💕**SQLi**??

**SQL language**

Structured Query Language is a back-end language to communicate with the database, but it’s not a database itself.

Examples of databases that use SQL language:

* Oracle
* SQL Server
* MySQL
* PostgreeSQL

**Starting SQL on Linux**

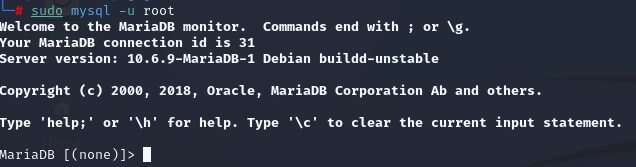
Starting mysql server:



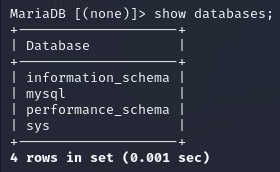
Starting mysql service:



Connecting to mysql and manage default databases:

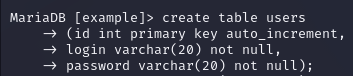


From here, we will only use mysql commands to communicate with the data bases.

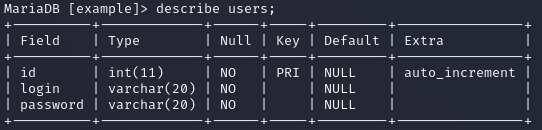


**A few commands examples:**

* **show databases;**
  + Shows available databases
* **select database();**
  + Shows in which database are we now
  + If it shows “null”, we don’t have any database selected
* **select user();**
  + Shows it which user are we logged on
* **select version();**
  + Shows our currently mysql version
* **CREATE DATABASE example();**
  + Creates a database named example
  + The command doesn’t need to be in caps lock
* **USE example;**
  + Select the database named example to be used
* **SHOW TABLES;**
  + Show tables on the select database
* **CREATE TABLE users**
  + Create a table named users
  + As we didn’t use **;** we’re still in the command, we can use this to create the fields in the table:



* **DESCRIBE users;**
  + Describers the information from a table named users



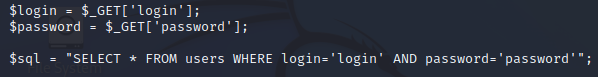
* **SELECT ID FROM users;**
  + Select the field id from table users to check for information
* **INSERT INTO users VALUES (‘’,’’,’’);**
  + Insert values into the table users, the amount of values to be insert must be the same as the fields in the table (in this case,three: id, login, password)



* **SELECT \* FROM users;**
  + Selects everything from table users
* **SELECT \* FROM users where id=”1”;**
  + Select everything from table users, where id = 1
* **SELECT \* FROM users order by id;**
  + Select everything from users, order by id
* **DELETE FROM users where id=”7”;**
  + Delete from table users where id = 7
* **SELECT @@version;**
  + Shows the database version
* **SELECT LOAD\_FILE (‘var/www/html/index.html’);**
  + Loads the file index.html
* **SELECT SLEEP(2);**
  + Waits 2 seconds to return
  + You can check in a pentest with sleep, if the database takes too long to return, it means it is accepting our commands
* **SELECT CHAR(65);**
  + Brings us the letter, converting table ASCII
* **SELECT LENGTH (test);**
  + Shows the size of the string test
* **SELECT CONCAT (login, ‘:’, password) FROM users;**
  + Select login and password separated with :
* **SELECT \* FROM table1 UNION SELECT \* FROM table2;**
  + Select the values both tables have in common
* **SELECT id FROM users LIMIT 2,4**
  + Select id from table users, starting by the second line (2) row count 4

**SQLi - SQL Injection**

SQL injection is a vulnerability in which the attacker can read, access and inject data on a database, without proper authorization. This vulnerability exists in applications that don’t validate the input, like in the follow example:



In this script example, the input is not being validated, everything the user types will directly access the database. Which means everything the user types will be executed, including if it is a malicious action.

**or 1=1;**

* This command is used by attackers and pentesters to test for SQLi vulnerabilities, you can put a statement like **select \* from users where login = ‘test’ or 1=1;** in this query, you’re asking to select everything from the table users where login = test, OR, 1=1, as 1=1 is always a true statement, if the application is vulnerable, you will be able to see the result of the query (information from the database)
* You can also put **#** in the end of the command, like: **or 1=1;#** so everything after the # is a commentary, which means it won’t be executed

**SQLi Example**

In this SQL injection attack example, we used a [vulnerable web application](http://testphp.vulnweb.com/artists.php?artist=1) made specifically for tests / studies purposes.

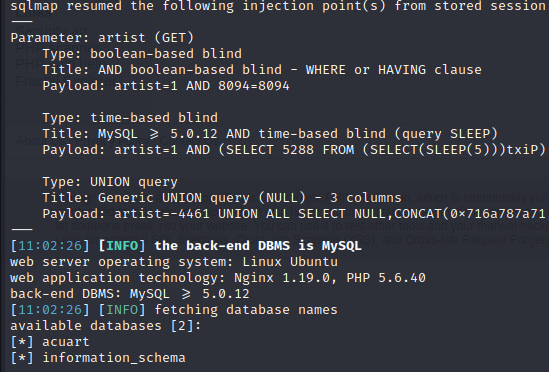
First thing you should do, is access the web site and look for **parameter entrance points**, like this one:



Next, we are going to use sqlmap, a tool to help us find SQLi vulnerabilities.



**-u** to specify the URL and **–dbs** to fetch database names.



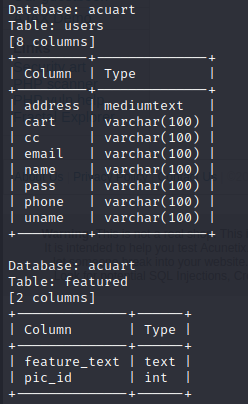
Sqlmap successfully found two databases: acuart and information\_schema, along with other information.

Now, we’re going to ask sqlmap to show us acuart database and show its columns.



**-D** for the selected database, **–columns** to bring its columns.

On the next image, we can see the result for the previous query, a couple of tables found on the selected database (acuart).



Next, we are going to use the **dump** command to view information on this database.



**-T** to select the table users in this case, **-C** to select the database column and **–dump** to “dump” the information.

With this, we were able to find one password stored on the column “pass”, like shown on the next image.

