

IS Project: Web Application Vulnerability Assessment and Mitigation



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**

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

1. Distributed Denial of Service (DDoS) Attack

Attack Description:

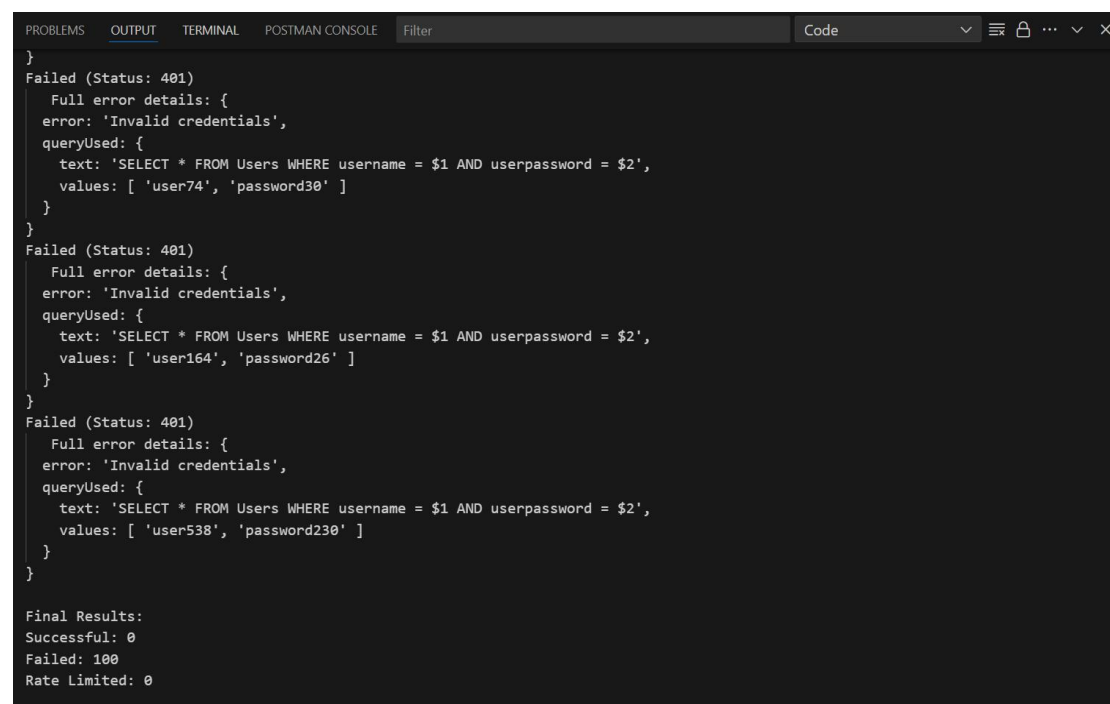
- Sent **100 requests in one minute** from a single IP to overwhelm the server.
- Observed server slowdown and potential crash.

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:

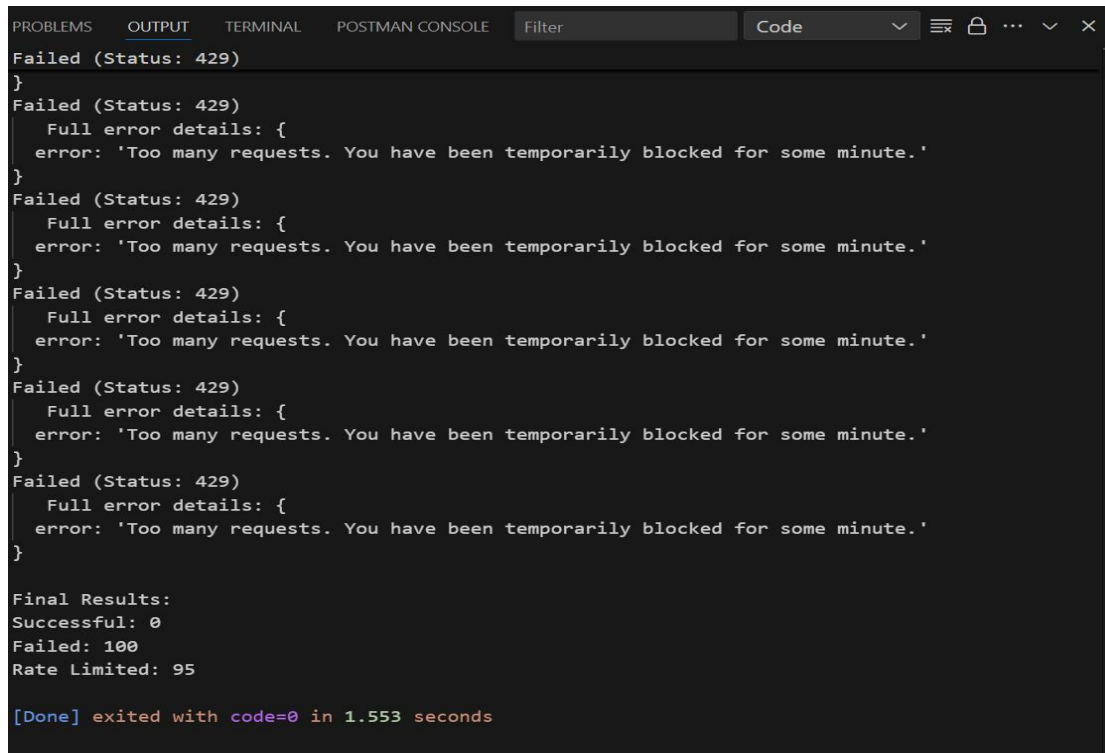
- **Before Fix:** Server logs showing 100 requests from a single IP.



```
PROBLEMS  OUTPUT  TERMINAL  POSTMAN CONSOLE  Filter  Code
}
Failed (Status: 401)
  Full error details: {
    error: 'Invalid credentials',
    queryUsed: {
      text: 'SELECT * FROM Users WHERE username = $1 AND userpassword = $2',
      values: [ 'user74', 'password30' ]
    }
  }
}
Failed (Status: 401)
  Full error details: {
    error: 'Invalid credentials',
    queryUsed: {
      text: 'SELECT * FROM Users WHERE username = $1 AND userpassword = $2',
      values: [ 'user164', 'password26' ]
    }
  }
}
Failed (Status: 401)
  Full error details: {
    error: 'Invalid credentials',
    queryUsed: {
      text: 'SELECT * FROM Users WHERE username = $1 AND userpassword = $2',
      values: [ 'user538', 'password230' ]
    }
  }
}

Final Results:
Successful: 0
Failed: 100
Rate Limited: 0
```

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



```
PROBLEMS  OUTPUT  TERMINAL  POSTMAN CONSOLE  Filter  Code  v  ≡  🔒  ⋮  v  x
Failed (Status: 429)
}
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
Failed (Status: 429)
  Full error details: {
    error: 'Too many requests. You have been temporarily blocked for some minute.'
  }
}
Final Results:
Successful: 0
Failed: 100
Rate Limited: 95
[Done] exited with code=0 in 1.553 seconds
```

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
```

Login

Username

<h1>Congratulations!</h1><p>You've won a special reward!</p>

Password

password123

Login

Don't have an account? [Sign Up](#)

Shayan.uk

View your Bookings Call 24/7 +92 21-111-172-782 WhatsApp: +92 304 777 2782

One Way Return 1 adult Economy

Select Source Airport Select Destination Airport

Least Expensive Most Expensive Nonstop

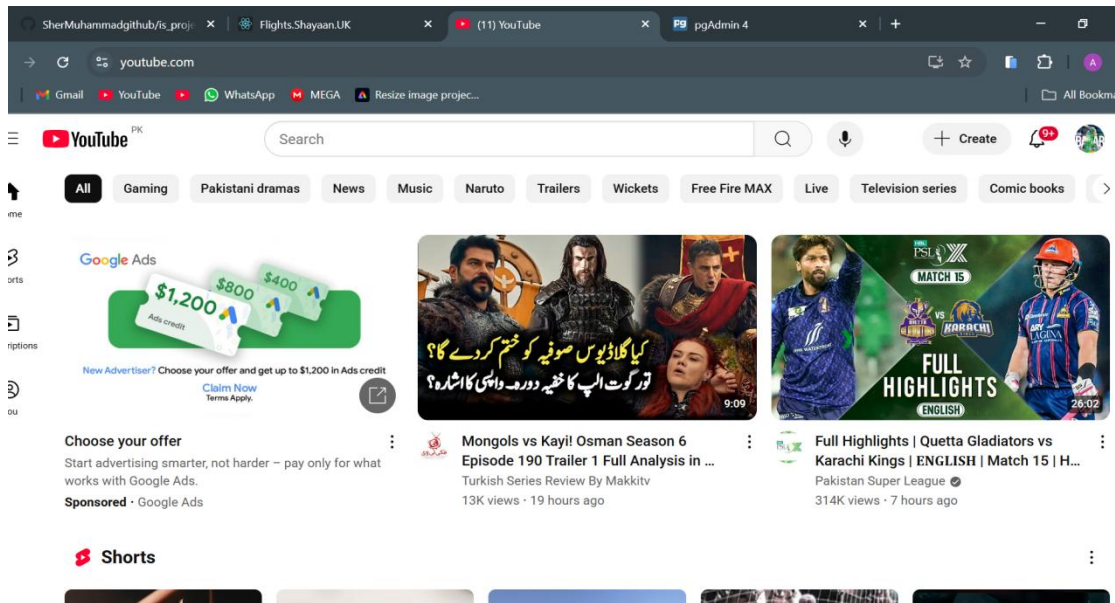
Welcome,
Congratulations!
You've won a special reward!

Claim Now

Logout

Cancel

Search



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.eyJpZCI6MjksInVzZXJuYWI1IjoiaWoiQ29uZ3JhdHVzYXRpb25zIV1vdSd2ZSB3b24gYSB2ZXJ5IHlwZWpYwWgcmV3YXJkIUNsYwltIE5vdyIsIm1hdCI6MTc0NTY0MzY2OSwiZXhwIjoxNzQ1NzMDY5fQ.KK0RLucvFH4kuK5yHmZ1PyU4x806NUfsA1c8gkdT70I'
}
Failed to create user with XSS payload.
```

3. Cross-Site Request Forgery (CSRF) Protection

Attack Description:

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

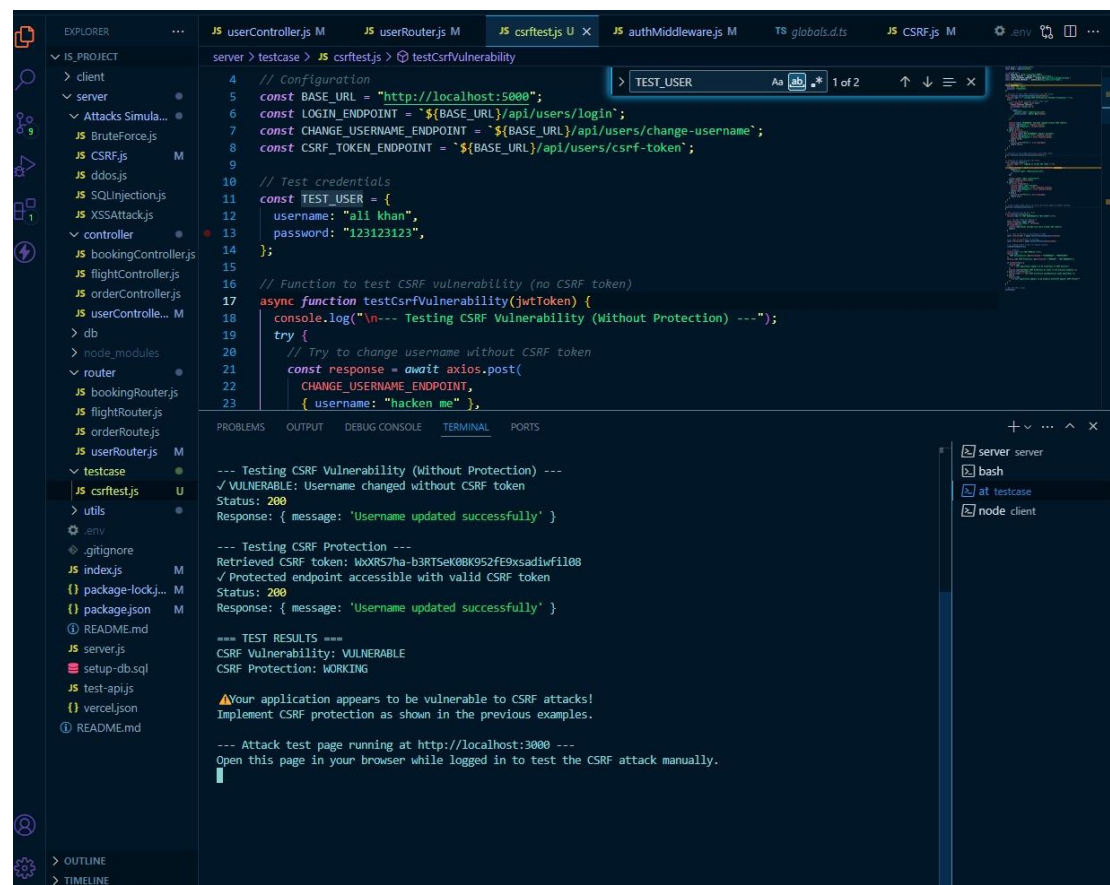
Solution Implemented:

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).



```
server > testcase > JS csrfTestjs > testCsrfVulnerability
4 // Configuration
5 const BASE_URL = "http://localhost:3000";
6 const LOGIN_ENDPOINT = `${BASE_URL}/api/users/login`;
7 const CHANGE_USERNAME_ENDPOINT = `${BASE_URL}/api/users/change-username`;
8 const CSRF_TOKEN_ENDPOINT = `${BASE_URL}/api/users/csrf-token`;
9
10 // Test credentials
11 const TEST_USER = {
12   username: "ali khan",
13   password: "123123123",
14 };
15
16 // Function to test CSRF vulnerability (no CSRF token)
17 async function testCsrfVulnerability(jwtToken) {
18   console.log("\n--- Testing CSRF Vulnerability (Without Protection) ---");
19   try {
20     // Try to change username without CSRF token
21     const response = await axios.post(
22       CHANGE_USERNAME_ENDPOINT,
23       { username: "hacken me" },
24     );
25   } catch (error) {
26     console.log(error);
27   }
28 }
29
30 // Test the vulnerability
31 testCsrfVulnerability(jwtToken).then(() => {
32   console.log("Test completed");
33 });
```

```
--- Testing CSRF Vulnerability (Without Protection) ---
✓ VULNERABLE: Username changed without CSRF token
Status: 200
Response: { message: 'Username updated successfully' }

--- Testing CSRF Protection ---
Retrieved CSRF token: WoxRS7ha-b3RTSeK0BK952fe9xsadiwf1l08
✓ Protected endpoint accessible with valid CSRF token
Status: 200
Response: { message: 'Username updated successfully' }

=== TEST RESULTS ===
CSRF Vulnerability: VULNERABLE
CSRF Protection: WORKING

⚠️Your application appears to be vulnerable to CSRF attacks!
Implement CSRF protection as shown in the previous examples.

--- Attack test page running at http://localhost:3000 ---
Open this page in your browser while logged in to test the CSRF attack manually.
```


- The screenshot displays the Visual Studio Code editor interface. On the left, the Explorer sidebar shows a project structure with folders like 'client', 'server', 'utils', and 'db'. The main editor area contains the file 'csrfTest.js' with the following code:

```
server > testcase > JS csrfTestjs > ...  
6 const LOGIN_ENDPOINT = `${BASE_URL}/api/users/login`;  
7 const CHANGE_USERNAME_ENDPOINT = `${BASE_URL}/api/users/change-username`;  
8 const CSRF_TOKEN_ENDPOINT = `${BASE_URL}/api/users/csrf-token`;  
9  
10 // Test credentials  
11 const TEST_USER = {  
12   username: "test user name",  
13   password: "123123123",  
14 };  
15  
16 // Function to test CSRF vulnerability (no CSRF token)  
17 async function testCsrfVulnerability(jwtToken) {  
18   console.log("\n--- Testing CSRF Vulnerability (Without Protection) ---");  
19   try {  
20     // Try to change username without CSRF token  
21     const response = await axios.post(  
22       CHANGE_USERNAME_ENDPOINT,  
23       { username: "hacken me" },  
24       { headers: {  
25
```

The bottom panel shows the TERMINAL output of running the script:

```
PS D:\self\is_project\server\testcase> node --no-deprecation d:\self\is_project\server\testcase\csrfTest.js  
=== CSRF VULNERABILITY TEST SCRIPT ===  
  
--- Logging in to get JWT token ---  
✓ Login successful  
JWT Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ejYpZCtGOSwidXNlcmShbWUiOiI0ZXNhIH4yZXBtIGFmZSIsIm1hcnQNTY3ODM0OwIdXhwijoxNzQ1ZDZlbnY0Ng4fQ.iCS1zp41_eKkHo6_nW_gmLUFGd-gp8A1P0J1MqgSVNQ  
  
--- Testing CSRF Vulnerability (Without Protection) ---  
X NOT VULNERABLE: Request failed  
Status: 403  
Response: { error: 'Invalid or missing CSRF token' }  
  
--- Testing CSRF Protection ---  
Retrieved CSRF token: XxK7BhwL-dG87HdwEBYG6vd-p5v--uok3ho  
✓ Protected endpoint accessible with valid CSRF token  
Status: 200  
Response: { message: 'Username updated successfully' }  
  
=== TEST RESULTS ===  
CSRF Vulnerability: PROTECTED  
✧ CSRF Protection: WORKING  
  
✔ Your application appears to be properly protected against CSRF attacks!  
  
--- Attack test page running at http://localhost:3000 ---  
Open this page in your browser while logged in to test the CSRF attack manually.
```

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

```
PROBLEMS OUTPUT TERMINAL POSTMAN CONSOLE Filter Code
[171] STATUS: 401 | ERROR: Invalid credentials
[172] STATUS: 401 | ERROR: Invalid credentials
[173] STATUS: 401 | ERROR: Invalid credentials
[174] STATUS: 401 | ERROR: Invalid credentials
[175] STATUS: 401 | ERROR: Invalid credentials
[176] STATUS: 401 | ERROR: Invalid credentials
[177] STATUS: 401 | ERROR: Invalid credentials
[178] STATUS: 401 | ERROR: Invalid credentials
[179] STATUS: 401 | ERROR: Invalid credentials
[180] STATUS: 401 | ERROR: Invalid credentials
[181] STATUS: 401 | ERROR: Invalid credentials
[182] STATUS: 401 | ERROR: Invalid credentials
[183] STATUS: 401 | ERROR: Invalid credentials
[184] STATUS: 401 | ERROR: Invalid credentials
[185] STATUS: 401 | ERROR: Invalid credentials
[186] STATUS: 401 | ERROR: Invalid credentials
[187] STATUS: 401 | ERROR: Invalid credentials
[188] STATUS: 401 | ERROR: Invalid credentials
[189] STATUS: 401 | ERROR: Invalid credentials
[190] STATUS: 401 | ERROR: Invalid credentials
[191] STATUS: 401 | ERROR: Invalid credentials
[192] STATUS: 401 | ERROR: Invalid credentials
[193] STATUS: 401 | ERROR: Invalid credentials
[194] STATUS: 200 | RESPONSE: {"message": "Login successful!", "user": {"id": 17, "username": "aqib", "userpassword": "193",
"created_at": "2025-04-22T00:05:04.308Z"}, "queryUsed": {"text": "SELECT * FROM Users WHERE username = $1 AND userpassword = $2",
"values": ["aqib", "193"]}}
```

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

```
minute.
[2990] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2991] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2992] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2993] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2994] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2995] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2996] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2997] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2998] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[2999] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.
[3000] STATUS: 429 | ERROR: Too many requests. You have been temporarily blocked for some
minute.

[Done] exited with code=0 in 14.529 seconds
```


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
```

6. Location-Based Access Restriction

Attack Description:

Requests from Israel were allowed by default.

Solution Implemented:

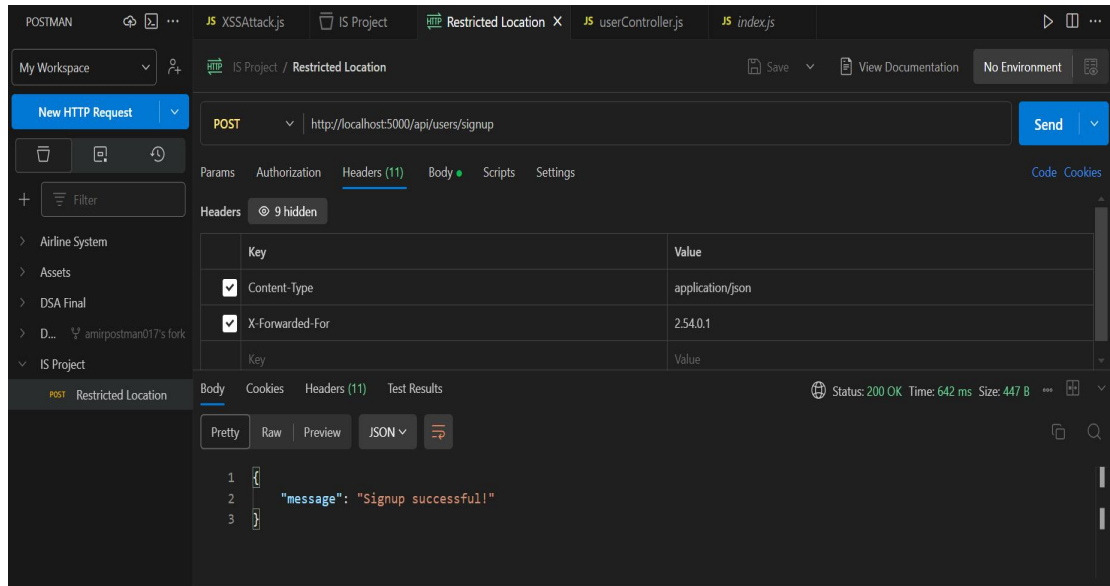
Geo-Blocking:

- Detected IP geolocation.

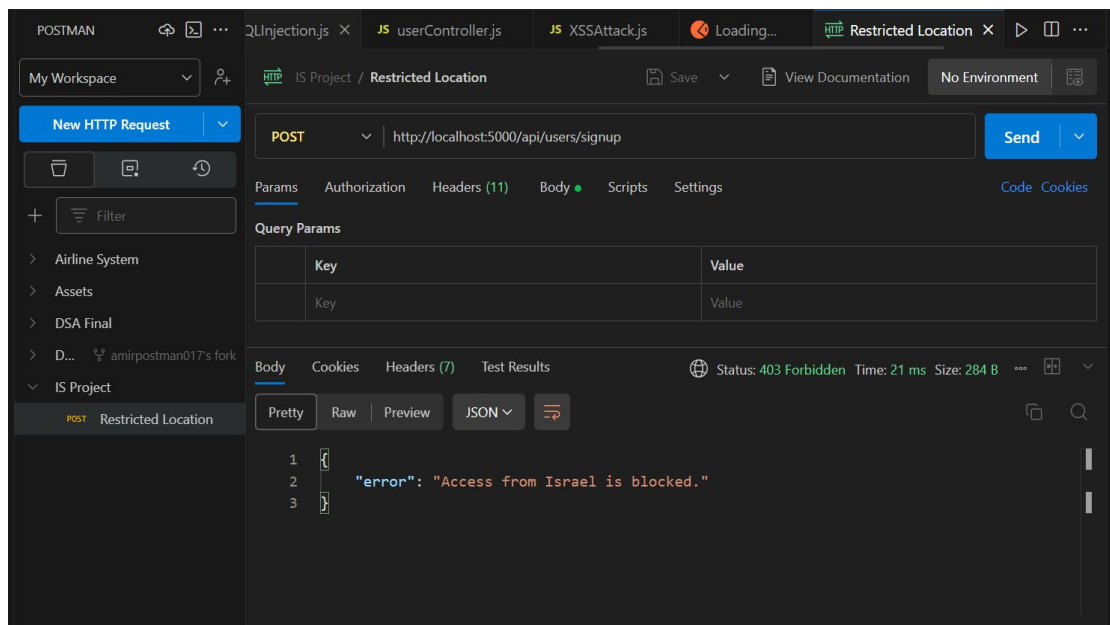
- Blocked requests from **Israel**.

Wireframes:

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



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



7. Password Encryption (Feistel Cipher)

Attack Description:

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

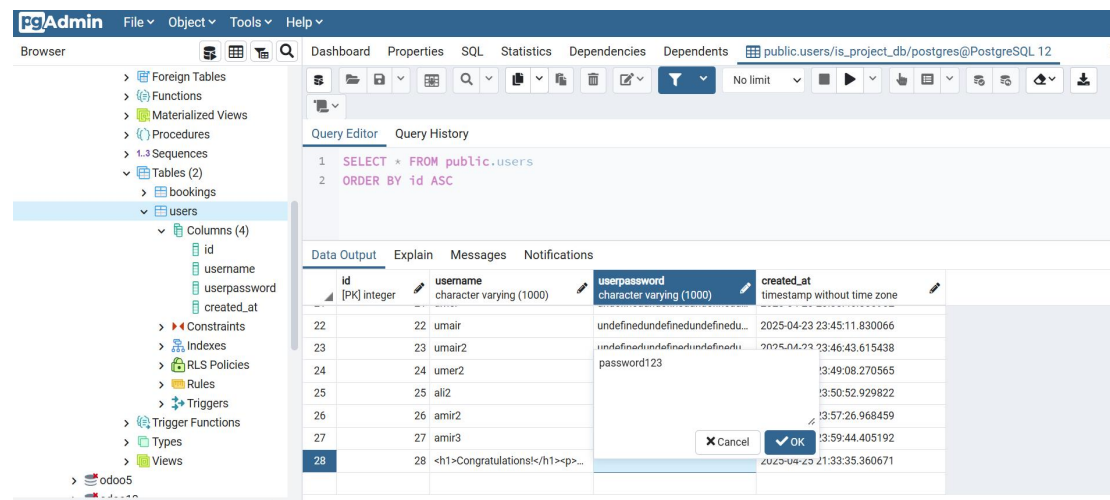
Solution Implemented:

Feistel Cipher Encryption:

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

Wireframes:

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



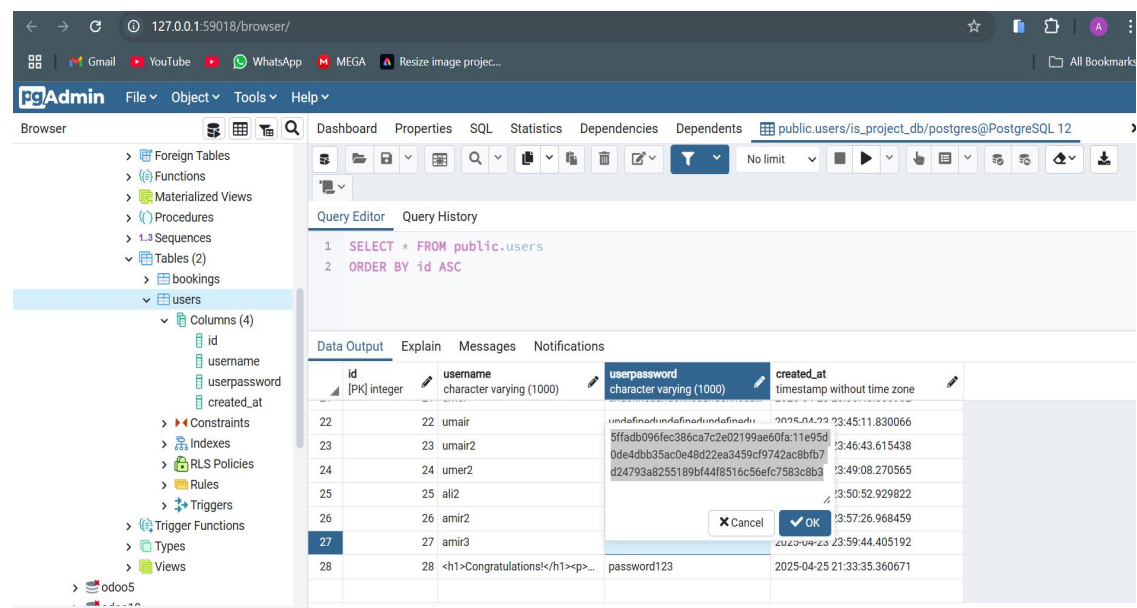
The screenshot shows the pgAdmin interface with a query editor containing the following SQL:

```
1 SELECT * FROM public.users
2 ORDER BY id ASC
```

The 'Data Output' tab displays the results of the query. The columns are: id, username, userpassword, and created_at. The data shows several users, with their passwords stored in plaintext.

id	username	userpassword	created_at
22	umair	undefinedundefinedundefined...	2025-04-23 23:45:11.830066
23	umair2	undefinedundefinedundefined...	2025-04-23 23:46:43.615438
24	umer2	password123	2025-04-23 23:49:08.270565
25	ali2		2025-04-23 23:50:52.929822
26	amir2		2025-04-23 23:57:26.968459
27	amir3		2025-04-23 23:59:44.405192
28	<h1>Congratulations</h1><p>...	password123	2025-04-23 21:33:35.360671

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



The screenshot shows the pgAdmin interface with the same query as before. The 'Data Output' tab displays the results of the query. The columns are: id, username, userpassword, and created_at. The data shows several users, with their passwords now stored as encrypted strings.

id	username	userpassword	created_at
22	umair	undefinedundefinedundefined...	2025-04-23 23:45:11.830066
23	umair2	5ffadb096fec386ca7c2e02199ae60fa11e95d...	2025-04-23 23:46:43.615438
24	umer2	0de4dbb35ac0e48d22ea3459cf9742ac8bf7...	2025-04-23 23:49:08.270565
25	ali2	d24793a8255189bf44f851ec56efc7583c8b3...	2025-04-23 23:50:52.929822
26	amir2		2025-04-23 23:57:26.968459
27	amir3		2025-04-23 23:59:44.405192
28	<h1>Congratulations</h1><p>...	password123	2025-04-25 21:33:35.360671

4. Conclusion & Lessons Learned

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