

METHODES POUR FAIRE FACE AUX INJECTIONS SQL ET AUX ATTAQUES XSS

1. Introduction

Avec le développement des applications web, la sécurité informatique est devenue une préoccupation majeure. Les sites web manipulent quotidiennement des informations sensibles telles que les identifiants des utilisateurs, les mots de passe, les données personnelles et les informations financières. Une mauvaise sécurisation du code peut permettre à un attaquant d'exploiter des vulnérabilités afin de compromettre l'application.

Parmi les attaques les plus répandues figurent les **injections SQL** et les attaques **Cross-Site Scripting (XSS)**. Ces deux vulnérabilités sont classées parmi les risques majeurs des applications web et peuvent entraîner le vol de données, la modification d'informations ou encore la prise de contrôle de comptes utilisateurs.

L'objectif de ce document est de présenter ces deux types d'attaques ainsi que les principales méthodes permettant de les prévenir.

2. Les injections SQL

2.1 Définition

Une injection SQL est une attaque qui consiste à insérer des commandes SQL malveillantes dans les données saisies par un utilisateur. Lorsque ces données sont directement intégrées dans une requête SQL sans contrôle, elles peuvent modifier le comportement normal de la base de données.

2.2 Exemple de code vulnérable

```
$email = $_POST['email'];  
$password = $_POST['password'];
```

```
$sql = "SELECT * FROM utilisateurs
      WHERE email='$email'
      AND password='$password'";
```

Un attaquant pourrait saisir :

```
' OR '1'='1
```

La requête deviendrait alors toujours vraie et pourrait permettre un accès non autorisé.

2.3 Conséquences

Les injections SQL peuvent permettre de :

- consulter des informations confidentielles ;
- modifier les données enregistrées ;
- supprimer des tables entières ;
- contourner le système d'authentification ;
- créer de nouveaux comptes administrateurs ;
- compromettre totalement la base de données.

3. Prévention des injections SQL

3.1 Utiliser les requêtes préparées

La meilleure solution consiste à utiliser les requêtes préparées.

Exemple :

```
$stmt = $pdo->prepare(
"SELECT * FROM utilisateurs
WHERE email=:email");

$stmt->execute([
```

```
'email'=>$email
```

```
]);
```

Cette technique sépare la requête SQL des données fournies par l'utilisateur.

3.2 Valider les entrées

Toutes les données doivent être contrôlées avant leur utilisation.

Exemple :

```
$id = filter_input(
INPUT_GET,
'id',
FILTER_VALIDATE_INT
);
```

3.3 Filtrer les caractères dangereux

Les données doivent respecter un format attendu.

Par exemple :

- adresse e-mail valide ;
- numéro entier ;
- longueur maximale ;
- caractères autorisés.

3.4 Limiter les privilèges MySQL

Le compte utilisé par l'application ne doit disposer que des permissions réellement nécessaires.

Par exemple :

- SELECT

- INSERT
- UPDATE

Il est préférable d'éviter les privilèges :

- DROP
- ALTER
- GRANT

3.5 Masquer les erreurs SQL

Les messages d'erreur détaillés ne doivent jamais être affichés aux utilisateurs.

Incorrect :

```
echo mysqli_error($connexion);
```

Correct :

- enregistrer l'erreur dans un journal (log) ;
- afficher un message générique.

4. Les attaques XSS

4.1 Définition

Une attaque XSS (Cross-Site Scripting) consiste à injecter du code JavaScript dans une page web. Ce code sera exécuté dans le navigateur des autres utilisateurs.

4.2 Exemple

Un utilisateur malveillant saisit :

```
<script>
```

```
alert('XSS');
```

```
</script>
```

www.ista-kinshasa.com

Si cette donnée est affichée sans protection, le navigateur exécutera automatiquement le script.

4.3 Conséquences

Une attaque XSS peut permettre de :

- voler les cookies ;
- détourner une session ;
- modifier une page web ;
- afficher de faux formulaires ;
- rediriger les visiteurs ;
- récupérer des informations confidentielles.

5. Protection contre les attaques XSS

5.1 Échapper les données

Avant d'afficher les données dans une page HTML :

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

Les balises HTML sont alors affichées comme du texte.

5.2 Valider les données

Contrôler :

- la longueur ;
- le format ;
- les caractères autorisés.

5.3 Éviter le JavaScript dynamique

Ne jamais insérer directement une donnée utilisateur dans un script JavaScript.

5.4 Utiliser une politique CSP

Exemple :

```
header(
"Content-Security-Policy:
default-src 'self';
script-src 'self';"
);
```

Cette politique limite les scripts pouvant être exécutés.

5.5 Sécuriser les cookies

Configurer les cookies avec :

- HttpOnly ;
- Secure ;
- SameSite.

Ces paramètres réduisent les risques de vol de session.

6. Bonnes pratiques générales

Pour développer une application web sécurisée, il est recommandé de :

- utiliser les requêtes préparées ;
- valider toutes les données provenant des utilisateurs ;
- filtrer les fichiers téléversés ;
- limiter le nombre de tentatives de connexion ;

- mettre régulièrement à jour PHP, MySQL et les bibliothèques utilisées ;
- sauvegarder régulièrement la base de données ;
- enregistrer les événements de sécurité dans des journaux ;
- mettre en œuvre une authentification robuste ;
- utiliser HTTPS pour chiffrer les échanges ;
- réaliser des tests de sécurité avant la mise en production.

7. Comparaison entre les injections SQL et les attaques XSS

Critère	Injection SQL	XSS
Cible	Base de données	Navigateur de l'utilisateur
Objectif	Manipuler les requêtes SQL	Exécuter du JavaScript malveillant
Risque principal	Vol ou destruction des données	Vol de session et usurpation d'identité
Protection principale	Requêtes préparées	htmlspecialchars() et CSP

8. Conclusion

Les injections SQL et les attaques XSS représentent deux des vulnérabilités les plus critiques dans le développement des applications web. Leur exploitation peut entraîner des pertes importantes de données, une atteinte à la confidentialité des informations et une dégradation de la confiance des utilisateurs.

L'adoption des bonnes pratiques de développement sécurisé, telles que les requêtes préparées, la validation des données, l'encodage des contenus affichés, la mise en œuvre d'une politique de sécurité du contenu (CSP) et la gestion sécurisée des sessions, constitue une protection efficace contre ces attaques. La sécurité doit être intégrée dès les premières phases de conception d'une application et faire l'objet d'une amélioration continue tout au long de son cycle de vie.