**Logical and Security Testing Bug Report**

**S.no:** 1

**Bug Id:** SEC\_BUG\_001

**Bug Title**: SQL Injection in ‘/client\_registeration’ Endpoint

**Description:** The `/client\_registeration` endpoint is vulnerable to SQL injection due to un-sanitized user input in the `email` parameter. This allows attackers to manipulate the database query, potentially leading to data compromise.

**Steps to Reproduce:**

1. Send the following POST request to `/client\_registeration`:

- URL: <http://127.0.0.1:5000/client_registeration>

{

"fullName": "Test",

"userName": "testuser",

**"email": "' OR '1'='1",**

"password": "password123",

"phone": "1234567890"

}

2. Observe the server response indicating successful registration despite the malicious input.

**Actual Behavior:**

The server registers the user and stores the payload in the database, demonstrating successful SQL injection.

**Expected Behavior:**

The server should reject the input as invalid and respond with an error like: {"msg": "Invalid Data"}

**Impact**

Severity: Critical

Likelihood: High

Impact: Major

Attackers can exploit this to bypass authentication, access sensitive data, or modify database entries.

**Risk Scoring**

CVSS v3.1 Score: 9.8 (Critical)

Attack Vector: Network

Attack Complexity: Low

Privileges Required: None

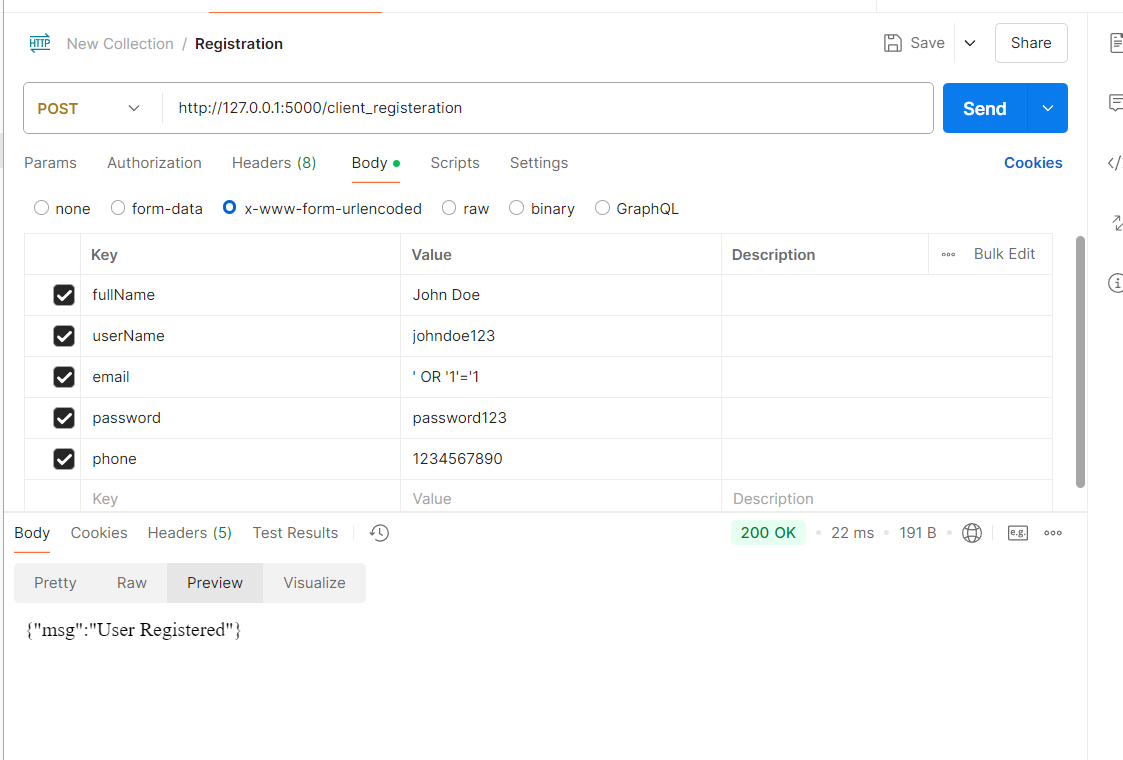
User Interaction: None

Confidentiality Impact: High

Integrity Impact: High

Availability Impact: High

**Screenshot**:



**S.no:** 2

**Bug Id:** SEC\_BUG\_002

**Bug Title:** Login Allows Authentication with Wrong Username but Correct Email and Password

**Description**

The /client\_login endpoint authenticates a user based on the email, but the logic is flawed. It checks the email or username independently, but does not ensure both match in the database during login. This allows attackers to log in with a valid email and valid password, but a wrong username.

**Steps to Reproduce**

Register a new user using a valid email and username. For example:

{

"fullName": "Test User",

"userName": "testuser",

"email": "testuser@example.com",

"password": "password123",

"phone": "1234567890"

}

Try to log in with the correct password and correct email, but incorrect username. Example:

Username: wrongusername

Email: testuser@example.com

Password: password123

**Expected Behavior:** The login should fail with an error message such as "Incorrect username or password".

**Observed Behavior**: The login is successful, and a token is returned, even though the username is wrong.

**Impact**

Severity: High

Likelihood: Medium (depends on whether the attacker knows the email)

Impact: Major

An attacker who knows the email and password could successfully authenticate without needing the correct username, allowing unauthorized access to the system.

This vulnerability might lead to unauthorized account access or privilege escalation if the application grants access based on email alone.

**Risk Scoring**

CVSS v3.1 Score: 8.3 (High)

Attack Vector: Network

Attack Complexity: Low

Privileges Required: None

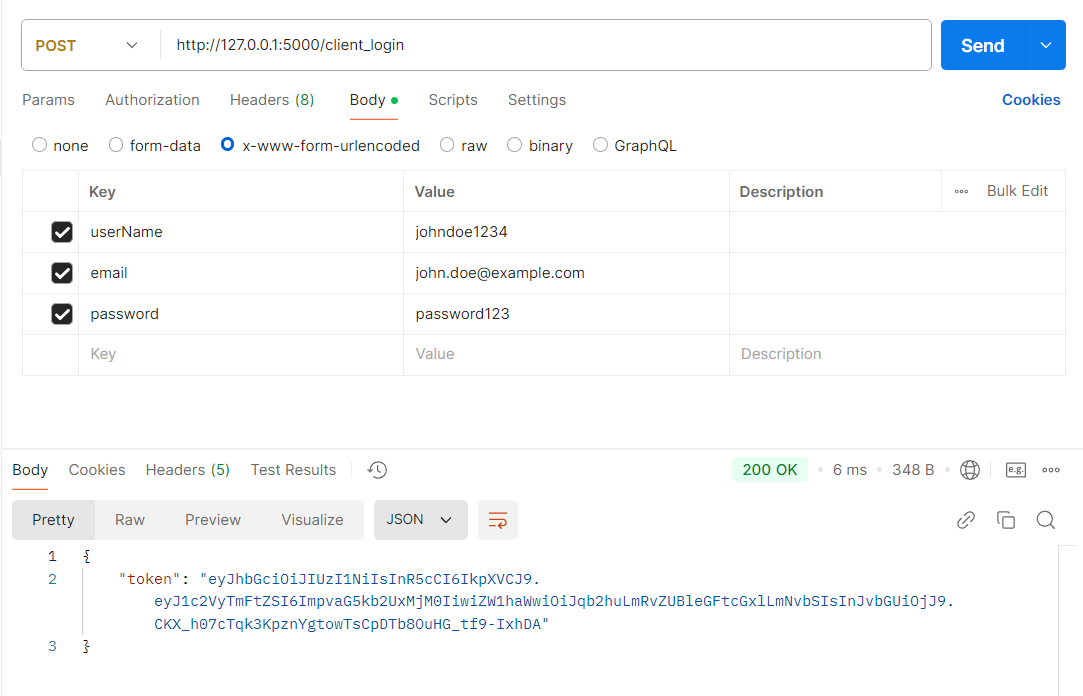
User Interaction: None

Confidentiality Impact: High (attackers can gain unauthorized access)

Integrity Impact: High (attackers can manipulate data)

Availability Impact: Low

**Screenshot**:



**S.no:** 3

**Bug Id:** SEC\_BUG\_003

**Bug Title:** Duplicate Username Registration with Different Emails but same username

**Description**

The application allows users to register with the same username but different email addresses. This creates a logical vulnerability because usernames should be unique across the system. Allowing multiple users with the same username but different emails can lead to confusion, identity mismanagement, and security concerns, especially if the username is used as a key identifier for user authentication.

**Steps to Reproduce**

Register the first user with a username testuser and an email user1@example.com:

json

{

"fullName": "Test User 1",

"userName": "testuser",

"email": "user1@example.com",

"password": "password123",

"phone": "1234567890"

}

Register another user with the same username testuser but a different email user2@example.com:

{

"fullName": "Test User 2",

"userName": "testuser",

"email": "user2@example.com",

"password": "password123",

"phone": "0987654321"

}

**Expected Behavior:** The second registration should fail with an error message such as "Username already exists".

**Observed Behavior:** The second user is successfully registered with the same username but a different email.

**Observed Behavior**

The system allows multiple users to register with the same username but different emails. This could result in:

Confusion when users try to log in, especially if the username is used as a unique identifier.

Potential privilege escalation or impersonation issues if the system relies on the username for user identification or access control.

**Expected Behavior**

The system should enforce that usernames are unique across all users.

If a user attempts to register with a username that already exists (regardless of the email), the registration should fail and return an appropriate error message:

{"msg": "Username already exists"}

**Impact**

Severity: High

Likelihood: Medium (depends on user behavior)

Impact: Major

This vulnerability can lead to account confusion and security risks if the username is used as a unique identifier for login or access control.

If an attacker is able to manipulate usernames and emails, they might impersonate legitimate users or cause data integrity issues.

**Risk Scoring**

CVSS v3.1 Score: 7.5 (High)

Attack Vector: Network

Attack Complexity: Low

Privileges Required: None

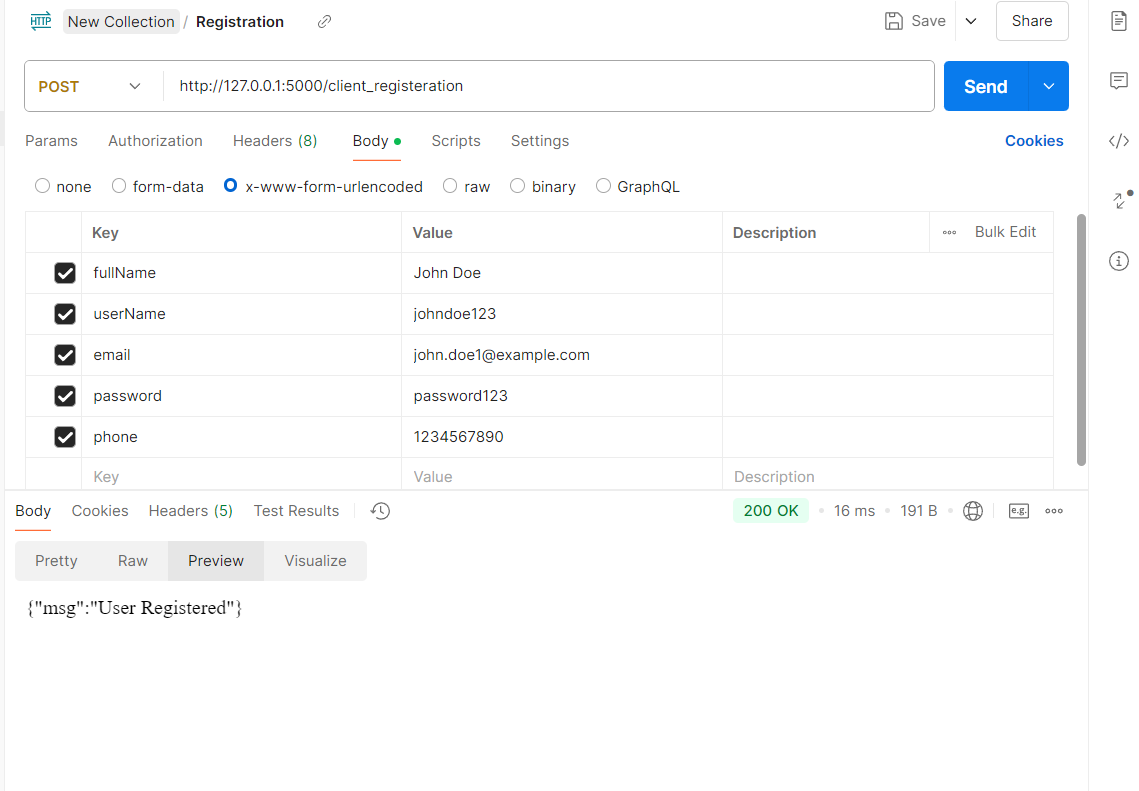
User Interaction: None

Confidentiality Impact: Moderate (attacker can impersonate or confuse users)

Integrity Impact: High (multiple users can have the same username)

Availability Impact: Low

**Screenshot**



**S.no:** 4

**Bug Id:** SEC\_BUG\_004

**Bug Title:** Weak Password Combinations Allowed During Registration

**Description**

The /client\_registeration endpoint permits the registration of user accounts with weak passwords (e.g., 12345, password, admin, etc.). This compromises account security, increasing the likelihood of unauthorized access and data breaches.

**Steps to Reproduce**

Use the /client\_registeration endpoint to register a new user with a weak password:

{

"fullName": "Test User",

"userName": "weakuser",

"email": "weakuser@example.com",

"password": "12345", # Weak password

"phone": "1234567890"

}

Observe that the registration is successful despite the weak password.

**Observed Behavior**

The application does not enforce any password strength requirements. Users can register with weak passwords such as 12345, password, or single-character passwords like a1234 etc..

**Expected Behavio**r

The application should enforce a strong password policy that: (Example)

Requires passwords to meet certain criteria (e.g., minimum length, complexity).

Rejects weak or commonly used passwords.

Example Password Policy:

Minimum length: 8 characters.

Must include at least one uppercase letter, one lowercase letter, one digit, and one special character.

Reject passwords from a list of commonly used or easily guessed passwords (e.g., 123456, password, admin).

**Impact**

Severity: High

Likelihood: High

Impact: Major

Users with weak passwords are more susceptible to brute-force attacks.

Compromised accounts can lead to unauthorized access, data theft, and other security breaches.

**Risk Scoring**

CVSS v3.1 Score: 7.4 (High)

Attack Vector: Network

Attack Complexity: Low

Privileges Required: None

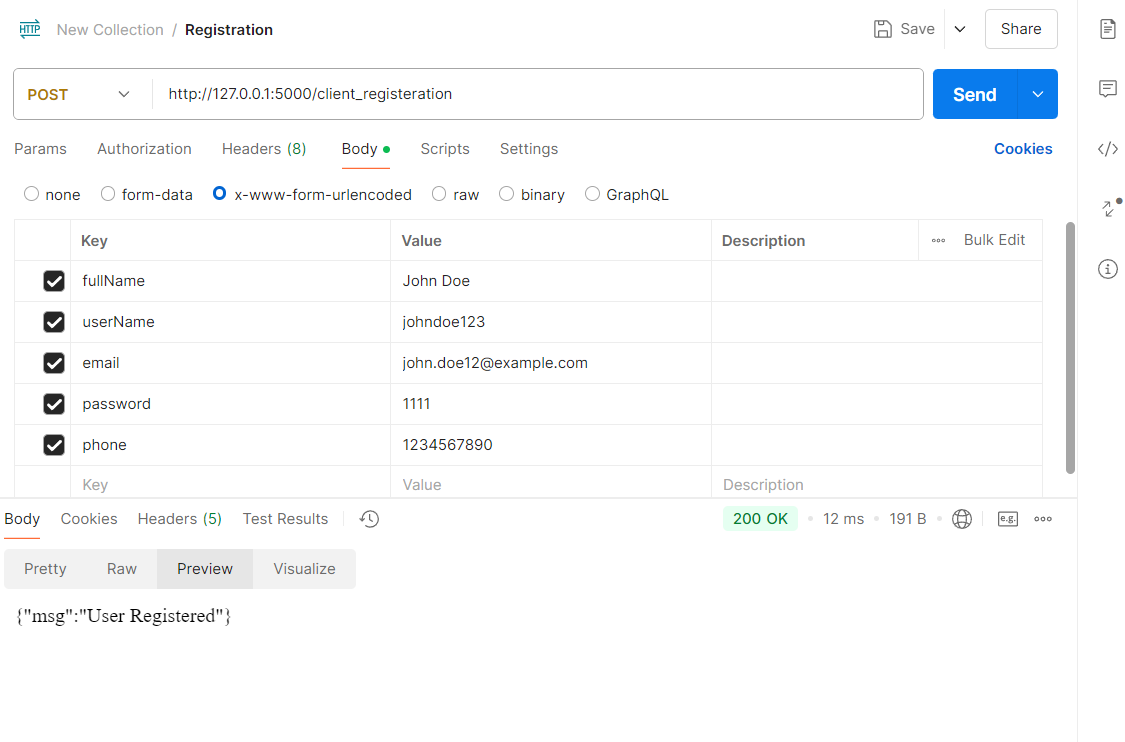
User Interaction: None

Confidentiality Impact: High

Integrity Impact: High

Availability Impact: Low

**Screenshot**



**S.no:** 5

**Bug Id:** SEC\_BUG\_005

**Bug Title**: Storing Passwords Unencrypted in Database

**Description**

The application stores passwords in the database without any encryption or hashing. This practice exposes user credentials to potential breaches. If an attacker gains access to the database, they can view or misuse the stored passwords.

**Steps to Reproduce**

Register a new user via the /client\_registeration endpoint:

Provide a password during registration (e.g., "password123").

The password is stored as plain text in the database.

Access the database and retrieve the password.

Example query:

SELECT \* FROM users WHERE email = 'user@example.com';

**Observed Behavior**

The password is stored as plain text in the database. It is not encrypted or hashed in any way.

**Expected Behavior**

Passwords should be hashed before being stored in the database, and never stored in plain text. This ensures that even if the database is compromised, the attacker cannot easily access the original passwords.

**Impact**

Severity: Critical

Likelihood: High

Impact: Major

If an attacker gains access to the database, they can easily view or misuse all stored passwords.

This could lead to unauthorized access, account takeovers, and data breaches.

**Risk Scoring**

CVSS v3.1 Score: 9.8 (Critical)

Attack Vector: Network (if the database is remotely accessible) or Local (if physical access to the database is possible).

Attack Complexity: Low (No special skills are needed to exploit this vulnerability).

Privileges Required: None (an attacker can access passwords if they gain access to the database).

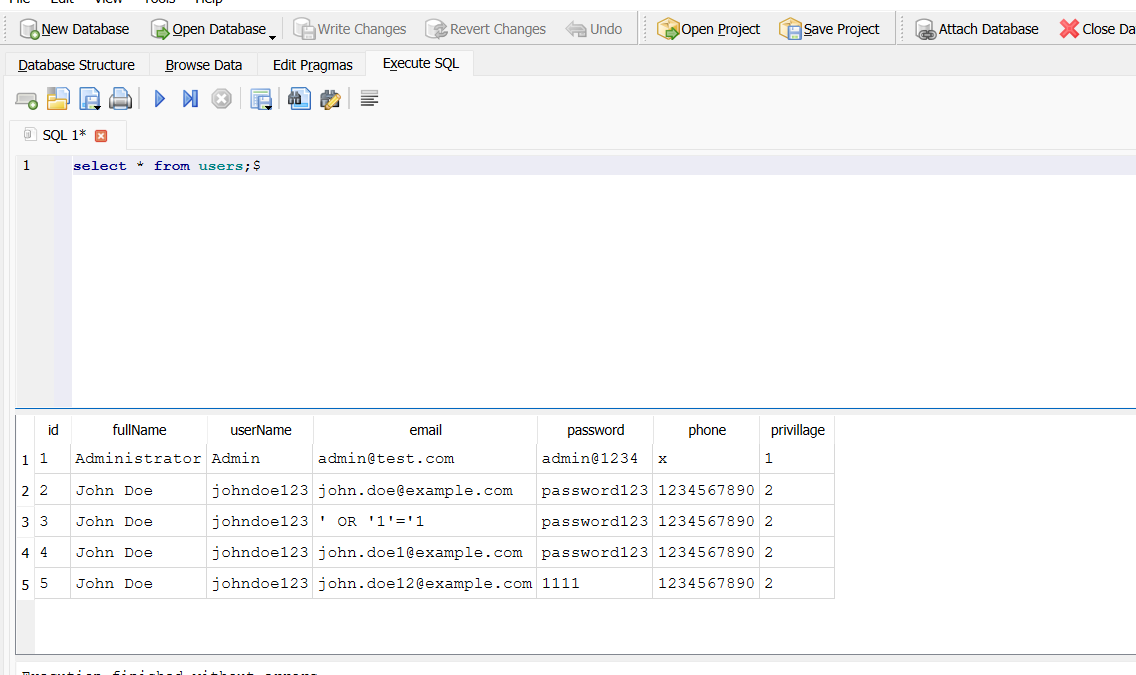
User Interaction: None

Confidentiality Impact: High (All user passwords are exposed).

Integrity Impact: High (Passwords can be modified by unauthorized users).

Availability Impact: Low (This issue doesn’t directly affect availability).

**Screenshot**



**S.no:** 6

**Bug Id:** SEC\_BUG\_006

**Bug Title**: Sensitive User Information Can Be Easily Decoded using access token

**Description**

The login access token can be easily decrypted into email and password using online tools like JWT online decoder.

**Steps to Reproduce**

Login using a valid user via the /client\_login endpoint:

Copy generated access token

Go to <https://jwt.io/>

Decrypt copied access token

If the password is stored as plain text, use online tools to decode or retrieve it.

If the email or phone number is stored in plain text, use the information for further attacks (e.g., phishing or social engineering).

**Observed Behavior**

Access token decrypts to username, email and role

**Expected Behavior**

Access token should not be decrypted easily

**Impact**

Severity: Critical

Likelihood: High

Impact: Major

Exposure of sensitive data: User passwords, emails, and phone numbers can be easily retrieved and decoded.

Account compromise: Attackers can use these details to gain unauthorized access to user accounts or perform phishing attacks.

Privacy violations: Storing sensitive data unprotected violates user privacy and can lead to severe legal consequences.

**Risk Scoring**

CVSS v3.1 Score: 9.8 (Critical)

Attack Vector: Network

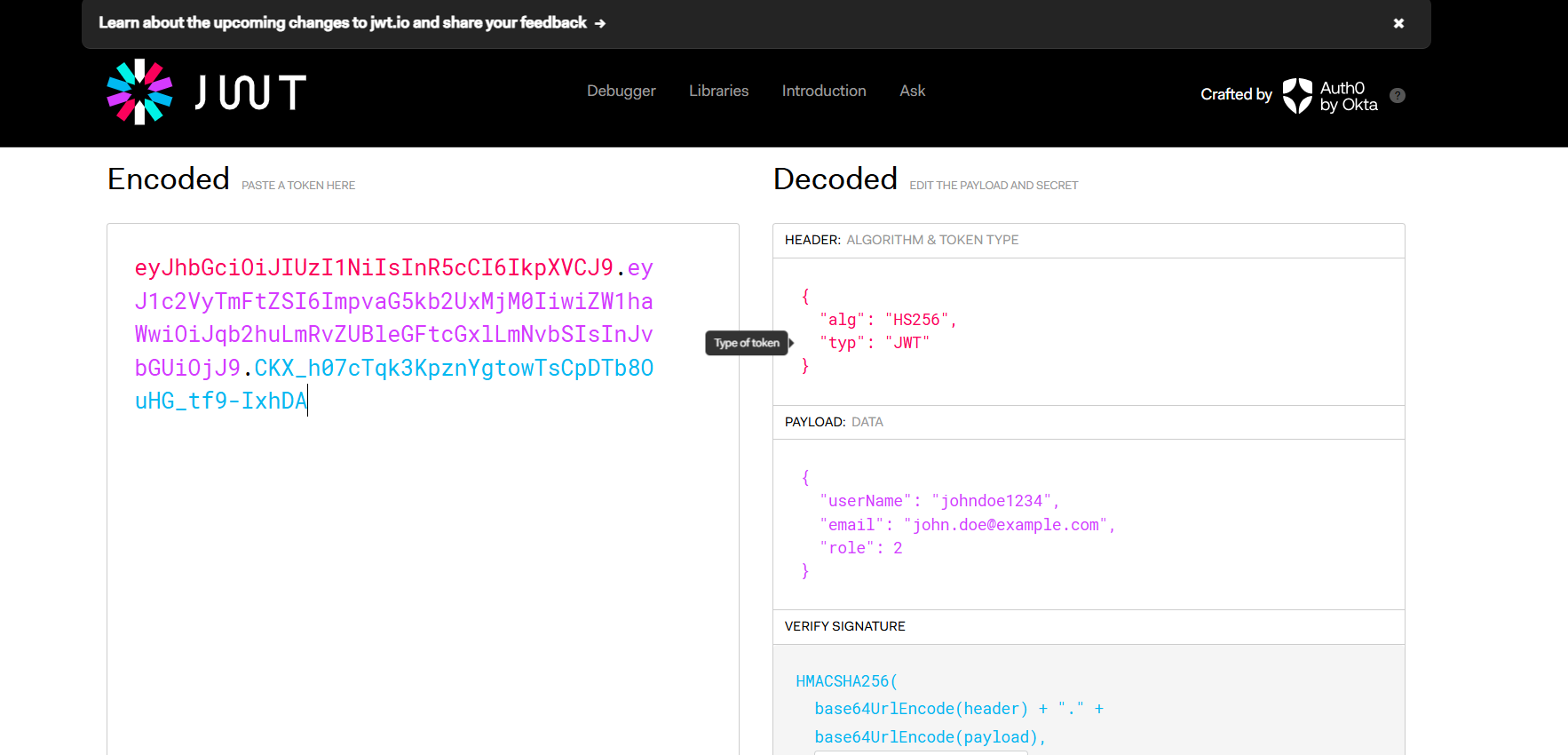
Attack Complexity: Low (No advanced skills are required to retrieve or decode data from an unprotected database).

Privileges Required: Valid login credentials

Integrity Impact: Medium (Data could be altered, but that would require further exploits).

Availability Impact: Low (This issue does not affect availability).

**Screenshot**



**S.no:** 7

**Bug Id:** SEC\_BUG\_007

**Bug Title:** Login Allowed with Empty Username

**Description**

The application allows a login request with an empty username, which should not be allowed as it poses a security risk. Users should not be able to submit login requests without providing valid authentication credentials. An attacker could potentially exploit this issue by sending login requests with empty usernames and bypassing basic authentication checks.

**Steps to Reproduce**

Send a POST request to the /client\_login endpoint with empty username and valid email and password.

Example Request:

{

"userName": "",

"email": "testuser@example.com",

"password": "password123"

}

Check the Response: The system should return an error or validation message stating that the username is required, but instead, it logs the user in without validating the empty username.

**Observed Behavior**

The system allows login attempts with an empty username (i.e., userName: "").

The application does not validate or reject the request, which could lead to unauthorized access if email and password match a valid account.

**Expected Behavior**

Usernames should not be allowed to be empty. The system should return a validation error if an empty username is provided, such as:

"msg": "Username is required"

The application should ensure that both email and username are non-empty before processing the login.

**Impact**

Severity: High

Likelihood: High

Impact: Major

Unauthorized Access: If the username is not validated properly, attackers could potentially bypass authentication by submitting empty usernames and valid email/password combinations.

User Confusion: Legitimate users may experience confusion if they are able to login without providing a username, which could indicate a flaw in the application.

Exploitation Risk: This flaw could be exploited to gain access to accounts with known emails and passwords, compromising the security of user data.

**Risk Scoring**

CVSS v3.1 Score: 8.1 (High)

Attack Vector: Network (remote authentication can be triggered over the network).

Attack Complexity: Low (Sending a request with an empty username is straightforward).

Privileges Required: None (No special privileges are required to exploit this flaw).

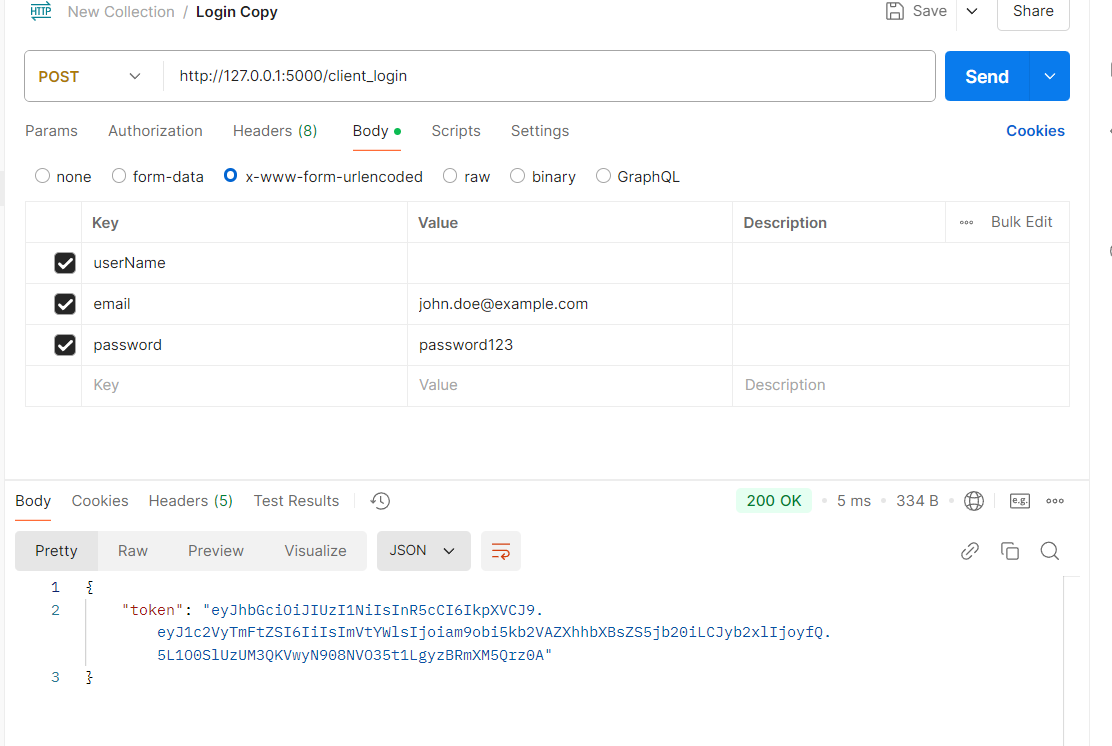
User Interaction: None (The issue is purely a backend vulnerability).

Confidentiality Impact: High (User accounts can be compromised by bypassing the authentication checks).

: Medium (Integrity Impact Could allow unauthorized users to gain access to the account, leading to data corruption).

Availability Impact: Low (This does not affect the system’s availability).

**Screenshot**



**S.no:** 8

**Bug Id:** SEC\_BUG\_008

**Bug Title**: Reflected XSS Vulnerability in /client\_registeration Endpoint

**Description**:

The application does not properly sanitize user input for the fullName and userName fields in the user registration process. This allows a user to inject JavaScript code (XSS payloads), which is reflected in the response and executed on the frontend when viewed by the user. This vulnerability can be exploited by attackers to execute malicious scripts, steal sensitive data, hijack sessions, or perform other malicious actions.

**Steps to Reproduce:**

Open Postman.

Create a POST request to the /client\_registeration endpoint with the following JSON payload:

{

"fullName": "<script>alert('XSS');</script>",

"userName": "<script>alert('XSS');</script>",

"email": "testuser@example.com",

"password": "password123",

"phone": "1234567890"

}

Send the request.

Check the response body. It contains the injected script as-is:

{

"msg": "User Registered",

"fullName": "<script>alert('XSS');</script>",

"userName": "<script>alert('XSS');</script>"

}

Inspect the response and observe that the script is reflected in the response body without sanitization or encoding.

**Expected Behavior:**

The application should sanitize or encode user input to prevent malicious scripts from being injected or reflected in the response. Specifically:

Any HTML or JavaScript tags should be escaped or removed before being sent back in the response.

The response body should not include raw script tags or executable code.

**Actual Behavior:**

The user input, which contains JavaScript code (<script>alert('XSS');</script>), is reflected back in the response without being sanitized. This causes the script to execute when rendered in a browser.

**Impact**

Severity: High

Risk: Critical

Endpoint: /client\_registeration

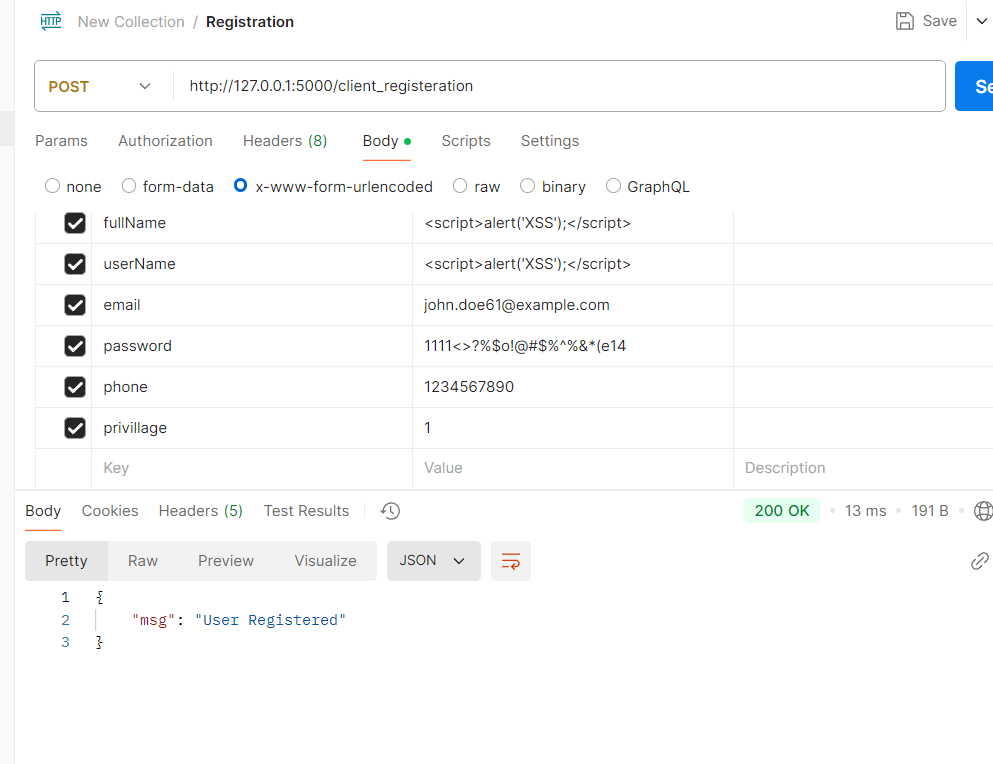
User Data Theft: Malicious scripts could steal sensitive user information like session cookies, login credentials, or personal data.

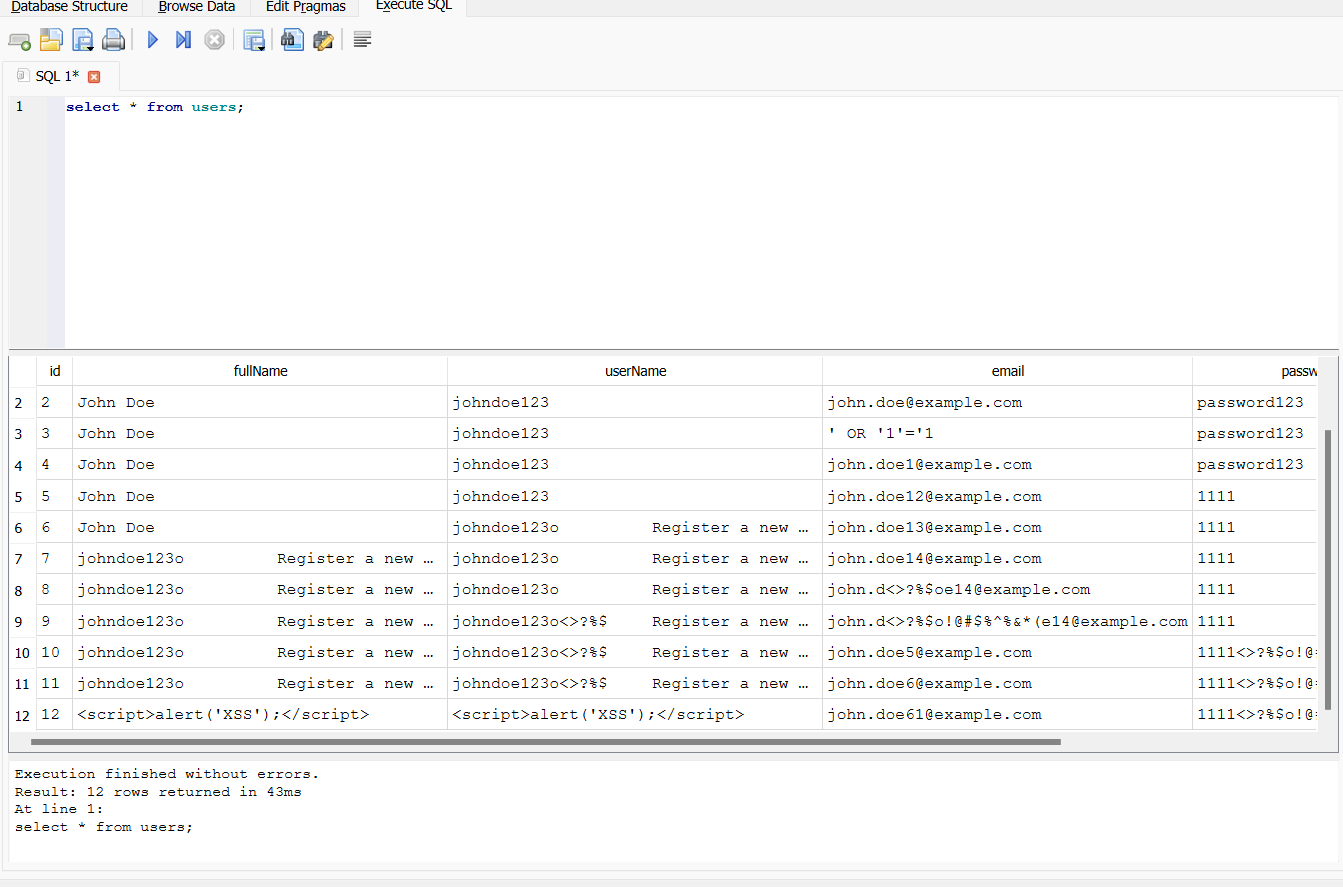
Session Hijacking: An attacker could execute a script to steal a session token, enabling them to impersonate users.

Defacement or Malicious Actions: Attackers can manipulate user interactions by injecting malicious JavaScript (e.g., redirecting users to phishing sites, modifying content, or sending unauthorized actions).

Cross-Site Request Forgery (CSRF): If combined with other vulnerabilities, XSS could be used to perform actions on behalf of users without their consent.

**Screenshot**





**S.no:** 9

**Bug Id:** SEC\_BUG\_009

**Bug Title:** Incorrect message for wrong email and password

{

    "msg": "In correct email or password"

}

**Description**

The message you provided is not grammatically correct. Here's the corrected version:

**Incorrect email or password.**

**Expected Behavior**

The message on incorrect email or password while login should be {Incorrect email or password}.

**Observed Behavior**

The message on incorrect email or password while login is {In correct email or password}.

**Impact**

Severity: Low

Impact: Low