Project title:

SQL INJECTION ATTACK ON TODAY'S TECHNOLOGY USINNG MYSQL DATABASE QUEIRIES

Prepared by **Guided by**

Pratiksha Deshmukh **Zakir Hussain**

Index

1 . Introduction

SQL Injection (SQLi) is one of the most prevalent and severe vulnerabilities in modern web applications. It occurs when an attacker injects malicious SQL statements into a database query via an input field, URL, or cookie to manipulate a database. SQLi can lead to data breaches, unauthorized access, or even the deletion of critical data. In this project, we simulate a real-world SQL Injection scenario using MySQL queries across different databases to understand how this attack works and explore strategies for prevention.

This report presents the structure of the databases used, the queries implemented, and the methods to detect and mitigate SQL Injection attacks in web applications.

2. project objective

The primary objectives of this project are as follows:

1. Simulate SQL Injection attacks on a MySQL database system.
2. Demonstrate how an attacker can manipulate user input to execute malicious SQL code.
3. Analyze the impact of SQL Injection on user authentication, product management, and employee information systems.
4. Propose and implement techniques to prevent SQL Injection attacks, ensuring database security.

3. Literature Review

Definition of SQL Injection

SQL Injection is a code injection technique where an attacker inserts malicious SQL code into a query. It takes advantage of improper handling of user input in web applications, bypassing authentication mechanisms and exposing sensitive data.

Types of SQL Injection Attacks

1. **Classic SQL Injection:** The attacker directly manipulates SQL queries through input fields.
2. **Blind SQL Injection:** The attacker doesn't receive direct feedback but infers information based on responses (e.g., true/false).
3. **Time-Based SQL Injection:** The attacker uses delays in query execution to gather information.
4. **Out-of-Band SQL Injection:** Data is extracted via a separate communication channel (e.g., HTTP).

Consequences of SQL Injection Attacks

The consequences of SQLi include:

- Data breaches, exposing sensitive information like usernames, passwords, or credit card numbers.
- Unauthorized access to administrative controls or other user accounts.
- Data manipulation, such as deletion, alteration, or adding fraudulent data.

4.Project Scope

This project focuses on simulating SQL Injection attacks on three key databases:

1. **User Authentication Database:** To simulate SQLi in login and user roles management.
2. **Product Information Database:** To understand how SQLi affects product listings, reviews, and categories.
3. **Employee Information Database:** To demonstrate SQLi attacks that target employee data.

Each database is structured with several tables to emulate a real-world scenario, making the attack simulation more relevant. The project also includes preventive measures like input validation and prepared statements to mitigate SQLi risks.

4. Database Design

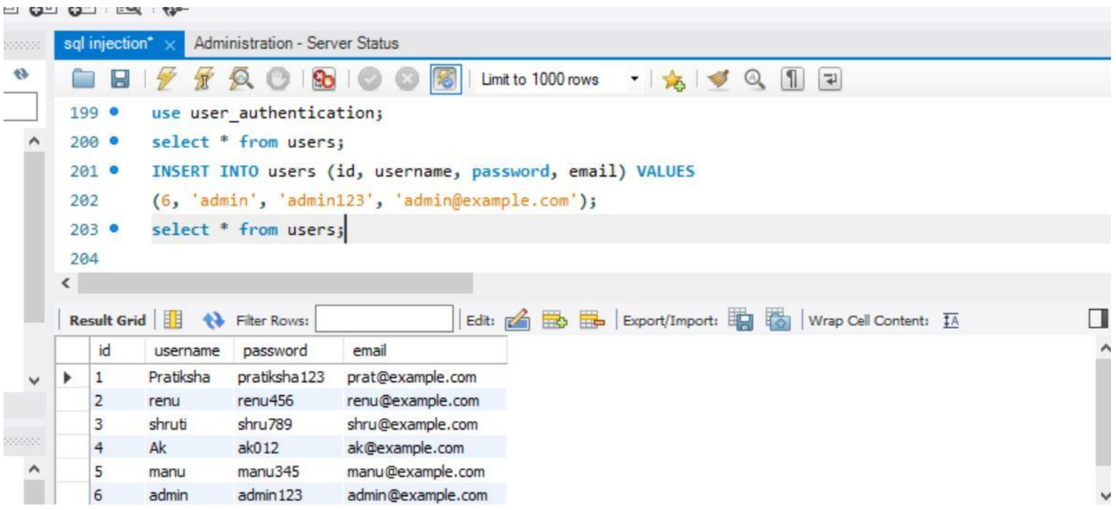
Database 1: User Authentication

Tables:

1. **users** (id, username, password, email)
2. **user_roles** (id, role_name, description)
3. **login_attempts** (id, username, attempt_date, success)
4. **user_sessions** (id, user_id, session_start, session_end)
5. **password_resets** (id, user_id, reset_date, reset_token)

Example Data Insertions for users table:

- (1, "admin", "password123", "admin@example.com")
- (2, "user1", "password456", "user1@example.com")



```
use user_authentication;
select * from users;
INSERT INTO users (id, username, password, email) VALUES
(6, 'admin', 'admin123', 'admin@example.com');
select * from users;
```

id	username	password	email
1	Pratiksha	pratiksha123	prat@example.com
2	renu	renu456	renu@example.com
3	shruti	shru789	shru@example.com
4	Ak	ak012	ak@example.com
5	manu	manu345	manu@example.com
6	admin	admin123	admin@example.com

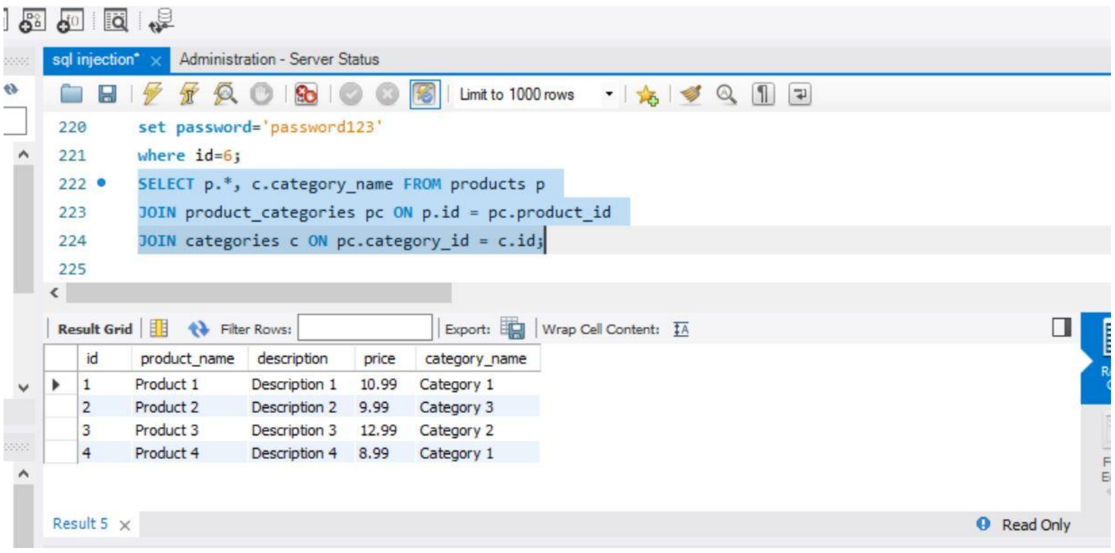
Database 2: Product Information

Tables:

1. **products** (id, product_name, description, price)
2. **categories** (id, category_name, description)
3. **product_categories** (id, product_id, category_id)
4. **product_reviews** (id, product_id, review_date, rating)
5. **product_images** (id, product_id, image_url)

Example Data Insertions for products table:

- (1, "Product 1", "Description 1", 10.99)
- (2, "Product 2", "Description 2", 9.99)



```
set password='password123'
where id=6;
SELECT p.*, c.category_name FROM products p
JOIN product_categories pc ON p.id = pc.product_id
JOIN categories c ON pc.category_id = c.id;
```

id	product_name	description	price	category_name
1	Product 1	Description 1	10.99	Category 1
2	Product 2	Description 2	9.99	Category 3
3	Product 3	Description 3	12.99	Category 2
4	Product 4	Description 4	8.99	Category 1

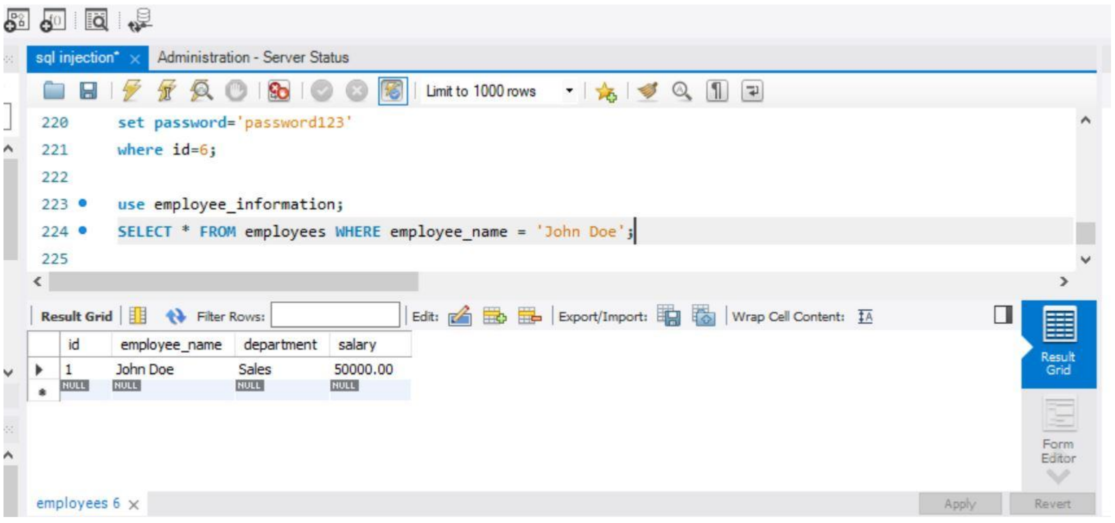
Database 3: Employee Information

Tables:

1. **employees** (id, employee_name, department, salary)
2. **departments** (id, department_name, description)
3. **employee_roles** (id, role_name, description)

Example Data Insertions for employees table:

- (1, "John Doe", "Sales", 50000)
- (2, "Jane Doe", "Marketing", 60000)



```
set password='password123'
where id=6;

use employee_information;
SELECT * FROM employees WHERE employee_name = 'John Doe';
```

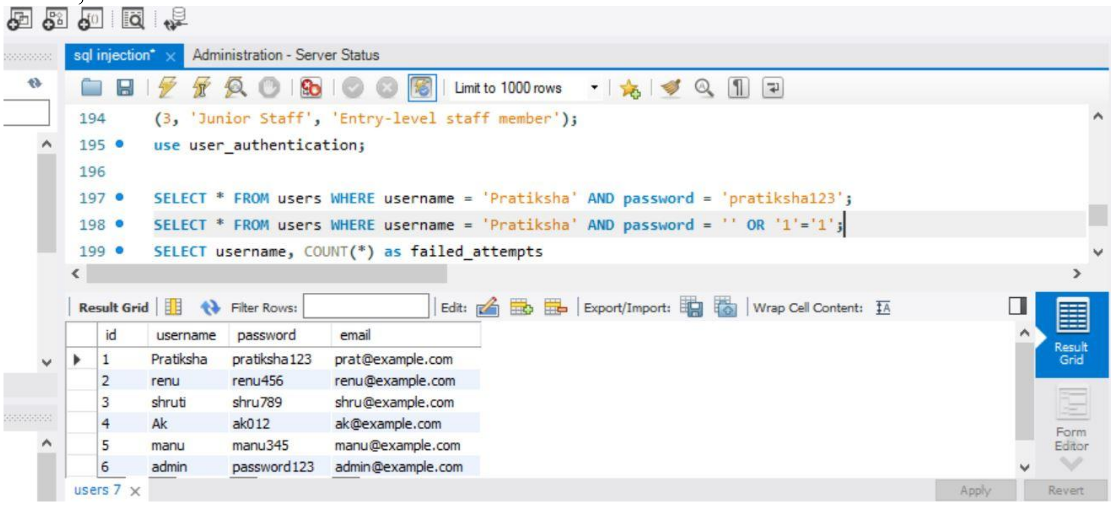
id	employee_name	department	salary
1	John Doe	Sales	50000.00
NULL	NULL	NULL	NULL

4. SQL Injection Attack Simulations

SQL Injection in User Authentication:

Example query:

SELECT * FROM users WHERE username = 'Pratiksha' AND password = '' OR '1'='1';

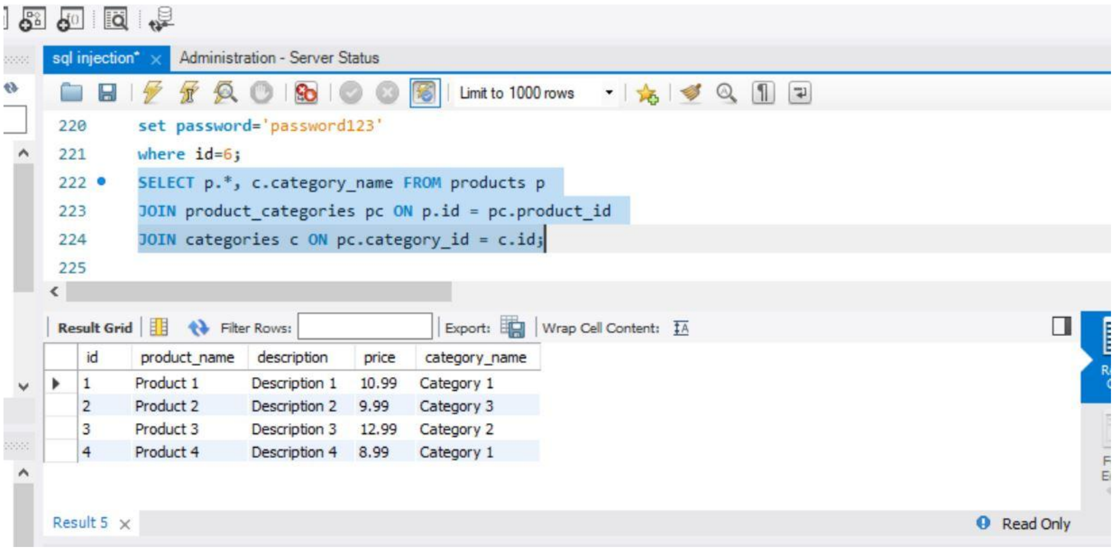


```
(3, 'Junior Staff', 'Entry-level staff member');
use user_authentication;

SELECT * FROM users WHERE username = 'Pratiksha' AND password = 'pratiksha123';
SELECT * FROM users WHERE username = 'Pratiksha' AND password = '' OR '1'='1';
SELECT username, COUNT(*) as failed_attempts
```

id	username	password	email
1	Pratiksha	pratiksha123	prat@example.com
2	renu	renu456	renu@example.com
3	shruti	shru789	shru@example.com
4	Ak	ak012	ak@example.com
5	manu	manu345	manu@example.com
6	admin	password123	admin@example.com

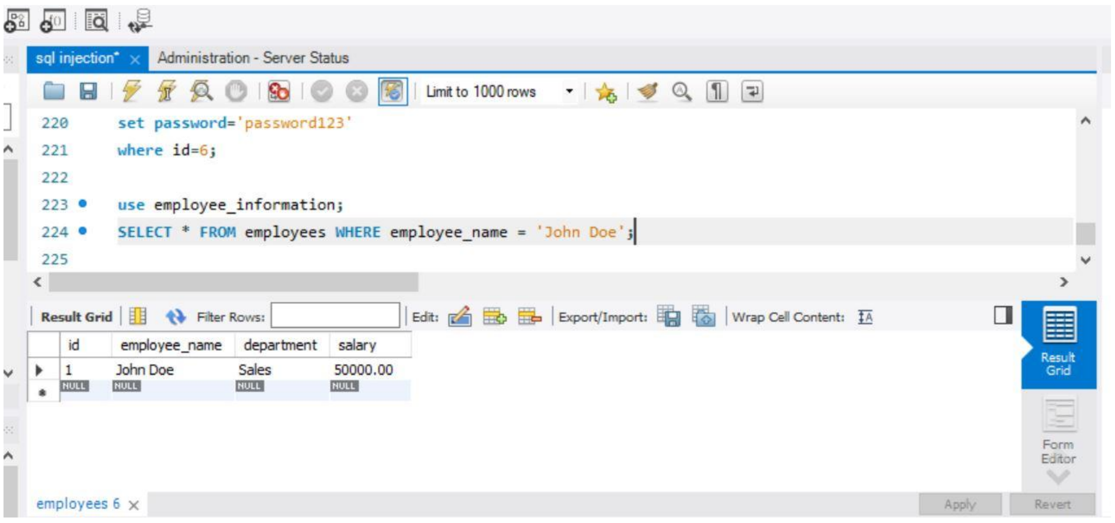
SELECT p.*, c.category_name FROM products p
JOIN product_categories pc ON p.id = pc.product_id
JOIN categories c ON pc.category_id = c.id;



```
set password='password123'
where id=6;
SELECT p.*, c.category_name FROM products p
JOIN product_categories pc ON p.id = pc.product_id
JOIN categories c ON pc.category_id = c.id;
```

id	product_name	description	price	category_name
1	Product 1	Description 1	10.99	Category 1
2	Product 2	Description 2	9.99	Category 3
3	Product 3	Description 3	12.99	Category 2
4	Product 4	Description 4	8.99	Category 1

SQL Injection in Employee Information:



```
set password='password123'
where id=6;

use employee_information;
SELECT * FROM employees WHERE employee_name = 'John Doe';
```

id	employee_name	department	salary
1	John Doe	Sales	50000.00
NULL	NULL	NULL	NULL

Conclusion

SQL Injection remains a significant threat to web applications, particularly those that handle sensitive data. Through this project, we were able to highlight the dangers of SQLi and demonstrate how simple coding practices like prepared statements and input validation can prevent such attacks. By adopting these security measures, developers can greatly reduce the risk of SQL Injection, safeguarding both their systems and users' data