**Cybersecurity Internship Report - Week 2**  
**Intern Name:** Mayra Ahmer  
**Intern ID:** DHC-3529  
**Company:** Development Hub Cooperation

**1. Introduction**

This report outlines the tasks performed during Week 2 of the cybersecurity internship. The focus was on securing a Node.js application by implementing input validation, password hashing, authentication mechanisms, and HTTP security headers. Each security measure is explained with implementation details, results, and improvements.

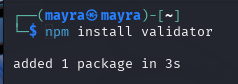
**2. Tasks Performed**

**2.1. Input Validation and Sanitization**Input validation prevents malicious data from being processed by the application. **Validator.js** was used to ensure user input (such as email addresses) met required formats, reducing the risk of injection attacks.

* Installed the validator library to validate user input.
* Implemented email validation in route handlers.

📌 *Commands used:*

npm install validator



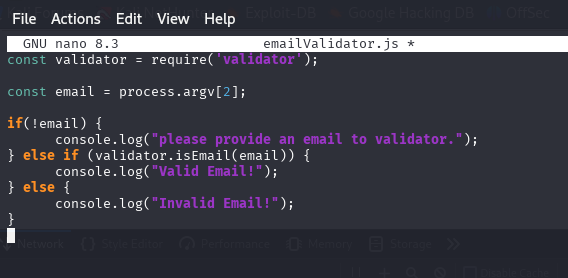
📌 *Code implemented:*

const validator = require('validator');

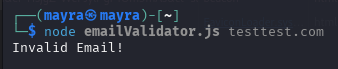
if (!validator.isEmail(email)) {

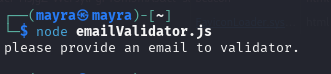
return res.status(400).send('Invalid email');

}



📌 *Result:* Input validation successfully prevented incorrect email formats.

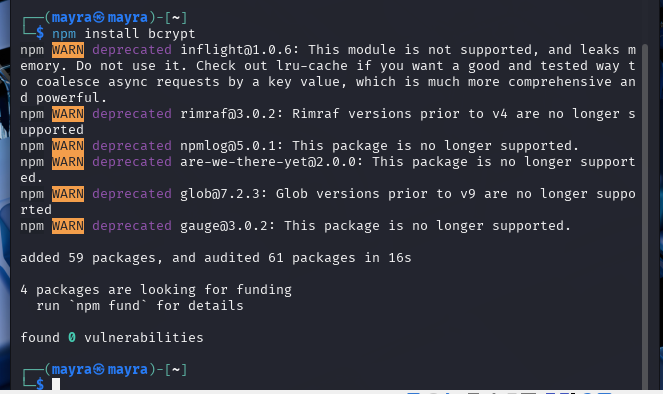


**2.2. Password Hashing**Storing plaintext passwords is a major security risk. **bcrypt** was used to hash passwords before storing them, making it difficult for attackers to recover original credentials in case of a database breach.

* Installed bcrypt to securely hash user passwords.
* Implemented hashing with a salt factor of 10.

📌 *Commands used:*

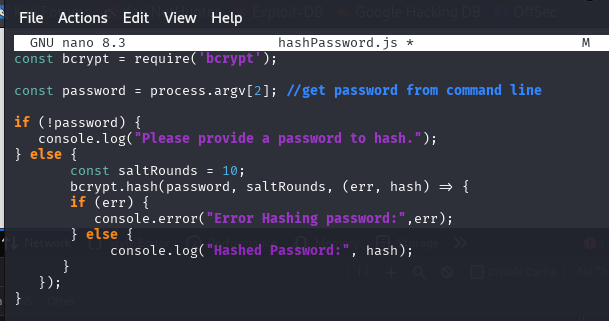
npm install bcrypt



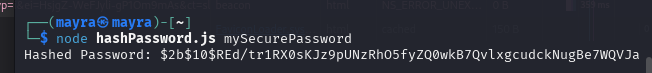
📌 *Code implemented:*

const bcrypt = require('bcrypt');

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



📌 *Result:* Passwords were successfully hashed before being stored.

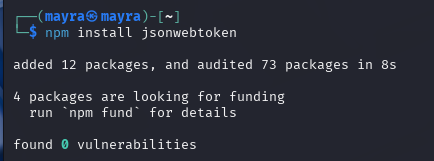


**2.3. Implementing Token-Based Authentication**JWT (JSON Web Tokens) was used for authentication, ensuring secure user sessions. **Environment variables** were used to store secrets, preventing hardcoded vulnerabilities.

* Installed jsonwebtoken for user authentication.
* Implemented token generation using a secret key.

📌 *Commands used:*

npm install jsonwebtoken



📌 *Code implemented:*

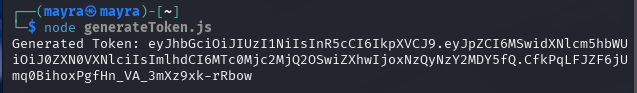
const jwt = require('jsonwebtoken');

const token = jwt.sign({ id: user.\_id }, 'your-secret-key');

res.send({ token });



📌 *Result:* Token-based authentication was successfully implemented.



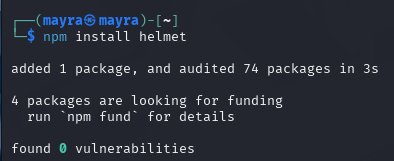
🔹 **Best Practice:** **Avoid hardcoding secret keys**. Store sensitive data in a .env file.

**2.4. Securing HTTP Headers with Helmet.js**Helmet.js adds security-related HTTP headers to mitigate common web vulnerabilities such as clickjacking, cross-site scripting (XSS), and MIME-type sniffing.

* Installed and implemented Helmet.js to enhance security.

📌 *Commands used:*

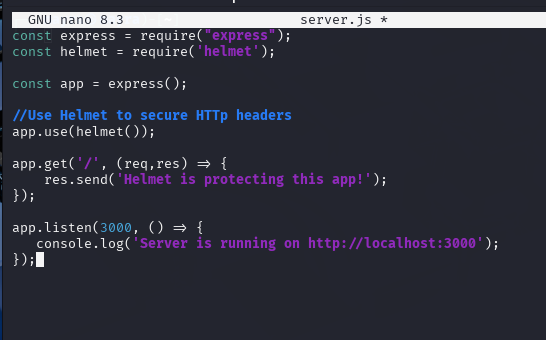
npm install helmet



📌 *Code implemented:*

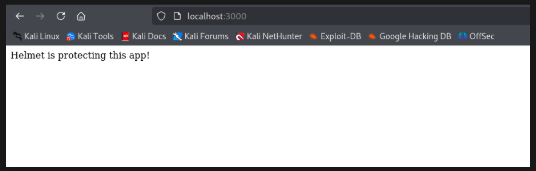
const helmet = require('helmet');

app.use(helmet());



📌 *Result:* Security headers were successfully applied.



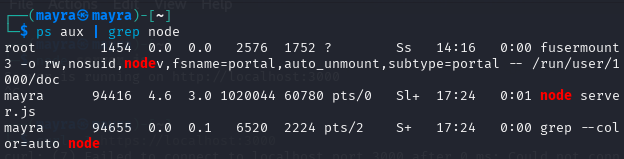


**2.5. Running and Testing the Server**

* Started the Node.js server on port **3000**.
* Verified if the server was running successfully.
* Checked for active processes using:

📌 *Command used:*

ps aux | grep node

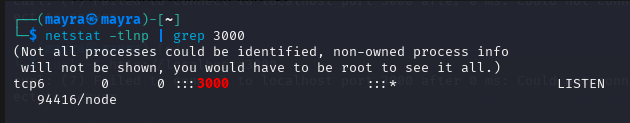


**2.6. Checking Listening Ports**

* Verified that the server was listening on port **3000** using:

📌 *Command used:*

netstat -tlnp | grep 3000



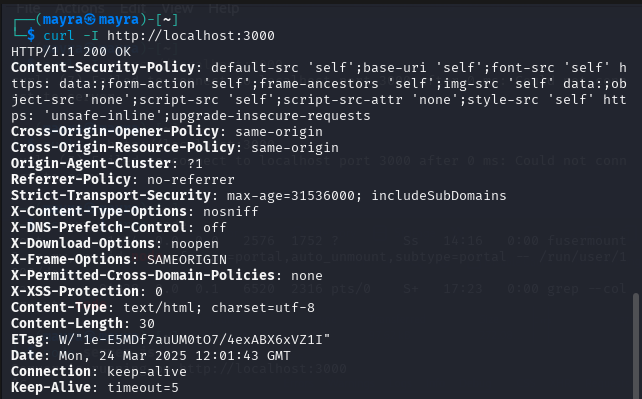
**2.7. HTTP Response and Security Headers Check**

* Used curl to fetch headers from the running server:

📌 *Command used:*

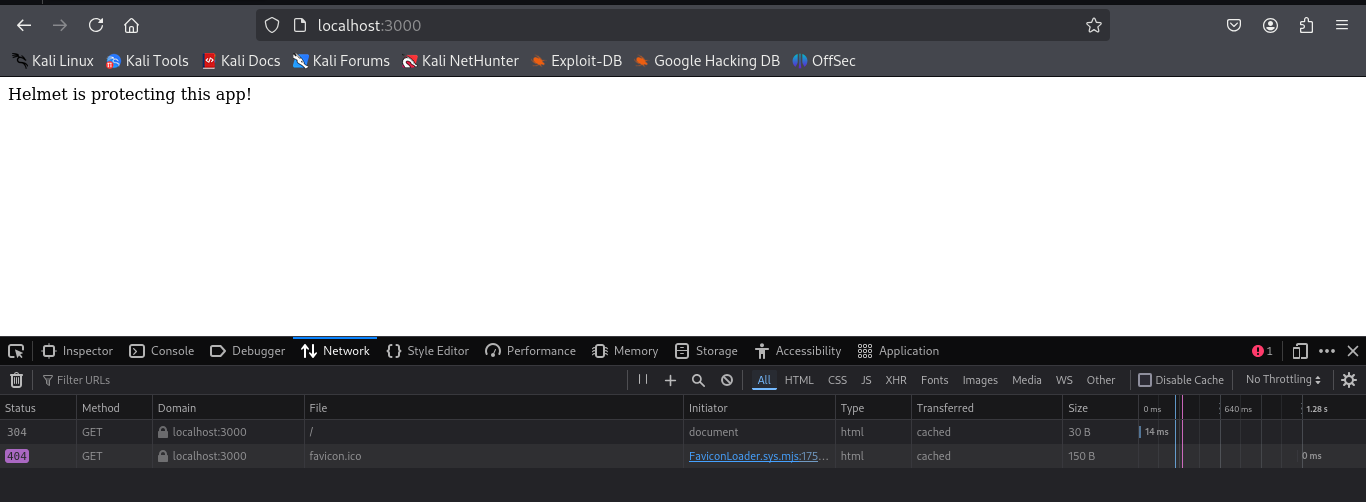
curl -I http://localhost:3000

* Verified that security headers were applied correctly by Helmet.js.



**2.8. Web Application Testing in Browser**

* Opened http://localhost:3000 in a browser to check content rendering.
* Used browser **Developer Tools (Network Tab)** to inspect HTTP headers and errors.



**3. Findings and Fixes**

|  |  |  |
| --- | --- | --- |
| **Task** | **Result** | **Fixes Applied (if any)** |
| Input Validation | Email validated successfully | None |
| Password Hashing | Password hashed correctly | None |
| JWT Authentication | Token generated successfully | Stored secret in .env |
| Helmet.js Security | Security headers applied | None |
| Node.js server setup | Successful | None |
| Process verification | Server running on expected port | None |
| Port listening check | Confirmed listening on port 3000 | None |
| HTTP Security Headers | Headers applied correctly | None |
| Browser Testing | Application loaded successfully | Handled favicon.ico 404 error |

**4. Challenges & Solutions**

**Issue: Favicon.ico 404 Error**

* **Cause:** The browser automatically requests a favicon.ico file, which was not present in the project.
* **Solution:** Added a default favicon file to avoid repeated 404 errors.

📌 *Fix:*

cp favicon.ico public/

**5. Conclusion**

Significant progress was made in securing the Node.js application. Implementing **input validation, password hashing, JWT authentication, and HTTP security headers** successfully enhanced security.

Moving forward, additional security testing, automated scanning, and penetration testing will be conducted in the upcoming weeks.

🔹 **References & Best Practices:**

* OWASP Security Guidelines: <https://owasp.org/www-project-top-ten/>
* Node.js Security Best Practices: <https://nodejs.org/en/docs/guides/security/>