Sécurité des Dossiers Web

Protégez vos fichiers et vos données sensibles

Sommaire

1 Introduction

La sécurité d'une application web ne se limite pas au code PHP ou aux requêtes SQL. Elle inclut aussi la **protection des dossiers et des fichiers** stockés sur le serveur.
Si un dossier ou un fichier sensible est accessible publiquement, un attaquant peut :

- Lire vos **fichiers de configuration** (contenant des mots de passe).
- Télécharger des **documents confidentiels** (listes de clients, factures).
- Exécuter des **scripts malveillants** téléchargés.

> 💡 **Objectif :** Apprendre à protéger les dossiers et les fichiers sensibles de votre application web.

2 🧠 Pourquoi Protéger les Dossiers ?

Les dossiers de votre application contiennent souvent des ressources critiques :

- **Fichiers de configuration** : **config.php**, **.env**, **database.ini** (contiennent des mots de passe).
- **Fichiers de données** : **.sql**, **.json**, **.csv** (contenant des données sensibles).
- **Fichiers d'upload** : **upload/**, **photos_clients/**, **cv/** (doivent être gérés avec soin).
- **Fichiers de logs** : **logs/error.log** (contiennent des traces d'erreurs ou de requêtes).

> ⚠️ **Risque :** Si ces dossiers ne sont pas protégés, les attaquants peuvent accéder à des données confidentielles ou exploiter des failles.

3 🚨 Risques Associés

Risque	Description	Exemple de Fichier/ Dossier
Vol de mots de passe	Accès à un fichier de configuration contenant des identifiants de base de données.	**config/database.php**

Risque	Description	Exemple de Fichier/ Dossier
Exposition de données sensibles	Téléchargement d'une base de données brute ou d'une liste de clients.	**data/backup.sql**, **documents/ liste_clients.csv**
Injection de code malveillant	Un attaquant télécharge un script PHP dans un dossier d'upload et l'exécute.	**upload/** (si non sécurisé)
Divulgation d'erreurs système	Accès aux logs peut révéler des détails techniques.	**logs/error.log**

4 ✅ Bonnes Pratiques de Sécurité

- **Ne placez jamais de fichiers sensibles dans le dossier racine web** (ex: `www`, `htdocs`).
- **Utilisez des dossiers en dehors de la racine web** pour les configurations ou les sauvegardes.
- **Restreignez l'accès aux dossiers critiques** via des règles de serveur ou du code PHP.
- **Validez et filtrez les fichiers téléchargés** (type MIME, extension, taille).
- **Désactivez le listing des fichiers** dans les dossiers non protégés.
- **Utilisez des noms de dossiers non évidents** pour les zones sensibles (ex: `private_data` au lieu de `backup`).

5 🔧 Protection avec Apache (.htaccess)

Sur les serveurs Apache, vous pouvez utiliser un fichier **.htaccess** pour bloquer

l'accès à un dossier ou à certains types de fichiers.

Exemple 1 : Bloquer l'accès à un dossier entier

Créez un fichier nommé **.htaccess** dans le dossier que vous voulez protéger (ex: `config/` ou `data/`).

```
 # .htaccess
# Bloquer l'accès à tous les fichiers du dossier
Order Deny,Allow
Deny from all
```

Exemple 2 : Bloquer l'accès à des types de fichiers spécifiques

```
 # .htaccess
# Bloquer l'accès aux fichiers .env et .sql

  Order Deny,Allow
  Deny from all



  Order Deny,Allow
  Deny from all
```

Exemple 3 : Désactiver le listing des fichiers

```
 # .htaccess
# Désactiver l'affichage de la liste des fichiers
Options -Indexes
```

> ✅ **Astuce :** Le fichier **.htaccess** s'applique au **dossier où il est placé** et à **tous ses sous-dossiers**.

6 Protection avec PHP

Vous pouvez aussi gérer l'accès aux fichiers via du code PHP.

Exemple 1 : Servir un fichier privé en fonction de l'utilisateur connecté

```
 < ? php
session_start();

// Vérifier si l'utilisateur est connecté et autorisé
if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'admin') {
    die("Accès refusé.");
}

// Récupérer le nom du fichier demandé
$filename = $_GET['f'] ?? '';
if (empty($filename)) {
    die("Fichier non spécifié.");
}

// Sécurité : Empêcher la traversée de dossiers (ex: ../../config.php)
$filename = basename($filename);

// Chemin vers le fichier dans un dossier privé
$filePath = __DIR__ . '/private_documents/' . $filename;

// Vérifier que le fichier existe et est dans le bon dossier
if (!file_exists($filePath) || strpos(realpath($filePath), realpath(__
    die("Fichier non trouvé ou non autorisé.");
}

// Envoyer le fichier au navigateur
header('Content-Type: application/octet-stream');
header('Content-Disposition: attachment; filename="' . basename($fileP
readfile($filePath);
exit;
? >
```

Exemple 2 : Valider un téléchargement de fichier via formulaire

```
 < ? php
if ($_POST && isset($_FILES['fichier'])) {
    $file = $_FILES['fichier'];
    $uploadDir = 'uploads/';
    $allowedTypes = ['image/jpeg', 'image/png', 'application/pdf'];
```

```
    if ($file['error'] === UPLOAD_ERR_OK) {
        $fileType = mime_content_type($file['tmp_name']);
        $fileExtension = strtolower(pathinfo($file['name'], PATHINFO_E

        if (in_array($fileType, $allowedTypes) && $fileExtension !== '
            $destination = $uploadDir . basename($file['name']);
            if (move_uploaded_file($file['tmp_name'], $destination)) {
                echo "✅ Fichier téléchargé avec succès.";
            } else {
                echo "❌ Erreur lors du déplacement du fichier.";
            }
        } else {
            echo "❌ Type de fichier non autorisé ou danger (PHP).";
        }
    } else {
        echo "❌ Erreur de téléchargement.";
    }
}
? >
```

> 💡 **Astuce :** Utilisez **basename()** pour empêcher les attaques de type "traversée de dossiers".

7 🏗️ Structure de Dossiers Recommandée

Voici une structure de projet qui sépare les fichiers sensibles du dossier accessible publiquement.

```
/mon_projet/
│
├── public/                 # Contenu accessible via le web (index.php
│   ├── index.php
│   ├── assets/
│   │   ├── css/
│   │   ├── js/
│   │   └── images/
│   └── uploads/            # Fichiers uploadés (à sécuriser)
│       ├── photos_clients/
│       └── documents/
```

```
│
├── private/                # Contenu non accessible via le web
│   ├── config/
│   │   └── db.php          # Fichier de configuration (mots de passe,
│   ├── logs/
│   │   └── error.log
│   ├── data/
│   │   └── backup.sql
│   └── includes/
│       └── functions.php
│
└── vendor/                 # Dépendances (ex: Composer)
```

Dans cette structure, seul le dossier **public/** est exposé à l'URL racine du serveur web.

8 📝 Exercices Pratiques

Exercice 1 : Créer un .htaccess de sécurité

Créez un dossier **config** dans votre projet et placez-y un fichier **.htaccess** pour empêcher l'accès à tous les fichiers du dossier.

Exercice 2 : Servir un fichier privé avec PHP

Créez un script PHP (**download.php**) qui vérifie si l'utilisateur est connecté avant de servir un fichier PDF depuis un dossier privé.

Exercice 3 : Valider un fichier uploadé

Créez un formulaire d'upload et un script PHP qui vérifie le type MIME et l'extension du fichier. Refusez les fichiers PHP ou exécutables.

Exercice 4 : Désactiver le listing des fichiers

Ajoutez la directive **Options -Indexes** dans un fichier **.htaccess** à la

racine de votre dossier d'uploads. Vérifiez que vous ne pouvez plus voir la liste des fichiers.

Exercice 5 : Protéger un dossier de sauvegarde

Créez un dossier **backup** dans votre projet et protégez-le avec un **.htaccess** qui interdit l'accès.
