Secure Coding Review – Task 3

**Name:** Aksin Abdullayev

**Date:** 2025-09-06

**Language & Framework:** Node.js / Express.js

## **1. Introduction / Objective**

The purpose of this report is to identify security vulnerabilities in a Node.js web application and suggest recommendations for secure coding practices. The application analyzed is a simple login/register/dashboard website built with Node.js, Express, EJS, and CSS. The goal is to perform manual and practical code review, identify vulnerabilities, and document remediation steps.

## **2. Scope**

* Manual code inspection
* Practical testing in browser and terminal
* Use of static analysis tools (e.g., npm audit)
* Screenshot documentation with detailed explanation for each vulnerability

## **3. Methodology**

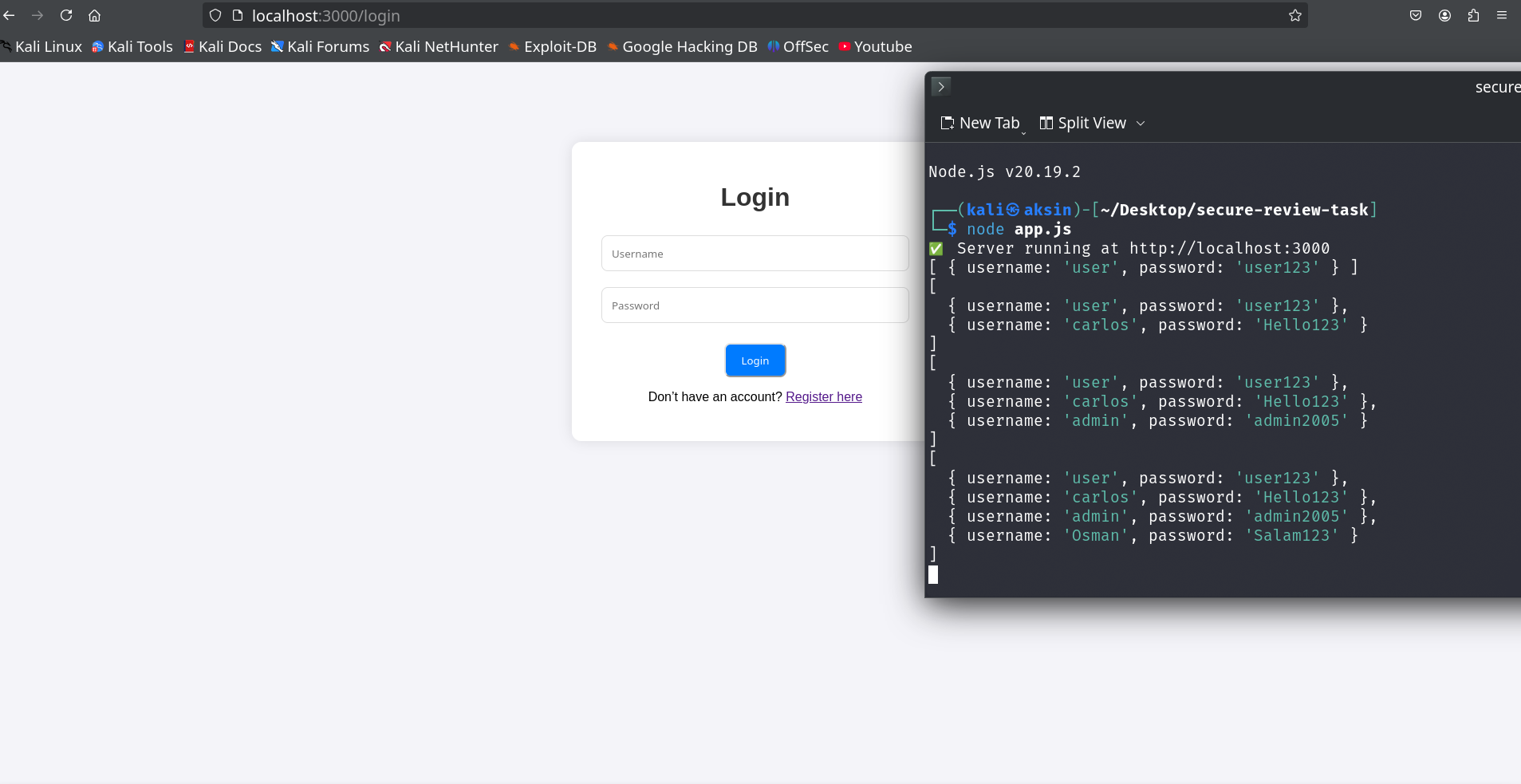
1. Launch the application locally
2. Register and login as a test user
3. Observe vulnerabilities such as plaintext password storage, session handling issues, and input validation weaknesses
4. Document each finding with a screenshot and detailed explanation
5. Provide remediation recommendations

## **4. Findings**

### **Vulnerability 1: Plaintext Passwords**

**Description:** User passwords are stored in plain-text in the server memory (users array in app.js). This exposes all passwords if server access is obtained.

**Screenshot:**



**Detailed Explanation:**

* When a new user registers, the terminal shows the users array containing both username and password in plain text.
* This is highly insecure because an attacker who gains access to server memory or logs can see all passwords.
* Screenshot should show terminal with users array.

**Risk Level:** High

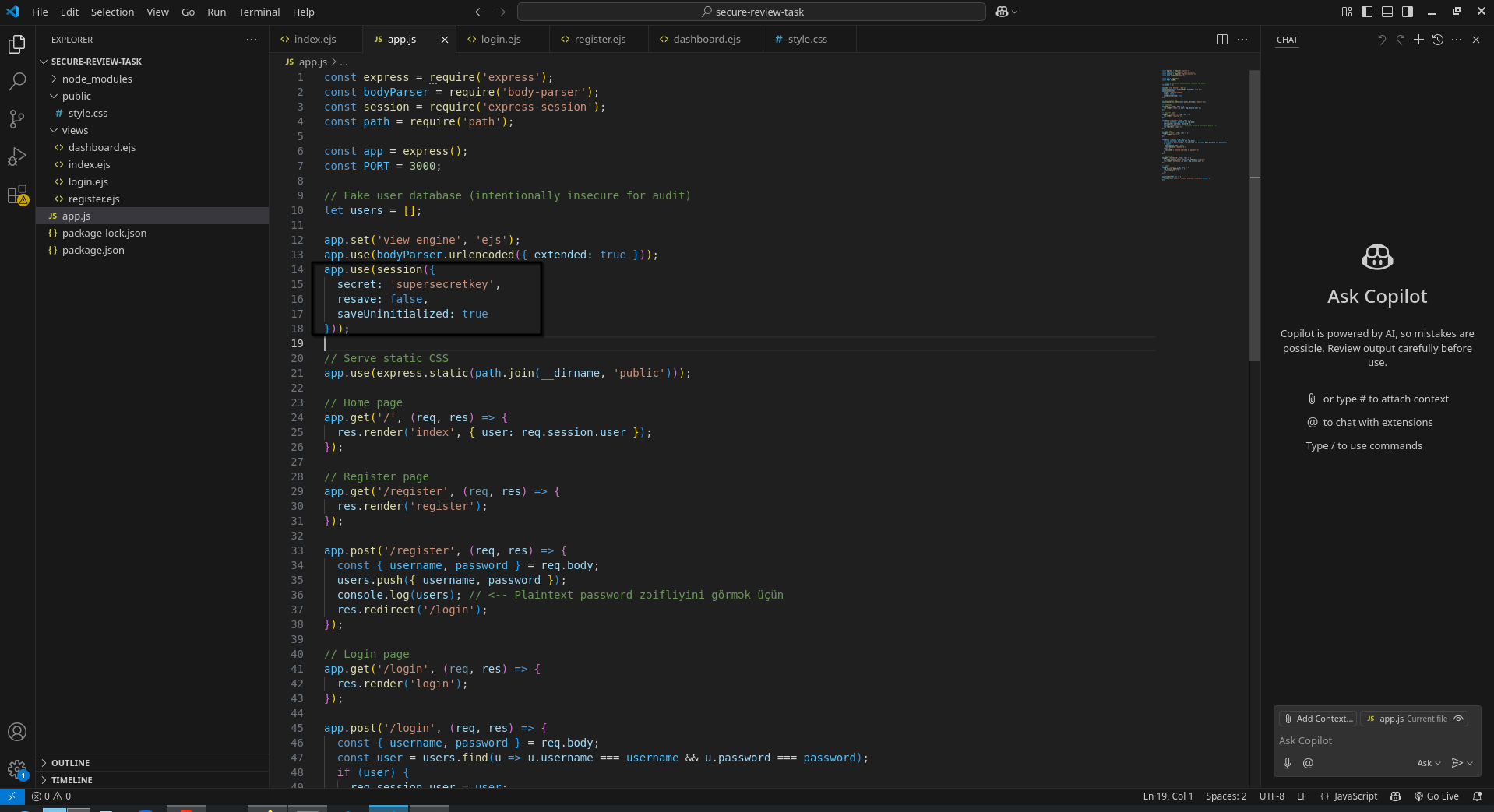
**Recommendation:**

* Use bcrypt or argon2 to hash passwords before storing them. Example: const hashedPassword = await bcrypt.hash(password, 10);

### **Vulnerability 2: Hardcoded Session Secret**

**Description:** The session secret is hardcoded in app.js, which could lead to session hijacking if code is leaked.

**Screenshot:**



**Detailed Explanation:**

* Session secret is currently: supersecretkey
* If attacker obtains source code, they can craft valid session cookies.
* Screenshot should show session configuration in app.js.

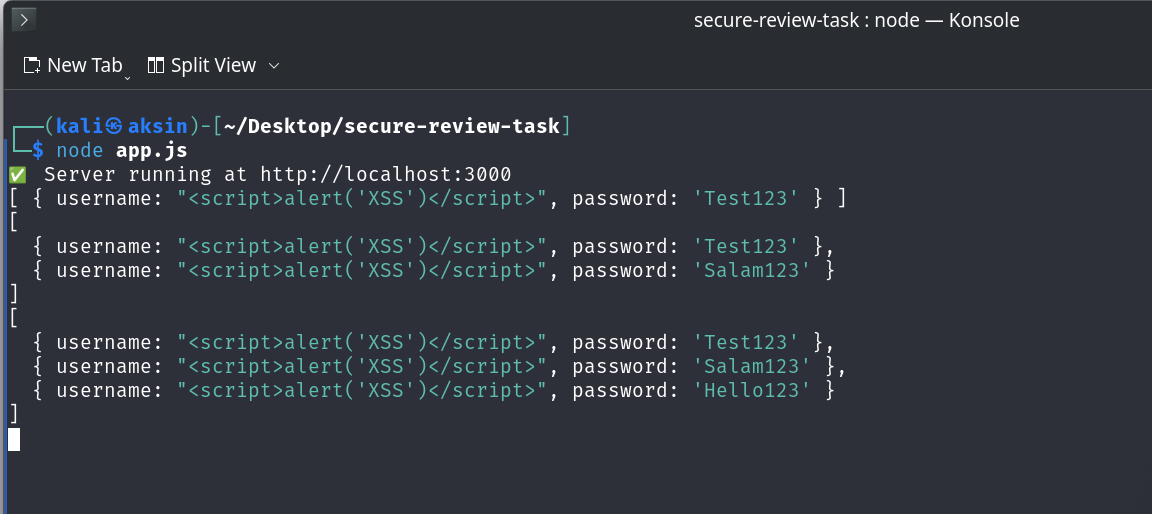
**Risk Level:** High

**Recommendation:** Move session secrets to environment variables using .env file and process.env.SECRET\_KEY.

### **Vulnerability 3: No Input Validation**

**Description:** User inputs are not validated or sanitized, making XSS or injection attacks possible.

**Screenshot:**



**Detailed Explanation:**

* User input is not validated or sanitized before being stored.
* Payload <script>alert('XSS')</script> entered in the username field is stored on the server and could execute in the browser if unsafe rendering is used.
* Screenshot should show the payload stored in the server (e.g., terminal console.log(users)) and/or the result in the browser if rendered unsafely.

**Risk Level:** Medium

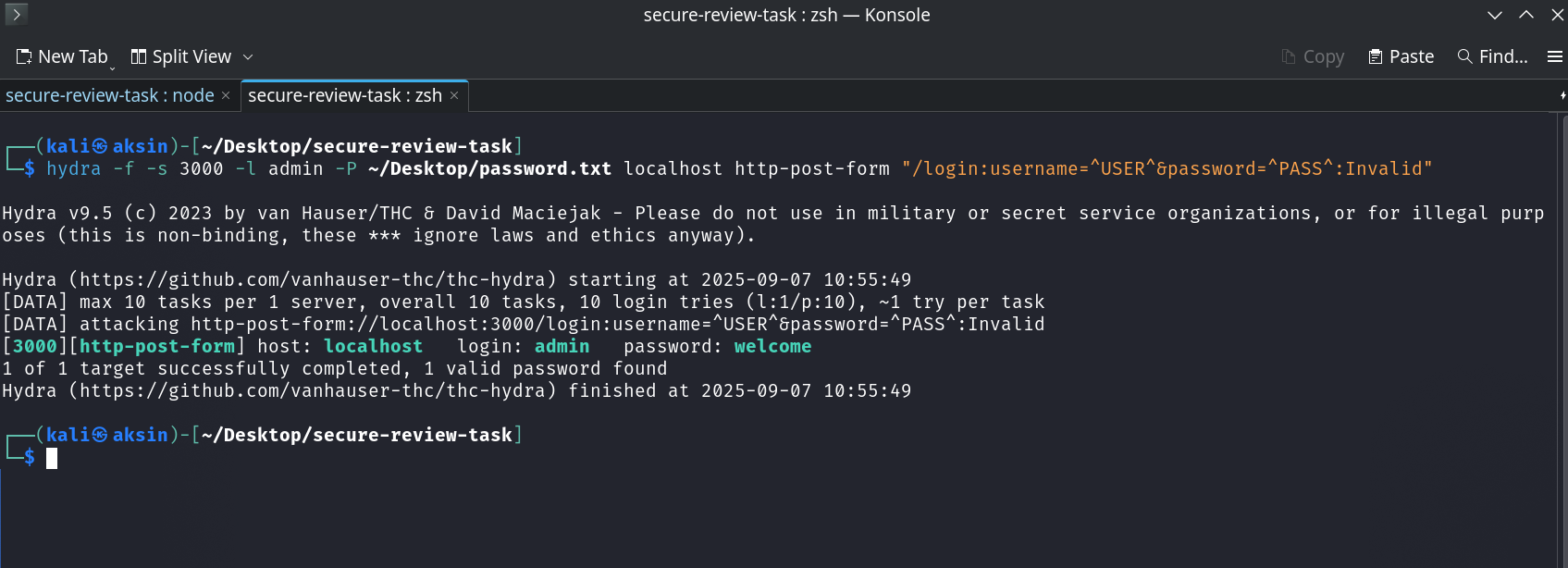
**Recommendation:**

* Always validate user input on the server side.
* Sanitize output before rendering in HTML using libraries like validator.js or escape-html.
* Avoid using unsafe rendering (<%- %>) in production.

### **Vulnerability 4: No Rate Limiting**

**Description:** Login endpoint allows unlimited attempts, enabling brute-force attacks.

**Screenshot:**



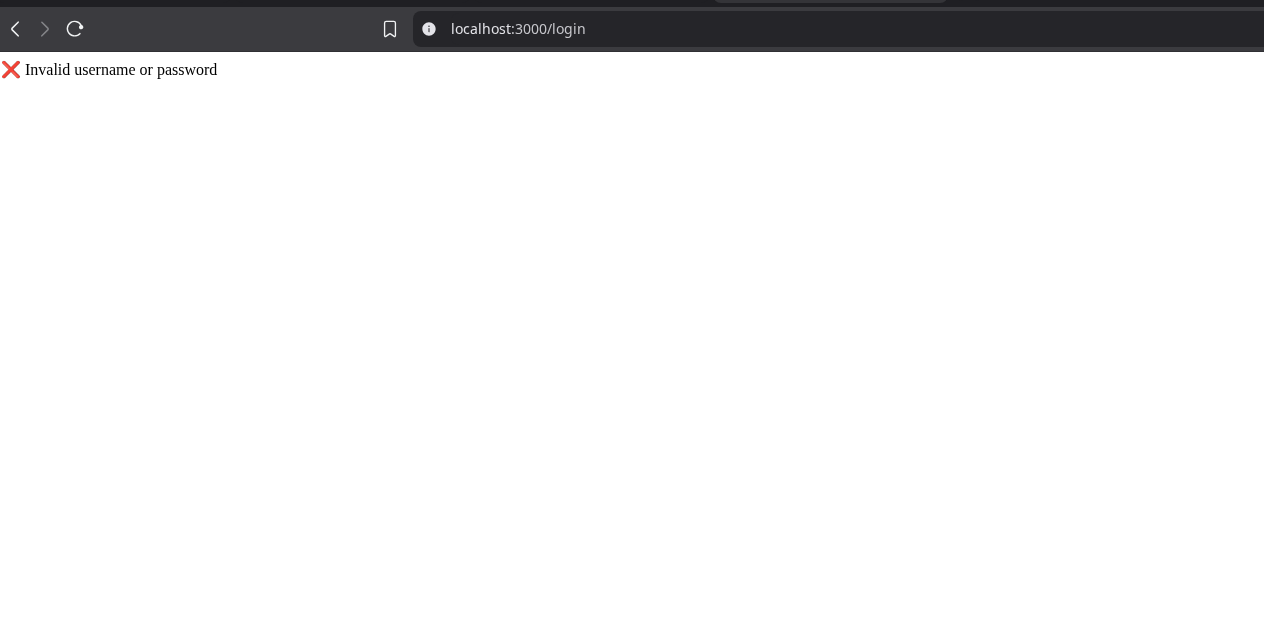
**Detailed Explanation:**

* Tested by submitting multiple incorrect login attempts.
* Risk Level: Medium
* Recommendation: Implement rate limiting using express-rate-limit to limit failed login attempts.
* Screenshot should show repeated attempts in browser or network console.

### **Vulnerability 5: Error Handling**

**Description:** Application displays raw error messages, potentially revealing sensitive information.

**Screenshot:**



**Detailed Explanation:**

* Example: Invalid login shows message directly, may reveal logic or system info.
* Risk Level: Low
* Recommendation: Add custom error handling middleware and generic error messages to avoid leaking sensitive info.
* Screenshot should show error message displayed in browser.

## **5. Tools Used**

* Node.js / Express.js
* EJS & CSS
* npm audit
* Browser developer tools (console & network)
* Manual inspection of app.js

## **6. Conclusion / Recommendations**

The application contains several security vulnerabilities, including plaintext password storage, hardcoded session secrets, lack of input validation, absence of rate limiting, and improper error handling. Recommendations:

* Hash passwords using secure algorithms before storing.
* Move session secrets to environment variables.
* Validate and sanitize all user inputs.
* Implement rate limiting on login or sensitive endpoints.
* Add proper error handling middleware with generic messages.