Fiche Recap - Injection SQL

Introduction

Une injection SQL va être le fait d'insérer (ou d'injecter) d'une partie d'une requête SQL, voire une requête SQL complète depuis un client (généralement un navigateur) vers un serveur.

Une attaque SQLi fonctionnelle permet globalement d'exécuter des morceaux arbitraires de requêtes SQL et ainsi de lire de la donnée sensible en provenance de la base de donnée mais l'idée va aussi être de venir modifier, ajouter et/ou supprimer des données déjà existantes.

Il va même être possible d'opérer des actes l'administration, tel que l'extinction ou le redémarrage du serveur et ça peut même aller jusqu'à l'exécution arbitraire de commandes sur le système d'exploitation du serveur distant.

> Note : Les fonctionnalités telles que l'écriture de fichiers ou le lancement de commandes concernent essentiellement des serveurs Microsoft SQL Server et ne sont plus d'actualité dans la configuration par défaut.

Modèle de menace

- Une injection SQL permet à un attaquant d'usurper une identité, d'altérer une donnée existante, entamer la répudiation d'un acte. Permet la découverte et la suppression de l'ensemble des données d'un serveur.
- Une injection SQL est une faille très courante dans les applications PHP et ASP en raison de la prévalence de vieilles interfaces de fonctionnement. Les applications en .Net ont tendance à utiliser des interfaces d'échanges obsolètes et sont plus généralement soumises à ce type de failles.
- La sévérité d'une injection SQL n'est limité que par la compétence de l'attaquant ainsi que son imagination et dans une moindre mesure par les mesures de défenses en profondeurs quelconques que l'on pourrait mettre en place du côté du serveur (limitation des privilèges, procédures stockées, ...).

Description

Une injection SQL survient lorsque :

- Une donnée non attendue entre dans un programme depuis une source considérée comme non étant de confiance
- Cette donnée est utilisée pour construire une requête

> Note : On considère comme source de "non confiance" n'importe quelle donnée issue d'une entrée utilisateur (pas forcément uniquement d'un formulaire mais dans les entêtes HTTP également).

Les conséquences de ce type d'attaques sont multiples et touchent l'ensemble des points que l'on cherche à protéger dans un modèle de sécurité informatique.

Confidentialité : Etant donné que les bases de données SQL contiennent des données (potentiellement sensibles), la perte de confidentialité est souvent considéré comme étant l'impact le plus fort lors d'une injection SQL.

Authentification : Il est très commun d'avoir un système SQL pour le stockage des comptes utilisateurs, une requête forgée sans traitement de l'entrée utilisateur peut permettre à un attaquant de bypass le système d'authentification en place et se connecter en tant que quelqu'un d'autre sans même avoir connaissance de son mot de passe.

> Note : Il est même parfois possible de se connecter en tant qu'un utilisateur qui n'existe pas.

Autorisation : Il n'y a pas que des questions d'authentification (et donc d'accès), si les permissions sont également stockées en base, l'exploitation d'une vulnérabilité de type injection SQL permet de procéder à de l'élévation de privilège (généralement applicative).

Intégrité : Dans la mesure ou il est possible de lire des données sensibles, il est généralement également possible de les modifier ou de les supprimer et donc impacter l'intégrité et la disponibilité.

Facteurs de risque

Les plateformes affectées par ce type d'attaques peuvent être multiples étant donné que l'ensemble des plateformes opérant des interactions avec un système SQL vont être concernées.

Comme le XSS avec les formulaires, les failles de type injection SQL sont devenues un standard d'exploitation au sein des sites web "database-driven".

Le défaut est cependant facilement traçable (aussi bien pour un attaquant que pour un défenseur).

Exemple

Une requête SQL va ressembler à ça :

```
SELECT id, login, password FROM accounts;
```

On va avoir tout un langage de sélection. Dans le cadre d'une authentification, la requête va devoir évoluer, par exemple :

```
SELECT id, login, password FROM accounts WHERE login = "admin" AND password =
"admin";
```

Dans l'idée, un code source de traitement très simple (et donc vulnérable) va ressembler à :

```
$login = $_REQUEST["login"];
$password = $_REQUEST["password"];

$query = "SELECT id, login, password FROM accounts WHERE login = '" . $login . "'
AND password = '" . $password . "';";
```

> Note : Ce qu'il est très important de noter, c'est que la requête est forgée dans le code source et a pour vocation d'être envoyée complète à la base de données (sans savoir ce qui provient de la requête de base ou de l'entrée utilisateur).

Le fait de rallonger un peu le mot de passe avec des instructions SQL telles que : `admin' or 'a' = 'a` va permettre de forcer la requête à contenir sur conditions vraie pour toutes les lignes de la base de données et donc travailler comme si les conditions n'étaient pas la.

Il est d'ailleurs possible dans la plupart des cas de procéder à du "Query Chaining" et simplement même envoyer une deuxième requête SQL complètement arbitraire en les séparant simplement par un `;`.

De manière générale, nous allons alimenter nos injections avec du contenu SQL arbitraire. Par exemple, il est courant d'ajouter des commentaires (`-- -`) pour que le reste de la requête soit ignoré.

Types d'injection

Union

Une injection SQL Union est un type d'attaque par injection SQL in band qui utilise l'operateur UNION SQL pour extraire aisément les informations requises de la base de données ciblée.

L'opérateur UNION permet à l'utilisateur de retirer des données simultanément sur de multiples tableaux constitués du même nombre de colonnes et types de données identiques.

> Note : Il est possible de concaténer des colonnes pour en réduire le nombre ou encore de créer de fausses colonnes pour en ajouter et atteindre le bon nombre pour l'UNION.

Exemple d'une requête UNION :

```
SELECT id_client AS id, nom, email FROM clients UNION SELECT id_prospect AS id,
nom, email FROM prospects;
```

Error Based

Un autre type d'attaque par injection SQL in-band est l'injection SQL error-based.

Cette technique permet à un attaquant de pondérer les messages d'erreur retournés par les serveurs pour obtenir des informations sur la structure du serveur ciblé. L'objectif en tant qu'attaquant sera donc de volontairement saisir des éléments invalides pour déclencher des messages d'erreur.

> Note : Générer une erreur est d'ailleurs la manière la plus simple de détecter une vulnérabilité d'injection SQL. Une simple `'` unique qui génère une erreur 500 sur le serveur indique que l'on a réussi à injecter du contenu.

Blind Time Based

Une injection SQL temporisée est une technique qui repose sur l'envoi d'une demande SQL à une base de données pour évaluer les résultats d'une demande.

La demande en question force la base de données à patienter avant de retourner un résultat, qui sera soit TRUE soit FALSE.

> Note : Cette attaque repose sur la fonction SLEEP() du langage SQL.

L'inconvénient principal de ce type d'injection SQL est le temps qu'elle prend à réaliser car le pirate doit énumérer la base de données caractère par caractère (même si il est possible d'optimiser les requêtes).

Blind Boolean Based

Un injection SQL blind boolean-based est une technique d'injection inférentielle très similaire à l'injection SQL blind time-based par laquelle les attaquants envoient une commande SQL à la fois pour tenter d'énumérer la base de données.

En fonction de la réponse qu’ils reçoivent, ils vérifient si leur charge utile a été envoyée correctement. Mais plutôt que de temporiser leurs commandes, ils combinent des expressions TRUE et FALSE.

Comme avec les injections SQL time-based, ces attaques peuvent être très lentes en particulier lorsqu’un attaquant cible une large base de données.

Exemple d'exploitation :

```
?id=1 AND 1=1 -- (vrai)
?id=1 AND 1=2 -- (faux)
```