# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/23/2025** | **Shah Ali Omor** | **Final version of the vulnerability assessment report** |

## 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

Shah Ali Omor

**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 designs personalized financial plans and handles sensitive client data, so secure communications are essential to maintain trust and protect confidential information. Ensuring that data is encrypted in transit and at rest prevents eavesdropping and tampering, which is critical given the financial nature of the application.

* **Value of Secure Communications:**

The value lies in preventing data breaches and unauthorized access, which could expose client financial and personal details. Encryption via HTTPS/TLS safeguards these communications and ensures compliance with industry standards.

* **International Transactions:**

Artemis may serve clients from various regions, meaning the application might handle cross-border transactions. This introduces complexities such as currency conversion and adhering to different regional data protection regulations.

* **Governmental Restrictions:**

There are strict regulatory requirements (like GDPR, PCI-DSS, or local financial regulations) governing data protection and secure communications. The application must implement these security controls to avoid legal repercussions and maintain operational integrity.

* **External Threats:**

Currently, threats such as injection attacks, XSS, and unauthorized API access are prominent. In the immediate future, zero-day vulnerabilities in third-party libraries and evolving attack methods pose additional risks. Robust security measures must be in place to counter these threats.

* **Modernization Requirements:**

With rapid advancements in open-source libraries and web technologies, the application must be designed to easily integrate updates and patches. Adopting modern, secure coding practices and regularly updating dependencies will be key to staying ahead of potential vulnerabilities.

This comprehensive understanding of Artemis Financials needs and potential threats guides the overall security strategy, ensuring that the application is both resilient and compliant.

**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.

After reviewing this diagram and considering Artemis Financials RESTful application, the following security areas are most relevant:

* **Architecture Review**

Analyzing the overall design ensures the application uses a defense-in-depth approach. This is important for a financial organization handling sensitive data, as it clarifies where security controls (such as authentication and encryption) must be placed in the system’s architecture.

* **Input Validation**

Artemis’s API endpoints receive user input, which must be validated and sanitized to prevent injection attacks (like SQL injection or cross-site scripting). Strict input validation helps ensure only safe and properly formatted data enters the application.

* **APIs (Secure API Interactions)**

The application’s core functionality is exposed via RESTful endpoints. Ensuring these APIs require authentication and proper access control is critical to protect confidential financial data from unauthorized users or automated threats.

* **Cryptography**

Sensitive client information and transaction details must be encrypted at rest and in transit. Using robust encryption libraries and protocols (e.g., TLS for network communications) helps safeguard data against interception and unauthorized access.

* **Client/Server Security**

The client-server model should be protected by enforcing HTTPS, using secure session handling, and potentially restricting requests to trusted networks or gateways. This prevents attackers from hijacking sessions or injecting malicious requests.

* **Code Error (Secure Error Handling)**

Proper error handling ensures that no sensitive details are revealed in error messages or logs. For example, stack traces can inadvertently disclose file paths or configuration secrets, which attackers could exploit.

* **Code Quality**

Following secure coding practices (like parameterized queries, consistent naming, and modular design) reduces the likelihood of introducing vulnerabilities. High-quality code is also easier to maintain and audit for security flaws.

* **Encapsulation**

Ensuring that data structures and classes are properly encapsulated prevents unintended data exposure. This is especially vital in financial applications where only authorized methods should modify or access sensitive fields.

By focusing on these areas, Artemis Financial can create a more resilient application, capable of handling evolving security challenges while protecting clients’ confidential financial data.

**3. Manual Review**

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

By manually inspecting each file in the Artemis Financial code base, the following vulnerabilities were identified:

1. **No Input Validation**
   * **Location:** GreetingController.java
   * **Description:** The greeting endpoint directly uses the name parameter without validating its content or length. This could enable attackers to inject malicious input or excessive data.
2. **Lack of Output Encoding**
   * **Location:** GreetingController.java
   * **Description:** The user-provided name is embedded in the response as-is, risking cross-site scripting (XSS) if this output is rendered by a web client.
3. **Hard-Coded Database Credentials**
   * **Location:** DocData.java (within read\_document method)
   * **Description:** The code uses "root", "root" for the JDBC connection, which can be easily discovered if the source code is compromised.
4. **Insecure Error Handling**
   * **Location:** DocData.java (catch block in read\_document)
   * **Description:** Calling e.printStackTrace() can reveal internal details, including file paths and stack traces, to potential attackers.
5. **No Authentication on Endpoints**
   * **Location:** GreetingController.java and CRUDController.java
   * **Description:** These controllers expose endpoints without requiring user authentication, allowing unauthorized access to application features.
6. **Incomplete Logic in Date Handling**
   * **Location:** myDateTime.java
   * **Description:** Methods retrieveDateTime() and setMyDateTime() are placeholders with no validation or real implementation, leaving potential gaps if they handle critical data in the future.
7. **Lack of Prepared Statements**
   * **Location:** DocData.java (potential future queries)
   * **Description:** The code references JDBC connections but does not use prepared statements, making it susceptible to SQL injection if user input is later concatenated into queries.
8. **Insecure HTTP Method Usage**
   * **Location:** GreetingController.java (uses GET for data submission)
   * **Description:** Sensitive or state-changing operations should not rely on GET, which can expose query parameters in URLs and logs.
9. **Poor Naming and Encapsulation**
   * **Location:** customer.java
   * **Description:** The class name “customer” and non-private fields do not follow standard Java conventions, risking unintended data exposure or accidental modifications.
10. **Potentially Outdated Dependencies**

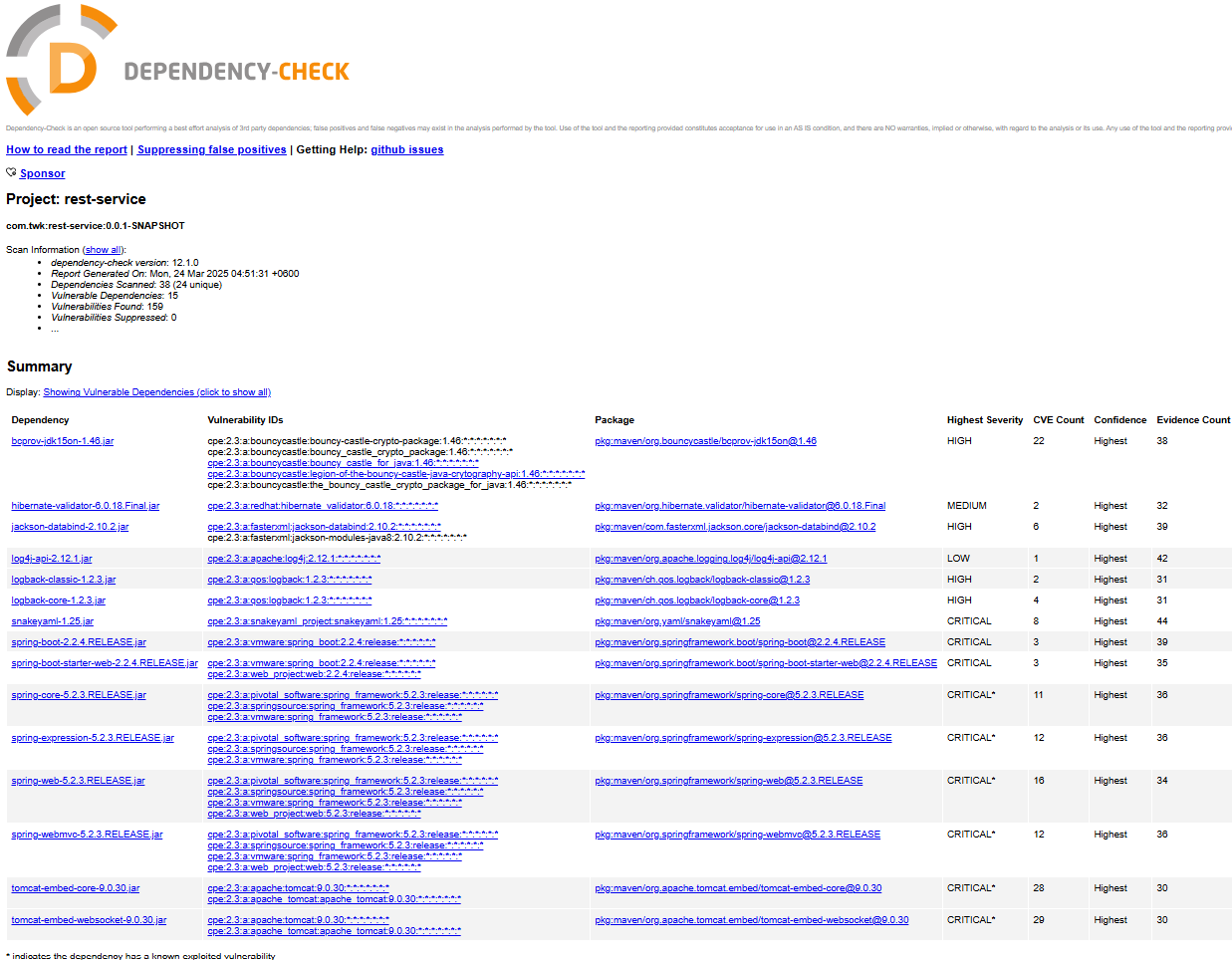
* **Location:** pom.xml
* **Description:** The Maven file references older library versions (e.g., Spring Boot 2.2.4, bcprov-jdk15on 1.46). These may have known vulnerabilities if not updated regularly.

Each of these issues represents a point where attackers could compromise data or functionality. Addressing them will significantly strengthen the security of Artemis Financials application.

**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



After running the OWASP Dependency-Check on the application, the tool flagged several vulnerable dependencies. Below is a summary of the key findings from the dependency-check report:

* **bcprov-jdk15on-1.46.jar**
  + Vulnerabilities: This older version contains known flaws in certificate validation, ECC handling, and other cryptographic functions. Attackers may exploit these weaknesses to perform man-in-the-middle attacks or cause resource exhaustion.
  + Description & Recommendation: Upgrading to bcprov-jdk15on version 1.76 or higher is strongly recommended. Newer releases address these security gaps and improve overall cryptographic resilience.
  + Attribution: Flagged by OWASP Dependency-Check, referencing data from the National Vulnerability Database (NVD) and related advisories.
* **spring-webmvc-5.2.3.RELEASE.jar**
* Vulnerabilities: Multiple CVEs, such as CVE-2020-5398, which may allow remote code execution or file download attacks.
* Description & Recommendation: These issues can compromise the application’s integrity and confidentiality. Updating to Spring Boot 2.7.x (or later) ensures a more secure Spring Framework version that addresses these vulnerabilities.
* Attribution: Reported by OWASP Dependency-Check, referencing the NVD and official Spring advisories.
* **jackson-databind-2.10.2.jar**
* Vulnerabilities: Includes CVE-2020-25649, CVE-2020-36518, CVE-2021-46877, among others. These flaws can lead to XXE and unsafe deserialization, exposing the application to data leaks or remote code execution.
* Description & Recommendation: Upgrading to Jackson 2.14.x mitigates these risks by providing improved parsing and secure defaults.
* Attribution: Findings are based on OWASP Dependency-Check results, which reference multiple NVD advisories.
* **snakeyaml-1.25.jar**
* Vulnerabilities: Includes CVE-2017-18640 and additional CVEs, making it vulnerable to YAML entity expansion attacks (e.g., “Billion Laughs”) that can cause denial-of-service.
* Description & Recommendation: Update to SnakeYAML 1.30 or later to avoid resource exhaustion and other parser-related exploits.
* Attribution: Documented by OWASP Dependency-Check using data from the NVD.
* **log4j-api-2.12.1.jar**
* Vulnerability: For example, CVE-2020-9488, which poses remote code execution risks if misconfigured.
* Description & Recommendation: Although only the API library is present, removing it if unnecessary or upgrading to Log4j 2.17.1+ is advised for better security.
* Attribution: Identified by OWASP Dependency-Check referencing NVD advisories and the known Log4j vulnerabilities.
* **logback-classic-1.2.3.jar & logback-core-1.2.3.jar**
* Vulnerabilities: Include CVE-2023-6378, CVE-2021-42550, and others related to insecure configuration and potential JNDI lookups.
* Description & Recommendation: Upgrade to Logback 1.2.10 or later to prevent unauthorized code execution or configuration-based exploits.
* Attribution: Reported by OWASP Dependency-Check, referencing official security advisories and the NVD.
* **hibernate-validator-6.0.18.Final.jar**
* Vulnerabilities: CVE-2023-1932, CVE-2020-10693, among others, which may allow bypassing of certain input validation rules.
* Description & Recommendation: Move to Hibernate Validator 6.1.x or 6.2.x to close these gaps and ensure reliable input checks.
* Attribution: Discovered by OWASP Dependency-Check, with references to the NVD and Hibernate documentation.
* **Additional Dependencies (spring-core, spring-expression, embedded Tomcat)**
* Vulnerabilities: Various CVEs affecting the older versions in use. These can include remote code execution, denial-of-service, or data exposure.
* Description & Recommendation: Upgrading to the latest Spring Boot version typically brings updated spring-core, spring-expression, and Tomcat libraries that address known vulnerabilities.
* Attribution: Listed in the OWASP Dependency-Check report, referencing NVD data.

Regular updates to these dependencies and ongoing scans are recommended to ensure the application remains secure against emerging threats.

**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.

Interpreting the manual review and static testing results, I propose the following action steps to remediate the vulnerabilities in Artemis Financials application:

* **Enhance Input Validation & Output Encoding:**
  + Implement robust input validation on all endpoints (e.g., in GreetingController) using frameworks like Hibernate Validator.
  + Sanitize and encode all outputs to prevent injection and XSS attacks.
* **Secure Database Credentials and Querying:**
  + Remove hard-coded database credentials from DocData.java. Use environment variables or secure configuration management.
  + Update future database queries to use prepared statements to protect against SQL injection.
* **Improve Error Handling:**
  + Replace printStackTrace() with a secure logging framework (e.g., SLF4J with Logback) to log errors without exposing internal details.
  + Implement a global exception handler (using Spring’s @ControllerAdvice) to return generic error messages to clients.
* **Enforce Authentication and Authorization:**
  + Integrate Spring Security to require authentication for all API endpoints.
  + Implement role-based access controls to ensure that users only access permitted data.
* **Refine HTTP Method Usage:**
  + Reassess endpoint designs and switch sensitive or state-changing operations from GET to POST/PUT/PATCH as appropriate.
  + Ensure that sensitive parameters are not exposed in URLs or logs.
* **Complete and Secure Incomplete Implementations:**
  + Finish and secure the implementations in myDateTime.java by adding proper logic and validation for date and time handling.
* **Refactor Code Quality:**
  + Follow standard naming conventions and encapsulate class fields properly (e.g., rename “customer” to “Customer” and make fields private).
  + Ensure that only required data is exposed via getters/setters.
* **Upgrade Outdated Dependencies:**
  + Upgrade bcprov-jdk15on from version 1.46 to 1.76 or higher.
  + Update Spring Boot (including spring-webmvc and related libraries) to 2.7.x (or later) to leverage improved security features.
  + Upgrade Jackson Databind to version 2.14.x, SnakeYAML to 1.30+, and update logging libraries (Log4j and Logback) as recommended.
  + Upgrade Hibernate Validator to a later version (6.1.x or 6.2.x).
  + Establish a process for regular dependency scans and updates to ensure ongoing security.
* **Secure Communication and Data Protection:**
  + Enforce HTTPS across all client-server communications.
  + Plan for encrypting sensitive data at rest (e.g., using strong hashing algorithms for passwords).
* **Ongoing Testing and Audits:**
  + Develop comprehensive unit and integration tests to verify that validation, authentication, and error handling are functioning correctly.
  + Re-run dependency-check scans regularly and monitor for new vulnerabilities to keep the application secure.

By following these steps, Artemis Financials application will be significantly hardened against both current and emerging threats. Regular audits and updates will help maintain a robust security posture over time.