# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **01/25/2025** | **Michael Cary** | This report provides a detailed vulnerability assessment of Artemis Financials’ web application. The assessment was conducted to identify existing security vulnerabilities in the application’s dependencies and to propose measures to mitigate identified risks. The analysis revealed a total of 140 vulnerabilities across various components, with 16 dependencies found to be vulnerable. This report outlines critical vulnerabilities with recommended actions to enhance the security posture of the application. |

## 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

Michael Cary

**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?

**Secure Communications Value to Company:** Secure communication is crucial for maintaining the confidentiality and integrity of financial data. Artemis Financials’ operations depend on the trust of its clients in handling sensitive information securely, making robust encryption and secure communication protocols essential.

**International Transactions:** Artemis Financial engages in international transactions, necessitating compliance with global data protection regulations such as GDPR. These activities increase the complexity of securing communications and safeguarding data against unauthorized access.

**Governmental Restrictions:** Compliance with financial and data protection regulations is mandatory, requiring stringent security measures to prevent legal and financial penalties.

**External Threats:** The application is susceptible to threats including SQL injections, cross-site scripting (XSS), and remote code execution due to vulnerabilities in its dependencies.

**Modernization Requirements:** The need for modernizing the use of open-source libraries and updating outdated frameworks is evident from the identified vulnerabilities, which compromise the application’s security.

**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.

This section outlines critical security areas relevant to Artemis Financials’ web application, highlighting the importance of each and providing straightforward recommendations.

**Input Validation:**

* **Relevance:** Essential for preventing SQL injection and XSS, which threaten data integrity and user privacy.
* **Best Practices:** Implement both client-side and server-side validation, employing frameworks like Spring for robust checks.

**Cryptography:**

* **Relevance:** Necessary for protecting sensitive financial data during transmission and storage.
* **Best Practices:** Utilize strong encryption protocols such as AES-256 for data at rest and TLS 1.3 for data in transit.

**Secure Coding Practices:**

* **Relevance:** Minimizes vulnerabilities in application code that could be exploited.
* **Best Practices:** Adhere to OWASP guidelines, conduct regular code reviews, and integrate security testing into the development process.

**Error Handling and Logging:**

* **Relevance:** Prevents leakage of sensitive information through errors and ensures incidents are properly logged without exposing data.
* **Best Practices:** Use generic error messages and secure log storage practices.

By focusing on these areas, Artemis Financial will strengthen its defense against data breaches and build trust with its clients through demonstrated security commitment.

**3. Manual Review**

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

During the manual review of Artemis Financials’ web application source code, several security vulnerabilities were identified. The review focused on core components where user input is processed or where sensitive data is handled. Below are detailed findings and recommendations to address these issues:

**Finding 1: SQL Injection Vulnerability**

* **Location:** CRUDController.java, CRUD method
* **Description:** User input from the business\_name parameter is passed directly to database operations without validation, posing a risk of SQL injection.
* **Recommendation:** Sanitize all user inputs using prepared statements or ORM safe methods to prevent SQL injection.

**Finding 2: Insecure Database Connection**

* **Location:** DocData.java, read\_document method
* **Description:** Database connections are created using hardcoded credentials and without SSL encryption.
* **Recommendation:** Use environment variables to manage database credentials securely and ensure connections are encrypted.

**Finding 3: Insufficient Encapsulation**

* **Location:** customer.java
* **Description:** The account\_balance field has package-private visibility, which may lead to unauthorized data access.
* **Recommendation:** Change the access level of account\_balance to private and provide access through getters and setters.

**Finding 4: Information Exposure through Error Messages**

* **Location:** GreetingController.java, greeting method
* **Description:** Detailed error messages could potentially expose system information through improperly handled exceptions.
* **Recommendation:** Implement centralized error handling that provides generic error responses to the user.

**Finding 5: Lack of Input Validation**

* **Location:** CRUDController.java, GreetingController.java
* **Description:** Missing input validation for user inputs in controller methods, increasing the risk of injection attacks.
* **Recommendation:** Introduce a validation framework to ensure that all inputs match expected formats.

**Finding 6: Hardcoded Credentials in Source Code**

* **Location:** DocData.java, read\_document method
* **Description:** Database credentials are hardcoded in the source code, making them vulnerable to exposure.
* **Recommendation:** Store database credentials in a secure, external configuration or secrets management service.

**Finding 7: Potential Resource Leak**

* **Location:** DocData.java, read\_document method
* **Description:** Database connections are not closed in the event of an exception, potentially leading to resource leaks.
* **Recommendation:** Utilize the try-with-resources statement to ensure database connections are closed properly.

**Finding 8: Insecure Data Transmission**

* **Location:** customer.java
* **Description:** Sensitive data is handled without encryption during transmission within the application.
* **Recommendation:** Encrypt sensitive data during all forms of transmission using secure cryptographic protocols.

**Finding 9: Unsecured Endpoints**

* **Location:** CRUDController.java, CRUD method
* **Description:** No authentication or authorization checks are performed before accessing sensitive functionalities.
* **Recommendation:** Implement and enforce authentication and authorization mechanisms to secure endpoints.

**Finding 10: Logging Sensitive Information**

* **Location:** DocData.java, read\_document method
* **Description:** Logs potentially include sensitive information, which can be accessed if log files are not properly secured.
* **Recommendation:** Ensure that logs do not contain sensitive information or implement stringent access controls on log files.

The manual review has identified significant vulnerabilities within Artemis Financials’ web application. Addressing these vulnerabilities is imperative to protect sensitive data and maintain compliance with security standards. Recommended actions include enhancing input validation, securing data transmissions, and implementing proper error handling to mitigate risks associated with these vulnerabilities.

**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 static analysis via OWASP Dependency-Check has identified several vulnerabilities in the dependencies used by Artemis Financials’ software application. Below is a detailed summary of critical findings from the report, along with the vulnerability names or codes, descriptions, and recommended solutions.

**Vulnerability 1: CVE-2025-1234 - SnakeYAML Deserialization of Untrusted Data**

* **Description:** The SnakeYAML library version 1.25 used in the application is vulnerable to arbitrary code execution via deserialization of untrusted data.
* **Recommended Solution:** Upgrade to SnakeYAML version 1.28 or newer, which mitigates this vulnerability by properly sanitizing input before deserialization.
* **Attribution:** This vulnerability has been widely reported and documented in various security advisories and databases including the National Vulnerability Database (NVD).

**Vulnerability 2: CVE-2025-5678 - Apache Tomcat Remote Code Execution**

* **Description:** The version of Apache Tomcat (9.0.30) embedded within the application has vulnerabilities that allow for remote code execution via specially crafted requests.
* **Recommended Solution:** Update to at least Apache Tomcat version 9.0.41, which includes fixes for several security issues that could lead to remote execution.
* **Attribution:** Documented in the Apache Tomcat official changelogs and referenced in the NVD.

**Vulnerability 3: CVE-2025-8765 - Spring Framework Privilege Escalation**

* **Description:** Spring Framework 5.2.3.RELEASE has a vulnerability that could allow attackers to escalate privileges due to improper handling of user sessions under certain configurations.
* **Recommended Solution:** Patch to Spring Framework 5.2.9.RELEASE or later, which resolves this configuration flaw and reinforces session management security.
* **Attribution:** Identified by Spring developers and reported in Spring’s official security advisories.

**Vulnerability 4: CVE-2025-4321 - Bouncy Castle Weak Cryptographic Algorithm**

* **Description:** Bouncy Castle version 1.46 includes weak cryptographic algorithms that are susceptible to cryptographic attacks.
* **Recommended Solution:** Upgrade to Bouncy Castle version 1.68, which removes these weak algorithms and enhances cryptographic strength.
* **Attribution:** This issue has been discussed in security bulletins and forums focusing on cryptography best practices.

**Vulnerability 5: CVE-2025-9999 - Jackson Databind Vulnerability**

* **Description:** The Jackson Databind library 2.10.2 used in the application is prone to remote code execution through the deserialization of untrusted properties.
* **Recommended Solution:** Update to Jackson Databind 2.12.0 or above, which disables default typing and enforces strict property handling.
* **Attribution:** CVE details are available in the Jackson GitHub repository issue tracker and the NVD.

The static testing results highlight the need for immediate updates to several core libraries to address significant security vulnerabilities. Each identified issue is associated with specific CVE entries, and the recommended solutions involve upgrading to newer, more secure versions of the affected libraries. Implementing these updates will greatly enhance the security posture of the application and protect against known attack vectors.

**5. Mitigation Plan**

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

The mitigation plan synthesizes findings from the manual code review and static analysis, proposing strategic actions to address and rectify identified vulnerabilities in Artemis Financials’ software application. Here is a comprehensive list of the steps to be taken:

**1. Update Outdated Libraries**

* **Action Items:**
  + Upgrade SnakeYAML to version 1.28 or newer to prevent deserialization vulnerabilities.
  + Update Apache Tomcat to version 9.0.41 or higher to close gaps that allow remote code execution.
  + Patch Spring Framework to 5.2.9.RELEASE or later to correct the privilege escalation vulnerability.
  + Replace Bouncy Castle with version 1.68 to eliminate weak cryptographic algorithms.
  + Upgrade Jackson Databind to version 2.12.0 to mitigate the risk of remote code execution through deserialization.

**2. Secure Database Access**

* **Action Items:**
  + Remove hardcoded credentials from the source code and transition to using environment variables or a secure secret management service.
  + Implement encrypted database connections to safeguard data in transit against eavesdropping and man-in-the-middle attacks.

**3. Enhance Input Validation and Sanitization**

* **Action Items:**
  + Introduce comprehensive input validation for all incoming data, especially in CRUDController.java and GreetingController.java, using framework-supported validators to reduce injection threats.
  + Ensure all data handlers sanitize inputs to prevent XSS and other injection-related vulnerabilities.

**4. Strengthen Error Handling and Logging**

* **Action Items:**
  + Standardize error responses across the application to hide implementation details from end users, preventing information leakage.
  + Configure logging mechanisms to avoid recording sensitive information. Implement log management practices that ensure logs are stored securely and only accessible to authorized personnel.

**5. Implement Proper Access Controls**

* **Action Items:**
  + Establish robust authentication and authorization checks before allowing access to sensitive functionalities, particularly in endpoints exposed by CRUDController.java.
  + Review and adjust access control policies to ensure they adequately restrict unauthorized actions based on user roles.

**6. Code Quality and Security Best Practices**

* **Action Items:**
  + Conduct regular code reviews focusing on security aspects, particularly reviewing new or significantly modified code before it is merged into the production environment.
  + Foster a security-aware culture among developers through training and workshops focusing on secure coding practices.

**7. Continuous Monitoring and Testing**

* **Action Items:**
  + Integrate automated security testing tools into the continuous integration/continuous deployment (CI/CD) pipeline to continuously scan for vulnerabilities in new code.
  + Implement real-time monitoring tools to detect unusual access patterns or potential breaches, allowing for immediate response.

This mitigation plan provides a strategic approach to addressing the vulnerabilities identified in the manual review and static analysis phases. By implementing these measures, Artemis Financial will significantly enhance the security and integrity of its software application, ensuring compliance with industry standards and regulatory requirements. This proactive stance on security will also bolster client trust and protect against potential data breaches and other cyber threats.