Information Security Project:
Web Application Vulnerability Assessment and Mitigation

Session 2023-27

Submitted by:

Amir Hashmi	2023-CS-11
Sher Muhammad	2023-CS-15
Mobeen Butt	2023-CS-28

Supervised by:
Dr. Faiza Iqbal

Course:
Information Security

Department of Computer Science
University of Engineering and Technology Lahore
Pakistan

Table of Contents

1. Introduction

This project focuses on identifying, exploiting, and mitigating common security vulnerabilities in a self-developed website. The goal was to simulate real-world attacks, implement security measures, and verify their effectiveness.

Objectives:

- Identify vulnerabilities in a web application.
- Perform attacks to exploit these vulnerabilities.
- Implement security solutions to mitigate risks.
- Re-test the application to ensure vulnerabilities are patched.

2. Vulnerabilities Exploited & Mitigation Strategies

2.1.1 Distributed Denial of Service (DDoS) Attack

Attack Description:

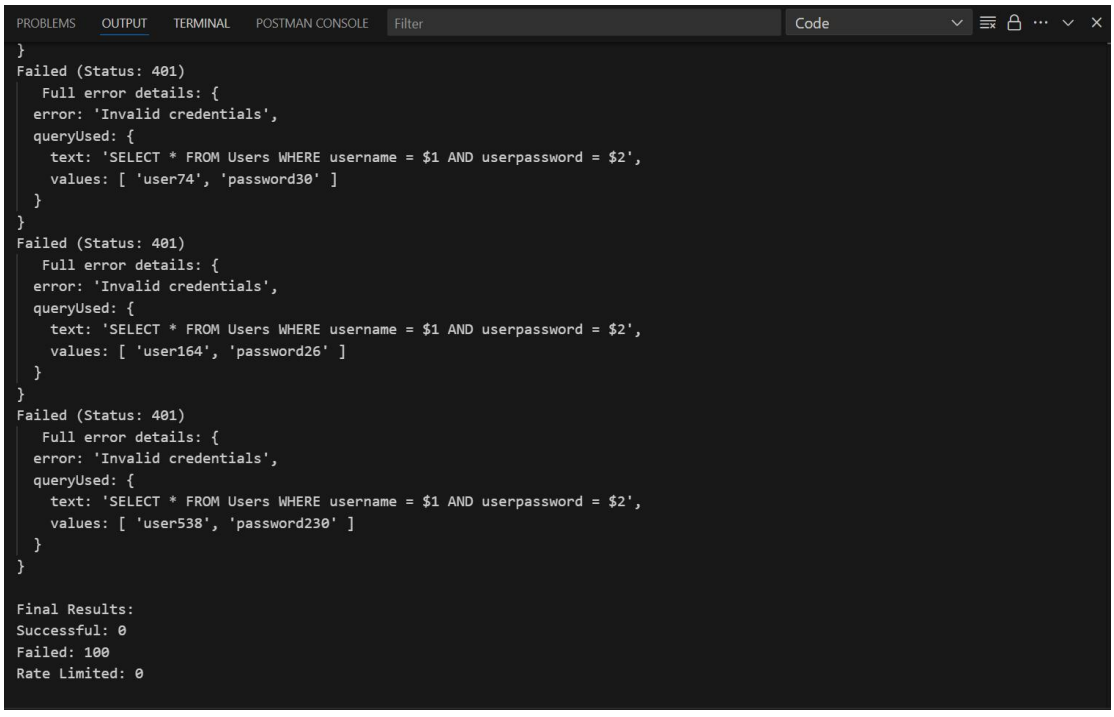
- Sent **100 requests in one minute** from a single IP to overwhelm the server.
- Observed server slowdown and potential crash.

Solution Implemented:

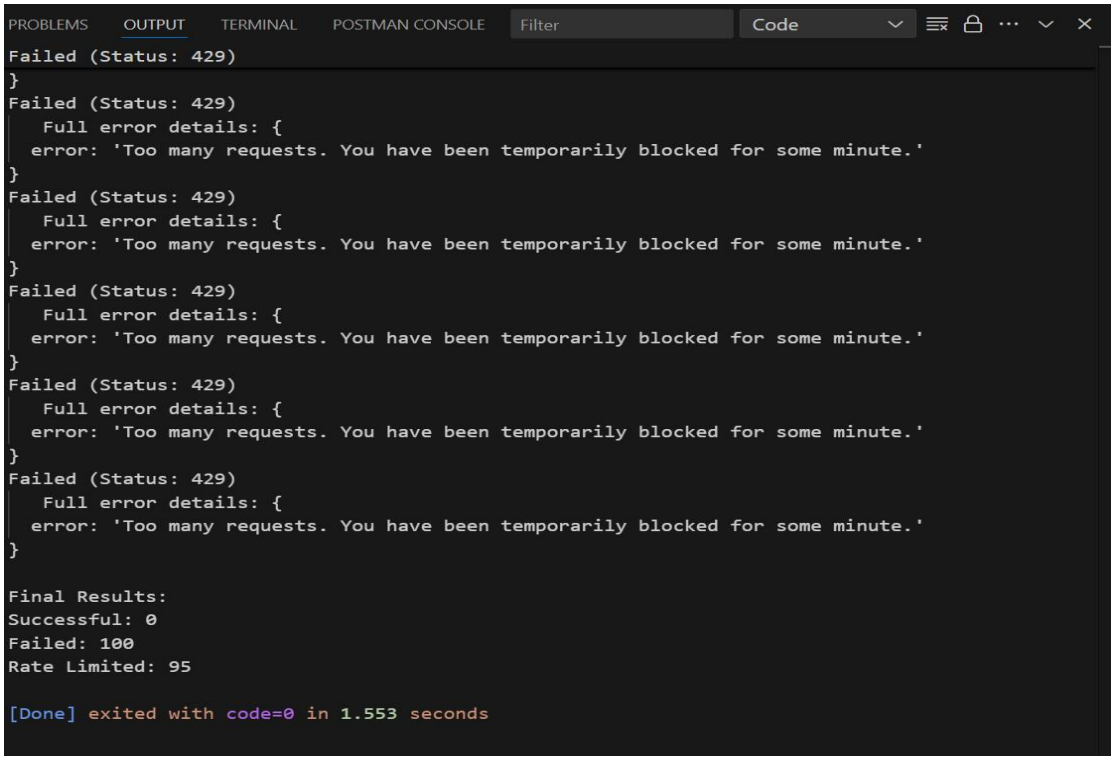
- **Rate Limiting:** Blocked IP if it exceeds **5 requests per minute**.
- **Temporary Blocking:** Banned the IP for **1 minute** upon exceeding the limit.

Wireframes:

- **Before Fix:** Server logs showing 100 requests from a single IP.



- **After Fix:** Logs showing blocked IP after 5 requests.



2.1.2 Cross-Site Scripting (XSS) Attack

Attack Description:

- Injected a malicious script in the **username field**:
  <script>alert('Collect Your Reward!'); window.location='http://scam-website.com';
  </script>
- When rendered, it displayed a fake reward button, redirecting users to a phishing site.

Solution Implemented:

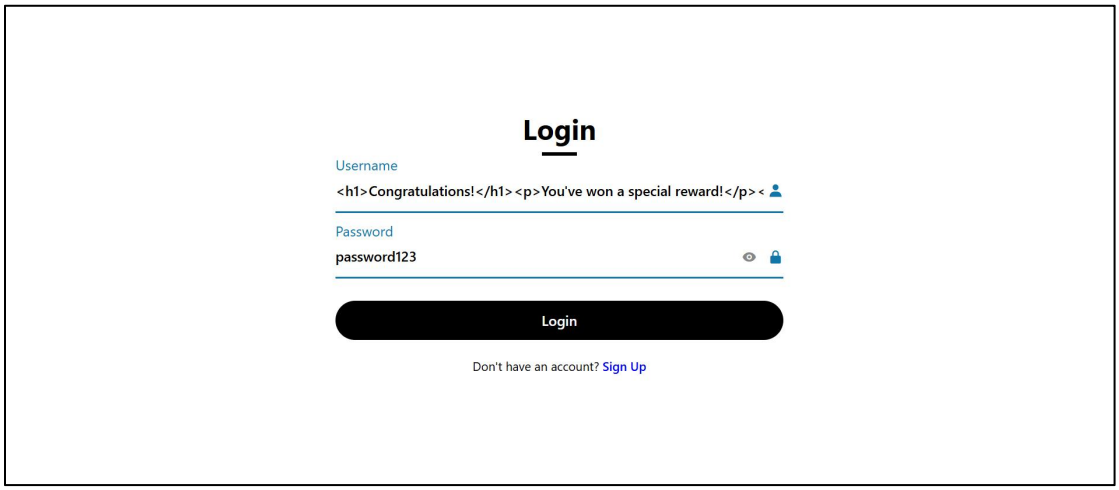
- **Input Sanitization:** Removed HTML/JS tags before database insertion.
- **Output Encoding:** Rendered user inputs as text, not executable scripts.

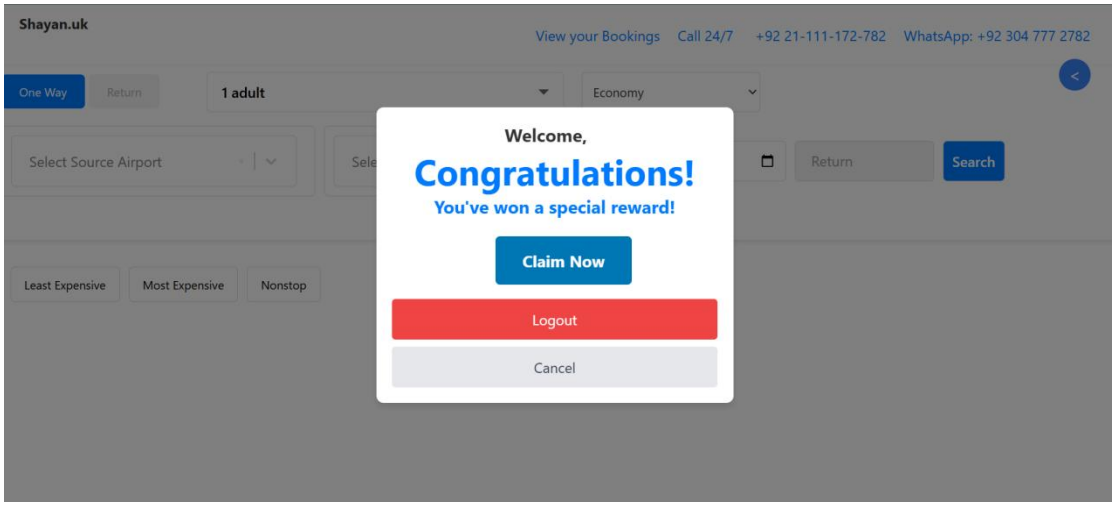
Wireframes:

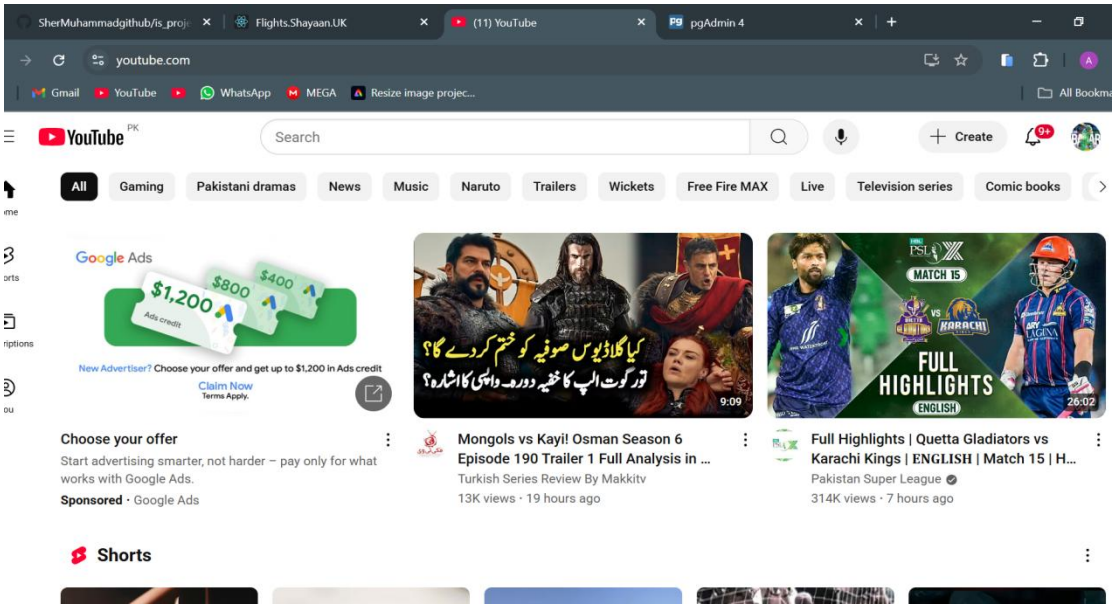
- **Before Fix:** Injected script execution.

```
[Running] node "d:\Github Repos\is_project\Volunerable server\Attacks Simulation\XSSAttack.js"
Step 1: Creating a user with XSS payload as username...
Payload: {
  username: `<h1>Congratulations!</h1><p>You've won a special reward!</p><a href="https://www.youtube.com/"
  style="display:inline-block;background-color:#0077B5;color:white;padding:15px 30px;text-decoration:none;font-size:18px;
  border-radius:5px;margin-top:20px" target="_blank">Claim Now</a></div>`,
  password: 'password123'
}
Signup Response Status: 200
Signup Response Data: { message: 'Signup successful!' }
Malicious user created successfully! XSS payload stored in database.

[Done] exited with code=0 in 1.266 seconds
```







Unwanted Redirection

- **After Fix:** Sanitized output (plain text).

```
[Running] node "d:\Github Repos\is_project\server\Attacks Simulation\XSSAttack.js"
Step 1: Creating a user with XSS payload as username...
Payload: {
  username: `<h1>Congratulations!</h1><p>You've won a very special reward!</p><a
  href="https://www.facebook.com/" style="display:inline-block;background-color:#0077B5;
  color:white;padding:15px 30px;text-decoration:none;font-size:18px;border-radius:5px;
  margin-top:20px" target="_blank">Claim Now</a></div>`,
  password: 'password123'
}
Signup Response Status: 201
Signup Response Data: {
  message: 'Signup successful!',
  user: {
    id: 29,
    username: "Congratulations!You've won a very special reward!Claim Now"
  },
  token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
  eyJpZCI6MjksInVzZXJuYW1lIjoiQ29uZ3JhdHVsYXRpb25zIVlvdSd2ZSB3b24gYSB2ZXJ5IHNwZWNpYWwgcmV3Y
  XJkIUNsYWltIE5vdyIsImlhdCI6MTc0NTY0MzY2OSwiZXhwIjoxNzQ1NzMwMDY5fQ.
  KK0RLucvFH4kuK5yHmZ1PyU4x806NUfsA1c8gkdT70I'
}
Failed to create user with XSS payload.
```

2.1.3. Cross-Site Request Forgery (CSRF) Protection

Attack Description:

Without CSRF tokens, attackers could forge requests (e.g., unauthorized fund transfers).

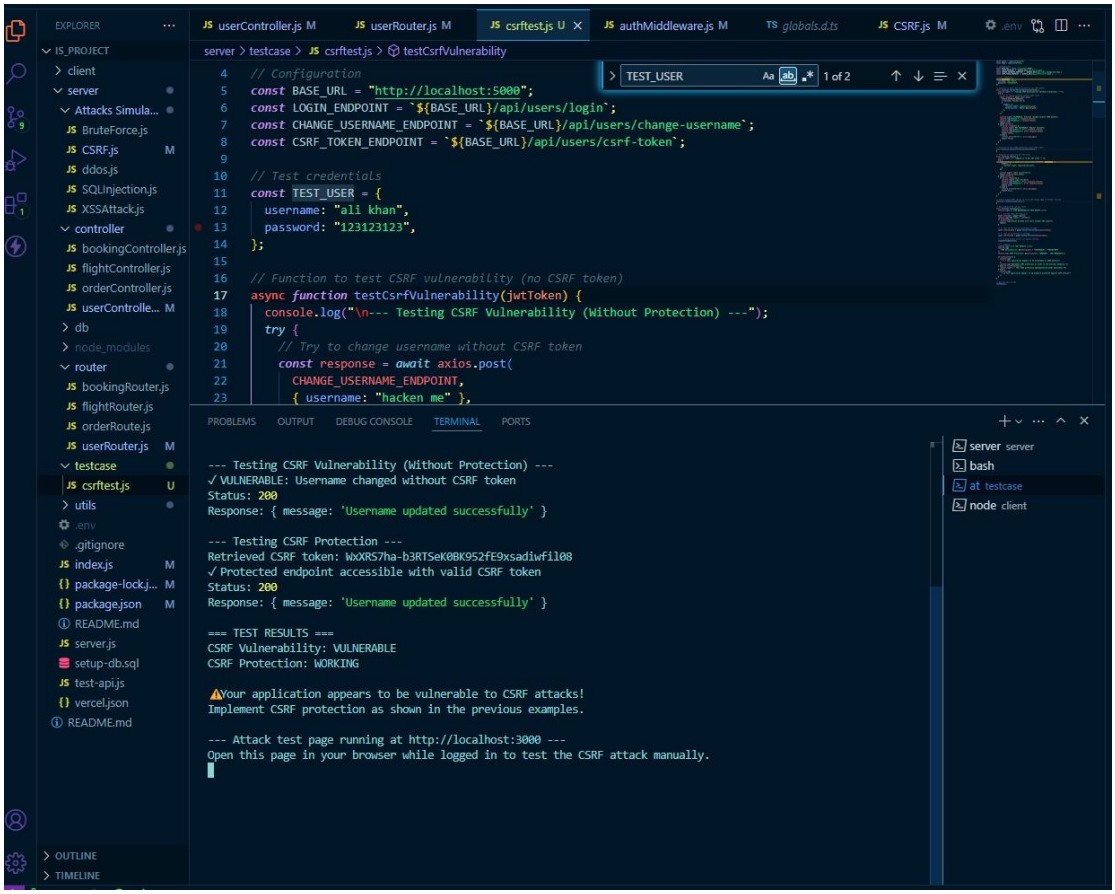
Solution Implemented:

CSRF Tokens:

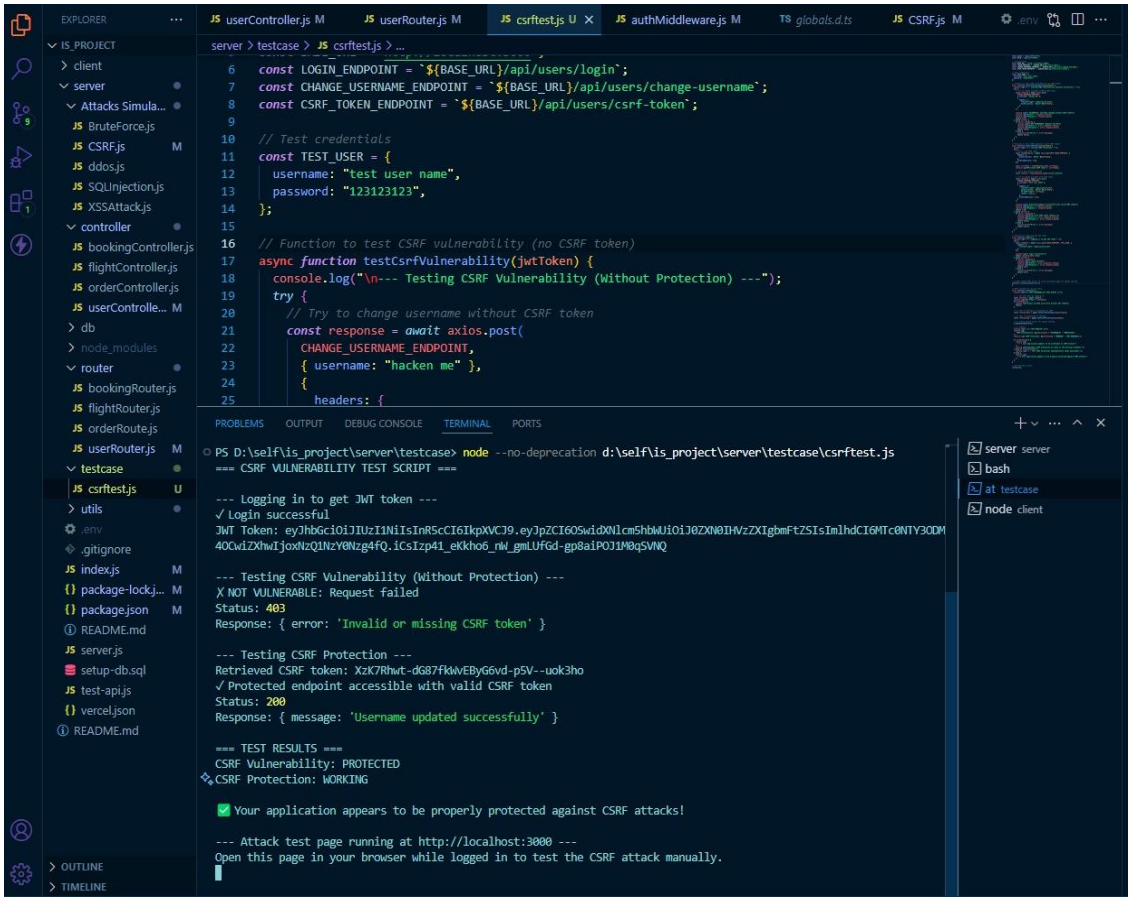
- Generated a unique token upon login.
- Required token for sensitive API requests.
- Rejected requests without valid tokens.

Wireframes:

- **Before Fix:** Successful forged request (e.g., POST without token).



- **After Fix:** Request rejected due to missing/invalid token.



2.1.4. Brute Force Password Attack

Attack Description:

- Used a **dictionary of 1000 passwords** to guess a **3-digit PIN**.
- Successfully logged in after multiple attempts.

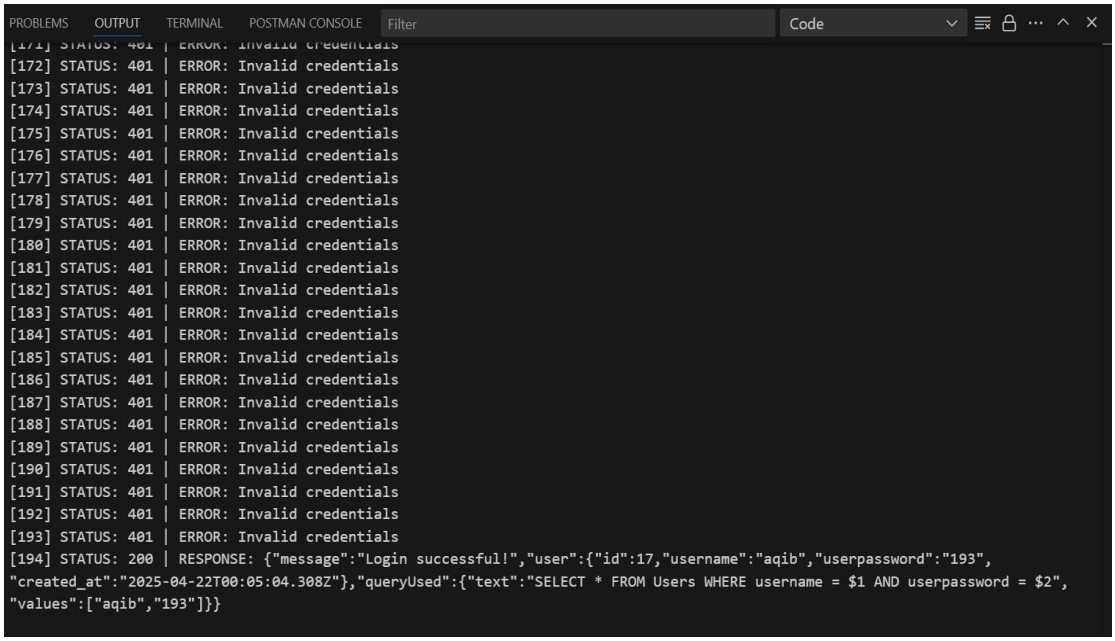
Solution Implemented:

Account Lockout Policy:

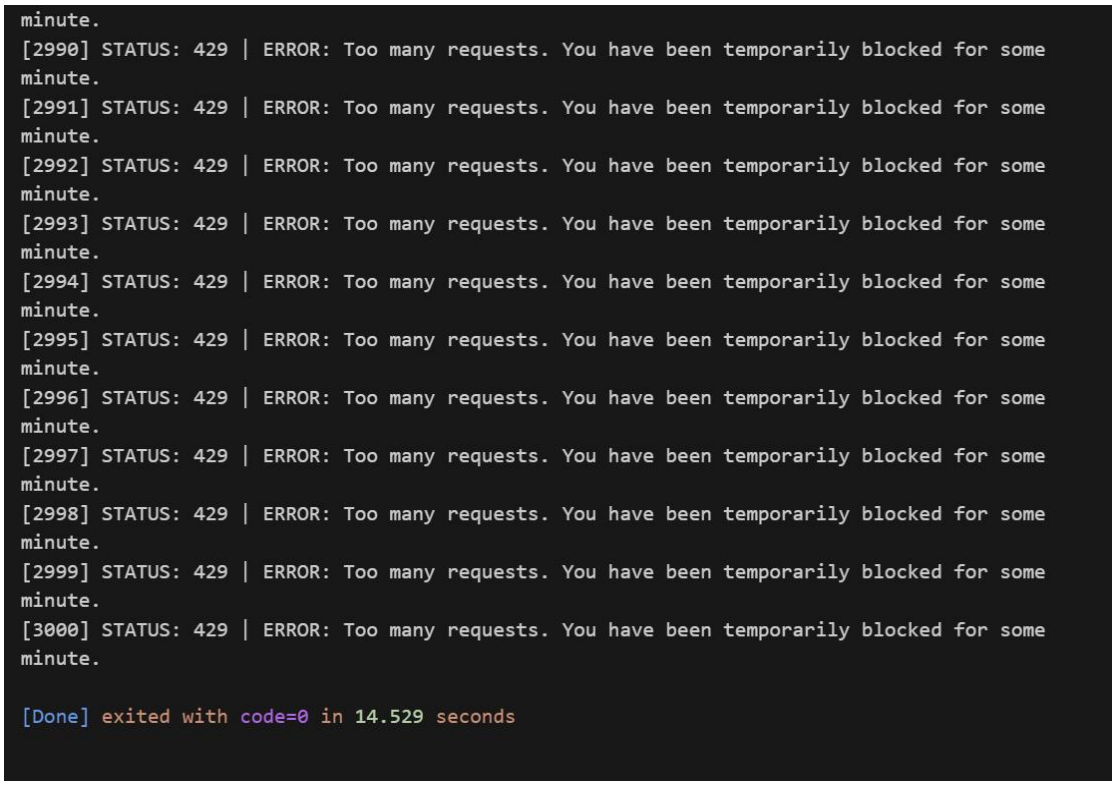
- Blocked user after **5 failed attempts**.
- Lock duration: **5 minutes**.

Wireframes:

- **Before Fix:** Successful login after brute-forcing.



- **After Fix:** Account locked after 5 attempts.



2.1.5. SQL Injection Attack

Attack Description:

Injected `' OR 1=1 --` in the login form, bypassing authentication.

Solution Implemented:

Parameterized Queries:

- Used prepared statements to separate SQL logic from user input.

Wireframes:

- **Before Fix:** Successful login with SQL injection.

```
[Running] node "d:\Github Repos\is_project\Volunerable server\Attacks Simulation\SQLInjection.js"
Attempting SQL injection attack...
Payload: { username: "admin' OR '1'='1", password: 'anything' }
Response Status: 200
Response Data: {
  message: 'Login successful!',
  user: {
    id: 1,
    username: 'testuser',
    userpassword: 'testpassword',
    created_at: '2025-04-20T21:40:54.770Z'
  },
  queryUsed: '\n' +
    '    SELECT * FROM Users \n' +
    "    WHERE (username = 'admin' OR '1'='1') \n" +
    "    OR (username = 'admin' AND 'anything' = 'anything')\n" +
    '  '
}
SQL Injection SUCCESSFUL! Authentication bypassed.
```

- **After Fix:** Login fails with malicious input.

```
[Running] node "d:\Github Repos\is_project\Volunerable server\Attacks Simulation\SQLInjection.js"
Attempting SQL injection attack...
Payload: { username: "admin' OR '1'='1", password: 'anything' }
Error: {
  error: 'Invalid credentials',
  queryUsed: {
    text: 'SELECT * FROM Users WHERE username = $1 AND userpassword = $2',
    values: [ "admin' OR '1'='1", 'anything' ]
  }
}
SQL Injection failed. The application is likely protected against this attack.

[Done] exited with code=0 in 0.794 seconds
```

2.1.6. Location-Based Access Restriction

Attack Description:

Requests from **Israel** were allowed by default.

Solution Implemented:

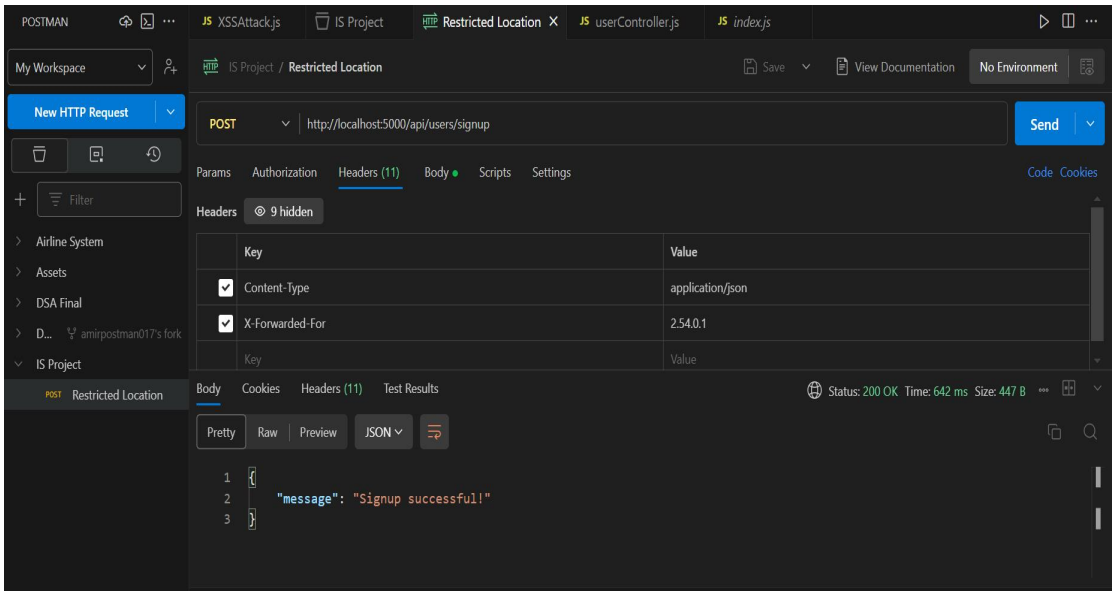
Geo-Blocking:

- Detected IP geolocation.

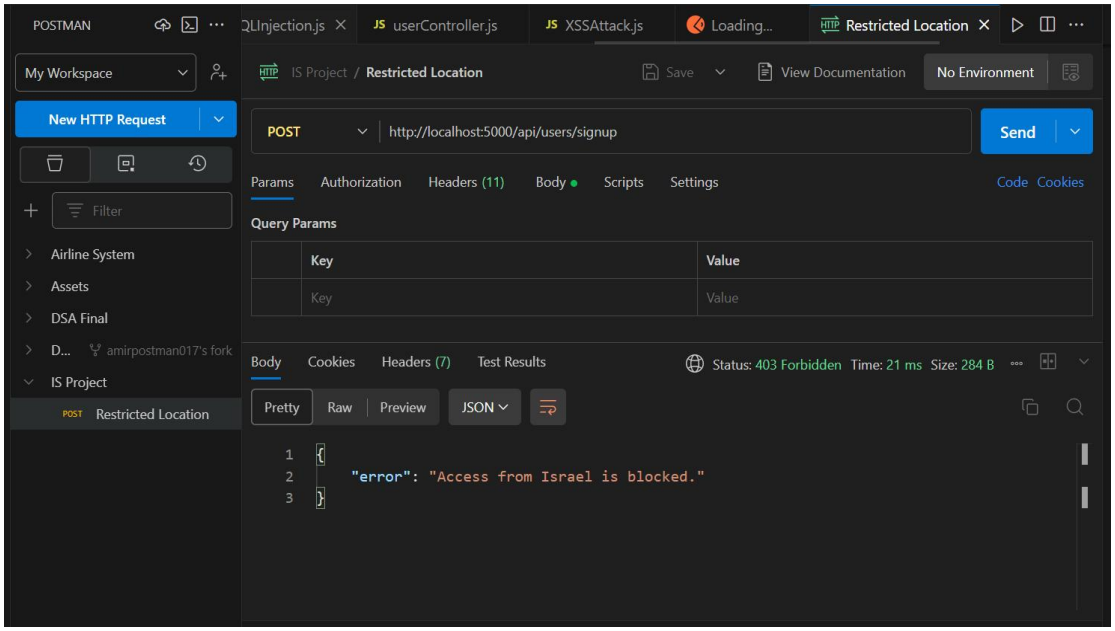
- Blocked requests from **Israel**.

Wireframes:

- **Before Fix:** Successful access from Israel.



- **After Fix:** "Access Denied" for Israeli IPs.



2.1.7. Password Encryption (Feistel Cipher)

Attack Description:

Passwords stored in **plaintext**, vulnerable to database leaks.

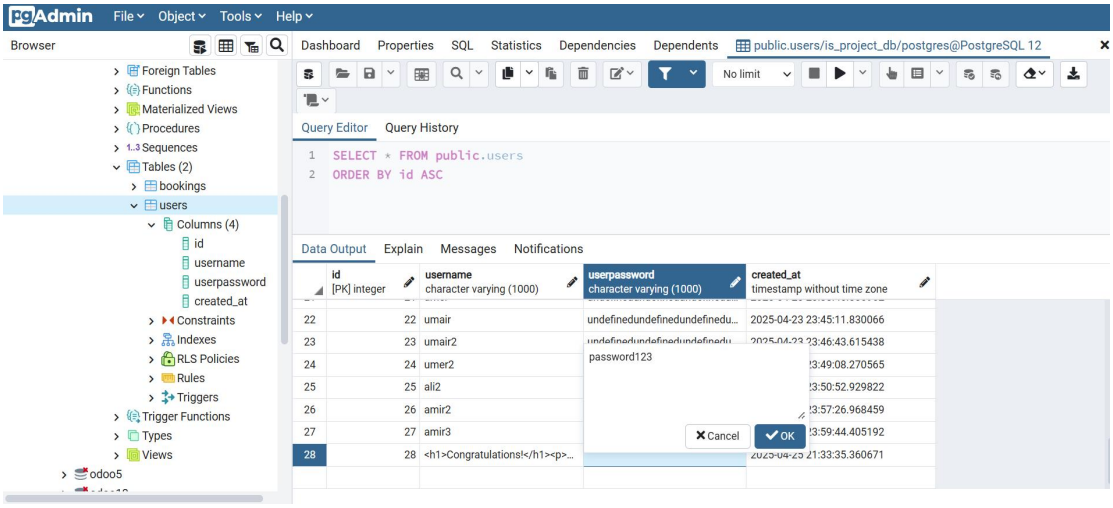
Solution Implemented:

Feistel Cipher Encryption:

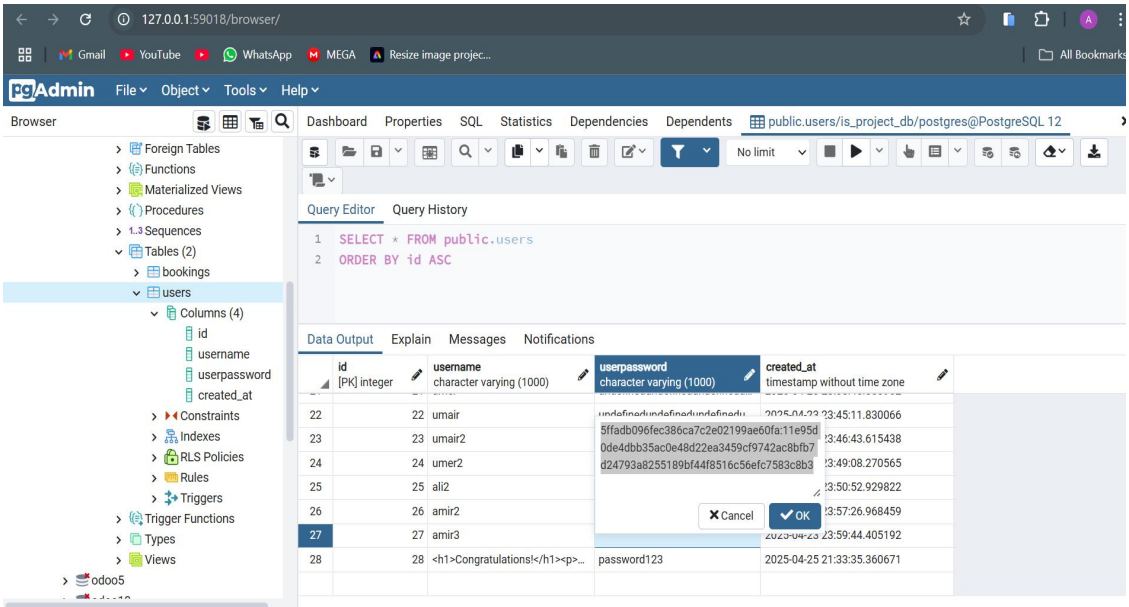
- Encrypted passwords before storage.
- Decrypted only during verification.

Wireframes:

- **Before Fix:** Database showing plaintext passwords.



- **After Fix:** Database showing encrypted passwords.



3. Conclusion:

- Security is a continuous process – new threats emerge constantly.
- Input validation and encryption are critical.
- Rate limiting and geo-blocking can prevent abuse.

4. References:

GitHub: https://github.com/SherMuhammadgithub/is_project