# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7-18-2025** | **Patricia Randol** |  |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Patricia Randol

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

Artemis Financial requires a thorough security evaluation of its software application to safeguard against various threats and vulnerabilities. As a financial services provider managing sensitive customer data and transactions, security is paramount for Artemis Financial. The company must identify possible security weaknesses within its application architecture layers and devise suitable mitigation strategies.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

According to the Vulnerability Assessment Process Flow Diagram, relevant security domains for Artemis Financials’ software application include:

2.1 Input Validation

Ensuring secure input handling: All user inputs must undergo rigorous validation to guard against injection attacks (SQL, XSS, CSRF) while maintaining data integrity throughout the application, critical for an application handling sensitive financial data.

2.2 Architecture Review

Analyze Application Architecture: Assessing overall design and structure helps uncover architectural-level security flaws by reviewing security architecture, data flow, and controls against industry standards for financial applications.

2.3 Code Review

Conduct a code review at multiple levels:

- Views: Interfaces for user interaction.

- Models: Data structures and business logic.

- Controllers: Logic managing user requests.

- Data Access: Database interactions.

- Services: Integrations with external services.

- Plug-ins: Third-party components.

- APIs: Interfaces for internal/external communication.

2.4 Security Focus Areas

Based on the Vulnerability Assessment Process Flow Diagram, focus on these critical domains:

- Secure Distributed Composing: Ensure safe communication between distributed components.

- Code Error/Secure Error Handling: Implement error handling practices that prevent information leaks while maintaining stability.

- Code Quality/Secure Coding Practices: Adhere to established secure coding practices aimed at minimizing vulnerabilities.

- Encapsulation/Secure Data Structures: Properly encapsulate sensitive data while implementing robust data structures.

- Cryptography/Encryption Use: Apply strong encryption techniques for both stored data and during transmission with effective key management.

- APIs/Secure API Interactions: Ensure all API interactions comply with secure protocols and authentication measures.

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

The architecture review and optional output from static testing will determine which manual code reviews are necessary. For Artemis Financial, the manual code review should focus on:

Input Validation: Analyze input processing code for adequate validation and sanitization.

Authentication & Authorization: Examine mechanisms safeguarding access controls.

Data Protection: Evaluate encryption implementations alongside key management strategies.

Error Handling: Scrutinize error handling code to ensure no sensitive information is exposed.

API Security: Inspect API endpoints regarding authentication, authorization, and input validation standards.

Session Management: Assess session handling codes relative to potential vulnerabilities.

Third-party Components: Review integrations with external libraries/services thoroughly.

**4. Static Testing**

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from the dependency-check report. Include the following items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

The scan indicated 189 vulnerabilities across 48 dependencies (with 35 being unique), identifying that out of these dependencies, 20 were vulnerable—ranging from LOW to CRITICAL severity levels—with several having known exploits posing significant risks to overall application security. Static testing serves as a pivotal component in detecting potential issues without executing code; this report delivers an extensive overview of static testing outcomes, highlighting found vulnerabilities along with their implications, as well as a prioritized remediation strategy.

Key Findings

Critical Vulnerabilities

Remote Code Execution (RCE) Vulnerabilities:

CVE-2022-1471 in snakeyaml-1.25.jar: This critical vulnerability allows unrestricted types during deserialization, enabling remote code execution. The vulnerability is caused by the SnakeYaml library not restricting Java types when deserializing objects using the Constructor. Deserializing YAML content provided by an attacker can lead to remote code execution. This vulnerability has been listed as a CISA Known Exploited Vulnerability, affecting multiple products.

CVE-2022-22965 ("Spring4Shell") in Spring Framework components (spring-context, spring-core, spring-web, spring-webmvc): This vulnerability enables remote code execution via data binding in Spring MVC or WebFlux applications running on JDK 9+11. It affects functions that use request mapping annotation and Plain Old Java Object (POJO) parameters within the Spring Framework. Unit 42 has observed this vulnerability being exploited in the wild.

Security Bypass Vulnerabilities:

CVE-2023-20873 in Spring Boot components: This vulnerability allows a security bypass for applications deployed to Cloud Foundry. In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, applications deployed to Cloud Foundry could be susceptible to a security bypass.

File Access Vulnerabilities:

CVE-2020-1938 ("Ghostcat") in Tomcat components: This vulnerability permits unauthorized access to files via the AJP connector. Ghostcat is found in Apache Tomcat versions 6.x, 7.x, 8.x, and 9.x 21 and is a file read/inclusion vulnerability in the AJP connector that is enabled by default with a default configuration port of 8009. The vulnerability is caused by the AJP connector within the Java Servlet being unable to process the read/inclusion of file inputs.

High Severity Vulnerabilities

Data Processing Vulnerabilities:

CVE-2020-25649 in Jackson-databind: This vulnerability enables XML external entity (XXE) attacks due to improper entity expansion security.

CVE-2020-36518, CVE-2022-42003, CVE-2022-42004 in jackson-databind: These vulnerabilities can cause resource exhaustion and stack overflow via nested objects.

Code Execution Vulnerabilities:

CVE-2023-6378, CVE-2021-42550 in logback components: These vulnerabilities allow arbitrary code execution through serialization issues and LDAP servers.

Comprehensive Remediation Plan

Priority 1: Immediate Actions

Update Critical Dependencies:

Upgrade snakeyaml from 1.25 to 2.0 or later to address CVE-2022-14715. If upgrading isn't immediately possible, use SnakeYaml's SafeConstructor when parsing untrusted data.

Upgrade Spring Framework components from 5.2.3. RELEASE to secure versions to mitigate CVE-2022-22965. This is especially important for applications running on JDK 9+15.

Upgrade Tomcat components from 9.0.30 to the latest version to address CVE-2020-1938. If an immediate upgrade is not possible, disable the AJP connector if not required. This vulnerability is exploitable only if port 8009 is exposed and AJP is installed.

Priority 2: Short-term Actions

Address High-Severity Vulnerabilities:

Upgrade Jackson-databind from 2.10.2 to a secure version to address XXE vulnerabilities (CVE-2020-25649) and resource exhaustion vulnerabilities (CVE-2020-36518, CVE-2022-42003, CVE-2022-42004).

Upgrade logback components to secure versions to mitigate serialization and LDAP vulnerabilities (CVE-2023-6378, CVE-2021-42550).

Implement Security Best Practices:

Configure Tomcat securely, ensuring proper handling of HTTP headers and implementing appropriate security controls.

Enforce a strict Content Security Policy (CSP) to mitigate XSS vulnerabilities.

Implement robust input validation, especially for untrusted data processed by vulnerable components, to prevent injection attacks.

Priority 3: Long-term Actions

Establish Security Processes:

Integrate dependency checking into the CI/CD pipeline to identify vulnerabilities early in the development process.

Establish a process for monitoring security advisories for the identified components (e.g., Spring Framework, Tomcat, SnakeYAML).

Implement a systematic approach for handling security patches and updates to ensure timely remediation of vulnerabilities.

False Positives Management

The Dependency-Check report provides XML suppression tools for handling false positives. For vulnerabilities deemed as false positives after thorough verification, use the XML suppression rules provided in the report. However, exercise caution when suppressing findings and ensure proper documentation of the justification.

Example of a suppression XML entry:

xml

<suppress>

<notes><! [CDATA[

file name: example.jar

]]></notes>

<vulnerabilityName>CVE-2023-XXXX</vulnerabilityName>

</suppress>

It's important to review these suppressions regularly and remove them when the underlying issue is resolved.

Next Steps in the Security Assessment Process

According to the Vulnerability Assessment Process Flow Diagram, the static testing results should inform the scope of manual code reviews:

Manual Code Reviews focusing on:

Input validation and secure input representations

Secure error handling

Secure coding practices

Cryptography implementation and vulnerabilities

API security and interactions

Component-Specific Reviews:

Review controllers for proper request handling and security

Review models for secure data handling and structures

Review services for secure business logic

Review data access components for secure database operations

Review APIs for secure external interactions

Review plug-ins for potential security issues

Architecture Review:

Analyze application architecture considering identified vulnerabilities

Review secure distributed composing practices

Identify any systemic issues that may be contributing to security weaknesses

Final Reporting:

The most pressing issues involve remote code execution vulnerabilities in SnakeYaml and Spring Framework components, which have known exploits. Immediate attention should be given to upgrading these dependencies to secure versions.

Following the prioritized remediation plan will significantly improve the security posture of the application. By addressing these vulnerabilities and implementing the recommended security practices, it will be better protected against potential security breaches, helping to safeguard sensitive data and maintain the integrity of the application.

Regular static testing should be incorporated into the development lifecycle to identify and address security vulnerabilities early. This proactive approach to security, combined with thorough manual code reviews informed by static testing results, will ensure that it remains secure against evolving threats.

**5. Mitigation Plan**

Interpret the results from the manual review and the static testing report. Then, identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

The vulnerability assessment process has identified several key areas that require attention in Artemis Financials’ software application. By following the structured approach outlined in the Vulnerability Assessment Process Flow Diagram, we can ensure comprehensive coverage of all security domains.

The mitigation plan should be implemented in the following phase:

Phase 1: Address critical vulnerabilities in input validation and authentication

Phase 2: Enhance data protection and encryption

Phase 3: Improve API security and error handling

Phase 4: Conduct comprehensive testing and validation

By implementing this mitigation plan, Artemis Financial can significantly enhance the security posture of its application, protecting sensitive customer data and financial transactions from potential security threats.

The advantages of using this structured vulnerability assessment process include:

Comprehensive Coverage: Addresses security at all layers of the application

Risk-based Approach: Prioritizes vulnerabilities based on risk

Modular Structure: Allows for focused security reviews

Security-by-Design Integration: Embeds security throughout the development lifecycle

This systematic approach ensures that Artemis Financials’ application undergoes thorough security scrutiny across all layers and components, resulting in more secure software with fewer vulnerabilities