

Injection SQL

WORKSHOP INFOSEC

By Alexandre Wagner

Le SQL, c'est quoi ?

Le SQL (***Structured Query Language*** ou ***langage de requête structurée***)
Est langage informatique c'est à échangé avec une base de données
(relationnel)

En informatique, une **base de données relationnelle** est une
base de données où l'information est organisée dans des
tableaux à deux dimensions appelés des *relations* ou *tables*

Composer d'un SGBD (Système de gestion de base de données
Ou en anglais *DBMS* pour *database management system*) permet
d'inscrire, de retrouver, de modifier, de trier, de transformer ou
d'imprimer les informations de la base de données.



La place de l'injection SQL dans la sécurité des sites internet.

The Top 10 OWASP vulnerabilities in 2020 are:

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access control
- Security misconfigurations
- Cross Site Scripting (XSS)
- Insecure Deserialization
- Using Components with known vulnerabilities
- Insufficient logging and monitoring

OWASP (The Open Web Application Security Project)
<https://owasp.org/>



La faille SQL injection (SQLi)

Une faille SQL injection (SQLi) est une vulnérabilité qui permet à un attaquant d'exécuter des commandes SQL arbitraires sur une base de données en injectant des commandes malveillantes dans des requêtes Web. Avec une faille SQLi, un attaquant peut :

- Lire des données sensibles comme des mots de passe, des informations personnelles, etc.
- Modifier les données en insérant, mettant à jour ou supprimant des enregistrements.
- Exécuter des commandes système pour prendre le contrôle du serveur web ou d'autres ressources réseau.
- Utiliser la base de données comme base pour des attaques ultérieures, par exemple en utilisant les informations de connexion obtenues pour accéder à d'autres systèmes ou en utilisant les informations de l'utilisateur obtenues pour envoyer des messages de phishing ciblés.

Il est très important de protéger vos applications contre les failles SQLi, en utilisant des techniques de validation et d'échappement appropriées et en mettant à jour régulièrement vos logiciels et vos composants tiers.



Exemples des commande SQL arbitraire valide


ORACLE



Exemples des commande SQL arbitraire valide

1. Injection de commandes pour lister les utilisateurs de la base de données :

```
' OR 1=1; --
```

 Copy code


Cette commande va ajouter une condition qui sera toujours vraie dans la requête SQL originale, ce qui signifie que tous les enregistrements seront sélectionnés. Le `--` à la fin de la commande sert à commenter tout ce qui suit, de sorte que si la requête SQL originale contenait des instructions supplémentaires, elles seront ignorées.

ORACLE



Exemples des commande SQL arbitraire valide

1. Injection de commandes pour lister les noms des tables :

 Copy code

```
' UNION SELECT table_name FROM information_schema.tables --
```


Cette commande va ajouter une instruction UNION à la requête SQL originale pour sélectionner les noms des tables de la base de données à partir du schéma information_schema. Cela va permettre à l'attaquant de connaître les tables existantes dans la base de données.

ORACLE



Exemples des commande SQL arbitraire valide

1. Injection de commandes pour lire les informations de l'utilisateur :

 Copy code

```
' UNION SELECT username, password FROM users --
```


Cette commande va ajouter une instruction UNION à la requête SQL originale pour sélectionner les informations de l'utilisateur (nom d'utilisateur et mot de passe) à partir de la table users . Cela va permettre à l'attaquant de connaître les informations d'authentification des utilisateurs.

ORACLE



Exemples des commande SQL arbitraire valide

1. Injection de commandes pour ajouter un utilisateur administrateur :

 Copy code

```
' ; INSERT INTO users (username, password, role) VALUES ('attacker', 'password',  
'admin'); --
```


Cette commande va ajouter une instruction SQL qui permet à l'attaquant d'insérer un nouvel utilisateur avec les informations d'authentification de son choix (username : 'attacker', password : 'password') et un role administrateur dans la table users.

ORACLE



Exemples des commande SQL arbitraire valide

1. Injection de commandes pour Modifier les données en insérant, mettant à jour ou supprimant des enregistrements :

 Copy code

```
' ; UPDATE orders SET status = 'shipped' WHERE id = 1; --
```

Cette commande va ajouter une instruction SQL qui permet à l'attaquant de mettre à jour les données dans la table orders en changeant le statut de la commande ayant l'id 1 à 'shipped'.

ORACLE



Exemple : MySQL

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| Database |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> USE Database;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

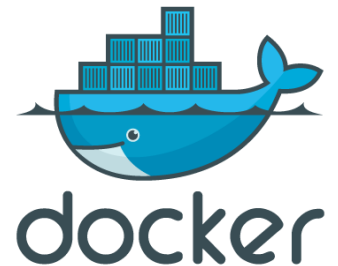
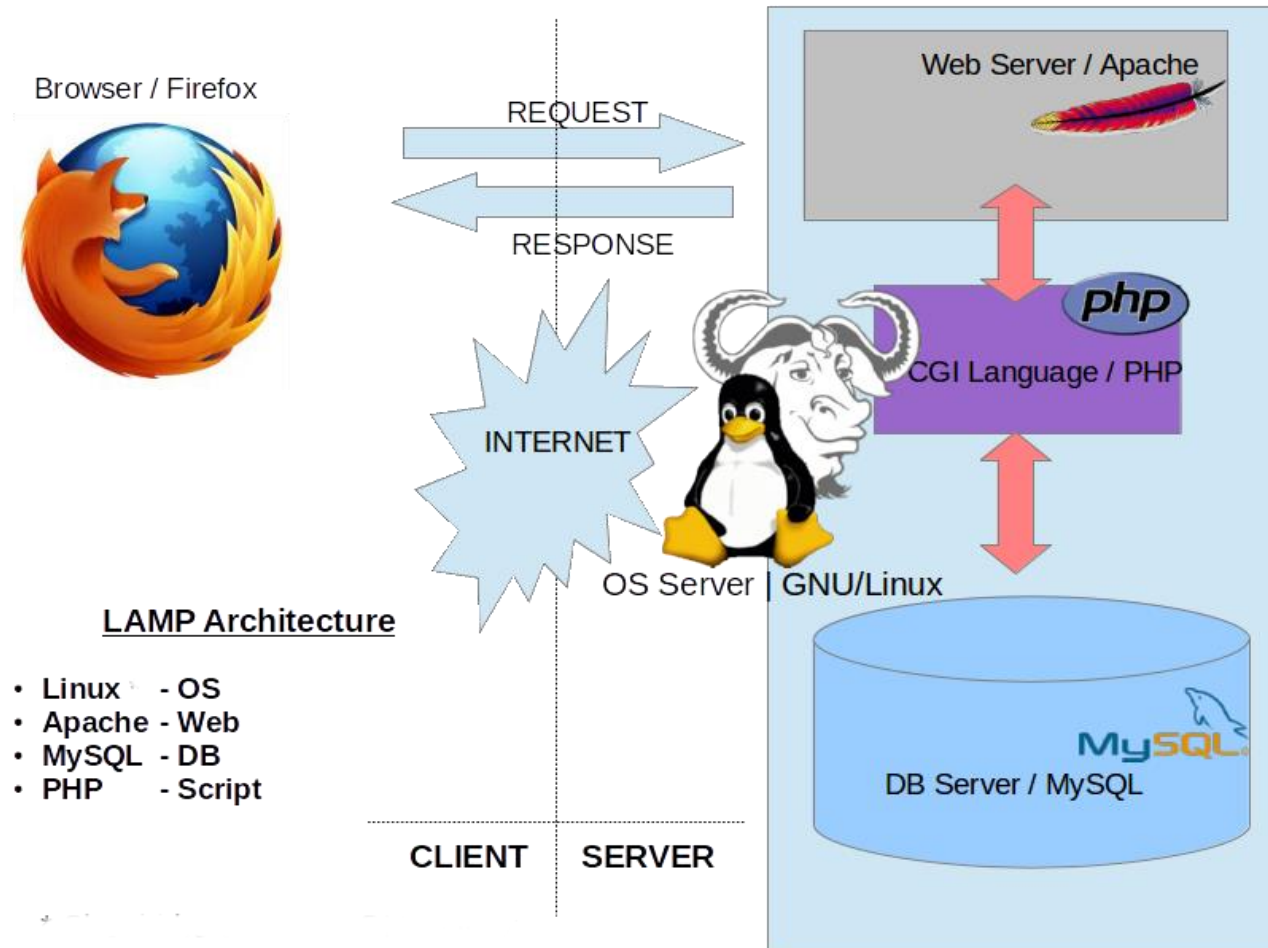
Database changed
mysql> SELECT * FROM User;
+-----+-----+-----+
| id | name | password |
+-----+-----+-----+
| 1 | admin | YWRtaW4= |
| 2 | user | MTIzNA== |
| 3 | John-the-retard | whatisapassword??? |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> 
```



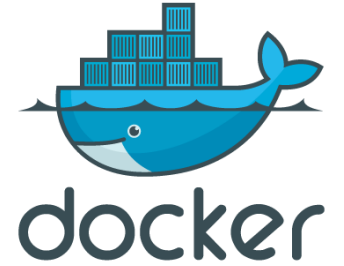
Petit Exercice :

Sous Docker avec un environnement LAMP :



Petit Exercice :

Intégrer avec l'environnement docker :



Le lien du repo : <https://github.com/wagnerwave/docker-lamp/>

Crée et lancer l'environnement :

```
~/docker-lamp$ docker-compose up --build
```

Rentré et intégrer dans l'environnement bash d'Apache

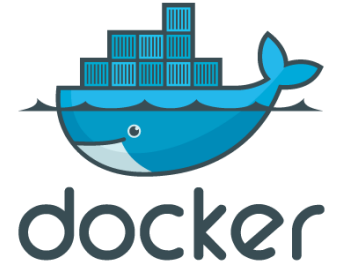
```
~/docker-lamp$ docker exec -it docker-lamp_www_1 bash
```

Rentré et intégrer avec l'environnement mysql :

```
~/docker-lamp$ docker exec -it docker-lamp_db_1 mysql -u root -p
```

Petit Exercice :

Objectif :

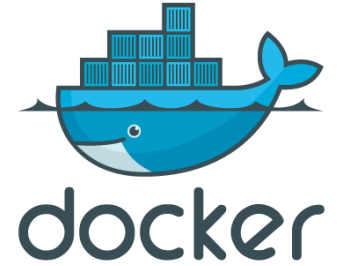


Injection SQL Workshop

- Open web browser to look at simple form at <http://localhost:8001>
- Try to understand why, this code contain a code failure, and try to make a injectin sql.

Petit Exercice :

Correction :



-> `SELECT * FROM User WHERE name='$username'`
-> `$username` étant égale à l'input user (`$_POST['username']`)
-> `SELECT * FROM User WHERE name="" OR 1=1 #`
-> le `#` met le reste de la requête en commentaire
-> dans le code PHP, on est connecté quand la condition est vraie [`' OR 1=1#`] retourneras toujours True.

Petit challenge ...

Find the good username:

Submit Query

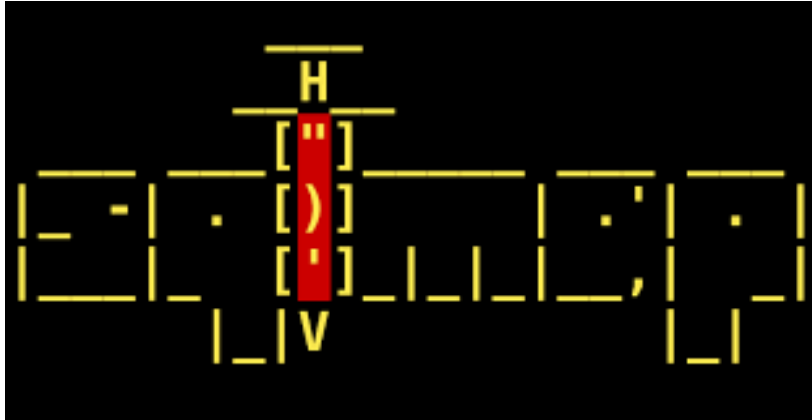
Comment corriger les failles d'injections SQL ?

- -> Préparer les requests SQL
- -> No trust mentality.
- -> toujours vérifier les retours SQL et les conditions associés.
- -> think like attackers
- -> mettre à jour les CMS et framework.

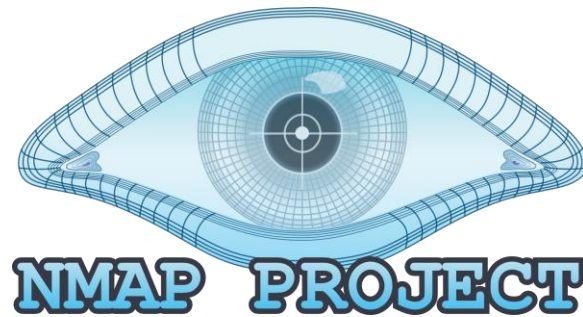


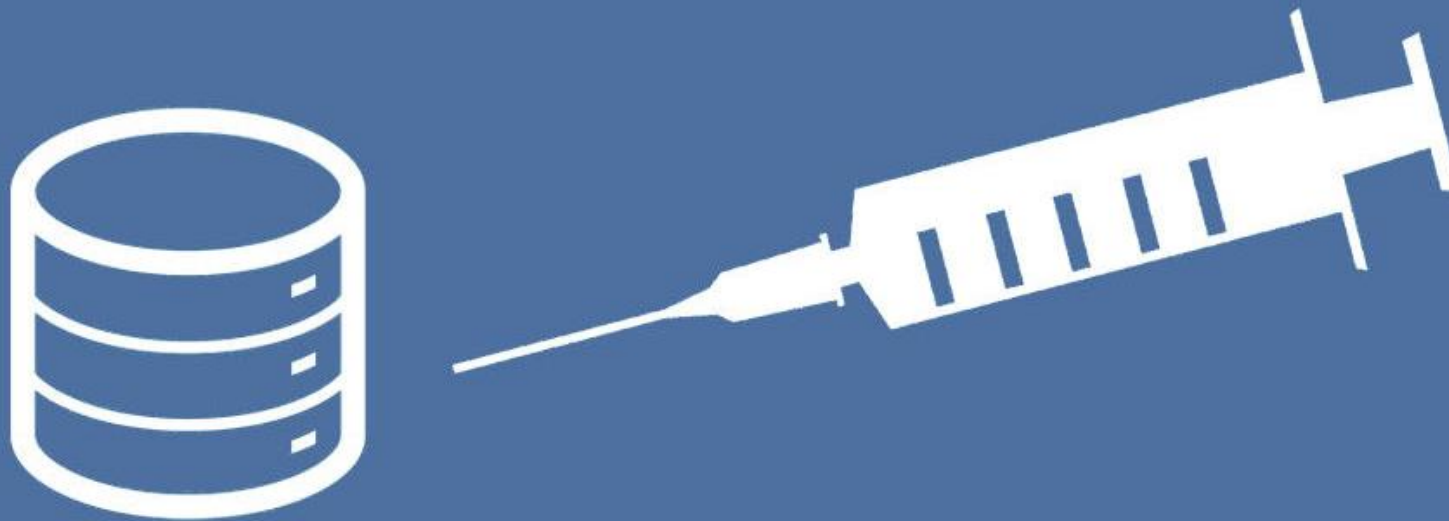
Les outils :

Sqlmap : RTFM (le -h
quoi).



Nmap scripts: Certains
scripts nmap.





Injection SQL

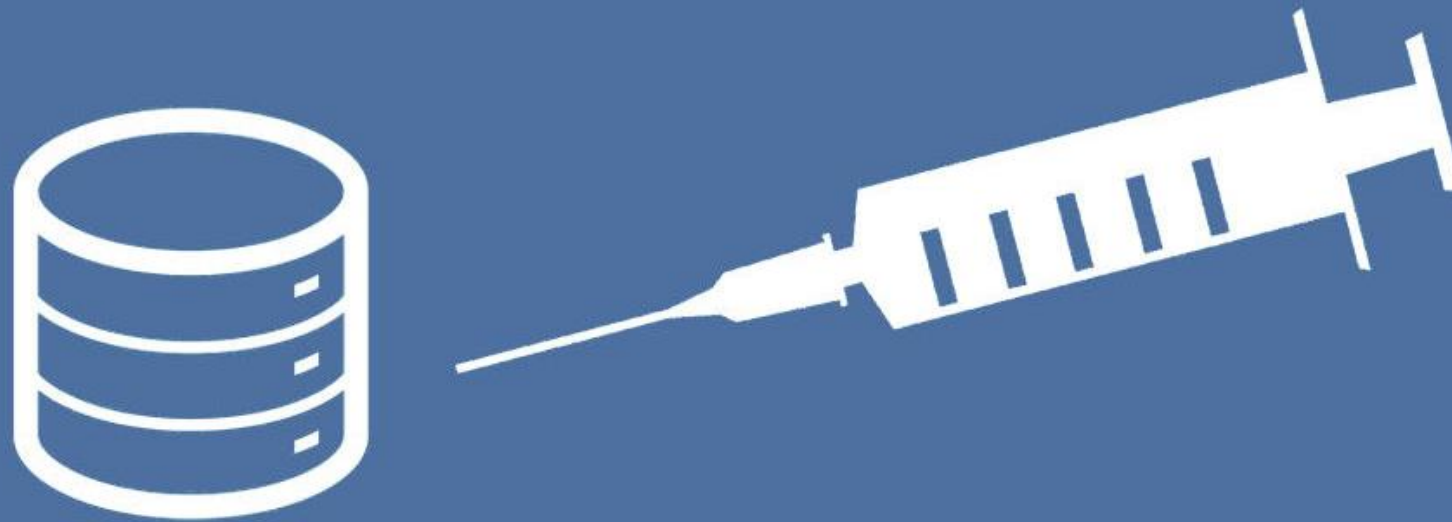
Exercice

By Alexandre Wagner

Exercices

✗	SQL Injection - Time based	2%	4374	45		ycam	😊	8	11 septembre 2015
✓	SQL Injection - String	7%	16880	30		g0uZ	😊	7	24 décembre 2012
✓	SQL Injection - Numérique	5%	11654	35		g0uZ	😊	7	24 décembre 2012
✗	SQL Injection - Lecture de fichiers	2%	4816	40		Arod	😊	7	19 octobre 2014
✗	SQL Injection - Insert	1%	2434	40		sambecks	😊	10	23 février 2015
✓	SQL Injection - Error	3%	5985	40		sambecks	😊	8	4 mars 2015
✓	SQL Injection - En aveugle	3%	6064	50		g0uZ	😊	10	27 février 2011
✗	SQL Injection - Contournement de filtres	1%	1791	80		sambecks	😊	5	21 juillet 2014
✓	SQL Injection - Authentification - GBK	3%	7614	30		dvor4x	😊	7	2 décembre 2015
✓	SQL Injection - Authentification	13%	36304	30		g0uZ	😊	10	27 février 2011
✗	SQL Truncation	2%	5368	35		Geluchat	😊	5	1er mai 2015





Injection SQL

POWERPOINT LINK : <https://wagnerwave.github.io/>

By Alexandre Wagner