Que20

Zeste de savoir

Les injections SQL

12 août 2019

Table des matières

Pour ce premier article, j'ai décidé d'aborder une faille classique et malheureusement encore trop présente : les injections SQL.

Le but de cet article n'est pas d'apprendre à pirater un site (ce qui, comme vous vous en doutez, est illégal) mais plus à comprendre le problème afin de s'en prémunir. Il s'agit donc plus d'une introduction plutôt qu'un tutoriel détaillant comment faire une injection SQL.

1. Qu'est-ce qu'une injection SQL ?

Une injection SQL, comme son nom l'indique, consiste à injecter du code SQL dans une donnée afin de continuer ou plutôt de détourner la requête et de lui faire faire autre chose que ce pour quoi elle a été conçue. Cela permet de manipuler la base de données et d'accéder, par exemple, à des données "normalement" inaccessibles (tables des utilisateurs avec tout ce qu'elle contient : login, mot de passe, adresse mail etc...) ou encore d'effectuer des opérations qu'un utilisateur classique ne devrait pas pouvoir faire (suppression de la base de données, ajout/modification d'enregistrements, création/lecture de fichiers etc...).

Passons maintenant à la démonstration de 2 cas.

2. Premier cas : injection SQL sur une chaîne de caractères

En langage SQL, une chaîne de caractère est entourées de guillemets (simples ou doubles).

Examinons la requête suivante :

```
1 $query = "SELECT id, titre, texte FROM articles WHERE titre LIKE
      '%".$_GET['titre']."%'";
```

2. *Premier cas : injection SQL sur une chaîne de caractères*

Cette requête va rechercher l'article dont le titre contient le terme envoyé par la variable titre, variable de type GET (mais ça aurait pu être POST, cela ne change rien).

> Une variable de type GET est présente dans l'URL sous la forme nom_variable=valeur.

Imaginons maintenant que nous avons un article intitulé "L'article de la semaine". Nous allons le rechercher en utilisant le mot "**article**" :

```
http://www.monsite.com/article.php?titre=article
```

Ce qui donnera la requête SQL suivante :

```
SELECT id, titre, texte FROM articles WHERE titre LIKE '%article%'
```

Rien à signaler, tout se passe comme prévu.

Maintenant tentons de le rechercher en utilisant cette fois le mot "**L'article**"

```
http://www.monsite.com/article.php?titre=L'article
```

Et là ... nous avons une erreur !

Pourquoi ?

Regardons nos 2 requêtes SQL de plus près...

Premier cas :

```
SELECT id, titre, texte FROM articles WHERE titre LIKE '%article%'
```

Second cas :

```
SELECT id, titre, texte FROM articles WHERE titre LIKE
    '%L'article%'
```

Avez-vous trouvé l'erreur ou plutôt ce qui cause cette erreur ?

Le responsable est en fait notre guillemet présent dans le terme recherché. Ce guillemet va être considéré comme la fin de la chaîne de caractère et ce qui suit va être interprété comme du code SQL. La commande **article%'** n'existant pas, cela produit une erreur.

Mais que se passerait-t-il si la commande existait et que la requête serait syntaxiquement juste ?

La réponse est simple : il n'y aurait aucune erreur et la requête serait interprétée.

Essayons d'injecter ceci :

```
http://www.monsite.com/article.php?titre=article' AND 1=1 --
```

Pas d'erreur et l'article s'affiche bien.

> *i* Le - - à la fin est un commentaire. En effet, si nous ne le mettons pas il restera un bout de la requête initiale, ce qui risque fort de provoquer une erreur de syntaxe... à moins que l'on s'arrange pour que notre injection concorde syntaxiquement.
> Mais l'on peut faire beaucoup plus simple : s'arranger pour ignorer totalement cette partie et c'est précisément à ça que sert ce commentaire placé juste après notre injection.

Tentons maintenant cela :

```
http://www.monsite.com/article.php?titre=article' AND 1=2 --
```

De nouveau pas d'erreur... mais cette fois l'article ne s'affiche pas.

Cela n'a rien de magique si l'on y réfléchit bien : la première condition est vérifiée, la seconde par contre ne l'est pas (1 n'est pas égal à 2). Le AND exige que les 2 conditions renvoient TRUE pour que le tout le soit, or là nous avons une condition renvoyant FALSE d'où l'absence de résultat. Ceci étant dit la requête est syntaxiquement juste et la présence ou l'absence de résultat nous prouve que le code SQL a bien été exécuté et qu'il est donc possible d'en injecter afin de détourner la requête initiale.

2.1. Comment s'en protéger ?

Pour le cas des chaînes de caractères, il faut les échapper comme on dit. C'est à dire qu'un antislash (\ ) sera ajouté devant les caractères potentiellement dangereux comme les guillemets ainsi que d'autres caractères. Cet antislash signifiera que le caractère qui suit doit être interprété comme du simple texte. Le guillemet injecté ne sera donc plus considéré comme la fin de la chaîne mais comme un guillemet. Il sera donc impossible au pirate de fermer la chaîne et par conséquent, impossible d'exécuter son injection SQL.

Pour ça, il faut utiliser des fonctions comme mysqli_real_escape_string() ou la préparation de requête si vous utilisez des outils comme PDO (encore une fois, à adapter si vous utilisez autre chose).

Nous avons abordé le cas d'une injection SQL sur une chaîne de caractères mais ce n'est pas le seul existant. Dans l'exemple suivant, vous comprendrez que l'échappement des chaînes n'est pas forcément suffisant pour contrer ce type d'attaque.

3. Second cas : injection SQL sur un nombre

L'échappement est une bonne chose mais n'est pas toujours suffisant et nous allons en apporter la preuve avec cet exemple.

Cette fois nous rechercherons notre article sur la base de son identifiant, identifiant qui est un nombre entier.

```
$query = "SELECT id, titre, texte FROM articles WHERE id =
    ".$_GET['id'];
```

Cette requête va afficher l'article correspondant à l'id qu'on lui a passé en paramètre.

```
http://www.monsite.com/article.php?id=1
```

L'article possédant l'id 1 sera affiché.

```
SELECT id, titre, texte FROM articles WHERE id = 1
```

La principale différence avec une injection sur une chaîne de caractères réside dans le fait qu'un nombre n'a pas forcément besoin d'être entouré de guillemets (bien qu'il peut l'être). Le pirate n'a donc ici plus besoin de chercher à fermer la chaîne. Il peut directement injecter du code SQL après l'identifiant.

De ce fait, tant qu'il n'utilise pas de chaînes de caractères (et donc des guillemets) son injection sera prise en considération sans aucun problème. L'échappement n'est donc pas suffisant dans ce cas ci pour se protéger.

Exemple :

```
http://www.monsite.com/article.php?id=1 AND 1=2 --
```

ce qui donnera :

```
SELECT id, titre, texte FROM articles WHERE id = 1 AND 1=2 --
```

Aucun guillemet n'a été utilisé et notre requête a bien été exécutée.

3.1. Comment alors réellement s'en protéger dans tous les cas ?

Il y a un dicton dans le milieu informatique qui dit :

> Ne faites JAMAIS confiance aux données provenant de l'utilisateur !

En bref, vous DEVEZ vérifier que la donnée reçue correspond bien à ce que vous attendez et prévoir que l'utilisateur peut tenter d'injecter des caractères non prévus (caractères alphanumériques, guillemets, slash, signes de ponctuation, etc.). Ces caractères ne doivent pas affecter le comportement de votre requête sinon ça signifie qu'il est potentiellement probable qu'on puisse la détourner.

Par exemple, si l'identifiant de votre article est un nombre et que vous savez que ça sera toujours un nombre, alors vérifiez que la variable reçue est bien un nombre (en PHP cela peut se faire notamment avec la fonction is_numeric). Si vous recevez une chaîne de caractères, assurez-vous de la gérer et ne permettez pas à l'utilisateur de pouvoir fermer cette chaîne et, par conséquent, d'injecter ce qu'il veut à la suite (en bref : échappez-là).

> Les liens qui suivent se rapportent à MySQL et à PHP mais des fonctions similaires devraient logiquement exister pour les autres langages/SGBD.

MySQLi : http ://php.net/manual/fr/mysqli.real-escape-string.php

PDO : http ://php.net/manual/fr/pdo.quote.php

PDO et requêtes préparées : http ://php.net/manual/fr/pdo.prepare.php

Ce n'est bien souvent pas grand chose mais une erreur de ce type peut vous coûter très cher. Et contrairement à ce que l'on peut croire, ça ne touche pas que les "petits" sites. De nombreux outils très répandus sont touchés par ce fléau : on peut citer notamment Wordpress, Joomla et plus récemment Drupal (ce qui représente juste ... quelques millions de sites).

Les injections SQL sont, comme beaucoup d'autres failles, dues à un manque de vérification de la part du développeur. Elles peuvent mener à des conséquences désastreuses et beaucoup d'entreprises ont notamment été victimes de chantage suite au vol de leur base de données.

De plus, les logiciels automatisant cette attaque sont de plus en plus nombreux et par conséquent le nombre d'attaquants également plus élevé.

Il n'est donc pas étonnant que cette faille se retrouve dans le top du classement des failles web. Elle peut être malgré tout facilement évitée avec de bonnes pratiques comme nous venons de le voir.