# CS 305 Project One

Dusty Cook / CS305 / 05-25-2025 / Professor Bolton

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/25/2025** | **Dusty Cook** |  |

## Client



## Developer

Dusty Cook

**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 focuses on developing tailored financial plans that cover savings, retirement, investments, and insurance. Given the management of sensitive information such as Social Security numbers and tax data, secure communication is crucial. While the company’s location remains unspecified, it is likely to engage in international transactions, necessitating adherence to global data security regulations. Safeguarding trade secrets and private client information is a significant legal priority. Implementing robust encryption is essential to defend against external threats. To enhance modernization, Artemis should emphasize regular updates and proactively address vulnerabilities in its third-party libraries and components.

**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.

Several security aspects identified in the vulnerability assessment are directly relevant to the Artemis Financials web application. Input validation is crucial since the application receives user input that could be subject to injection attacks if not adequately sanitized. The APIs responsible for transmitting sensitive financial data must be safeguarded with authentication, authorization, and throttling measures to mitigate the risk of abuse. Cryptography ensures that all data, whether in transit or at rest, is encrypted using contemporary standards. The client/server architecture should facilitate secure communication protocols and effective session management to protect against man-in-the-middle attacks and unauthorized access. Strong error handling is essential to prevent the application from revealing stack traces or other internal details that could benefit attackers. Implementing secure coding practices will lower the chances of introducing vulnerabilities during development. Finally, encapsulation helps safeguard sensitive data structures from unintended access, reinforcing access control throughout the codebase. These security considerations are critical for preserving the Artemis Financials application's confidentiality, integrity, and availability.

**3. Manual Review**

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

A manual examination of the codebase—specifically GreetingController.java, CRUDController.java, and RestServiceApplication.java—reveals multiple vulnerabilities in the vulnerability assessment procedure. In GreetingController.java, the name parameter is inserted directly into a string template without input validation or sanitization, posing a risk under Input Validation. The endpoint is unauthenticated and lacks protection, violating API Security best practices. There is no error handling for malformed input or failures, nor is there any rate limiting, making the endpoint prone to misuse.

In CRUDController.java, the /read endpoint accepts user input for business\_name and uses it to create a DocData object but only returns doc.toString() without verification or sanitization. This also presents a risk under Input Validation. Furthermore, there is no authorization, allowing access to potentially enumerate or probe business data, violating API Security principles. There is no error handling if DocData fails to initialize or raises an internal exception. This code lacks logging, CORS control, and output sanitization, relating to Code Quality and Secure API Practices.

In RestServiceApplication.java, while the main method appears safe, it lacks safeguards like secure configuration for TLS and startup validation. If misconfigured, the application might run in insecure development mode or expose sensitive configurations—issues related to Client/Server and Cryptography in production environments.

All three files lack authentication, authorization, and logging, which are crucial for security monitoring and access control. Additionally, input length constraints, output encoding, and structured error responses are missing, raising concerns under Input Validation, Code Error, and Encapsulation. These vulnerabilities highlight the need to implement secure coding practices from the outset, even in seemingly trivial endpoints.

**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 dependency-check report for Artemis Financials software application highlighted several known vulnerabilities in key third-party libraries. Significant vulnerabilities include CVE-2021-42550 affecting `logback-classic-1.2.3.jar`, which allows unsafe deserialization; resolving this requires an upgrade to version 1.2.9 or later. Additionally, `log4j-api-2.12.1.jar` has multiple critical issues, including CVE-2021-44832 and CVE-2020-9488, related to remote code execution through JNDI lookups; it is advised to upgrade to version 2.17.1 or later. The `jackson-databind-2.10.2.jar` is impacted by deserialization issues like CVE-2020-8840 and CVE-2020-9547 and should be updated to version 2.10.5.1 or newer. Similarly, `tomcat-embed-core-9.0.30.jar` contains CVE-2020-1938 vulnerability, which can leak internal files; upgrading to 9.0.31 or higher mitigates this risk. The `hibernate-validator-6.0.18.Final.jar` has CVE-2020-10693, which permits information disclosure due to inadequate constraint validation and should be resolved by upgrading to 6.1.5 or later. Furthermore, all Spring Framework components (such as `spring-web`, `spring-webmvc`, `spring-context`) at version 5.2.3 are affected by CVE-2020-5398, which causes path matching issues; upgrading to at least version 5.2.9 addresses this. The `snakeyaml-1.25.jar` library is vulnerable to CVE-2017-18640, a denial-of-service exploit initiated by deep recursion, and an upgrade to SnakeYAML 2.0 or higher is recommended. These vulnerabilities have been formally documented through publicly accessible sources, including the National Vulnerability Database (NVD), the MITRE CVE database, and official vendor advisories from Apache, Spring, FasterXML, Red Hat, and others.

**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.

Several strategic enhancements need to be executed to address the identified security vulnerabilities in Artemis Financials software. First, all user inputs on endpoints, such as greeting and reading, must be validated and cleaned to prevent injection and deserialization attacks. This involves enforcing input length limits and blocking potentially harmful characters. Authentication and authorization processes must be applied to all accessible routes to limit unauthorized access to the API. Additionally, rate limiting and logging should be introduced to detect and thwart brute-force attacks, while ensuring request traceability for audits. Standardizing exception handling is essential to avoid exposing internal stack traces to users. About static testing, all outdated dependencies flagged in the OWASP Dependency-Check report must be upgraded to secure versions. Specifically, crucial libraries such as log4j, jackson-databind, tomcat-embed-core, hibernate-validator, and all spring-\* modules must be updated to the latest patched versions as noted in the CVE summaries. Moreover, unused or unnecessary libraries should be removed to minimize the attack surface. Security measures should be established at the server level, including enforcing HTTPS and applying secure startup profiles. Regular scanning and patching of dependencies and automated checks in the CI/CD pipeline must be part of ongoing maintenance. Implementing these actions will enhance the application’s defenses against known vulnerabilities and emerging threats.