|  |
| --- |
| Mickaël BRENOIT | Michaël FUCHS  11/09/2017 |

|  |
| --- |
| UNIVERSITE DE HAUTE-ALSACE |
| Sécurisation bases de données |
| Mini-projet |



SOMMAIRE

[1. Présentation du mini-projet 2](#_Toc493019965)

[2. Création de la base de données 3](#_Toc493019966)

[2.1. Table « users » 3](#_Toc493019967)

[2.2. Table « accounts » 3](#_Toc493019968)

[2.3. Script SQL 3](#_Toc493019969)

[3. Les différentes failles 4](#_Toc493019970)

[3.1. Les injections SQL 4](#_Toc493019971)

[3.1.1. Définition et comment les utiliser 4](#_Toc493019972)

[3.1.2. Comment se protéger 5](#_Toc493019973)

# Présentation du mini-projet

Le mini-projet présenté dans ce rapport est le fruit de connaissances récoltées lors de nos recherches internet sur les différentes failles possibles sur un site internet. Il nous a été demandé de :

1. Créer une base de données qui contient deux tables :
   1. Une table « users » représentant les utilisateurs de notre application. Les utilisateurs auront un id, un login et un mot de passe.
   2. Une table « accounts » représentant les comptes bancaires que dispose un utilisateur. Les comptes auront un id, un type, un montant et une clé faisant référence à l’id de l’utilisateur à qui appartiennent les comptes.
2. Réaliser un formulaire en PHP qui se connecte à notre base de données et qui demande le login et mot de passe de l’utilisateur.
3. Rechercher les différentes failles importantes qui existent et les tester sur notre formulaire.
   1. Tester les failles « injections SQL » vues en cours
   2. Rechercher d’autres types de faille sur internet
4. Explorer les solutions sur le web et décrire au moins deux méthodes importantes de protection contre ces failles.
5. Illustrer les scénarios avec failles et sans failles.

# Création de la base de données

## Table « users »

La première table créée est « user ». Elle contient :

* Un champ « id » : identifiant unique d’un utilisateur, incrémenté automatiquement et non nul. C’est la clé primaire de cette table.
* Un champ « login » : pseudonyme de l’utilisateur, il est déclaré unique et non null
* Un champ « password » : mot de passe de l’utilisateur haché, non null.

De base, nous aurons deux entrées :

* Id : 1, login : mickael et password : md5(aA4\*7777)
* Id : 2, login : framboise et password : md5(strawberry)

## Table « accounts »

La seconde table créée est « accounts ». Elle contient :

* Un champ « id » : identifiant unique d’un compte, incrémenté automatiquement et non nul. C’est la clé primaire de cette table.
* Un champ « type » : intitulé du compte bancaire, unique et non null
* Un champ « amount » : montant représenté par un entier réel positif, non null
* Un champ « iduser » : identifiant de l’utilisateur qui détient ce compte, non null

De base, nous aurons quatre entrées :

* Id : 1, type : Livre bleu, amount : 1100 et iduser : 1
* Id : 2, type : Compte Courant, amount : 1600 et iduser : 1
* Id : 3, type : Livret Framboise 1, amount : 600 et iduser : 2
* Id : 4, type : Livret Framboise 2, amount : 750 et iduser : 2

## Script SQL

Image

# Les différentes failles

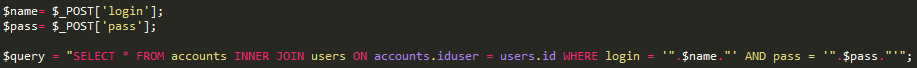
## Les injections SQL

### Définition et comment les utiliser

Les injections SQL (SQLi) se réfère à une attaque d'injection dans laquelle un attaquant peut exécuter des instructions SQL malveillantes qui contrôlent le serveur de base de données d'une application Web (également communément appelé système de gestion de base de données relationnelle - SGBDR). Étant donné qu'une vulnérabilité SQL Injection pourrait affecter n'importe quel site Web ou application Web qui utilise une base de données basée sur SQL, cette vulnérabilité est l'une des vulnérabilités les plus anciennes, les plus répandues et les plus dangereuses.

Pour exécuter des requêtes SQL malveillantes contre un serveur de base de données, un attaquant doit d'abord trouver une entrée dans l'application Web qui permet d’inclure une requête SQL. Un attaquant peut alors insérer une partie de code SQL qui sera incluse dans la requête SQL et s'exécuter en fonction du serveur de la base de données.

Le pseudo-code suivant du côté serveur sert à authentifier les utilisateurs sur l'application Web.



Le script ci-dessus est un exemple simple d'authentification d'un utilisateur avec un nom d'utilisateur et un mot de passe contre une base de données avec une table nommée utilisateurs et une colonne de nom d'utilisateur et de mot de passe. Le script ci-dessus est vulnérable à l'injection SQL, car un attaquant peut envoyer des entrées malveillantes de manière à modifier l'instruction SQL exécutée par le serveur de base de données. Un exemple simple d'une charge utile d'injection SQL pourrait être quelque chose d'aussi simple que le champ de mot de passe en password 'OR 1 = 1**.**

Cela entraînerait que la requête SQL suivante soit exécutée contre le serveur de base de données.

**SELECT id FROM users WHERE username=’username’ AND password=’password’ OR 1=1’**

Dans le cas où le contournement d'authentification est possible, l'application enregistrera probablement l'attaquant avec le premier compte à partir du résultat de la requête - le premier compte dans une base de données est habituellement un utilisateur administratif.

### Comment se protéger

Selon OWASP « Open Web Application Security Project (OWASP) est une communauté en ligne travaillant sur la sécurité des applications Web. Sa philosophie est d'être à la fois libre et ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web. », la mise en place de trois défenses est obligatoire pour lutter contre les injections SQL.

Voici la liste des trois défenses :

* Utilisation des requêtes préparées
* Utilisation des procédures stockées
* Vérifier toutes les entrées utilisateurs

Les requêtes préparées garantissent qu'un attaquant ne peut pas modifier l'intention d'une requête, même si les commandes SQL sont insérées par un attaquant. Dans l'exemple de sécurité vu précédemment, si un attaquant devait entrer l'ID utilisateur de tom 'ou' 1 '=' 1, la requête paramétrée ne serait pas vulnérable et rechercherait un nom d'utilisateur qui correspondait littéralement à la chaîne entière tom 'ou' 1 '=' 1.

L’utilisation des procédures stockées est très similaire à l’utilisation des requêtes préparées. La seule différence entre les instructions préparées et les procédures stockées est que le code SQL pour une procédure stockée est défini et stocké dans la base de données elle-même, puis appelé à partir de l'application.

Vérifier toutes les entrées utilisateurs consiste à échapper les saisies des utilisateurs avant de les mettre dans une requête SQL. Pour cela on utilise des fonctions spéciales qui vont filtrer les données en prenant soin de virer les balises <elem></elem> ainsi que les apostrophes.