}  
  
if ($autorise == 1) {  
    // Accès à des choses importantes  
}
```

- php.ini : register\_globals , **toujours à OFF**
  - Définit si oui ou non les variables EGPCS (Environment, GET, POST, Cookie, Server) seront enregistrées comme des variables globales.
- Risque important :
  - variable \$autorise non initialisée
  - envoyer dans l'URL ? autorise =1

# Sécurité des sessions

---

```
session_start();

if (!isset($_SESSION['initiated']))
{
    session_regenerate_id();
    $_SESSION['initiated'] = true;
}
```

- Cookies et sessions
  - prédiction du cookie
  - capture du cookie
  - fixation du cookie : générer un autre cookie à la place du système

# Sécurité des données de la base

---

- Ne pas se faire "voler" ses données
- Ne pas autoriser des requêtes SQL dynamiques
  - risque d'injection SQL
  - analyser finement les entrées
  - **que des requêtes préparées**
- Se rappeler que TOUT passe en clair sur le réseau, même les mots de passe saisis dans un formulaire
  - solution : le cryptage, comme tout simplement https
  - https : le choix de Google !

# Sécurité : ne pas s'exposer (aux problèmes)

---

- Penser qu'on est sur le WEB :
  - accessible à des millions de personnes
  - accessible à des robots
  - toute donnée en entrée possible peut arriver un jour
- Maîtriser les droits du système d'exploitation
- Maîtriser les paramètres du serveur WEB
  - Ex: Apache etc/php.d/security.ini, placer expose\_php=Off, qui n'informe plus sur la version PHP utilisée ...
  - Ne pas afficher en clair les erreurs dans le serveur en exploitation  
display\_errors=Off
  - Interdire les ouvertures de fichiers hors du site  
allow\_url\_fopen=Off  
allow\_url\_include=Off
  - Interdire certaines fonction dangereuses  
disable\_functions =exec,passthru,shell\_exec,system,proc\_open

# Failles de sécurité : a savoir

---

- Sécurité : un problème **humain** et pas seulement technique
- SQL injection : pas de requête dynamiques
- XSS (Cross-site scripting) : vérification de toutes données de l'extérieur (ex: htmlspecialchars())
- Téléchargement de fichiers (upload) : limiter type, taille, droits, ...
- Pas de production de code dynamique
  - ne jamais utiliser la fonction **eval()**
- **Include** :
  - inclusion dynamique de code : **énorme** faille de sécurité
  - restreindre fortement les fichiers à inclure
  - pas d'inclusion hors du site !
- Limiter la **quantité** de code, augmenter sa **qualité** (clarté)
  - les failles et les erreurs sont proportionnel à la taille du code
  - réutiliser du code, limiter les bibliothèques,
- Se tenir informer : <http://www.cyberciti.biz/tips/php-security-best-practices-tutorial.html>