Security Testing

1. **SQL Injection (Logical Vulnerability)**:
   - **Issue**: The application constructs SQL queries using string concatenation, which is vulnerable to SQL injection.
   - **Test**: Try injecting SQL commands via user inputs, such as:
     - email = 'a@a.com' OR 1=1 --
     - userName = 'admin' OR 1=1 --
   - **Risk**: High. An attacker can bypass authentication and extract sensitive data.
   - **Recommendation**: Use parameterized queries (e.g., dbCursor.execute('SELECT * FROM users WHERE email = ?', (email,))) to prevent SQL injection.
2. **Password Handling (Security Vulnerability)**:
   - **Issue**: Passwords are stored in plain text and transmitted over HTTP.
   - **Test**: Attempt login with known credentials and check if passwords are being exposed or stored in plain text.
   - **Risk**: High. Storing passwords in plain text can lead to data breaches.
   - **Recommendation**: Hash passwords using algorithms like bcrypt, Argon2, or PBKDF2 before storing them in the database. Use HTTPS for secure transmission.
3. **JWT Token Weakness (Security Vulnerability)**:
   - **Issue**: The JWT is signed using a static secret key ('123456'), which is weak and easily guessable.
   - **Test**: Decode the JWT to verify if the payload is accessible without signature verification.
   - **Risk**: Medium. An attacker can forge a JWT and impersonate a user.
   - **Recommendation**: Use a stronger secret key and enable signature verification. Rotate keys periodically.
4. **Token Bypass (Logical Vulnerability)**:
   - **Issue**: The /update_info endpoint uses a JWT for authentication but decodes it without verifying its signature (verify_signature=False).
   - **Test**: Use a forged token or a token without proper validation.
   - **Risk**: High. Attackers can perform unauthorized actions using a forged token.
   - **Recommendation**: Always verify JWT signatures using a proper secret key and don't disable signature verification.
5. **Insecure Deserialization (Security Vulnerability)**:
   - **Issue**: There is no check for token tampering or expiration in JWT tokens.
   - **Test**: Manipulate the token and try to access protected routes with tampered data.
   - **Risk**: Medium. Attackers may alter the JWT to gain unauthorized access.
   - **Recommendation**: Use JWT expiration and verify the integrity of the token.

6. **Input Validation (Security Vulnerability)**:
    - **Issue**: The application does not validate user inputs effectively for the /client_registeration and /client_login endpoints.
    - **Test**: Test with special characters in inputs, such as <script>, ', ", or non-UTF characters.
    - **Risk**: Medium. Lack of input sanitization can lead to Cross-Site Scripting (XSS) or other injection attacks.
    - **Recommendation**: Sanitize and validate all user inputs, especially those coming from user registration and login.

Logical Testing

1. **Duplicate Registration**:
    - **Test**: Attempt to register with an already existing email.
    - **Expected Outcome**: The system should return a message indicating that the email already exists ('Email already Exist').
    - **Risk**: Low, this is a logical flaw that should be handled properly.
2. **Invalid Data Handling**:
    - **Test**: Send incomplete data (e.g., missing fullName, userName, etc.) for registration.
    - **Expected Outcome**: The system should return an error message like 'Invalid Data'.
    - **Risk**: Low. Ensure that registration only proceeds if all required fields are provided.
3. **Login Logic (Username vs. Email)**:
    - **Test**: Attempt to login with only a userName or only an email, or both, and verify the response.
    - **Expected Outcome**: The login should succeed only if either the username or email matches, and the correct password is provided.
    - **Risk**: Medium. There is a potential for confusion between username and email input, so ensure the logic correctly handles both.
4. **Privilege Handling**:
    - **Test**: Check how the system handles the role (privillage) during registration and login. For example, verify that an admin role (privilege = 1) can perform restricted actions.
    - **Expected Outcome**: The system should correctly identify the role and handle permissions accordingly.
    - **Risk**: Medium. Ensure that roles and privileges are properly enforced.
5. **Successful Login Flow**:
    - **Test**: Try logging in with correct credentials for both email and username scenarios.

  - **Expected Outcome**: The system should generate a valid JWT token and return it to the user.
  - **Risk**: Low. This should work fine if the database and inputs are valid.

Bug Reporting (with Risk Scoring)

Bug	Description	Risk	Recommendation
SQL Injection	Vulnerable SQL queries are used with string concatenation.	High	Use parameterized queries to prevent SQL injection attacks.
Password Storage	Passwords are stored in plain text.	High	Hash passwords using strong algorithms before storing them in the database.
Weak JWT Key	The JWT secret is weak ('123456').	Medium	Use a stronger secret key and rotate it periodically.
Token Bypass	JWT tokens are decoded without signature verification.	High	Always verify the signature of JWT tokens.
Input Validation	User inputs are not properly sanitized or validated.	Medium	Sanitize inputs to prevent injection and XSS attacks.
Duplicate Registration	No checks for duplicate email during registration.	Low	Implement proper checks for duplicate data during registration.
Privilege Handling	Admin roles may not be enforced correctly.	Medium	Ensure proper role-based access control (RBAC) is in place.