**DVNA Security Enhancement Report**

**Week 1 to Week 3 Summary**

This report outlines the process of identifying, exploiting, and ultimately securing vulnerabilities in the Damn Vulnerable NodeJS Application (DVNA). Over the course of three weeks, the approach moved from hands-on exploration of web security flaws to actively fixing and validating those fixes. Below is a week-by-week breakdown of tasks, outcomes, and implementations.

**Week 1: Exploring the Vulnerabilities**

The first week focused on understanding DVNA's architecture and performing basic exploitation techniques. After setting up the environment and database using npm start and SQLite, the initial steps involved finding and exploiting security issues in the app.

The main vulnerability targeted was Cross-Site Scripting (XSS). Input fields such as those on login and registration pages were tested for XSS by injecting payloads. One successful example allowed for the injection and execution of malicious scripts, confirming that the application lacked proper sanitization or output encoding.

Session hijacking was also explored. By inspecting cookies using browser developer tools (like Firefox’s storage tab), the session ID (connect.sid) was captured. This was later reused to impersonate a valid user, proving the lack of secure session handling.

Basic usage of OWASP ZAP was also attempted to automate attack scanning, though performance limitations and application hang-ups were encountered.

**Week 2: Fixing the Vulnerabilities and Strengthening the App**

Week 2 focused on applying fixes to the known vulnerabilities and improving the codebase with basic security best practices.

First, input validation was introduced using the validator npm library. Email formats and password length were verified in files like authHandler.js. The code was updated to ensure that fields like login and email were not only present but also safely structured.

Password storage was also hardened. Existing bcrypt usage was extended, and password checks were enforced with minimum length rules (at least 6 characters). Feedback messages were made more specific so users knew if their input failed due to format or mismatch.

Helmet.js was implemented in server.js to set secure HTTP headers. This provided protection against common vulnerabilities like content-sniffing and clickjacking.

All these changes were tested by trying invalid form submissions and monitoring console logs and redirects. The app correctly flagged weak passwords and invalid emails and handled them gracefully.

**Week 3: Testing, Logging, and Final Review**

In the final week, the emphasis was on confirming the security fixes and adding observability to the system.

Penetration testing was done using tools like **Nikto** and **Nmap**. Nikto was run against the local DVNA server (http://localhost:9090), which identified a few minor issues like missing security headers and open login/register endpoints but confirmed the absence of critical vulnerabilities.

To monitor potentially malicious activity, **Winston** was integrated into the app. Logs were configured to record both console outputs and persistent logs in a file (security.log). Suspicious actions, such as untrusted input in routes like /ping, were now being logged with appropriate context.

The forgotten password functionality was also secured. A reset token was generated using md5(login) and embedded in a reset link. This token was validated before rendering the reset form or updating the password, preventing misuse of the reset endpoint.

By the end of the week, a security checklist was created and cross-verified:

* Input validation: Added using validator
* Password hashing: Enforced with bcrypt
* Secure HTTP headers: Enabled with Helmet
* Session management: Already in place
* Logging: Implemented with Winston
* Testing: Conducted via Nikto and Nmap

**Final Thoughts**

Over these three weeks, the DVNA application was transitioned from a vulnerable sandbox environment to a more secure and structured platform. Each week built upon the last—starting with vulnerability discovery, moving into real security patching, and concluding with validation and logging.

The journey not only strengthened the application itself but also deepened hands-on experience with both offensive and defensive web application security. These practices reflect real-world strategies that developers and security engineers rely on to secure modern applications.