Vulnerability Assessment

Muhammad Haseeb
Cybersecurity Intern

DevelopersHub Corporation

🔒 Web Application Vulnerability Assessment Report

Target Application: Web_Project (GitHub)
Assessment Type: Static & Dynamic Analysis
Date of Assessment: 18 April 2025
Tools Used: OWASP ZAP, Browser Developer Tools, Manual Testing

Introduction

This report presents the results of a security assessment performed on a Node.js-based web application that implements user management features such as registration, login, and profile viewing. The objective was to identify and document common web application vulnerabilities and provide actionable remediation recommendations.

Environment Setup

The following steps were taken to prepare the application for testing:

Repository Cloning

```
git clone https://github.com/MH4S33B/Web_Project.git
cd Web_Project
```

Dependency Installation

```
npm install
```

Application Deployment

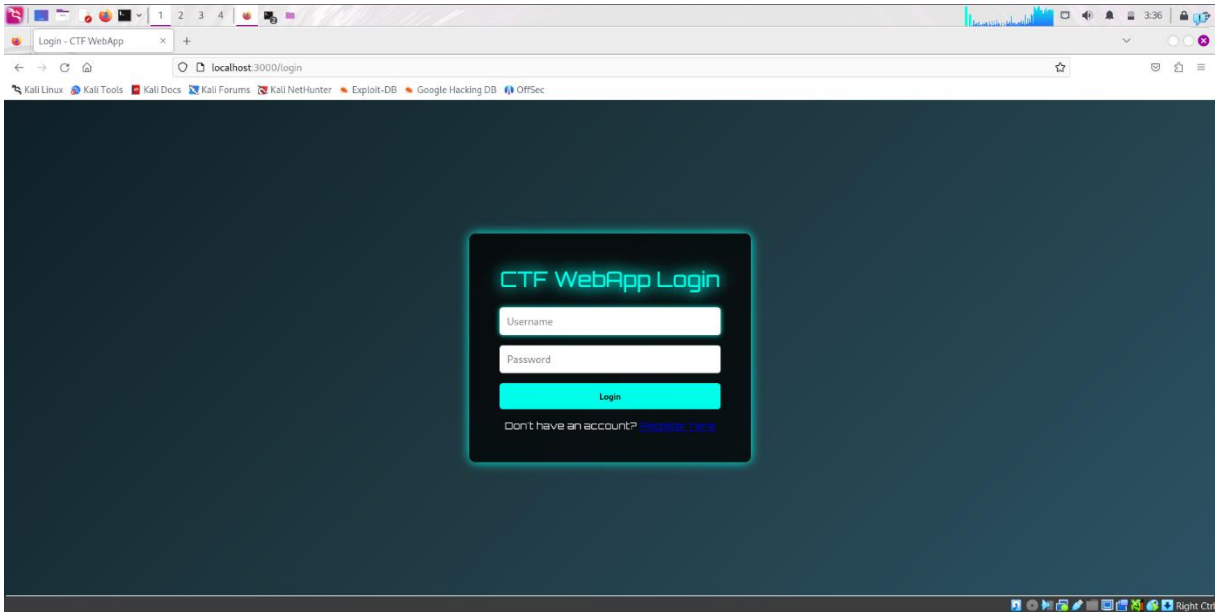
```
┌──(haseeb㉿vbox)-[~/Desktop/Web-Project]
└─$ npm start


> ctf-webapp@1.0.0 start
> node index.js

Executing (default): SELECT name FROM sqlite_master WHERE type='table' AND name='Users';
Executing (default): SELECT name FROM sqlite_master WHERE type='table' AND name='Sessions';
Executing (default): PRAGMA INDEX_LIST(`Users`)
Executing (default): PRAGMA INDEX_LIST(`Sessions`)
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_Users_1`)
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_Sessions_1`)
Executing (default): SELECT name FROM sqlite_master WHERE type='table' AND name='Sessions';
Executing (default): PRAGMA INDEX_LIST(`Sessions`)
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_Sessions_1`)
Server running on http://localhost:3000
```

Manual Functional Verification

The application was accessed at: http://localhost:3000



Features Verified:

- User Registration
- User Login
- Profile Access

Dummy accounts were created to validate business logic and navigation.

Vulnerability Assessment Methodology

The following tools and techniques were utilized:

Tool / Method	Purpose
OWASP ZAP	Automated dynamic vulnerability scanning
Browser Developer Tools	Manual DOM manipulation, script injection
Manual Testing	Authentication bypass, SQL injection checks

Identified Vulnerabilities

Cross-Site Scripting (XSS)

Risk Level: High
Test Method: Manual

Description:
An input such as <script>alert('XSS');</script> was injected into the "Name" field during registration. Upon form submission, the payload executed, confirming the application is susceptible to stored or reflected XSS.

Impact:

- Malicious script execution in user browsers
- Theft of session cookies
- Redirection to malicious websites

Recommendation:

- Sanitize and encode all user inputs before rendering them in HTML
- Use libraries like DOMPurify
- Implement Content Security Policy (CSP)

SQL Injection

Risk Level: Critical
Test Method: Manual

Payload Used:

```
admin' OR '1'='1
```

Description:
Login was successful without valid credentials, indicating the application directly incorporates user inputs into SQL queries without sanitization.

Impact:

- Full authentication bypass
- Possible data exfiltration or database modification
- Remote code execution in certain DB engines

Recommendation:

- Use parameterized queries or ORM frameworks
- Never concatenate user input directly into SQL strings

Missing Security Headers

Risk Level: Medium
Tool: OWASP ZAP
Description:
The application does not return security-related HTTP headers such as:

- X-Frame-Options
- Strict-Transport-Security
- X-XSS-Protection

Impact:

- Increased exposure to clickjacking
- Lack of HTTPS enforcement
- Absence of browser-based XSS filters

Recommendation:

- Implement middleware like helmet in Express:

```
const helmet = require('helmet');
app.use(helmet());
```

Sensitive File Enumeration

Risk Level: Medium
Tool: OWASP ZAP
Description:
Requests made to sensitive paths such as /WEB-INF/web.xml and /WEB-INF/applicationContext.xml returned 404 errors, suggesting such routes are accessible, even if the files don't exist.
Impact:

- Internal path disclosure
- Potential access to config or backup files if present

Recommendation:

- Restrict access to internal directories using server configurations
- Return 403 Forbidden for these types of paths

Broken Static Assets (404 Errors)

Risk Level: Low
Tool: OWASP ZAP

Description:
Multiple CSS and JS file requests resulted in 404 errors, indicating broken links or missing assets.

Impact:

- Poor user experience
- Possible entry point for social engineering or MITM attacks

Recommendation:

- Audit frontend resource paths
- Use a consistent static file delivery mechanism (e.g., express.static)

Summary of Findings

#	Vulnerability	Risk Level	Confirmed
1	Cross-Site Scripting (XSS)	High	✅
2	SQL Injection	Critical	✅
3	Missing Security Headers	Medium	✅
4	Sensitive File Enumeration	Medium	✅
5	Broken Asset References	Low	✅

Recommendations

Category	Recommended Action
Input Validation	Implement input sanitization and encoding
Authentication	Use prepared statements or ORM for database interactions
HTTP Headers	Use helmet to configure security headers
Access Control	Deny access to sensitive internal paths
Password Handling	Store hashed passwords using bcrypt/scrypt
Asset Management	Ensure all static paths are valid and protected

Conclusion

The assessment identified multiple high-impact security flaws in the application, particularly in areas related to input handling and database interaction. Immediate remediation is advised to prevent exploitation.

Security Posture: 🚨 At-Risk — Critical issues found

It is highly recommended that the application undergoes a full code review and follow-up penetration testing once fixes are implemented.

Connect With Me

- 💼 **LinkedIn**
- 📝 **Medium**
- 📧 **Email: mhaseebraja2006@gmail.com**
- 💻 **GitHub**