# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **20 July 2025** | **Dylan Dunagan** |  |

## 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

Dylan Dunagan

**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 are extremely valuable to Artemis Financial because the company handles sensitive information such as customer financial data. They will need to protect this information during transmission to maintain integrity, confidentiality, and compliance. Since Artemis Financial will help people with their own individualized financial plans, they could easily produce international transactions meaning that the program’s infrastructure should be prepared to support those international transactions securely. There are a multitude of government restrictions to consider. There are Unites States regulations, both federal and state, that require financial institutions to protect customer data, including through electronic communications. The Federal Trad Commission also mandates that trade data must be encrypted during transit. Internationally there are regulations such as the General Data Protection Regulation that requires encryptions and secure communications for all data belonging to the European Union. Also, depending on the country, personal data can be restricted internationally unless specific protections are put in place such as contracts and agreements. Artemis Financial will likely face common web application attacks, such as man-in-the-middle and injections. They could also face insecure authentication and potential supply chain compromises. Artemis Financial’s modernization must include secure management of open-source libraries, updated encryption protocols, and zero trust architecture, while adhering to compliance regulations.

**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.

The areas within the vulnerability assessment process flow diagram that apply to Artemis Financial’s software are input validation, APIs, cryptography, client/server, code error, and encapsulation. Strong input validation prevents data integrity issues, preventing injection attacks and cross-site scripting. Since the software will be driven by API and APIs are frequently targeted by attack, Artemis Financial must secure their interactions, such as authentication, authorization, and rate limiting. Cryptography ensures that the sensitive financial information is protected during transmission by encrypting it. This prevents data breaches as well as adhering to financial regulations. Artemis Financial’s software will be web-based requiring it have secure client and server communications. This will protect data in transit and prevent unauthorized access by securing session tokens, API keys, and backend service communications. Proper error handling ensures that the system does not expose details through different exploits, such as unhandled exceptions. Lastly, proper data encapsulation protects the internal system and sensitive information that could be involved from being tampered with by an unauthorized person.

**3. Manual Review**

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

While following the vulnerability assessment process flow diagram, I started reviewing for input validation. I checked the POM.xml file for any kind of validator, such as Apache, but found none. I then checked the GreetingController, and I found that all input is accepted straight from the user without any form of validation or sanitation. Accepting raw user input without validation allows for potential future exploitation. Next, I evaluated the APIs. I found that the input is passed through the URL, which poses security risks as this data can be exposed through server logs and history. Sensitive or critical information should be transmitted through POST methods to avoid unnecessary exposure. I found no evidence of encryption being used. This means that neither stored data nor transmitted data is protected. Given the financial nature of the program, Artimis Financial would need to implement proper encryption to meet industry and international compliance. When reviewing error handling, I found that DocData contains a basic try-catch block, but it is minimal. Beyond that, I did not see any consistent error handling throughout the code.

**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

Bcprov-jdk15on-1.46.jar – This version includes outdated cryptographic algorithms and known security flaws allowing attackers to exploit weaknesses in encryption, leading to potential data exposure or compromise.

Hibernate-validator-6.0.18.Final.jar – Older versions can contain vulnerabilities related to improper input validation, allowing attackers to bypass validation logic or exploit deserialization flaws. This could lead to injection vulnerabilities if untrusted inputs are not adequately validated.

Jackson-core-2.10.2.jar – Could be susceptible to denial-of-service attacks due to resource exhaustion vulnerabilities during parsing.

Jackson-databind-2.10.2.jar – Known deserialization vulnerabilities exist in this version, allowing attackers to exploit arbitrary code through input.

Log4j-api-2.12.1.jar – Older versions pose risks related to improper input handling, which could allow remote code execution.

Logback-classic-1.2.3.jar – May expose applications to insecure deserialization and improper input sanitation.

Logback-core-1.2.3.jar – May pose risks relating to insecure serialization.

Snakeyaml-1.25.jar – Can be vulnerable to remote code execution through deserialization of malicious YAML payloads, especially when untrusted input is involved.

Spring-boot-2.2.4.RELEASE.jar – may contain security vulnerabilities, such as input validation, deserialization, and HTTP request smuggling.

**Spring-boot-starter-web-2.2.4.RELEASE.jar – May expose the application to outdated dependencies. Vulnerabilities involve improper HTTP request parsing, validation bypasses, or component deserialization issues.**

**Spring-context-5.2.3.RELEASE.jar – Older versions can suffer from deserialization vulnerabilities, expression language injection, and insecure bean manipulation.**

**Spring-expression-5.2.3.RELEASE.jar- Vulnerabilities in SpEL handling can allow attackers to inject or execute unintended expressions, leading to arbitrary code execution if expression inputs are not properly controlled or sanitized.**

**Spring-wbmvc-5.2.3.RELEASE.jar – Can be vulnerable to path traversal, improper input handling, or deserialization attacks depending on configurations and user input handling.**

**Tomcat-embed-core-9.0.30.jar – This version is known to have vulnerabilities including request smuggling, HTTP header injections, and denial-of-service attacks since Tomcat handles incoming requests and responses.**

**Tomcat-embed-websocket-9.0.30.jar – May expose vulnerabilities related to improperly handled WebSocket connections. Exploits could involve denial-of-service or unauthorized access attacks.**

**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 current state of Artemis Financial’s software presents significant security vulnerabilities, particularly regarding unvalidated inputs, unsecured APIs, lack of encryption, and dependency risks. To mitigate these risks, they must implement a strong input validation and sanitization mechanism for all inputs, especially inputs passed through URLs. They must secure their API design by switching sensitive data handling from GET to POST to avoid exposing data in URLs and use HTTPS to protect data in transit. They must encrypt sensitive data before storing it in databases. They must standardize error responses and generic error messages to prevent leaking implementation details. Lastly they need to upgrade and replace outdated libraries so that the system is inherently guarded against certain vulnerabilities.