Rapport de Projet : Démonstration d'une Injection SQL

Fait par **Terence AMBA, Mehdi BAKRI, Anatole MAGRAUD, Estéban De Olivera et Amadis TALAVERA.**

L'injection SQL est l'une des failles les plus connues et dangereuses dans les applications web. Ce projet a pour but de comprendre comment cette faille fonctionne, d'apprendre à l'exploiter dans un environnement contrôlé, puis de mettre en œuvre une méthode de correction efficace.

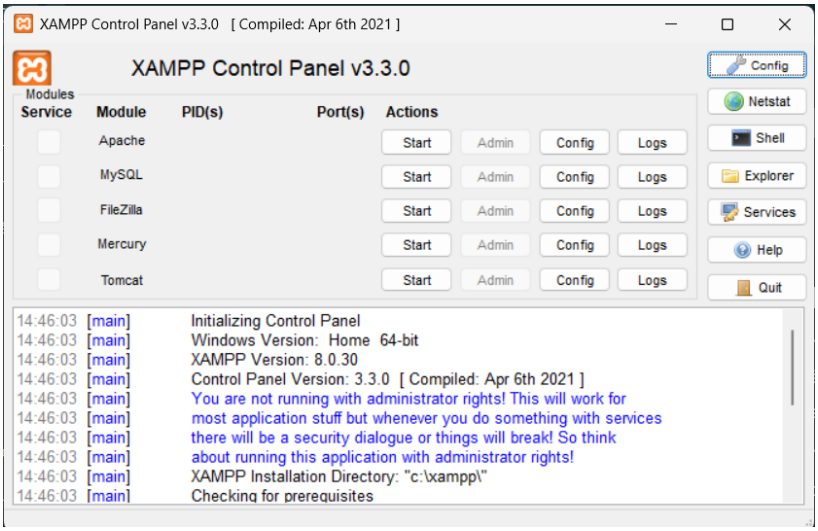
L'objectif n'était pas seulement de provoquer une injection, mais aussi de développer une solution sécurisée en comparaison.

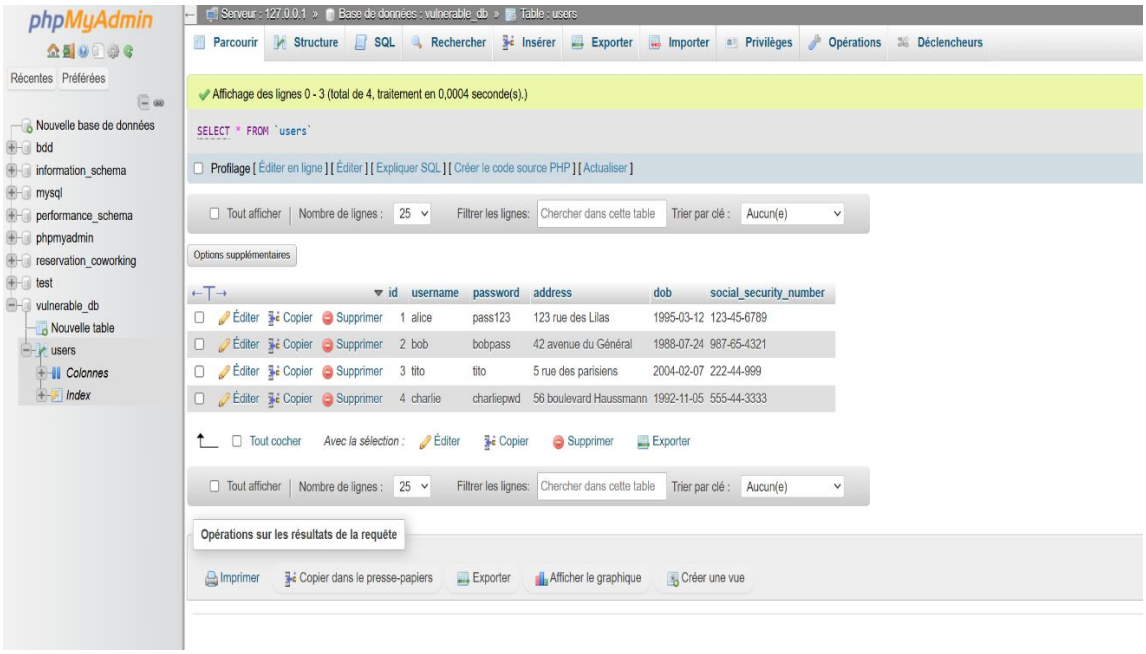
Contexte et objectifs

Dans ce travail, il nous a été demandé de concevoir une application web de type formulaire de connexion, volontairement vulnérable à l'injection SQL. Ce formulaire devait permettre à un attaquant de se connecter sans identifiants valides, voire de cibler un utilisateur spécifique. Ensuite, une version sécurisée du même formulaire devait être développée, cette fois-ci en respectant les bonnes pratiques de codage sécurisé.

Déroulement technique du projet

On a d'abord mis en place un environnement local de développement en utilisant XAMPP, qui regroupe un serveur Apache, une base de données **MariaDB** et PHP. Après avoir créé la base de données **vulnerable_db**, j'ai ajouté une table **users** contenant quelques utilisateurs factices (dont un administrateur) avec des données sensibles (numéro de sécurité sociale, adresse, date de naissance) à des fins de démonstration.





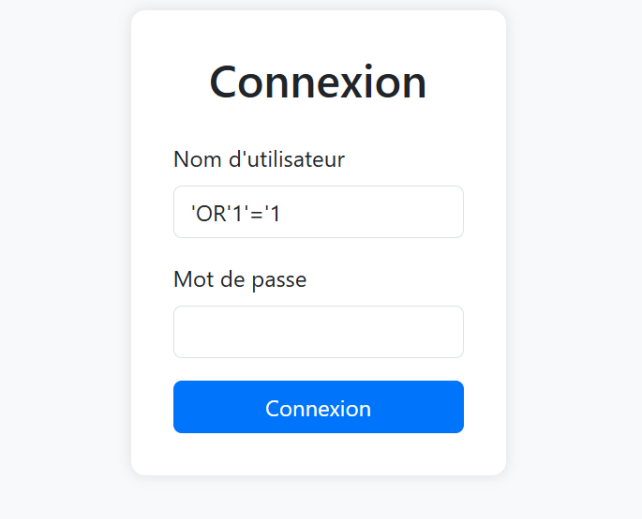
La première version du formulaire de connexion (login.php) récupère les entrées de l'utilisateur (nom d'utilisateur et mot de passe) et les intègre directement dans une requête SQL sans aucune vérification ni protection. Cela rend l'application extrêmement vulnérable, puisque le moindre caractère de contrôle SQL inséré dans les champs du formulaire peut détourner la logique de la requête.

Exploitation de la vulnérabilité

Dans la version non sécurisée de l'application, deux types d'injections SQL ont été testés avec succès.

La première consiste à contourner totalement l'authentification en injectant une condition toujours vraie dans les deux champs du formulaire. En saisissant ' OR '1'='1 comme nom d'utilisateur et mot de passe, la requête devient toujours valide, car elle est interprétée ainsi par le serveur SQL :

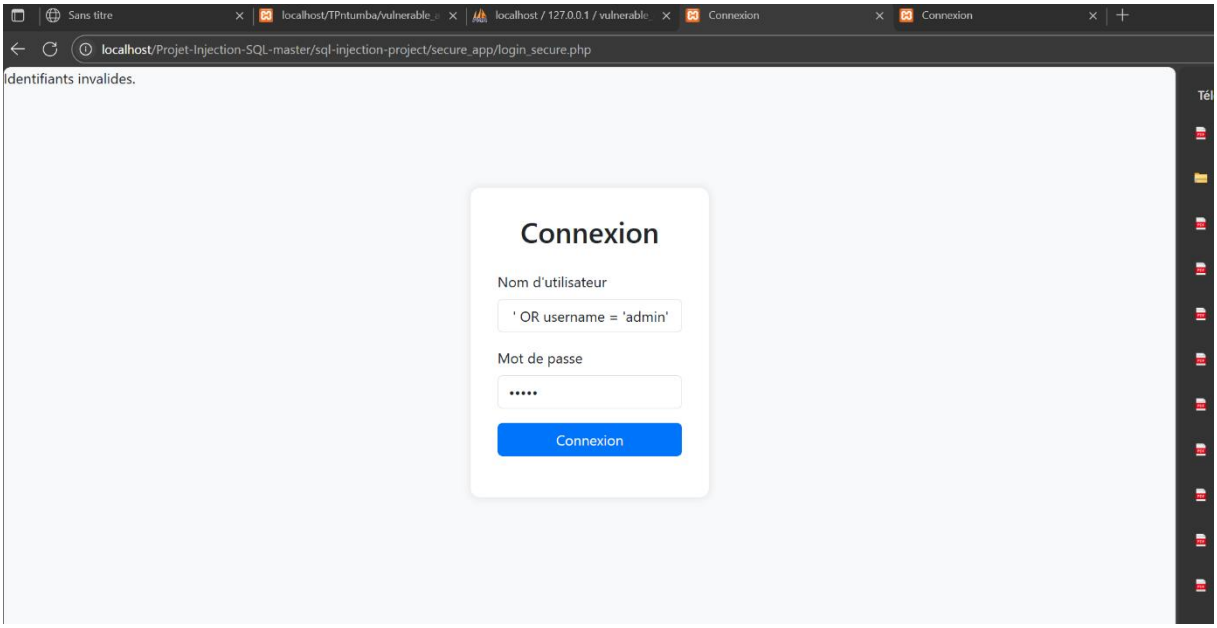
```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1'
```



Cela permet à n'importe qui de se connecter sans fournir de véritables identifiants. Cette forme d'attaque est typique et très répandue.

La deuxième injection testée est plus ciblée. Elle consiste à se connecter en usurpant l'identité d'un utilisateur spécifique, par exemple l'administrateur. En injectant dans le champ "**username**" une valeur comme **' OR username='admin' --,** le reste de la requête est ignoré grâce au double tiret (--) qui marque un commentaire SQL. Même si le mot de passe fourni est incorrect, la requête retourne tout de même les données de l'utilisateur "admin".

```
SELECT * FROM users WHERE username = '' OR username='admin' -- ' AND password = 'abc'
```



Résultats observés

Les deux injections ont été exécutées avec succès sur la version vulnérable. La première a permis de se connecter à n'importe quel compte, et la seconde à accéder précisément au compte de l'administrateur. Une fois connecté, les informations sensibles de l'utilisateur ciblé (adresse, date de naissance, etc.) sont affichées en clair à l'écran, ce qui montre à quel point cette faille peut être dangereuse.

Ces tests prouvent que sans la moindre protection, la base de données est exposée à des risques d'attaques critiques. Il suffirait qu'un pirate utilise ces techniques sur une application en ligne pour accéder à des données confidentielles, voire les manipuler ou les supprimer.

Mise en œuvre de la solution sécurisée

Afin de corriger cette faille, une deuxième version du formulaire de connexion a été réalisée (login_secure.php). Cette version repose sur l'utilisation de requêtes préparées (prepared statements)

avec l'extension mysqli. Au lieu d'insérer directement les données utilisateur dans la requête, on utilise des marqueurs **(?)** qui sont ensuite liés aux vraies valeurs via la méthode bind_param.

Grâce à cette méthode, le moteur SQL ne peut plus interpréter les caractères spéciaux comme ' ou --, car les entrées utilisateur sont traitées comme de simples valeurs, et non comme du code SQL.

Des tests similaires à ceux effectués précédemment ont été réalisés sur cette version sécurisée. Aucune des injections SQL n'a fonctionné : l'accès sans identifiants valides a été refusé, et la tentative de ciblage d'un utilisateur spécifique n'a pas abouti. L'application s'est comportée de façon conforme et sécurisée.

Analyse de sécurité

Ce projet nous a permis de comprendre que l'injection SQL n'est pas seulement une erreur de développement, mais une faille critique qui peut avoir des conséquences lourdes. Dans de nombreuses entreprises, cette faille a été la porte d'entrée de violations de données majeures. Il est donc essentiel d'adopter des réflexes de codage défensif dès la conception.

En plus de l'usage des requêtes préparées, d'autres bonnes pratiques de sécurité web peuvent être mises en œuvre pour renforcer l'application : validation stricte des champs, journalisation des tentatives de connexion, filtrage des entrées, chiffrement des mots de passe, gestion des erreurs serveur, et mise à jour régulière des composants du serveur.

Conclusion

Ce travail nous a offert une approche concrète et complète des injections SQL, à la fois du point de vue de l'attaquant et du développeur. Il m'a permis de mieux comprendre les risques liés à un mauvais contrôle des entrées utilisateur et surtout, de savoir comment les prévenir efficacement. Je retiens de ce projet l'importance de la rigueur dans le développement web sécurisé et la nécessité de toujours penser à la sécurité dès les premières lignes de code.