# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/23/2025** | **Sean Born** |  |

## 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

Sean Born

**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 important to Artemis Financial due to the sensitive nature of the financial data it handles, such as retirement, investments, and insurance information. Making sure the confidentiality, integrity, and availability of this data protects the company from financial fraud, reputational damage, and legal consequences. Secure communications such as through HTTPS, TLS, and end-to-end encryption help prevent interception or tampering by unauthorized parties during data transmission (Scarfone et al., 2008). Implementing secure communication protocols supports Artemis Financial’s trust with its clients.

Given Artemis Financial's global reach and its partnership with Global Rain, which serves clients worldwide, it is likely that the company engages in international transactions. These transactions may involve data transfers across borders, requiring compliance with data protection laws such as the General Data Protection Regulation (GDPR) in the European Union and other region-specific regulations (Voigt & Von dem Bussche, 2017). Such transactions heighten the need for robust encryption standards and secure APIs to maintain client trust and regulatory compliance across jurisdictions.

Yes, governmental restrictions on secure communications exist and must be considered. For example, some countries have limitations on encryption strength or require key disclosure to government agencies, which could impact Artemis Financial’s global operations. U.S. companies must also comply with export control laws such as the Export Administration Regulations (EAR) regarding the international use of cryptographic technology (BakerHostetler, 2021). These legal frameworks necessitate careful design of secure communication systems that follow both domestic and international legal requirements.

Artemis Financial faces a variety of external threats, including phishing, ransomware, SQL injection, and API endpoint attacks. As financial data is highly valuable, threat actors may increasingly target the company using sophisticated tactics like credential stuffing and zero-day exploits. The growing use of open-source components introduces risks from unpatched vulnerabilities and supply chain attacks (OWASP, 2023). Future threats may also include AI-driven cyberattacks and API abuse, emphasizing the need for proactive monitoring and layered defense strategies.

As Artemis Financial modernizes its systems, it must look into the integration of open-source libraries and frameworks, which provide flexibility and speed in development but also carry potential security risks if not properly managed. Using tools like Software Composition Analysis (SCA) can help identify vulnerabilities in third-party components (NIST, 2022). Adopting modern web application technologies such as microservices, containers, and serverless computing also requires updated security practices, including zero trust architecture and automated security testing within CI/CD pipelines. Making sure that security is being thought of throughout the software development lifecycle is important to sustaining modernization securely.

**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 elements of the Vulnerability Assessment Process Flow diagram that I find to be relevant to assess are input validation, APIs, code error, and code quality.

Input Validation is important because web applications accept data from users or external systems. These could potentially be malicious or malformed. If the system does not have proper validation attackers could exploit vulnerabilities such as injection attacks by submitting crafted inputs that change the intended behavior of the application (OWASP Foundation, 2021). Since Spring Expression Language allows dynamic evaluation of expressions, unsanitized inputs could lead to a security risks if attackers are able to inject malicious expressions (Seacord, 2013). By Validating all inputs it makes sure that only expected and safe data enters the system and protects it from unauthorized access or data corruption (OWASP Foundation, 2021).

APIs are the backbone of web applications. This allows for communication between the client and server or between various services. Secure APIs are important because they expose the application functionality and data to clients and possibly third-party systems. If API endpoints are not properly secured through authentication, authorization, and input validation they can be targeted to gain unauthorized access, leak sensitive data, or cause a denial of service (OWASP Foundation, 2021). Since spring-data-rest-webmvc automatically exposes Restful endpoints it is important to make sure that APIs enforce security policies such as role-based access control and limit data exposure (Fowler, 2019).

Code errors such as unhandled exceptions or logic bugs can be exploited by attackers or cause system crashes. In the Spring framework exceptions could inadvertently reveal sensitive stack traces or system information if not properly handled. Errors in logic could also lead to inconsistent application behavior or security bypasses. Error handling and logging can help maintain system stability, provide diagnostics for developers, and prevent leaking sensitive information that could help attackers (McGraw, 2006).

Maintaining high code quality is very important in secure coding. When the code is clean, well-structured, and tested it can help reduce the likelihood of vulnerabilities that could show up due to mistakes or misunderstandings. Poorly written code can have hidden bugs, race conditions, or logic errors that attackers can and will exploit (McGraw, 2006). In frameworks like Spring, sticking to coding standards, performing code reviews, and using automated tools for static analysis can help catch potential security flaws early. High-quality code also allows for easier maintenance and quicker response to vulnerabilities that do show up (Seacord, 2013).

**3. Manual Review**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Vulnerability Description** | **File Name** | **Line Number** | **Mitigation** |
| Java Version Outdated | pom.xml | 18 | Change version number to 11. |
| Outdated maven dependency | pom.xml | 60 | Change version number to 12.1.0 |
| Lack of configuration for https | application.properties |  | Add configuration for https on the application.properties file |
| File/Class names do not follow naming conventions | customer.java  myDateTime.java  CRUD.java  CRUDController.java |  | Rename class names to Customer.java, MyDateTime.java, Crud.java, CrudController.java; capitalizing the first letter of each word. |
| There is no input validation on inputting the date and time. | myDateTime.java |  | Implement validation of inputs for example using console.log(validateDateTime to check date and time inputs and make sure they follow formats and data limits |
| Access modifiers are not present like they are within all of the other classes | myDateTime.java |  | Implement access modifiers within the class to limit the accessibility of the class, methods, variables, and constructors. |
| Naming conventions for read\_document | DocData.java | 21 | Change to readDocument to follow proper naming conventions for java applications. |
| Username and password for default admin is exposed within the code. | DocData.java | 25-32 | Throw them into environment variables and then call them. |
| One variable is private and one is not | customer.java | 4 & 7 | I would make both private because event the accounts number could be sensitive information that needs to only be seen by authorized individuals |

**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

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dependency Name** | **Description** | **Severity** | **CVE Count** |
| [snakeyaml-1.25.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l16_8b6e01ef661d8378ae6dd7b511a7f2a33fae1421) | This reads YAML configuration files. Vulnerabilites could allow for arbitrary code execution when parsing malicious YAML content, especially if user input is not properly trusted. | CRITICAL | 8 |
| [spring-boot-2.2.4.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l17_225a4fd31156c254e3bb92adb42ee8c6de812714) | This is the main library for building Spring Boot applications. Vulnerabilities could allow attackers to gain unauthorized access. | CRITICAL | 3 |
| [spring-boot-starter-web-2.2.4.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l18_ec75d01d212b5229c16d872fb127744c0ed46ed8) | This includes all necessary web libraries for a Spring Boot web applications. Vulnerabilites could allow attackers to manipulate web inputs or access internal services. | CRITICAL | 3 |
| [spring-context-5.2.3.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l19_7750c95c96c7a1885c8b1b503ba915bc33ca579a) | This manages app configuration and dependency injection. Improper configuration or insecure bean definitions can cause privilege escalation or unsafe component loading. | CRITICAL\* | 34 |
| [spring-core-5.2.3.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l20_3734223040040e8c3fecd5faa3ae8a1ed6da146b) | This is the core functionality and utilities used by the entire spring framework. Vulnerabilities could allow for wide-reaching attacks like insecure deserilization or classpath manipulation. | CRITICAL\* | 12 |
| [spring-expression-5.2.3.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l21_d0c6bb10758805b2153c589686b8045554bfac2d) | This processes dynamic expressions within a spring application. Vulnerabilities could cause expression injection, which attackers might use to execute arbitrary cade or access sensitive data. | CRITICAL\* | 13 |
| [spring-web-5.2.3.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l22_dd386a02e40b915ab400a3bf9f586d2dc4c0852c) | This handles HTTP communications in Spring applications. This could make the system be exploited for cross-site scripting, cross-site request forgery, or injection attacks through malformed web request. | CRITICAL\* | 18 |
| [spring-webmvc-5.2.3.RELEASE.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l23_745a62502023d2496b565b7fe102bb1ee229d6b7) | This supports the model view controller pattern for web application. Vulnerabilities could allow attackers to bypass URL access restrictions or manipulate routes and data flow within the app. | CRITICAL\* | 13 |
| [tomcat-embed-core-9.0.30.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l24_ad32909314fe2ba02cec036434c0addd19bcc580) | This is the core part of the Tomcat server, which runs Java-based web applications. Vulnerabilities could allow for remote code execution directory traversal or misconfigured access controls which could compromise the entire application server. | CRITICAL\* | 40 |
| [tomcat-embed-websocket-9.0.30.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l26_33157f6bc5bfd03380ebb5ac476db0600a04168d) | This file allows for real-time communication features in web applications using WebSockets, such as live chats or notifications. Vulnerabilities could allow attackers to take over active sessions, inject malicious messages, or conduct denial of service attacks on live connections. | CRITICAL\* | 41 |
| [jackson-databind-2.10.2.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l6_0528de95f198afafbcfb0c09d2e43b6e0ea663ec) | This handles JSON conversion to and from Java objects. Attackers could create malicious JSON to trigger arbitrary code execution. | HIGH | 6 |
| [bcprov-jdk15on-1.46.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l1_991c96a4e31e6c19e2b9136c8955bd423f2dc4c7) | This is a cryptography library that provides encryption, decryption, and other security functions for Java applications. Possible vulnerabilities include weak or improperly implemented algorithms, padding attacks, or denial of service due to malformed input. This could allow attackers to bypass security controls. | HIGH | 18 |
| [logback-classic-1.2.3.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l13_7c4f3c474fb2c041d8028740440937705ebb473a) | This extends logback-cire with features for application logging. Vulnerabilities could expose sensitive system details which could be leveraged for further exploitation by attackers. | HIGH | 2 |
| [logback-core-1.2.3.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l14_864344400c3d4d92dfeb0a305dc87d953677c03c) | This provide foundational logging support. This could be used for log injection or information leakage, especially if logs include sensitive user data. | HIGH | 4 |
| [jackson-core-2.10.2.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l5_73d4322a6bda684f676a2b5fe918361c4e5c7cca) | This provides low-level tools for reading and writing JSON. Vulnerabilities could cause denial of service or parsing issues, especially when handling large or deeply nested data. | MEDIUM | 1 |
| [hibernate-validator-6.0.18.Final.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l3_7fd00bcd87e14b6ba66279282ef15efa30dd2492) | This performs input validation on data. Vulneribilies could allow attackers to bypass validation logic, leading to injection attacks, privilege escalation, or corrupted data storage. | MEDIUM | 3 |
| [log4j-api-2.12.1.jar](file:///C:\Users\lakin\eclipse-workspace\CS%20305%20Project%20One%20Code%20Base.zip_expanded\rest-service\target\dependency-check-report.html#l11_a55e6d987f50a515c9260b0451b4fa217dc539cb) | This supports the Log4j library. Vulnerabilities can allow attackers to execute arbitrary code remotely. | LOW | 1 |

**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.

Dependencies that should be focused on in initial efforts are tomcat-embed-websocket-9.0.30.jar, tomcat-embed-core-9.0.30.jar, spring-context-5.2.3.RELEASE.jar, and spring-web-5.2.3.RELEASE.jar should be prioritized due to their critical severity ratings and high CVE counts. These components play important roles in the system’s runtime, configuration, and web communication, and their compromise could result in remote code execution, session hijacking, or unauthorized access to sensitive components of the application.

Next, I would focus on the following dependencies snakeyaml-1.25.jar, spring-boot-2.2.4.RELEASE.jar, and spring-boot-starter-web-2.2.4.RELEASE.jar. These also have a critical severity but with fewer CVEs. Even though they have lower CVE counts, these dependencies are widely used in the app’s configuration and initialization phases, where improperly handled vulnerabilities such as code execution from malicious YAML files (CVE-2017-18640) or insecure auto-configuration could be exploited early in the application lifecycle (National Vulnerability Database, n.d.).

To mitigate these vulnerabilities the most effective approach is to upgrade to the latest stable versions of each dependency, as newer releases often patch known security issues. Additionally, for configuration-related libraries like SnakeYAML and Spring Context, it is important to limit the use of user-supplied input in configuration files. For web components such as Spring MVC and Web we can implement strict input validation and authentication controls (OWASP, 2024). Where upgrading is not immediately possible, using dependency constraints, application firewalls, and manual patches should be considered. For example, implementing network-level protections for Tomcat’s WebSocket support can help reduce the risk of session hijacking even if a vulnerable version is still in use.

References

BakerHostetler. (2021). *U.S. export controls on encryption*. <https://www.bakerlaw.com>

Fowler, M. (2019). *Patterns of enterprise application architecture*. Addison-Wesley.

McGraw, G. (2006). *Software security: Building security in*. Addison-Wesley Professional.

National Vulnerability Database. (n.d.). *CVE List*. <https://nvd.nist.gov/>

NIST. (2022). *Secure software development framework (SSDF)*. <https://csrc.nist.gov/publications/detail/white-paper/2022/02/04/ssdf/revision-1>

OWASP Foundation. (2021). *OWASP Top Ten 2021*. <https://owasp.org/www-project-top-ten/>

OWASP. (2023). *OWASP Top 10: The ten most critical web application security risks*. <https://owasp.org/Top10>

OWASP. (2024). *OWASP Top 10 – 2024: The Ten Most Critical Web Application Security Risks*. <https://owasp.org/www-project-top-ten/>

Scarfone, K., Souppaya, M., & Sexton, M. (2008). *Guide to secure web services*. NIST Special Publication 800-95. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf>

Seacord, R. C. (2013). *Secure coding in C and C++*. Addison-Wesley.

Voigt, P., & Von dem Bussche, A. (2017). *The EU General Data Protection Regulation (GDPR)*. Springer.