# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/26/2025** | **Abhilash Krishna Raj** |  |

## 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

Abhilash Krishna Raj

**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 communication is very important for Artemis Financial to ensure client trust and to comply with data protection laws. Financial data must be encrypted to prevent unauthorized access. Secure communications also mitigate risks and protect against potential security threats like data breaches.

Since Global Rian specializes in software development for companies around the world, if Artemis Financial is involved in international transactions, they would need to follow global data protection rules like CRA for the EU. Using secure communication protocols such as HTTPS and TLS/SSL will help them handle international data securely and avoid legal issues.

Governmental regulations, depending on where Artemis Financial operates, should be met. In the US, they might need to follow ECPA for secure data transmission. If they work with clients in Europe, CRA rules might also apply. These standards focus on encryption, secure storage, and safe transmission of sensitive data.

Artemis Financial's RESTful web API could possibly currently face threats like injection attacks, broken object-level authorization, and man-in-the-middle attacks, and this could compromise sensitive financial data. Future risks from the company’s desire to modernize their operations could include vulnerabilities in open-source libraries, and misconfigured cloud infrastructure.

Artemis Financial considers the use of open-source libraries to speed up development and regularly update and test them for vulnerabilities to modernize their operations. They also need to consider using newer technology like microservices or serverless architectures to help modernize their operations while still being able to ensure strong API security and monitoring of their software.

**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.

Input validation is a relevant area of security to consider as it’s very important to defend against common threats like injection attacks, which are a major risk for APIs that handle user data. Since Artemis Financial processes sensitive financial data through its web application, validating user input helps ensure that malicious or unexpected inputs are filtered out. This minimizes risks like SQL injection or cross-site scripting (XSS), as they both could lead to data breaches.

API security is also relevant to consider as Artemis Financial’s has a RESTful API. APIs are often targeted by attackers for broken object-level authorization or improper authentication vulnerabilities. Securing the API with proper authentication mechanisms like OAuth2 and ensuring endpoints follow the principle of least privilege can help protect client data and prevent any unauthorized access.

Cryptography is another revenant area of security to consider as the company deals with sensitive financial data, and this means encrypting that data at rest and in transmission is very important. Cryptographic protocols such as TLS/SSL for secure communication help ensure that data transmitted between clients and the server is not intercepted or altered by attackers. Proper encryption also helps meet legal and regulatory requirements like ECPA and CRA.

Client/server security is another relevant area to consider as the application operates in a client-server environment, and this means that secure communication and data transmission between these the client and server is very important. This includes using secure session management, preventing man-in-the-middle attacks, and securing the server against unauthorized access. Proper server configuration and network security controls are also very important to use to protect sensitive data from 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.

1. In the CRUDController.java file, an input validation vulnerability was found where the /read endpoint in CRUDController uses the @RequestParam annotation to take input for business\_name. The input is passed directly to the DocData object without validation, and this makes them vulnerable to a risk of injection attacks like SQL Injection.
2. In the GreetingController.java file, an input validation vulnerability was found where the /greeting endpoint processes user input through the name parameter using @RequestParam. The input is directly included in the response without sanitization, and this makes it vulnerable to Cross-Site Scripting (XSS).
3. In the CRUDController.java file, an API vulnerability was found where the /read endpoint in CRUDController lacks authentication and authorization. This makes it vulnerable to unauthorized access and attacks.
4. In the GreetingController.java file, an API vulnerability was found where the /greeting endpoint in GreetingController does not have any rate limiting or logging mechanisms. This makes it vulnerable to brute force or denial-of-service (DoS) attacks to go undetected.
5. In the RestServiceApplication.java file, a cryptography vulnerability was found where the RestServiceApplication class does not enforce HTTPS or TLS/SSL encryption for communication. This makes any client-server data exchanges vulnerable to interception through man-in-the-middle attacks.
6. In the RestServiceApplicationTests.java file, a client/server vulnerability was found where the RestServiceApplicationTests class has only a basic test to check if the application context loads. There are no tests to verify the security or integrity of client-server communications, making it vulnerable.
7. In the DocData.java file, an input validation vulnerability was found where the read\_document method processes inputs like key and value for reading database records. These inputs lack validation or sanitization, making it vulnerable and possible for malicious data to execute SQL injection attacks.

**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 (CVE-2024-34447)
  + Description and Solution: The Bouncy Castle Crypto library improperly validates certificates with host mismatch, making it vulnerable to man-in-the-middle attacks. It is recommended to update to the latest version and enable stricter hostname verification in SSL/TLS configurations.
  + Attribution: OSSIndex, CWE-297 (Improper Validation of Certificate with Host Mismatch)
* hibernate-validator-6.0.18.Final.jar (CVE-2023-1932)
  + Description and Solution: The SafeHtmlValidator in Hibernate Validator allows HTML Injection or Cross-Site Scripting (XSS) due to improper tag sanitization. It is recommended to upgrade to a patched version and implement additional input validation at the application level.
  + Attribution: OSSIndex, CWE-79 (Improper Neutralization of Input During Web Page Generation)
* jackson-databind-2.10.2.jar (CVE-2020-25649)
  + Description and Solution: Jackson Databind improperly secures entity expansion, making it vulnerable to XML External Entity (XXE) attacks. It is recommended to update to the latest version and validate all XML inputs before processing them.
  + Attribution: OSSIndex, CWE-611 (Improper Restriction of XML External Entity Reference)
* log4j-api-2.12.1.jar (CVE-2020-9488)
  + Description and Solution: Log4j’s SMTP appender improperly validates certificates, leading to man-in-the-middle attacks during SMTP logging. It is recommended to update to version 2.12.3 or later where this is fixed.
  + Attribution: OSSIndex, CWE-295 (Improper Certificate Validation)
* logback-classic-1.2.3.jar (CVE-2023-6378)
  + Description and Solution: A deserialization vulnerability in Logback's receiver component allows denial-of-service attacks using malicious data. It is recommended to upgrade to a secure version and disable deserialization of any untrusted inputs.
  + Attribution: OSSIndex, CWE-502 (Deserialization of Untrusted Data)
* snakeyaml-1.25.jar (CVE-2022-1471)
  + Description and Solution: SnakeYaml’s Constructor class allows unrestricted deserialization, enabling remote code execution. It is recommended to upgrade to version 2.0 or later and use SafeConstructor for YAML parsing.
  + Attribution: OSSIndex, CWE-502 (Deserialization of Untrusted Data), CWE-20 (Improper Input Validation)
* spring-boot-2.2.4.RELEASE.jar (CVE-2023-20873)
  + Description and Solution: Spring Boot deployed to Cloud Foundry can allow unauthorized access. It is recommended to upgrade to version 3.0.6 or 2.7.11 for a security fix.
  + Attribution: VMware Advisories, CWE-noinfo (Security Bypass)
* tomcat-embed-core-9.0.30.jar (CVE-2020-1938)
  + Description and Solution: The default AJP configuration in Tomcat enables file access and JSP execution, allowing remote code execution. It is recommended to upgrade version 9.0.31 or later and disable the AJP connector if unnecessary.
  + Attribution: Apache Tomcat Security Advisories, CWE-noinfo (Improper Privilege Management)
* spring-webmvc-5.2.3.RELEASE.jar (CVE-2022-22965)
  + Description and Solution: Spring MVC applications on JDK 9+ are vulnerable to remote code execution (RCE) via unsafe data binding. It is recommended to upgrade version 5.2.20 or later and avoid WAR deployments if unnecessary.
  + Attribution: CISA Known Exploited Vulnerabilities, CWE-94 (Improper Control of Code Generation)
* spring-expression-5.2.3.RELEASE.jar (CVE-2022-22965)
  + Description and Solution: Spring’s Expression Language component allows RCE attacks under specific conditions (e.g., JDK 9+ on Tomcat with WAR deployment). It is recommended to upgrade to version 5.2.20 or later and mitigate risks by deploying as an executable JAR.
  + Attribution: VMware Security Advisories, CWE-94.

**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.

1. Input Validation Vulnerabilities

* Location: CRUDController.java (/read endpoint), GreetingController.java (/greeting endpoint), DocData.java (read\_document method).
* Action Plan: Implement strict input validation and sanitization for all user inputs, such as using libraries like OWASP Java Encoder for encoding data to prevent injection attacks. Also, validate user inputs for acceptable formats, lengths, and characters using frameworks like Hibernate Validator.

1. API Security Vulnerabilities

* Location: CRUDController.java (/read endpoint), GreetingController.java (/greeting endpoint).
* Action Plan: Introduce authentication mechanisms, like OAuth2 or JWT, to restrict access to API endpoints. Enforce role-based access control (RBAC) as well to limit permissions based on user roles. Also, implement rate-limiting and logging mechanisms to monitor and prevent brute-force and denial-of-service (DoS) attacks.

1. Cryptography Vulnerabilities

* Location: RestServiceApplication.java (lack of TLS/SSL encryption).
* Action Plan: Enforce HTTPS for all client-server communications using TLS 1.2 or higher. Use HSTS (HTTP Strict Transport Security) headers to ensure secure connections persist.

1. Client/Server Security Vulnerabilities

* Location: RestServiceApplicationTests.java (insufficient testing for secure communications).
* Action Plan: Develop and include automated security tests to verify secure communication protocols. Use tools like OWASP ZAP or Burp Suite to test for client-server vulnerabilities during the build process.

1. Outdated Dependencies

* Location: Found in dependency-check report (jackson-databind-2.10.2.jar, log4j-api-2.12.1.jar, logback-classic-1.2.3.jar).
* Action Plan: Update all vulnerable dependencies to their latest secure versions, following the vulnerability’s advisory notes. Also, regularly run dependency checks using tools like OWASP Dependency-Check to identify new vulnerabilities.

1. Default Insecure Settings

* Location: tomcat-embed-core-9.0.30.jar (AJP connector vulnerability).
* Action Plan: Disable the AJP connector unless explicitly required and upgrade to Apache Tomcat version 9.0.31 or later for secure default configurations.

1. Serialization Vulnerabilities

* Location: logback-classic-1.2.3.jar, snakeyaml-1.25.jar.
* Action Plan: Use secure deserialization methods, such as enabling SafeConstructor in SnakeYAML to restrict the deserialization of malicious data. Also, disable or replace unnecessary deserialization mechanisms.

1. Remote Code Execution (RCE) Risks

* Location: spring-webmvc-5.2.3.RELEASE.jar, spring-expression-5.2.3.RELEASE.jar.
* Action Plan: Upgrade to patched versions of Spring Framework (5.2.20 or later). Also, avoid deploying the application as a WAR file to Tomcat when not necessary and instead use Spring Boot's executable JARs.

1. Man-in-the-Middle (MITM) Vulnerability

* Location: bcprov-jdk15on-1.46.jar, log4j-api-2.12.1.jar.
* Action Plan: Enable hostname verification in SSL/TLS configurations for affected libraries and replace/update libraries with known secure versions.