# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **September 20, 2025** | **Andriana Djurdjevic** | **Initial Vulnerability Assessment** |

## 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

Andriana Djurdjevic

**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’s web application handles highly sensitive information regarding client identities and financial information, retirement and investment details, and insurance planning data. These are all areas of high value and therefore need to be protected from interception and vulnerability making secure communications vital to the company’s structure. International transactions aren’t explicitly mentioned in the current platform, however investment and insurance methods are often intertwined with global markets or providers, and this makes international transactions also of crucial importance when developing the software security requirements.

As for government restrictions, since Artemis Financial is a financial services organization, they must comply with several governmental regulation and state-level regulations (Kost, 2025). These regulations emphasize the importance of safeguarding consumer data and encryption of customer information. The company may face phishing and credential theft as external threats, as well as injection attacks and API abuse. AS the company modernizes their systems, open-source libraries and encryption are of utmost importance to consider as well as cloud-readiness and evolving technologies and libraries.

**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.

Based on Artemis Financial’s RESTful web application, the following areas of security from the vulnerability assessment apply:

Input Validation: The API processes customer and financial data, which can be exploited and leaked if the inputs are not secure. Attackers could try using SQL injection or command injection through improper input handling, so there should be strong server side validation and processes in place to reject unexpected input.

APIs: Since the application relies on RESTful APIs to return financial information, they are a common attack path for unauthorized data access and exposure. Secure authentication and authorization checks as well as API gateway protections can keep users and the system safe.

Cryptography: The sensitive client data that this system uses must be encrypted both in transit and at rest. Using updated cryptographic methods could keep client data from being exposed. Utilizing strong cipher suites, secure key management and database encryption can all aid in security.

Client/Server Communication: Financial and investment planning demands secure exchanges of communications between clients and servers, for if communication is not protected then attackers may perform man in the middle attacks or session hijacking. Using session timeouts and multi-factor authentication for customer portals can aid in security.

Code Error Handling: Poorly handled code errors can expose stack traces, file paths, or sensitive data. Since the system handles financial records, this leakage could be exploited by attackers. Implementing user friendly but vague error messages while logging detailed errors for developers can help build continuous improvement in security.

Code Quality: Consistent and secure coding while using best practices reduces the likelihood of vulnerabilities during development. Since Artemis’s code base already uses open-source libraries, this makes following secure coding standards pertinent.

Encapsulation: Financial data must be protected from unauthorized access and requires proper encapsulation in classes and data models to keep unauthorized modification of sensitive information from happening. Using private fields with controlled access methods and validating data at the boundary level of the application can enforce secure structures.

**3. Manual Review**

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

1. No login or access control on the endpoints- In both CRUDController.java and GreetingController.java, the API methods are open to anyone. There’s nothing in the code that checks if a user is allowed to use them. This could mean anyone who finds the URL could run the code.
2. Request mapping is too broad- In CRUDController.java, the method uses @RequestMapping("/read"). From what I understand, this will accept any kind of HTTP request, and it would be better to limit this to only the correct type.
3. Input valuers aren’t checked- In both controllers, user input values go straight into the method and there is no check to make sure the input is safe or the right length.
4. Returning toString() directly- the code returns directly to toString() which might print more information than expected and could expose details about data.
5. No error handling- None of the controllers catch errors or handle them in a standard way, which could expose details of the system if a full error message shows that attackers can use to source information.
6. Fields in myDateTime aren’t private- there are no private fields which means that other parts of the program could change them directly.
7. No checks on time values- the methnod setMyDateTime doesn’t check if the numbers make sense.
8. No rate limit- there doesn’t seem to be a limit on how many requests can be sent, so if an attacker decided to spam the system there is no handling in place.

**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

|  |  |  |
| --- | --- | --- |
| Dependency Code | Vulnerability IDs | Description |
| bcprov-jdk15on-1.46.jar | cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46:\*:\*:\*:\*:\*:\*:\* cpe:2.3:a:bouncycastle:bouncy\_castle\_crypto\_package:1.46:\*:\*:\*:\*:\*:\*:\* | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7. |
| hibernate-validator-6.0.18.Final.jar | cpe:2.3:a:hibernate:hibernate-validator:6.0.18:\*:\*:\*:\*:\*:\*:\* cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\* | Hibernate's Bean Validation (JSR-380) reference implementation. |
| jackson-core-2.10.2.jar | cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\* | Core Jackson processing abstractions (aka Streaming API), implementation for JSON |
| jackson-databind-2.10.2.jar | cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\* | General data-binding functionality for Jackson: works on core streaming API |
| log4j-api-2.12.1.jar | cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\* | The Apache Log4j API |
| logback-classic-1.2.3.jar | cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\* | logback-classic module |
| logback-core-1.2.3.jar | cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\* | logback-core module |
| snakeyaml-1.25.jar | cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\* | YAML 1.1 parser and emitter for Java |
| spring-aop-5.2.3.RELEASE.jar | cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\* | Spring AOP |
| spring-boot-2.2.4.RELEASE.jar | cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\* | Spring Boot |
| spring-boot-starter-web-2.2.4.RELEASE.jar | cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:web\_project:web:2.2.4:release:\*:\*:\*:\*:\*:\* | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container |
| spring-context-5.2.3.RELEASE.jar | cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\* | Spring Context |

**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.

After looking at the code and the results of the dependency check, the following is a list of things that Artemis Financial can do to make their application safer:

1. Add login and permissions- as of right now there are no permissions for the endpoints, so adding security with logins and roles would ensure only authorized users can access them.
2. Using the right mapping- @RequestMapping should be changed so that the methods only accept the type of request that they are supposed to.
3. Check input values- Add validation to ensure inputs aren’t too long or filled with strange characters to protect integrity
4. Handle errors better- Add a global error handler so the user can see a friendly message if something goes wrong, but the technical details can be forwarded safely to the maintenance team.
5. Check time values- Add rules so seconds are only 0-59, minutes 0-59, and hours 0-23.
6. Add rate limiting- Put a limit on how many times someone can call the API in a short period so the system isn’t overloaded.

Dependency Fixes:

1. Update Lof4j to the newer version to address the serious issues.
2. Update SnakeYAML
3. Update Jackson Bouncy Castle
4. Update to the latest version of Spring Boot

Ongoing Maintenance:

1. Routinely review and run dependency checks so new vulnerabilities don’t pile up.
2. Avoid putting passwords or keys in the code.
3. Utilize logging and monitoring to watch for unwanted or suspicious activity.
4. Plan for regular updates to all dependencies and frameworks.

References

Jeganathan, S. (2019). DevSecOps: A Systemic Approach for Secure Software Development.

*ISSA Journal*, *17*(11), 20–27.

Kost, E. (2025, July 9). *Top 9 cybersecurity regulations for financial services*. UpGuard. https://www.upguard.com/blog/cybersecurity-regulations-financial-industry