**Week 2: Implementing Security Measures in DVNA (Damn Vulnerable NodeJS Application)**

**Overview**

During Week 2 of our internship, we focused on **identifying**, **fixing**, and **understanding** key vulnerabilities present in the DVNA (Damn Vulnerable NodeJS App). The goal was to patch common security risks outlined by OWASP and learn how to secure Node.js applications using real-world tools and libraries.

**🔧 What We Did**

**1. Input Validation and Sanitization**

We installed the validator library to help us clean and verify user inputs across the application.

* npm install validator

**Where we applied it:**

* core/authHandler.js (during password resets)
* core/passport.js (during login and signup)

**Fix Example:**

* if (!validator.isEmail(req.body.email)) {

return res.status(400).send('Invalid email format');

}

**✅ Result:**

* Prevented malformed or malicious data from entering the system.
* Reduced XSS and injection risks by ensuring strict input formats.

**2. Password Hashing with Bcrypt**

Passwords are now hashed securely using bcrypt.

* npm install bcrypt

**Code Example:**

* const hashedPassword = await bcrypt.hash(password, 10);

**Where:** Used in both registration and password reset flows.

**✅ Result:**

* Prevented plaintext password storage.
* Mitigated risks related to credential leakage.

**3. Added Helmet.js for Security Headers**

We used helmet to help secure our Express app by setting various HTTP headers.

* npm install helmet

**Code Added to server.js:**

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

**✅ Result:**

* Blocked many basic attacks like clickjacking and sniffing.
* Improved browser-level security policy enforcement.

**4. Fixing Sensitive Data Exposure**

The admin API previously exposed all user data, including passwords.

**Fix in appHandler.js:**

* db.User.findAll({

attributes: ['id', 'name', 'email', 'login', 'role'] // password excluded

})

**✅ Result:**

* Removed sensitive fields from public APIs.

**🔬 Testing & Results**

We used the browser and Postman to test validation, authentication, and API behavior:

* ✅ Tried invalid emails — received proper error messages.
* ✅ Attempted weak passwords (<6 chars) — blocked with clear feedback.
* ✅ Checked API endpoints like /app/admin/usersapi — no password field exposed.
* ✅ Confirmed headers added via browser DevTools (e.g., X-Frame-Options, Content-Security-Policy).

**🚧 Challenges Faced**

* Ran into issues with helmet due to Node.js version — resolved by switching syntax.
* Faced disk space errors when installing packages — cleaned up unnecessary files.
* Validating input while maintaining user-friendly error messages took time to tweak.

**✅ Summary**

This week was all about **secure coding practices**. We:

* Hardened input fields and added validation.
* Secured passwords with hashing.
* Introduced headers to guard against common attacks.
* Prevented sensitive data leakage in APIs.

These changes made the DVNA app significantly more secure and aligned with OWASP recommendations.

Next week, we’ll dive deeper into vulnerabilities like SQL Injection, Command Injection, and Insecure Deserialization — and explore how to fix them at both the code and infrastructure level.