Step 1: Install MySQL Server

sudo apt update
sudo apt install mysql-server -y

```
student@student-VMware-Virtual-Platform:~$ sudo apt update
sudo apt install mysql-server -y
[sudo] password for student:
Hit:1 http://in.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:5 https://packages.microsoft.com/repos/code stable InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libaio1t64 libcgi-fast-perl libcgi-pm-perl libfcgi-bin libfcgi-perl
  libfcgi0t64 libhtml-template-perl libmecab2 libprotobuf-lite32t64
  mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-server-8.0 mysql-server-core-8.0
Suggested packages:
  libipc-sharedcache-perl tinyca
The following NEW packages will be installed:
  libaio1t64 libcgi-fast-perl libcgi-pm-perl libfcgi-bin libfcgi-perl
  libfcgi0t64 libhtml-template-perl libmecab2 libprotobuf-lite32t64
  mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-server mysql-server-8.0 mysql-server-core-8.0
0 upgraded, 17 newly installed, 0 to remove and 3 not upgraded.
Need to get 29.2 MB of archives.
```

Step 2: Secure MySQL Installation

sudo mysql_secure_installation

- Set root password
- Remove anonymous users
- Disallow remote root login
- Remove test DB
- Reload privilege tables

```
student@student-VMware-Virtual-Platform:~$ sudo mysql_secure_installation

Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD COMPONENT can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD component?

Press y|Y for Yes, any other key for No: Y

There are three levels of password validation policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary                 file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 0

Skipping password set for root as authentication with auth_socket is used by default.
If you would like to use password authentication instead, this can be done with the "ALTER_USER" command.
See https://dev.mysql.com/doc/refman/8.0/en/alter-user.html#alter-user-password-management for more information.

By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
```

Step 3: Login and Create a Secure User

sudo mysql -u root -p

CREATE USER 'secuser'@'localhost' IDENTIFIED BY 'StrongPassword!';
GRANT SELECT, INSERT ON testdb.* TO 'secuser'@'localhost';
FLUSH PRIVILEGES;

```
student@student-VMware-Virtual-Platform:~$ sudo mysql -u root -p

CREATE USER 'secuser'@'localhost' IDENTIFIED BY 'StrongPassword!';
GRANT SELECT, INSERT ON testdb.* TO 'secuser'@'localhost';
FLUSH PRIVILEGES;
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.41-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

Step 4: Enable Auditing in MySQL

For MariaDB or MySQL 5.7+, use:

sudo apt install libmysqlclient-dev

Enable logging:

sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

Add:
general_log = 1
general_log_file = /var/log/mysql/mysql.log

Restart MySQL:

sudo systemctl restart mysql

Check log:

sudo tail -f /var/log/mysql/mysql.log

```
student@student-VMware-Virtual-Platform:~$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
student@student-VMware-Virtual-Platform:~$ sudo systemctl restart mysql
sudo tail -f /var/log/mysql/mysql.log
\
Time                 Id Command    Argument
2025-04-05T11:07:57.497224Z         0 Execute   CREATE TABLE performance_schema.innodb_redo_log_files(
`FILE_ID` BIGINT NOT NULL COMMENT 'Id of the file.',
`FILE_NAME` VARCHAR(2000) NOT NULL COMMENT 'Path to the file.',
`START_LSN` BIGINT NOT NULL COMMENT 'LSN of the first block in the file.',
`END_LSN` BIGINT NOT NULL COMMENT 'LSN after the last block in the file.',
`SIZE_IN_BYTES` BIGINT NOT NULL COMMENT 'Size of the file (in bytes).',
`IS_FULL` TINYINT NOT NULL COMMENT '1 iff file has no free space inside.',
`CONSUMER_LEVEL` INT NOT NULL COMMENT 'All redo log consumers registered on smaller levels than this value, have already consumed this file.'
)engine = 'performance_schema'
```

Step 5: Test SQL Injection Prevention

Create a Database and Table

Login to MySQL: mysql -u root -p

In the MySQL shell:

```
CREATE DATABASE testdb;
USE testdb;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    password VARCHAR(50)
);

INSERT INTO users (username, password) VALUES ('admin', 'admin123'),
('user1', 'pass123');

EXIT;
```

Create Flask App Files

Create a project folder and move into it:

```
mkdir sql_injection_demo
cd sql_injection_demo
```

Create a file: app.py

Add Python Code

```
from flask import Flask, request
import mysql.connector

app = Flask(__name__)

# Connect to database
def get_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="your_mysql_password",
        database="testdb"
    )

# Vulnerable route
@app.route('/login-vulnerable', methods=['GET', 'POST'])
def login_vulnerable():
    if request.method == 'POST':
        username = request.form['username']

        query = f"SELECT * FROM users WHERE username =
'{username}'"
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute(query)
        result = cursor.fetchall()
        conn.close()
        return str(result) if result else "No user found"
    return '''
        <form method="post">
            Username: <input name="username"><br>
            <input type="submit">
        </form>
    '''

# Safe route using parameterized query
@app.route('/login-safe', methods=['GET', 'POST'])
def login_safe():
    if request.method == 'POST':
        username = request.form['username']
        conn = get_connection()
```

```
        cursor = conn.cursor()
        query = "SELECT * FROM users WHERE username = %s"
        cursor.execute(query, (username,))
        result = cursor.fetchall()
        conn.close()
        return str(result) if result else "No user found"
    return '''
        <form method="post">
            Username: <input name="username"><br>
            <input type="submit">
        </form>
    '''

if __name__ == '__main__':
    app.run(debug=True)
```

Replace "your_mysql_password" with your actual MySQL root password.

Run the Flask App python3 app.py

Open browser and go to:

http://localhost:5000/login-vulnerable

http://localhost:5000/login-safe

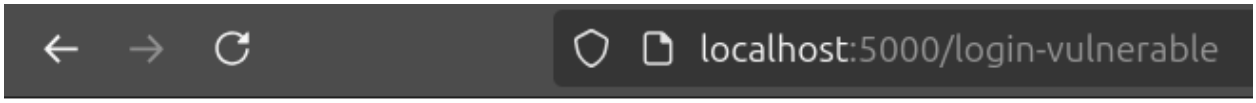
Test SQL Injection
Try entering this in the vulnerable form: ' OR '1'='1

It should return all users — showing it's vulnerable.

Try the same in the safe version, and it should block the injection.

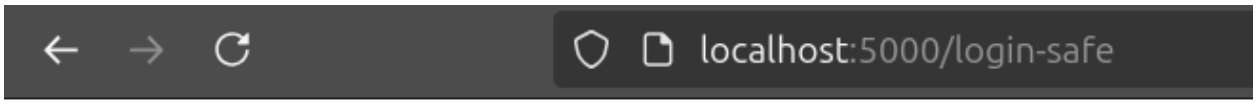
```
student@student-VMware-Virtual-Platform:~$ mkdir sql_injection_demo
cd sql_injection_demo
touch app.py
student@student-VMware-Virtual-Platform:~/sql_injection_demo$
```

Vulnerable Site attack:



[(1, 'admin', 'admin123'), (2, 'user1', 'pass123')]

Safe Site Attack:



No user found