International Journal of Academic Multidisciplinary Research (IJAMR)
ISSN: 2643-9670
Vol. 8 Issue 5 May - 2024, Pages: 174-179

Etude sur la politique de sécurité des applications Web contre les attaques par Injection SQL

Jacques KIZA DIMANDJA

Informatique de Gestion et SÃ©curitÃ© des donnÃ©es
University: UniversitÃ© Notre Dame du Kasayi (U.KA)

Abstract : *SQL injection attacks are one of the biggest threats facing web applications. This attack method allows attackers to exploit application security vulnerabilities to inject malicious SQL statements into SQL queries. This can lead to serious consequences such as unauthorized access to sensitive data, data alteration or deletion, or even the execution of malicious commands. In order to protect the appl.*

Résumé

Les attaques par injection SQL sont l'une des principales menaces auxquelles les applications web sont confrontées. Cette méthode d'attaque permet aux attaquants d'exploiter les failles de sécurité des applications pour injecter des instructions SQL malveillantes dans les requêtes SQL. Cela peut entraîner des conséquences graves telles que l'accès non autorisé aux données sensibles, l'altération ou la suppression de données, voire l'exécution de commandes malveillantes. Afin de protéger les applications web contre ces attaques, il est essentiel de comprendre la définition de l'injection SQL et de reconnaître l'importance de la sécurité des applications web.

L'injection SQL est une technique d'attaque courante qui exploite les défauts de conception des applications web mal conçues pour exécuter du code SQL malveillant. Voici un exemple concret pour illustrer comment cela fonctionne :

1. **Contexte** :
   - Supposons que vous ayez une application web avec un formulaire de connexion. Le formulaire accepte une adresse e-mail et un mot de passe, puis les soumet à un fichier PHP nommé index.php.
   - L'instruction SQL backend pour vérifier l'ID utilisateur est la suivante :

```
SELECT * FROM utilisateurs WHERE email = $_POST['email'] AND mot_de_passe = md5($_POST['mot_de_passe']);
```

Remarque : l'instruction ci-dessus utilise directement les valeurs du tableau $_POST [] sans les nettoyer. Le mot de passe est crypté à l'aide de l'algorithme MD5.

2. **L'attaque par injection SQL** :

Un attaquant peut exploiter cette vulnérabilité en manipulant les entrées de l'utilisateur. Voici comment cela se produit :

- L'attaquant fournit une adresse e-mail valide dans le champ de connexion.
- Dans le champ du mot de passe, l'attaquant saisit : ' OR '1'='1
- L'instruction SQL devient alors :

```
SELECT * FROM utilisateurs WHERE email = 'adresse_email_valide' AND mot_de_passe = md5('' OR '1'='1') ;
```

- Comme '1'='1' est toujours vrai, l'attaquant peut se connecter sans connaître le mot de passe réel.

On compte aujourd'hui trois types d'injections SQL[1] :

1. **Error-based**.
2. **Boolean based**.
3. **Time-based**.

Voyons ce qui les différencie : [2]

[1] Attaquez la base de données avec les injections SQL - Réalisez un test d'intrusion web - OpenClassrooms, visité le 17 fevrier 2024.

[2] https://openclassrooms.com/fr/courses/7727176-realisez-un-test-dintrusion-web/7917166-attaquez-la-base-de-donnees-avec-les-injections-sql#/id/r-7917204 , , visité le 17 fevrier 2024

a) Les injections SQL de type error-based[3]

Les **SQLi error-based** sont les plus faciles à identifier et à exploiter. On les trouve généralement moins souvent dans la nature : comme elles sont facilement détectables, elles sont généralement rapidement identifiées et corrigées.

La détection est assez simple : si le code est vulnérable, lorsque vous rentrez un caractère spécial (comme une apostrophe ' ou un guillemet ") dans un champ qui va être utilisé comme paramètre dans une requête SQL, la requête va buguer.

Si les erreurs sont affichées, vous aurez un message d'erreur qui vous indique parfois même ce qui ne va pas, à l'image de la réponse de l'application de démo :

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'test" at line 1
```

b) **Les injections de type Boolean-based**

L'injection SQL de type **Boolean-based** est une technique qui repose sur l'envoi d'une requête SQL à la base de données, en fonction de laquelle la technique force l'application à renvoyer des résultats différents. Le résultat permet à un attaquant de déterminer si la charge utile utilisée renvoie **vrai** ou **faux**. Bien que les données de la base de données ne soient pas récupérées, les résultats fournissent à l'attaquant des informations précieuses. En fonction du résultat booléen (VRAI ou FAUX), le contenu de la réponse HTTP changera ou restera identique. De plus, il s'agit d'une attaque lente, ce qui permet à l'attaquant d'énumérer la base de données[4]

Voici un exemple illustrant une injection SQL de type **Boolean-based** :

Supposons qu'une couche d'accès aux données vulnérable d'une application construise une requête SQL comme celle-ci à partir d'une requête URL :

SELECT title, description, body FROM items WHERE ID = 2 and 1=2

Si l'application est vulnérable à l'injection SQL, elle ne renverra rien. L'attaquant injectera ensuite une requête avec une condition vraie (1=1). Si le contenu de la page diffère de celui renvoyé lors de la condition fausse, l'attaquant peut en déduire que l'injection SQL fonctionne. À ce stade, l'attaquant peut vérifier s'il est prêt à utiliser d'autres méthodes d'injection SQL[5]
Utilisons cette méthode qui contourne l'attaque d'Injection SQL par Boolean-based :

```php
<?php
    String query = "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, request.getParameter("customerName"));
    ResultSet results = statement.executeQuery();
    // Traitement des résultats...
    } catch (SQLException e) {
        // Gestion des erreurs...
    }

?>
```

Les **injections SQL de type Time-based** sont une catégorie de vulnérabilités qui exploitent le temps de réponse des requêtes SQL pour déterminer si une condition est vraie ou fausse. Voici comment elles fonctionnent, accompagnées d'un exemple de requête SQL.

1. **Fonctionnement des injections SQL Time-based**:[6]
   - Lorsqu'une application web effectue une requête SQL, elle attend une réponse de la base de données.
   - Si la requête est vulnérable, un attaquant peut injecter du code SQL malveillant dans les paramètres de la requête.
   - L'attaquant utilise ensuite des techniques pour retarder la réponse de la base de données[7].

[3] https://openclassrooms.com/fr/courses/7727176-realisez-un-test-dintrusion-web/7917166-attaquez-la-base-de-donnees-avec-les-injections-sql#/id/r-7917205 , visité le 17 fevrier 2024
[4] https://beaglesecurity.com/blog/vulnerability/boolean-based-blind-sql-injection.html , visité le 29 Mars 2024
[5] Boolean based Blind SQL Injection (SQLi) (beaglesecurity.com), visité le 29 Mars 2024
[6] analyse-innovation-solution.fr/publication/fr/hacking/injection-sql-sqli-dorks, visité le 29 Mars 2024.
[7] Tutoriel d'injection SQL : comment apprendre avec un exemple (guru99.com), visité le 29 Mars 2024.

- En observant le temps de réponse, l'attaquant peut déterminer si une condition est vraie ou fausse.

2. **Exemple de requête SQL Time-based**: Supposons que nous ayons une page d'authentification avec un formulaire d'identification. Voici une requête SQL pour vérifier les informations d'identification :

SELECT * FROM users WHERE username = 'utilisateur' AND password = 'motdepasse';

L'attaquant peut injecter du code malveillant dans le champ de saisie du nom d'utilisateur (par exemple, en utilisant ' OR '1'='1'; --).
La requête modifiée ressemblerait à ceci :

SELECT * FROM users WHERE username = '' OR '1'='1'; --' AND password = 'motdepasse';

Si la base de données prend plus de temps à répondre (par exemple, en exécutant une requête complexe), l'attaquant sait que la condition '1'='1' est vraie.

L'attaquant peut alors accéder au compte sans connaître le mot de passe légitime.

3. **Prévention** :

Pour éviter les injections SQL Time-based, utilisez des requêtes préparées (paramétrées) avec des placeholders pour les valeurs.

Évitez d'interpoler directement les valeurs utilisateur dans les requêtes SQL.

3. **Prévention contre l'injection SQL** :

Pour éviter l'injection SQL, suivez ces mesures :

- **Validation des entrées utilisateur**

Nettoyez et échappez correctement les valeurs des formulaires avant de les utiliser dans les requêtes SQL. C'est-à-dire écrire une méthode qui permet de vider les champs de saisie après envoi des données dans la table concernée, cela évite qu'un attaquant les utilisent pour pirater votre application Web.

- **Utilisation de requêtes préparées**

Utilisez des requêtes préparées avec des paramètres pour éviter l'injection SQL. Elles permettent de crypter les données saisies dans le formulaire afin de les envoyer en toute sécurité. A la place d'utiliser cette instruction pour se connecter à la base de données [8] :

```php
<?php
//On demarre les sessions

//On se connecte a la base de donnee
$connect = mysql_connect('localhost', 'root', '') or die ("Erreur de connexion au Serveur");
// connexion à base de données
mysql_select_db('bddarchives_facs', $connect) or die ("Erreur de connexion à la base, veuillez
    reessayer ");

?>
```

Utiliser celle-ci à la place :

```php
<?php
    // Connexion à la base de données (exemple utilisant PDO)
        $pdo = new PDO("mysql:host=localhost;dbname=bddarchives_facs", "root", "");
?>
```

Exemple d'une requête préparée pour insérer un enregistrement dans la base de données en utilisant une requête préparée :

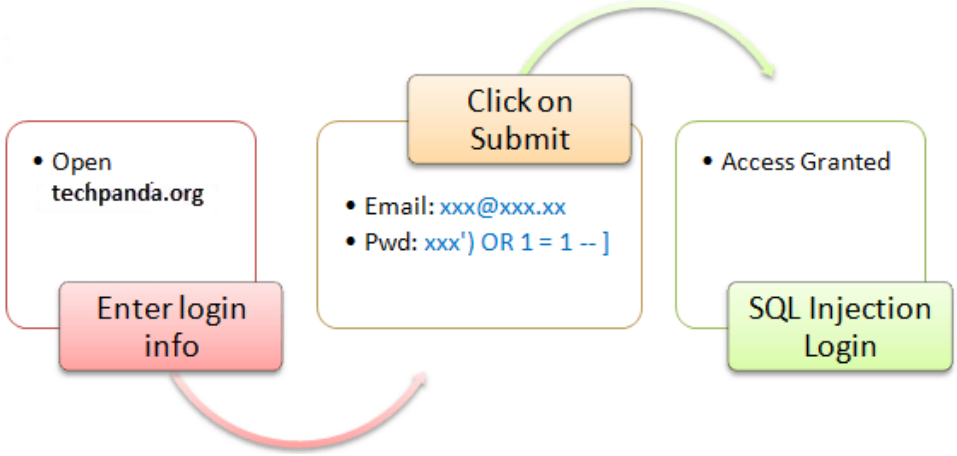
[8] Tutoriel d'injection SQL : apprendre avec un exemple (guru99.com), visité le 12 février 2024

```php
<?php
    // Requête preparée pour inserer le chemin d'accès dans la base de données
    $query = "INSERT INTO table (champ1,champ2, champ3) VALUES (:champ1, :champ2, :champ3)";
    // Préparer la requête
        $stmt = $pdo->prepare($query);

        // Lier la valeur du chemin d'accès à la requête
        $stmt->bindValue(":champ1", htmlentities($champ1));
        $stmt->bindValue(":champ2", htmlentities($champ2));
        $stmt->bindValue(":champ3", htmlentities($champ3));


    // Executer la requête preparée
        if($stmt->execute())
        {
          echo '<script> alert("Requete reussie") </script>';
        } else
        {
          echo '<script> alert("Echec de la Requete") </script>';
        }
?>
```

- **Évitez d'utiliser des fonctions d'encodage obsolètes** comme MD5 pour stocker les mots de passe. Utilisez plutôt des algorithmes de hachage plus sécurisés comme **bcrypt**.[9]

4. Activité de piratage : SQL injecte une application Web

Pour contourner la sécurité d'une application Web à partir d'un formulaire, Le schéma ci-dessous montre les étapes que vous devez suivre[10] :



Supposons qu'un attaquant fournisse la combinaison suivante :

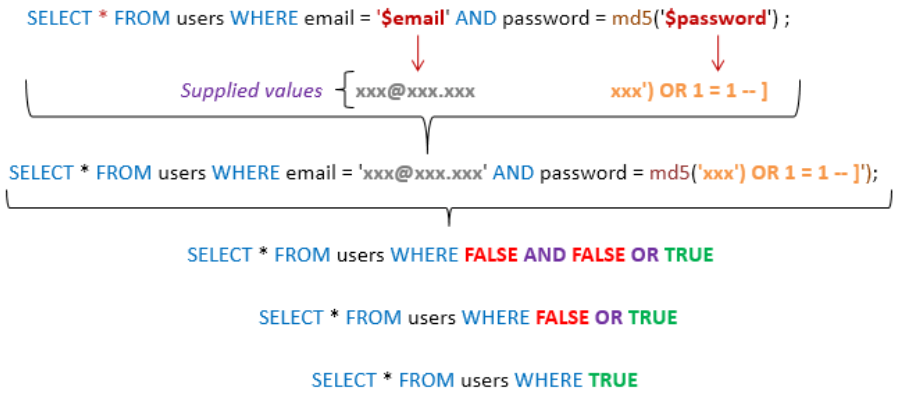
- Étape 1 : Saisissez xxx@xxx.xxx comme email propos : pour l'adresse mail, il saisit une adresse reelle ;
- Étape 2 : Il saisit ce code dans la zone de mot de passe : OU '1' = '1' --]

L'instruction SQL générée sera la suivante

SELECT * FROM utilisateurs OÙ email = 'xxx@xxx.xxx' ET mot de passe = md5('xxx') OU 1 = 1 —]');

Le diagramme ci-dessous illustre que la déclaration a été générée.

[9] Tutoriel d'injection SQL : apprendre avec un exemple (guru99.com), visité le 17 février 2024
[10] Attaquez la base de données avec les injections SQL - Réalisez un test d'intrusion web - OpenClassrooms, visité le 29 Mars 2024.



ICI,

- La déclaration suppose intelligemment que le cryptage **md5** est utilisé
- Complète le guillemet simple et la parenthèse fermante
- Ajoute une condition à l'instruction qui sera toujours vraie

En général, une attaque par injection SQL réussie utilise un certain nombre de techniques différentes.

5. Test de vulnérabilité avec le Havij

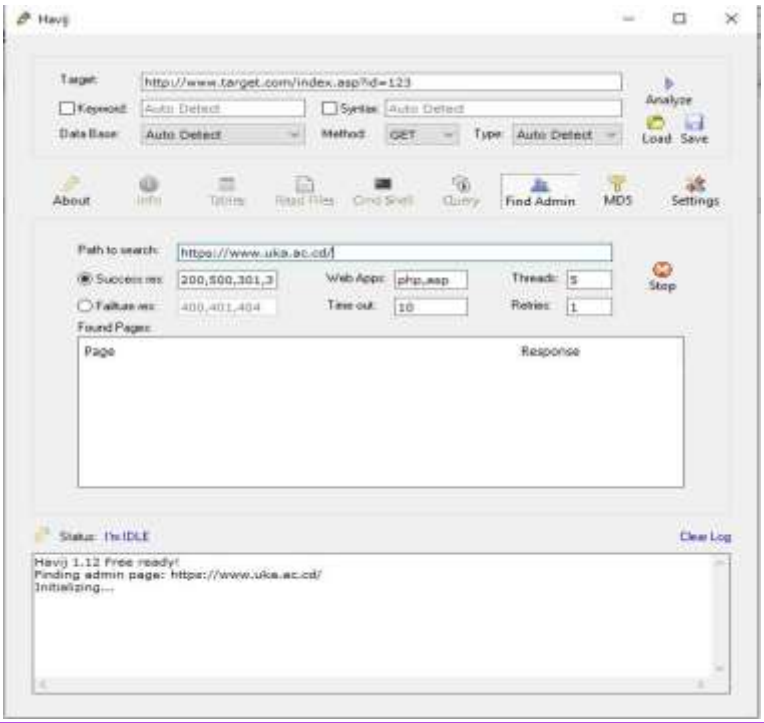
Présentation et usage de l'outil

Le **Havij Advanced SQL Injection** est un logiciel OpenSource qui permet d'analyser un site Web à la recherche de vulnérabilités. Dans le cas sous examen, nous avons choisi de tester le site Web de l'Université Notre-Dame du Kasayi **(U.KA)** afin de détecter les vulnérabilités et les améliorer pour garantir sa crédibilité et celle des Internautes sur le Net.

Remarque : votre programme antivirus peut le signaler en raison de sa nature. Vous devez l'ajouter à la liste des exclusions ou suspendre votre logiciel antivirus.

L'outil ci-dessus peut être utilisé pour évaluer la vulnérabilité d'un site/d'une application Web.

Nous avons pour exemple, le site de l'Université Notre Dame du Kasayi(U.KA) afin de démontrer les différentes vulnérabilités :



Résumé

- L'injection SQL est un type d'attaque qui exploite de mauvaises instructions SQL
- L'injection SQL peut être utilisée pour contourner les algorithmes de connexion, récupérer, insérer, mettre à jour et supprimer des données.
- Les outils d'injection SQL incluent **SQLMap**, **SQLPing** et **SQLSmack**, etc.
- Une bonne politique de sécurité lors de l'écriture d'une instruction SQL peut aider à réduire les attaques par injection SQL en utilisant les requêtes préparées énumérées ci-haut.

Conclusion

Les attaques par injection SQL sont l'une des principales menaces auxquelles les applications web sont confrontées. Cette méthode d'attaque permet aux attaquants d'exploiter les failles de sécurité des applications pour injecter des instructions SQL malveillantes dans les requêtes SQL. Cela peut entraîner des conséquences graves telles que l'accès non autorisé aux données sensibles, l'altération ou la suppression de données, voire l'exécution de commandes malveillantes. Afin de protéger les applications web contre ces attaques, il est essentiel de comprendre la définition de l'injection SQL et de reconnaître l'importance de la sécurité des applications web.