FINAL INTERNSHIP SECURITY REPORT (WEEKS 4–6)

Project Title: CyberShield V3: Advanced Security Portal

Intern Name: Syed Zunair Hussain

Environment: Kali Linux / Node.js / Docker

Submission Date: December 30, 2025

1. PROJECT OVERVIEW

This report summarizes the security transformation of the CyberShield Intern Portal. Over three weeks, the project moved from a vulnerable "legacy" state to a hardened, production-ready architecture. The work involved building defensive perimeters, conducting ethical hacking to find weaknesses, and performing final audits for secure deployment.

2. WEEK 4: THREAT DETECTION & API HARDENING

2.1 Intrusion Detection (Fail2Ban)

I implemented real-time monitoring to stop automated attacks.

- **Setup:** Configured Fail2Ban to watch my security.log (generated by Winston).
- **Logic:** If an IP triggers 5 AUTH_FAILURE warnings within 2 minutes, it is banned for 10 minutes.
- **Verification:** I simulated a brute-force attack; the system logged the failures, and the IP was successfully blocked at the firewall level.

2.2 API Security & Rate Limiting

To prevent script-based attacks, I added express-rate-limit.

- **Implementation:** Restricted the /api/ routes to 10 attempts per 10 minutes.
- **CORS:** Configured Cross-Origin Resource Sharing to ensure only authorized frontend origins can talk to the API.
- **Code Logic:** If a user exceeds the limit, the server returns a 429 TOO_MANY_REQUESTS error.

2.3 Security Headers & CSP

I used Helmet.js to set secure HTTP headers.

- **HSTS:** Enforced HTTPS to prevent protocol downgrade attacks.
- **Content Security Policy (CSP):** Implemented a strict CSP. Since I used Tailwind CSS via

CDN, I generated a **Nonce** (number used once) for every request to ensure only my specific scripts can execute, blocking malicious XSS injections.

3. WEEK 5: ETHICAL HACKING & VULNERABILITY MITIGATION

3.1 Reconnaissance

Before patching, I acted as an attacker using Kali Linux tools:

- **Nmap:** nmap -sV localhost -p 3000 confirmed the service was running and identified the version.
- **Nikto:** Identified that the early version was missing critical anti-clickjacking and XSS headers.

3.2 SQL Injection (SQLi)

- **The Attack:** I used sqlmap against the /api/login-vulnerable route. SQLMap confirmed the email field was vulnerable and could be used to dump the entire database.
- **The Fix:** I migrated the code to **Prepared Statements**. Instead of using string templates, I used db.get("SELECT * FROM users WHERE email = ?", [email], ...). This ensures the database treats input as data, not executable code.

3.3 Cross-Site Request Forgery (CSRF)

- **The Attack:** I used **Burp Suite** to intercept a request to /api/clear-logs. I proved that without a token, a malicious site could trick an admin into wiping logs.
- **The Fix:** Integrated csurf middleware. Now, every state-changing request (POST) requires a valid CSRF-Token in the header, which must match a secure HttpOnly cookie.

4. WEEK 6: AUDITS & SECURE DEPLOYMENT

4.1 Advanced Security Audits

I ran multiple industry-standard tools to verify compliance:

- **OWASP ZAP:** Flagged the SQLi in the vulnerable route as "Critical." After my patches, I re-ran the scan to confirm the "Secure" routes passed.
- **Nikto:** Verified that Helmet.js successfully added all required security headers.
- **Winston Logging:** Verified that all security events (successful logins, SQLi attempts, and CSRF failures) are written to security.log for auditing.

4.2 Dependency & Container Scanning

- **Snyk:** Ran npx snyk test to check for vulnerable npm packages. I updated bcrypt and inflight to resolve high-severity issues.
- **Docker Hardening:** I used node:18-slim as a base image to reduce the attack surface.
- **Container Scan:** snyk container test cybersec-app:v2 resulted in **0 Critical / 0 High**

vulnerabilities.

4.3 OWASP Top 10 Compliance

The project now addresses the following:

- **A01:2021-Broken Access Control:** Implemented CSRF and CORS.
- **A02:2021-Cryptographic Failures:** Used Bcrypt with 10 salt rounds for password hashing.
- **A03:2021-Injection:** Replaced vulnerable queries with Parameterized Queries.
- **A09:2021-Security Logging:** Integrated Winston for persistent auditing.

5. FINAL CONCLUSION

The CyberShield V3 Portal has undergone a full security lifecycle. By combining manual penetration testing (Burp Suite, SQLMap) with defensive coding (CSP, Rate Limiting, Prepared Statements) and automated audits (Snyk, ZAP), the application is now resilient against the most common web attacks. The system is fully documented, containerized, and ready for secure deployment.

End of Report