# Sécurité des bases de données

## Introduction

Les bases de données contiennent souvent les informations les plus sensibles d'une organisation : données personnelles, mots de passe, informations financières. Une faille de sécurité peut avoir des conséquences graves :

- **Fuite de données** : vol d'informations confidentielles
- **Modification de données** : altération malveillante
- **Suppression de données** : perte irréversible
- **Non-conformité légale** : sanctions RGPD

La sécurité repose sur **trois axes** : le réseau, les accès, et l'application.

## Réduire la surface d'exposition

La **surface d'exposition** représente l'ensemble des points d'entrée potentiels pour un attaquant. L'objectif est de la réduire au minimum.

### Bind address

Par défaut, configurez le serveur pour n'écouter qu'en local :

```
# /etc/mysql/mariadb.conf.d/50-server.cnf
bind-address = 127.0.0.1
```

> N'utilisez `0.0.0.0` (toutes les interfaces) que si des clients externes doivent se connecter, et uniquement combiné avec un firewall.

### Firewall

Limitez les IP autorisées à se connecter au port de la base de données :

```
# Autoriser uniquement le serveur web (192.168.1.10) à accéder à MariaDB
sudo ufw allow from 192.168.1.10 to any port 3306
sudo ufw deny 3306
```

## Ne jamais exposer directement sur Internet

Une base de données ne devrait **jamais** être accessible directement depuis Internet. Pour un accès distant, utilisez :

- Un **tunnel SSH**
- Un **VPN**
- Un **bastion host**

```
# Exemple : tunnel SSH pour accéder à la base distante
ssh -L 3306:localhost:3306 user@serveur-distant

# Puis connexion locale
mysql -h 127.0.0.1 -u utilisateur -p
```

# Gestion des utilisateurs et privilèges

## Principe du moindre privilège

Chaque utilisateur ne doit avoir que les droits **strictement nécessaires** à sa fonction.

```
-- À ÉVITER : donner tous les droits
GRANT ALL PRIVILEGES ON *.* TO 'mon_app'@'%';

-- PRÉFÉRER : droits limités à une base et aux opérations nécessaires
GRANT SELECT, INSERT, UPDATE, DELETE ON starwars.* TO 'mon_app'@'192.168.1.10';
```

## Utilisateurs dédiés par application

Créez un utilisateur distinct pour chaque application :

```
-- Utilisateur pour l'application web (lecture/écriture)
CREATE USER 'app_web'@'192.168.1.10' IDENTIFIED BY 'MotDePasse_Complexe!2024';
GRANT SELECT, INSERT, UPDATE, DELETE ON starwars.* TO 'app_web'@'192.168.1.10';

-- Utilisateur pour les rapports (lecture seule)
```

```
CREATE USER 'app_rapports'@'192.168.1.20' IDENTIFIED BY 'Autre_MotDePasse!2024';
GRANT SELECT ON starwars.* TO 'app_rapports'@'192.168.1.20';

-- Appliquer les changements
FLUSH PRIVILEGES;
```

### Ne jamais utiliser root pour les applications

Le compte `root` ne doit servir qu'à l'administration. Une application compromise avec un accès root = base de données compromise.

# Authentification

## Mots de passe forts

Utilisez des mots de passe longs et complexes :

- Minimum 16 caractères
- Mélange de majuscules, minuscules, chiffres, caractères spéciaux
- Uniques pour chaque utilisateur

## mysql_secure_installation

Après l'installation de MariaDB, exécutez cette commande :

```
sudo mysql_secure_installation
```

Elle permet de :

- Définir un mot de passe root
- Supprimer les utilisateurs anonymes
- Désactiver la connexion root à distance
- Supprimer la base de test

> Cette commande devrait être exécutée systématiquement sur tout nouveau serveur MariaDB.

# Injection SQL

L'**injection SQL** est l'une des vulnérabilités les plus courantes et les plus dangereuses. Elle figure dans le top 10 OWASP depuis des années.

# Principe

L'attaque consiste à insérer du code SQL malveillant dans une entrée utilisateur qui sera exécutée par la base de données.

Exemple vulnérable - recherche d'un personnage par nom :

```
-- L'utilisateur entre : Vader
SELECT * FROM personnages WHERE nom = 'Vader';
-- Résultat : Dark Vador est retourné ✓

-- L'utilisateur malveillant entre : ' OR '1'='1
SELECT * FROM personnages WHERE nom = '' OR '1'='1';
-- Résultat : TOUS les personnages sont retournés ✗

-- Pire, l'utilisateur entre : '; DROP TABLE personnages; --
SELECT * FROM personnages WHERE nom = ''; DROP TABLE personnages; --';
-- Résultat : la table est supprimée ✗
```

# Protection : les requêtes préparées

La solution est d'utiliser des **requêtes préparées** (prepared statements) qui séparent le code SQL des données.

### PHP avec PDO :

```
// □ VULNÉRABLE - Ne jamais faire ça
$nom = $_GET['nom'];
$sql = "SELECT * FROM personnages WHERE nom = '$nom'";
$result = $pdo->query($sql);

// □ SÉCURISÉ - Requête préparée
$nom = $_GET['nom'];
$stmt = $pdo->prepare("SELECT * FROM personnages WHERE nom = ?");
$stmt->execute([$nom]);
$result = $stmt->fetchAll();
```

### Python avec MySQL Connector :

```
# □ VULNÉRABLE - Ne jamais faire ça
nom = input("Nom du personnage: ")
cursor.execute(f"SELECT * FROM personnages WHERE nom = '{ nom }'")

# □ SÉCURISÉ - Requête préparée
nom = input("Nom du personnage: ")
cursor.execute("SELECT * FROM personnages WHERE nom = %s", (nom,))
```

> Ne **jamais** concaténer des entrées utilisateur dans une requête SQL, même après validation. Utilisez **toujours** des requêtes préparées.

# Bonnes pratiques - Résumé

| Axe | Action |
|---|---|
| **Réseau** | Bind sur 127.0.0.1, firewall, pas d'exposition Internet directe |
| **Accès** | Moindre privilège, utilisateurs dédiés, mots de passe forts |
| **Authentification** | Exécuter `mysql_secure_installation`, désactiver root distant |
| **Application** | Requêtes préparées systématiques, jamais de concaténation |

---

Revision #1
Created 15 January 2026 19:37:00 by Thibaud FRICHET
Updated 15 January 2026 19:37:00 by Thibaud FRICHET