Internship Progress Report

Name: Saiprakash Bollam
Internship Role: Research Intern
Duration: 20th November 2025 – 3rd December 2025
Organization: Computer Science Department, Binghamton University
Supervisor: Zerksis Umrigar
Email: umrigar@binghamton.edu

---

1. Introduction

This week, the primary focus was to perform a **complete security review and audit** of the Smart Contact Manager application.The security check targeted critical areas such as authentication, authorization, input validation, SQL query handling, sensitive data exposure, deprecated method usage, and the overall security posture of the application following significant framework and architecture upgrades.

The audit was driven by a combination of manual inspection and **AI-assisted static analysis**, ensuring thorough coverage and actionable outcomes.

---

2. Objectives

- Identify and document all potential vulnerabilities within the project.
- Detect deprecated or removed Spring Security APIs.
- Perform code-level and architectural security validation.
- Recommend and implement corrections aligned with OWASP Top 10 standards.
- Validate improved security posture through AI-assisted analysis.

---

3. Key Findings from the Security Audit

3.1 Strengths Identified

- Authentication and authorization flows remain logically structured.
- Transition to the Jakarta namespace was properly completed.
- Modern Spring Security configuration (lambda-based DSL) is in place.
- No use of the deprecated `WebSecurityConfigurerAdapter`.

3.2 Vulnerabilities Identified

- **Deprecated header configurations** (`xssProtection()`, `contentTypeOptions()`, `frameOptions()`).

- **Field-based dependency injection** detected across controllers and services, weakening testability and maintainability.

- Potential **SQL injection exposure** due to concatenated query strings.

- **Hardcoded credentials** discovered in early configuration layers.

- Sensitive information logged in multiple methods.

- Outdated or unnecessary annotations surfaced during static code analysis.

These findings indicate a combination of legacy code remnants and patterns incompatible with modern Spring Boot 3.x and Spring Security 6.x standards.

---

4. Fixes Implemented

4.1 Security Enhancements

- Removed deprecated header methods such as `xssProtection()`, `httpStrictTransportSecurity()`, and others incompatible with Spring 6.

- Replaced vulnerable query concatenations with **parameterized queries** to eliminate SQL injection exposure.

- Eliminated logging that revealed internal system details and sensitive parameters.

4.2 Code Modernization

- Replaced all field-based `@Autowired` usage with **constructor injection**, improving security, consistency, and test isolation.

- Upgraded `User.getAuthorities()` to follow modern Java 21 `.toList()` patterns.

- Updated deprecated annotations and removed redundant code fragments flagged by the security scan.

4.3 Authentication Provider Updates

- Created a dedicated, secure `DaoAuthenticationProvider` bean for user authentication.

- Ensured alignment with `PasswordEncoder`, userDetailsService, and the updated security filter chain.

---

5. AI's Role in the Security Audit

AI tools were heavily utilized throughout the audit and provided substantial support:

- Automated scanning for deprecated, insecure, or outdated code patterns.
- Detection of sensitive information exposure and unsafe logging practices.
- Generated fix-ready recommendations for replacing insecure APIs.
- Delivered detailed summaries of potential vulnerabilities.
- Offered prioritized remediation strategies based on severity.
- Assisted in validating that security improvements did not affect existing functionality.

AI significantly reduced manual review time, improved reliability, and supported continuous iteration.

---

6. Challenges Faced

- High volume of AI-generated fix suggestions required **manual validation** to avoid incorrect changes.

- Some security changes introduced **temporary build conflicts** that required dependency adjustments.

- Updating test cases to reflect constructor-based dependency injection.

- Ensuring security-related changes did not affect endpoint accessibility.
- Differentiating between AI-flagged false positives and actual vulnerabilities.

---

7. Outcomes & Learnings

Through the security audit, the system achieved a more secure and modern architecture.
Key outcomes include:

- Elimination of deprecated and insecure code blocks.
- Stronger authentication and authorization configurations.
- Enhanced structural maintainability due to constructor-based injection.
- Removal of potential attack vectors tied to SQL injection and sensitive logging.
- A complete report of vulnerabilities, their severity, and their fixes.

This week reinforced best practices in **secure coding, audit methodology, and AI-augmented analysis**, while deepening understanding of modern Spring Security mechanisms.