# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03-23-2025** | **Zac Harrington** |  |

## 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

Zac Harrington

**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?

For a financial organization like Artemis Financial, maintaining the confidentiality, integrity, and availability of sensitive client data is essential. Their RESTful web application interacts with customers and external systems, making secure communication a top priority. Financial transactions and personal data exchanges must be protected from man-in-the-middle attacks, eavesdropping, and data interception through encryption protocols like HTTPS and TLS. Any vulnerability in data transmission could result in severe financial and reputational damage. Given the potential for international outreach, Artemis Financial must also comply with global data regulations such as the GDPR or the Gramm-Leach-Bliley Act. These require institutions to handle, store, and share data responsibly and transparently. Threats like injection attacks, authentication bypass, or outdated components are particularly relevant. Furthermore, modernization is key—open-source libraries must be kept up to date, and the company must adopt modern authentication methods like token-based systems. As threats evolve, Artemis Financial needs a proactive security stance to maintain trust and protect their infrastructure.

**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.

Upon reviewing the Vulnerability Assessment Process Flow Diagram, several critical areas of security are applicable to Artemis Financial’s software. Input validation is vital, as the system accepts user input that, if left unchecked, could lead to injection attacks or malformed data impacting application integrity. Secure API interaction is another key focus, especially since the application uses a RESTful architecture. API endpoints must be protected against unauthorized access, and proper authentication tokens should be enforced. Additionally, data in transit must be encrypted, making secure client-server communication essential through protocols like TLS. Authentication and authorization mechanisms must be in place to ensure that users can only access permitted resources based on their roles. Lastly, error handling must be hardened; detailed error messages can inadvertently disclose system information that attackers could leverage. Ensuring these security areas are addressed helps fortify the application against both internal weaknesses and external threats.

**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 manual inspection of the source code uncovered several security concerns. One notable issue is the use of the GET method instead of POST for sensitive operations at the /greeting endpoint. While both methods transmit data, GET exposes parameters in the URL, which is more vulnerable to logging and shoulder-surfing. POST, on the other hand, includes parameters in the message body, offering more discretion. Additionally, there’s a lack of input validation in files such as customer.java and DocData.java, meaning malicious input could manipulate how data is handled. Endpoints in CRUDController.java perform sensitive actions like adding or deleting users without requiring any form of authentication or token check. Furthermore, sensitive fields are returned in full via the API, exposing internal data structures unnecessarily. There’s also no implementation of session timeout, logging of critical events, or validation on custom date parsing, all of which could be exploited if left unaddressed. These findings indicate the application requires significant hardening before it can be considered secure.

**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

**CVE-2020-9488 (Log4j 1.x)**  
*Issue*: Vulnerable to remote code execution due to misconfigured JMSAppender.  
*Fix*: Upgrade log4j-api to version 2.13.2 or later.

**CVE-2020-10693 (Hibernate Validator)**  
*Issue*: Invalid EL expressions can bypass input validation, enabling attackers to evaluate dangerous inputs.  
*Fix*: Use expression variable sanitization, context unwrapping, and Hibernate Constraint Validator Context.

**CVE-2018-5382 (BKS Keystore)**  
*Issue*: Uses a weak 16-bit HMAC, enabling attackers to tamper with or forge keystore data.  
*Fix*: Upgrade to Bouncy Castle version 1.47+ which uses a 160-bit HMAC.

**CVE-2021-22118 (Spring WebFlux)**  
*Issue*: File upload handling may allow local privilege escalation or overwriting arbitrary files.  
*Fix*: Update Spring Framework to version 5.3.7 or higher.

**CVE-2020-11996 (Apache Tomcat)**  
*Issue*: HTTP/2 connections can trigger excessive CPU usage, leading to denial-of-service.  
*Fix*: Upgrade tomcat-embed-core to version 9.0.36 or newer.

**CVE-2020-13935 (Apache Tomcat WebSocket)**  
*Issue*: Incorrect handling of WebSocket payloads can cause an infinite loop or DoS.  
*Fix*: Upgrade tomcat-embed-core to version 9.0.37+.

**CVE-2020-9484 (Apache Tomcat + FileStore)**  
*Issue*: Improper deserialization allows attackers to execute arbitrary code via PersistenceManager.  
*Fix*: Upgrade to tomcat-embed-core 9.0.39+ and avoid using sessionAttributeValueClassNameFilter="null".

**CVE-2020-1938 (Tomcat AJP Protocol)**  
*Issue*: AJP connector enabled by default may allow attackers to process any file as a JSP.  
*Fix*: Upgrade to version 9.0.31+ and disable AJP connector unless explicitly used.

**CVE-2020-11996 (Tomcat HTTP/2 CPU Spike)**  
*Issue*: Malicious HTTP/2 requests can cause server to become unresponsive.  
*Fix*: Upgrade to version 9.0.36+.

**CVE-2016-1000352 to CVE-2016-1000341 (Bouncy Castle JCE)**  
*Issue*: Multiple encryption and key handling flaws including weak Diffie-Hellman, DSA, ECDSA, and CBC padding issues.  
*Fix*: Upgrade bcprov-jdk15on to version 1.56 or later.

**CVE-2017-13098 ("ROBOT" Attack)**  
*Issue*: Exploitable Bleichenbacher oracle due to vulnerable RSA key exchange in TLS.  
*Fix*: Upgrade bcprov-jdk15on to 1.60+ and ensure modern TLS configurations.

**CVE-2020-25649 (FasterXML Jackson)**  
*Issue*: External entity expansion (XXE) vulnerability that could allow attackers to read arbitrary files.  
*Fix*: Add the settings disallow-doctype-decl=true and nonvalidating/load-external-dtd=false.

**CVE-2021-24122 (Tomcat NTFS Path Disclosure)**  
*Issue*: File.getCanonicalPath() can disclose internal NTFS paths via JSP resources.  
*Fix*: Upgrade to tomcat-embed-core 9.0.40+.

**CVE-2020-1935 (Tomcat Transfer-Encoding Header Bug)**  
*Issue*: Malformed Transfer-Encoding headers can lead to HTTP request smuggling.  
*Fix*: Upgrade to tomcat-embed-core 9.0.31+.

**CVE-2019-17569 (Tomcat Regression - Transfer-Encoding)**  
*Issue*: Regression led to invalid headers being misinterpreted.  
*Fix*: Upgrade to tomcat-embed-core 9.0.31+.

**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.

Many of the identified vulnerabilities can be addressed by applying updates and enforcing secure coding practices. First, updating outdated dependencies is essential—this includes upgrading log4j-api to version 2.13.2, spring-core to 5.2.9 or higher, and tomcat-embed-core to at least 9.0.43. Keeping libraries current is one of the most effective ways to eliminate known security flaws. Input validation must also be improved by implementing character whitelisting and enforcing maximum length limits on all user input fields, which helps mitigate injection and buffer overflow risks. For endpoints like /greeting, switching from GET to POST ensures that sensitive data isn’t exposed in URL parameters or server logs. Date parsing should also be hardened—particularly in response to CVE-2020-10693—by sanitizing EL expressions and processing them securely through Hibernate's Constraint Validator Context. Additionally, the application should use Data Transfer Objects (DTOs) to restrict API responses to only necessary data fields, thereby preventing overexposure of internal structures. Authentication checks need to be enforced with Spring Security annotations such as @PreAuthorize and token-based mechanisms to verify user access at each critical point. Lastly, robust logging should be implemented to track access attempts, user activities, and anomalies for improved traceability and forensic readiness. Applying these mitigation steps will significantly reduce Artemis Financial’s exposure to threats and align the software with modern security expectations.