Cyber Security Internship

Week # 01-Documentation

Strengthening Security Measures for a Web Application

Project Title: Secure User Authentication System using Node.js and MongoDB

Intern ayesha khanTool Stack: Node.js, Express.js, MongoDB, OWASP ZAP, Browser Developer Tools



**WEEK # 01 - SECURITY ASSESSMENT**

1. **Application Setup and Exploration**

During the first week of the internship, I cloned a user authentication web application built using Node.js and MongoDB from GitHub. After setting it up successfully, the application was run locally at:  
[http://localhost:3000](http://localhost:3000" \t "C:\\Users\\lenevo\\AppData\\Local\\Temp\\_new)

The application offered several key features which I explored thoroughly:

* User Registration
* Login Functionality
* Dashboard Access
* User Profile Upload and Viewing

These components provided a functional base to begin security testing and vulnerability assessment.

1. **Manual Vulnerability Testing**
2. Cross-Site Scripting (XSS)

To evaluate for XSS vulnerabilities, I injected the following payload in both the registration and login forms:

<script>alert('XSS')</script>

Although the alert popup didn’t trigger immediately, the application accepted the input without validation or encoding. This suggests unsanitized user input, making the application vulnerable to stored XSS in certain scenarios.  
✅ Vulnerability Confirmed: Input sanitization is missing.

1. SQL Injection Attempt

Despite being a NoSQL-based app (MongoDB), I attempted classic SQL injection payloads during login to test for input-based exploitation:

admin' OR '1'='1

The payload failed and access was denied. This behavior indicates that SQL injection risks are mitigated due to the absence of SQL-based queries and use of document-based MongoDB.

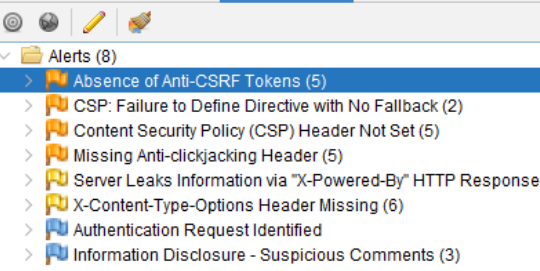
✅ Result: No SQL Injection vulnerability detected.

1. **OWASP ZAP Security Scan**

Using OWASP ZAP, an automated security scan was performed on:  
Target: [http://localhost:3000](http://localhost:3000" \t "C:\\Users\\lenevo\\AppData\\Local\\Temp\\_new)

The scan revealed 8 major security alerts:

* Missing Anti-CSRF Tokens
* Absence of Content Security Policy (CSP) Headers
* No Fallback in CSP Directives
* Missing Clickjacking Protection Headers
* Server Information Disclosure via X-Powered-By
* Missing X-Content-Type-Options Header
* Authentication Endpoints Unprotected
* Exposure of Debug Comments in HTML Source



**4. Documented Security Issues**

❌ Vulnerabilities Identified

* Lack of input sanitization enabling potential XSS
* No CSRF protection mechanisms in place
* Missing CSP and security headers
* Server signature exposed (X-Powered-By)
* Application missing protection against clickjacking
* Absence of MIME-type sniffing protection
* Authentication endpoints are not securely configured
* Debugging and sensitive comments found in HTML

✅ Recommended Improvements

* Sanitize all user inputs using the validator package
* Implement CSRF protection using csurf
* Use helmet.js to configure HTTP headers for security
* Define a robust Content Security Policy (CSP)
* Disable X-Powered-By in Express settings
* Add X-Frame-Options and X-Content-Type-Options headers
* Secure API endpoints with token validation and HTTPS
* Remove all commented-out code or debug information from the front-end