

Compte Rendu : Injection SQL avec Docker

Nom : Péniel LAWSON-BODY

Devoir : Cyber sécurité

1. Objectif

L'objectif est de démontrer comment une faille d'injection SQL permet de contourner une page de connexion. Pour ce faire, j'ai mis en place un environnement isolé utilisant **Docker**.

2. Configuration Docker

J'ai utilisé **GitHub Codespaces** pour faire tourner Docker sans ralentir mon ordinateur.

- **Conteneur App** : Une application Node.js sur le port 3000.
- **Conteneur DB** : Une base de données MariaDB pour stocker les utilisateurs.

3. Mise en place de la base de données

J'ai créé une table `users` et inséré un compte de test via le terminal Docker :

SQL

```
CREATE TABLE users (username VARCHAR(255), password VARCHAR(255));
INSERT INTO users (username, password) VALUES ('testuser', 'password123');
```

4. L'Attaque (Injection SQL)

Le code de l'application est vulnérable car il insère les entrées de l'utilisateur directement dans la requête SQL.

- **Méthode utilisée** : Dans le champ "Nom d'utilisateur", j'ai saisi : `testuser' OR '1='1`.

The screenshot shows a web browser window with the URL `redesigned-system-wr6g96rvjxxf9r4g-3000.app.github.dev`. The page title is "Connexion". There are two input fields: the first contains the value `testuser' OR '1='1`, and the second contains three dots (...). Below the inputs is a button labeled "Se connecter".

- **Résultat :** La condition '`1='1`' étant toujours vraie, le système me connecte sans avoir besoin du bon mot de passe.



A screenshot of a web browser window. The address bar shows the URL: `redesigned-system-wr6g96rvjnxf9r4g-3000.app.github.dev/login`. Below the address bar, the main content area displays the text: **Bravo ! Connexion réussie en tant que testuser**.

5. Conclusion

Ce projet montre que pour sécuriser une application, il ne faut jamais faire confiance aux entrées de l'utilisateur. La solution est d'utiliser des **requêtes préparées** pour empêcher l'exécution de code SQL malveillant.