Sécurisation des Applications PHP

Introduction

PHP est l’un des langages de développement web les plus utilisés pour créer des sites dynamiques. Cependant, sa popularité en a aussi fait une cible privilégiée pour les attaques. Une application PHP mal sécurisée peut exposer des données sensibles, permettre l’exécution de code malveillant, ou encore compromettre un serveur web. Ce cours explore les meilleures pratiques et techniques pour sécuriser vos applications PHP contre les attaques les plus courantes.

---

1. Principales menaces pour les applications PHP

1.1 Injection SQL

C’est l'une des attaques les plus répandues. Elle permet à un utilisateur malveillant d'injecter du code SQL via les champs de formulaire.

1.2 Cross-Site Scripting (XSS)

Le XSS consiste à injecter du code JavaScript malveillant dans des pages web vues par d'autres utilisateurs.

1.3 Cross-Site Request Forgery (CSRF)

Le CSRF force un utilisateur connecté à exécuter une action non désirée sur une application dans laquelle il est authentifié.

1.4 Inclusion de fichiers

Les failles d’inclusion permettent à un attaquant d'inclure des fichiers externes dans le script PHP.

---

2. Protection contre l’injection SQL

2.1 Ne jamais faire confiance aux données utilisateur

❌ Mauvais exemple :

```
$sql = "SELECT * FROM users WHERE email = '".$_POST['email']."'";
```

✅ **Bon exemple avec requêtes préparées :**

```
$stmt = $conn->prepare("SELECT * FROM users WHERE email = ?");
$stmt->bind_param("s", $_POST['email']);
$stmt->execute();
```

Les **requêtes préparées** sont la méthode la plus fiable contre les injections SQL.

---

3. Prévention du Cross-Site Scripting (XSS)

3.1 Échapper les sorties HTML

Utilisez `htmlspecialchars()` pour désactiver l'interprétation du HTML injecté :

```
echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
```

3.2 Validation côté client ET serveur

Ne vous fiez pas uniquement au JavaScript pour filtrer les entrées.

---

4. Protection contre le Cross-Site Request Forgery (CSRF)

4.1 Utiliser des tokens CSRF

Ajoutez un **jeton unique** dans chaque formulaire :

```
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

Dans le formulaire HTML :

```
<input type="hidden" name="csrf_token" value="<?php echo
$_SESSION['csrf_token']; ?>">
```

Et côté serveur :

```
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("Requête non autorisée !");
}
```

---

5. Sécurisation des fichiers et des répertoires

- Évitez de stocker des fichiers sensibles dans le dossier `www`.
- Protégez vos fichiers de configuration (`config.php`) en les plaçant hors de la racine web.
- Désactivez l'exécution de scripts PHP dans les dossiers de téléchargement.

---

6. Gestion sécurisée des sessions

- Utilisez `session_start()` en haut de chaque page protégée.
- Activez les cookies sécurisés :

```
session_set_cookie_params([
    'secure' => true,
    'httponly' => true,
    'samesite' => 'Strict'
]);
```

- Changez l'ID de session après une connexion :

```
session_regenerate_id(true);
```

---

7. Validation et filtrage des données utilisateur

- Utilisez `filter_var()` pour valider les e-mails, les URLs, etc.
- Validez toujours côté serveur, même si JavaScript fait un pré-filtrage.

```
if (!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
    die("Adresse email invalide.");
}
```

---

8. Erreurs et messages d'exception

- **Ne jamais afficher les erreurs PHP en production !**
- Utilisez un système de logs sécurisé :

```
error_reporting(0);
ini_set('display_errors', 0);
ini_set('log_errors', 1);
```

---

9. Utilisation de HTTPS

- **Chiffrez toutes les communications** entre le client et le serveur avec un certificat SSL/TLS valide.

- Redirigez automatiquement vers HTTPS :

```
if (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS'] !== 'on') {
    header('Location: https://' . $_SERVER['HTTP_HOST'] .
$_SERVER['REQUEST_URI']);
    exit;
}
```

---

10. Bonnes pratiques générales

- Gardez PHP et vos bibliothèques à jour.
- Utilisez des frameworks sécurisés comme Laravel ou Symfony.
- Définissez les permissions de fichiers avec prudence (`chmod`, `chown`).
- Testez votre application avec des outils de sécurité (OWASP ZAP, Burp Suite).

Conclusion

Sécuriser une application PHP ne repose pas sur une seule technique, mais sur une combinaison de bonnes pratiques, de validation, de filtrage, et de gestion rigoureuse des entrées et sorties. En respectant les principes abordés dans ce cours, vous renforcez considérablement la résilience de vos projets web contre les attaques courantes.