Rapport de Cryptographie

Les Injections SQL

Sommaire

Introduction

Ce rapport vise à expliquer ce qu'est une injection SQL. Nous allons pour cela procéder en plusieurs étapes. Dans un premier temps, nous allons expliquer pourquoi nous avons choisi ce sujet. Puis, nous allons expliquer rapidement les bases du langage SQL. Enfin nous allons montrer les injections SQL connues, la façon dont celles-ci procèdent et enfin essayer de mettre en avant des manières de procéder afin d'éviter d'être la cible de telles attaques.

Définition d'une injection SQL

Une injection SQL est un type d'exploitation d'une faille de sécurité d'une application web, en injectant une requête SQL non prévue par le système et pouvant compromettre sa sécurité.

Contexte

Depuis les dernières années, une importante explosion des sites internet a eu lieu, notamment les sites internet dynamiques. Pour la plupart, ces sites dynamiques utilisent une base de données. De là est venue l'apparition de la faille de sécurité dite « Injection SQL ».
Ce type de faille se trouve être critique, car celle-ci permet notamment d'accéder aux informations de bases du site : « les données ». Cela peut par exemple permettre de récupérer le mot de passe d'utilisateur (ce qui permettrait un éventuel usage de compte paypal ou autres).

Les sites web dits dynamiques ont souvent des formulaires autorisant l'utilisateur à rentrer des informations. Ces informations sont ensuite envoyées au serveur pour traiter la requête avec ces informations. Mais que se passe-t-il si l'utilisateur est mal intentionné et qu'il rentre des informations bien précises pour infiltrer votre site ?

Le langage utilisé : SQL

Le **SQL** (Structured Query Language) est le langage utilisé afin de récupérer des informations sur une base de données. Pour cela nous effectuons ce qu'on appelle des requêtes (des Query en anglais).

Une requête est formulée de la sorte :

CLE [nom des champs] TABLE [CONDITION]

CLE peut par exemple être le mot permettant :
- une sélection : « SELECT »
- une modification : « UPDATE »
- une suppression : « DELETE »
- une insertion de données : « INSERT »

Entre crochet (donc optionnel), on peut trouver le nom des champs. Par exemple si on ne veut sélectionner que certains champs, on peut taper : *SELECT champs1, champs2*.

Ensuite seulement, nous trouvons le nom de la table. Il faut préciser sur quelles tables sont effectuées les opérations.

Il est ensuite possible de mettre des conditions sur les résultats que l'on veut avoir. On peut notamment restreindre un des champs de nos résultats à une valeur :

WHERE champs= « valeur »

Les Injections SQL

Afin de faciliter la lisibilité de ce rapport, nous allons utiliser les légendes suivantes.

Représentera le code type de création de la requête sur le serveur

Données rentrée par l'utilisateur malveillant

Création de la requête avec les données rentrées par l'utilisateur

Explication de l'erreur

Manière de se protéger de cette attaque

Dans la partie du code type, nous allons écrire du PHP. En effet, celui-ci est un langage dynamique souvent utilisé sur internet. Celui-ci permet entre autre de récupérer la valeur d'un champ rentré par un utilisateur dans un formulaire fait à cet effet.

Cette récupération de valeur se fera de la manière suivante en PHP :

$_POST['nom_de_champs']

Les Hello World SQL Injections

Ce type d'injection SQL utilise une faille se situant à la construction même de la requête. Supposons que vous voulez effectuer une recherche sur un moteur de recherche, la requête ressemblerait à :

```
$query = " SELECT nom_site, url_site FROM table_site WHERE
nom_site LIKE ' %$_POST['search']% ' OR
url_site LIKE ' %$_POST['search']% ' "
```

Cette requête signifie que l'on souhaite récupérer le nom du site ainsi que l'adresse internet du site dont les sites possèdent dans leur nom ou dans leur URL la valeur du champ *search*.

Imaginons qu'un utilisateur mal intentionné vienne sur votre site, et veuille le mettre à rude épreuve :

```
a' OR 'a%' = 'a
```

```
$query = " SELECT nom_site, url_site FROM table_site WHERE
nom_site LIKE '%a' OR 'a%' = 'a%' OR
url_site LIKE '%a' OR 'a%' = 'a%' "
```

Syntaxiquement, la requête est bien construite, mais au lieu de nous renvoyer que les sites possédant dans leur URL ou dans leur nom a' OR 'a%' = 'a , la requête va nous renvoyer tous les sites. En effet, ici les ' ont été interprété, ce qui a trompé le serveur : 'a%'='a%' renverra toujours vrai.

Ici le principal problème vient du fait que ' a été interprété. Afin d'éviter ce type d'injections SQL, il faut faire comprendre au serveur de ne pas interpréter le ' comme celui de la syntaxe SQL. Il faut donc utiliser des fonctions d'échappement de caractères spéciaux.

Les formulaires d'authentification

Ici, nous allons montrer plusieurs méthodes pour s'authentifier avec un compte qui ne vous appartient pas. Cela peut être notamment très dangereux, si l'on arrive à s'authentifier sur le compte d'un administrateur.

```
'$query =  " SELECT name, password FROM utilisateurs WHERE
name = ' $_POST['name'] ' AND
password = ' $_POST['password'] ' "
```

Cette requête permet de vérifier le mot de passe et le login d'un utilisateur.

Premier Cas

```
a' OR 'a' = 'a
```

```
$query = " SELECT name, password FROM utilisateurs WHERE
name = 'a' OR 'a' = 'a' AND
password = 'a' OR 'a' = 'a' "
```

Syntaxiquement, la requête est bien construite, mais ici, nous allons passer le formulaire d'inscription sans posséder pour autant un compte. En effet 'a'='a' nous renverra toujours vrai.

Ici encore le a' OR 'a'='a a été interprété. Encore une fois, la solution est de mettre une fonction permettant d'échapper les caractères spéciaux, avant que la phrase soit injectée dans la requête SQL.

Deuxième Cas

```
Name :  Administrateur'/*
```

```
$query = " SELECT name, password FROM utilisateurs WHERE
name = 'Administrateur'/* AND
password = '' "
```

Ici on veut s'authentifier avec le nom Administrateur. Le /* est en SQL le début d'un commentaire. Donc la suite de la requête n'est pas prise en compte et l'utilisateur sera authentifié avec le compte ayant pour login Administrateur.

Ici encore le ' a été interprété. Une fonction d'échappement aurait suffit pour éviter ce problème. Mais d'autres solutions sont possibles (on peut par exemple bannir de notre langage /*, c'est-à-dire que avant de l'injecter dans la requête, si nous trouvons un /* il serait supprimé.

Dans le même genre nous aurions pu rentrer :

Name : Administrateur'--

Name : Administrateur'#

Dans ces deux autres cas, la partie après – ou # aurait été également ignoré de la même façon.

Troisième Cas

Dans le second cas, nous validions le champ de login, mais cela implique que nous connaissions le nom. Nous allons voir ici comment valider le login sans connaître de nom, et en ignorant le reste. Cela revient donc plus ou moins à un mixte des deux premiers cas.

Name : ' OR 1=1/*

$query = " SELECT name, password FROM utilisateurs WHERE
name = ' ' OR 1=1/* AND
password = '' "

Ici on ne rentre pas de nom d'utilisateurs, mais nous arrivons tout de même à valider ce champ grâce au 1=1 qui renverra toujours vrai, et nous nous permettons ensuite d'ignorer le mot de passe en utilisant les symboles de commentaires.

Ici encore, une fonction d'échappement aurait suffit. Nous aurions également pu penser à effectuer une requête dans laquelle on vérifierait d'abord le mot de passe et non le login. Ainsi, le commentaire figurant dans le champ *name* n'aurait eu que peu d'effet. A moins que l'utilisateur rentre quelque chose de similaire dans les deux champs afin de s'assurer que l'ordre de vérification ne le bloque pas…

```
Name : ‘ OR 1=1--
```

```
Name : ‘ OR 1=1#
```

Récupération des informations d’une table

Il est parfois très utile pour un hacker de connaître l’architecture d’une des tables de votre base de données. Il peut donc procéder de manière à ce que si votre site ne soit pas bien sécurisé, celui-ci lui informe des noms des colonnes de la table.

GROUP BY

Prenons le même exemple que tout à l’heure dans les formulaires d’authentification

```
$query = “ SELECT * FROM utilisateurs WHERE
name = ‘ $_POST[‘name’] ‘ AND
password = ‘ $_POST[‘password’] ‘ ”
```

Imaginons maintenant que la table utilisateurs comporte d’autres champs. Le premier serait le champ *id*, le second *name*, le troisième *password* et le dernier *mail*.

```
Name : ‘ HAVING 1=1--
```

```
$query = “ SELECT * FROM utilisateurs WHERE
name = ‘ ’ HAVING 1=1-- AND
password = ‘’ ”
```

```
Le serveur va alors nous retourner une erreur. Celle-ci sera du type : « Column utilisateurs.id is invalid [… ]». Nous avons abrégé délibérément le message d’erreur car le reste ne nous intéresse peu.
```

Mais de là, nous savons que le nom de la table est utilisateurs et que le premier champ est *id*. Maintenant si nous rentrons à nouveau dans le formulaire et que cette fois nous tapons :

```
Name : ‘ GROUP BY utilisateurs.id HAVING 1=1--
```

```
$query = “ SELECT * FROM utilisateurs WHERE
name = ‘ ’ GROUP BY utilisateurs.id HAVING 1=1-- AND
password = ‘’ ”
```

Le serveur va alors nous retourner une erreur. Celle-ci sera du type : « Column utilisateurs.name is invalid […]».

Nous savons maintenant que le second champ est *name* et si on tape maintenant :

‘ GROUP BY utilisateurs.id, name HAVING 1=1--

Et ainsi de suite… On continuera à recevoir des erreurs jusqu’à ce que nous ayons toutes les colonnes de la table.

Union

Maintenant que nous savons comment récupérer le nom des champs, il peut être intéressant de savoir leur type, c'est-à-dire savoir s’ils contiennent des entiers, des champs textes ou autre. Nous allons donc utiliser ici le mot UNION :

Name : ‘ UNION select sum(name) FROM utilisateurs--

$query = “ SELECT name, password FROM utilisateurs WHERE name = ‘ ’ UNION select sum (name) FROM utilisateurs-- AND password = ‘’ ”

Ici le serveur va nous retourner une erreur du type :
« The sum or average operation cannot take a varchar data type as an argument »

Ce message nous informe donc que le champ *name* est de type varchar (autrement dit du texte).

Encore une fois, une des solutions aurait été de protéger la chaine de caractères avec une fonction d’échappement. Mais il aurait été très simple de bloquer certains mot clés (le UNION dans notre cas).

Maintenant que l’utilisateur malveillant a récupéré toutes ces informations, il peut se donner certains droits facilement. En effet, qui dit formulaire d’authentification dit également la plupart du temps formulaire d’inscription

Formulaire d'inscription

Changer des valeurs

Lorsque vous désirez vous inscrire sur un site, une requête d'insertion de données est envoyée au serveur . Cette requête est du type suivant :

```
$query = "INSERT INTO utilisateurs VALUES('', '$name', '$password', '$mail', 1)"
```

Nous avons ici rajouté un champ à la table utilisateurs. Ce champ va représenter le niveau de privilège de l'utilisateur (le nom de la colonne est access).
1 = Utilisateurs
10 = Bannis
100 = Modérateurs
1000 = Administrateurs
Ici à l'inscription nous ne donnons donc que des droits simples à un nouvel utilisateur.

Maintenant, puisque la personne connait les champs de la table, ainsi que les données qu'il contient, voyons ce qu'il peut faire. Imaginons que dans le champ mail il rentre :

```
Mail : test@test.com',1000--
```

```
$query = « INSERT INTO utilisateurs VALUES('', 'test', 'test', 'test@test.com',1000--
, 1)
```

L'utilisateur va avoir dès son inscription les droits d'un administrateur.

La fonction d'échappement aurait encore marché sur cet exemple. Mais voyons à présent une autre solution marchant très bien : « les prepare statements ».
Ce système de Prepared Statement est un bon moyen de protection car toute chaine de caractères passée en paramètres est vraiment considérée comme une chaine, et les mots clés du SQL ne sont donc pas pris en compte.

Autres type de requêtes

Suppression d'une table

Imaginons qu'on veuille rechercher un article par un numéro. La requête serait du type :

```
$query = "SELECT * FROM articles WHERE id = '$_POST['id']'"
```

Le langage SQL permet notamment d'exécuter plusieurs requêtes de suite.

```
Id : ' ; DROP TABLE articles--
```

```
$query = "SELECT * FROM articles WHERE id = ''; DROP TABLE articles --'"
```

Le résultat de la requête sera donc la suppression de la table articles.

Les solutions sont identiques aux autres. La solution la plus simple à mettre en œuvre est encore une fois les fonctions d'échappement. Mais la mise en place d'un dictionnaire de mots clés interdit est également viable.
La solution la plus propre reste néanmoins encore les prepared statements

Les Injections SQL à l'aveugle (Blind SQL Injection)

Le principe de ces injections est de rentrer l'information que l'on veut valider dans la requête. Ces injections ne sont utilisées que dans des requêtes ne renvoyant vrai ou faux. Ces injections n'ont pas pour but d'afficher un quelconque résultat, mais d'obtenir des informations. Nous allons montrer un petit exemple :

Version du serveur

```
SELECT * FROM table WHERE champ ='$_POST['valeur']';
```

```
a' OR @@version > 3;
```

```
SELECT * FROM table WHERE champ = 'a' OR @@version > 3;
```

Si la requête renvoie une erreur, la version est inférieure à 3, sinon la version est supérieure à 3.

Les solutions sont les même que présentées précédemment :
- fonctions d'échappement
- bloquer des mots clés
- les Prepare Statement

De la même manière, on aurait pu tester si la version était supérieure à 4.

Récupération du nombre de champs

Nous avons vu précédemment comment récupérer tous les champs d'une table, mais d'autres solutions existent

```
SELECT id, champ1 FROM table WHERE id = '$_POST['valeur']';
```

```
$id_non_existant' GROUP BY 1;
```

```
SELECT id, champ1 FROM table WHERE id = '$id_non_existant' GROUP BY 1;
```

Si la requête retourne vrai, il y a au moins 1 champs.

On réitère le processus en changeant le GROUP BY 1, par GROUP BY 2, etc … La dernière fois où la requête ne renvoie pas d'erreurs nous donnera le nombre de champs dans la table.

Les solutions sont les même que présentées précédemment :
- fonctions d'échappement
- bloquer des mots clés
- les Prepare Statement

Résumé : comment protéger sa base de données

A comprendre

En aucun cas vous ne devez avoir confiance en l'utilisateur. Bien que beaucoup des utilisateurs du site n'aient pas d'intentions malveillantes, il existera un jour quelqu'un à qui il prendra l'idée de mettre votre serveur à rude épreuve.

Les protections possibles

- Désactiver les fonctions non utilisées

Désactivez toutes les fonctions que vous n'utilisez pas. Il n'est pas nécessaire de garder des fonctions de votre serveur que vous n'utilisez pas car elles sont potentiellement dangereuses pour vous.

- Message d'erreurs personnalisés

Lors de l'exécution de la requête, effectuez un test pour savoir si celle-ci renvoie une erreur. Si erreur il y a, mettez une erreur personnalisée. Les erreurs SQL donnent en effet trop d'informations à un utilisateur malveillant.

- Fonctions d'échappement

Les fonctions d'échappement sont très faciles à mettre en place, et permettent de sécuriser rapidement votre serveur contre la plupart des attaques.

- Interdire certains mots clés

N'autorisez pas certains caractères ou mot clés. Il existe en effet des mots que l'utilisateur n'a pas à rentrer dans une requête. Si ce genre de mot est rentré, il faut l'enlever, car celui-ci viendra probablement d'une attaque.

- Limiter la taille des données

Vous pouvez limiter la taille des données rentrées par l'utilisateur. On a vu que certaines injections demandaient un certains nombres de caractères (vous pouvez limiter les numéros d'identifiants à 5 caractères, un login à 15 par exemple).

- Utiliser les Prepare Statements

Utiliser les prepare statements. Cette solution reste la plus efficace pour se protéger des injections SQL. Elle vous demandera un peu plus de temps (pas beaucoup plus), mais sécurisera de manières efficaces votre serveur.

Conclusion

Ce rapport nous a permis de présenter quelques injections SQL parmi les plus connues, permettant ainsi de mettre en évidence l'importance de la protection d'une base de données. Ces injections permettant en effet d'avoir accès aux données, de les modifier voire de les supprimer, d'usurper le compte d'un utilisateur, d'accéder à la structure de la base, de contourner les règles de gestion, ou encore obtenir certaines informations concernant le serveur. Quelques protections simples permettent d'éviter que sa base de données soit victime de ces attaques telles que l'interdiction de certains mots clés, l'usage de fonctions d'échappement ou encore la désactivation de messages d'erreur un peu trop bavards…

Bibliographie

- Wikipédia en rapport aux injections SQL :
  http://fr.wikipedia.org/wiki/Injection_SQL

- SQL :
  http://fr.wikipedia.org/wiki/Structured_Query_Language

- Ghost in the Stacks, les bases :
  http://www.ghostsinthestack.org/article-8-les-bases-des-injections-sql.html

- Ghost in the Stacks, les blind SQL injections :
  http://www.ghostsinthestack.org/article-11-blind-sql-injections.html

- Knol de Marc boizeau :
  http://knol.google.com/k/marc-boizeau/le-risque-injection-sql/g6fqt9cl9ver/6#

- Site de Jeremy Amiot :
  http://www.jeremyamiot.com/erreurs_injections_sql.pdf