# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **01/20/2025** | **Maxwell J. Sciola** | **First edition** |

## 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

Maxwell J. Sciola

**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:** Secure communication is necessary, regardless of whether the software handles personal information or applications that do not involve sensitive information. A secure communication line plays a huge role in preventing unauthorized access and cyberattacks, especially when sensitive data is involved.

Our client, *Artemis Financial*, is a leader in personalized financial planning and insurance services and collects sensitive client information such as income details and investment portfolios. In order to have clients entrust *Artemis Financial* with sharing this type of information, it is a requirement to prioritize vigorous security measures. To increase Artemis Financials' services, open communication needs to be encouraged, and trust needs to be built by upholding confidentiality and keeping clients' personal information safe.

**International Transactions:** A thorough review of *Artemis Financials'* documents and existing codebase provided no evidence of international transactions within their operations. Nerveless, no documentation ruled out the possibility of international transactions. With no clear answer, it is necessary to take a cautious approach.

Even if international transactions aren't explicitly mentioned, it is best practice for security reasons to consider a potential future with international dealings. Even though there are no immediate plans, preparedness is needed with the lack of explicit exclusion. Artemis Financial should implement a robust and comprehensive security framework to secure all communications.

Within this framework, domestic and international communication and transactions must be addressed to ensure all sensitive information is properly protected. This mitigates the risk associated with international transactions and preserves the organization's operational side.

**Governmental Restrictions:** Within the United States, there is no current enforcement of specific regulations requiring formal evaluation of encryption or communication security; however, there has been a shift towards changing that. On January 16, 2025, President Biden issued an executive order aimed at promoting innovation in the nation's cybersecurity.

The executive order makes federal agencies increase the security and integrity of high-risk software, highlighting the need for software development to be more secure and have rigorous third-party risk management. In attempt to standardize encryption policies across the nation, a bill was introduced named "ENCRYPT Act of 2023 (H.R.5311)" where it aimed to apply state-level mandates on data security vulnerabilities and decryption requirements.

These actions demonstrate the government's new focus on our nation's cybersecurity and data protection. Proactively adopting security measures that comply with regulations may be in Artemis Financials’ best interest.

**External Threats:** Since *Artemis Financial* manages sensitive financial and personal information, there are significant external threats in this digital world. Individuals or small groups are the primary concern for *Artemis Financial* rather than high-level actors or even state-sponsored attackers.

Some of the threats come from different vulnerabilities, such as SQL injections, invalid inputs, and data leaks, such as usernames and passwords exposed through browser histories. Other reasons for the increased risk are dependency vulnerabilities and automated bots programmed to complete repeated login attempts, which cause denial-of-service attacks. Another potential danger comes from privilege escalation, which allows authorized users to access restricted data, in this case, an end-user getting administrator-level access.

To ease these potential risks, they must employ a vigorous API focused on preventing attacks, secure access control, and systematically addressing vulnerabilities.

**Modernization Requirements:** New vulnerabilities can be introduced by modernizing old systems, particularly from external software or when open-source libraries are integrated. Incorporating external stock and bond data to increase financial planning services, for example, may improve offerings and increase exposure to potential security flaws. If their APIs and underlying structures are inadequate, then modern software tools can help in functionality but also become liabilities. If an API is poorly designed, then it becomes a target for attackers, possibly compromising sensitive data. If an application requires permission or user data, as web applications technologies evolve, so do the risk and attack surfaces.

A way for *Artemis Financial* to adopt a proactive security strategy is to prioritize security measures, addressing vulnerabilities while planning and prepare for unexpected system failures.  The company can modernize in a successful manner while also guarding its system and sensitive data by implementing those precautions.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financials’ software application. Justify your reasoning for why each area is relevant to the software application.

**Input Validation:** Input validation is the foundation of secure software development, especially when processing user-provided data. This can be seen within the program CRUDController.java line 12 when it accepts input as a string, emphasizing the need to validate inputs to keep the application secure. Proper validation is needed to prevent SQL injection attacks and software failures. This means formatting, length and acceptable values should be verified. SQL injection lets attackers insert malicious SQL code, compromising the security of the database and releasing sensitive data. With comprehensive input validation, the application can prevent SQL injection and ensure security. Being proactive and mitigating possible malicious user behavior is vital for protecting the system.

**APIs:** Due to the fact that this application is designed to function in external environments, like web browsers, a strong API is necessary. API outlines how outside software interacts with the application, it identifies acceptable methods and data type. Dependencies like integration of third-party software increases the risks but a secure API can mitigate those risks.

**Cryptography:** In order to have secure international transfers and encrypted communications, cryptography plays a critical role. The U.S. and other countries classify encryption as an export item. A strong encryption signals confidentiality of sensitive information during oversea business to Artemis Financial clients. A dual focus on compliance and security allows the organization to have lawful operations in other countries while also remaining secure.

**Error Handling:** When processing user inputs, secure error handling is critical to application security. Vulnerabilities like privilege escalation can be exploited by malicious or malformed inputs. Proper error management is needed along with mitigating risks to ensure the user's experience. So, error messages must be understandable to the average person and not cryptic in technical details. A way to protect the application and its user from potential threats is by combining robust error management with rigorous input validation.

**Code Quality:** When dealing with input validation and APIs, it is important to have a high-level of code quality. If code is written well enough, it ensures methods not intended for public use remain inaccessible and prevents unintentional data exposure.

**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. Hardcoded Database Credentials**

**File:** DocData.java  
**Issue:** Database credentials are hardcoded in the connection string (“root”,” root”)

**Risk:** Attackers can gain unauthorized access to the database if this code gets exposed.

**2. Missing Input Validation**

**File:** CRUDController.java, GreetingController.java  
**Issue:** @RequestParam user inputs are not validated or sanitized

**Risk:** The application is exposed to injection attacks like SQL injection or script injections

**3. Insecure Default Values**

**File:** GreetingController.java  
**Issue:** Without validation, the default value “World” is set in the greeting endpoint.  
**Risk:** This can be manipulated and be used for script injection attacks.

**4. Lack of Exception Handling in Database Operations**

**File:** DocData.java  
**Issue:** SQL exceptions are caught by the read\_document method but only prints the stack trace.

**Risk:** This can lead to denial-of-service (DoS) attacks because there is no mechanism to recover or log critical issues securely.

**5. Missing Access Modifiers for Class Fields**

**File:** customer.java, myDateTime.java  
**Issue:** Appropriate access modifiers are missing in fields like account\_balance and mySecond.

**Risk:** Due to a lack of encapsulation, unintentional modifications of sensitive data could occur.

**6. Potential SQL Injection**

**File:** DocData.java  
**Issue:** There are placeholders from the read\_document method for SQL queries but doesn’t use prepared statements.

**Risk:** The application will be vulnerable to SQL injection attacks if raw SQL queries are implemented there.

**7. Redundant Code in CRUD Class**

**File:** CRUD.java  
**Issue:** The CRUD constructor prepares content with the same value as content2.  
**Risk:** This can lead to unexpected behavior in data handling or confusion.

**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

**1. Dependency: bcprov-jdk15on-1.46.jar**

* **Vulnerabilities**:

cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46:\*:\*:\*:\*:\*:\*:\*  
cpe:2.3:a:bouncycastle:bouncy\_castle\_crypto\_package:1.46:\*:\*:\*:\*:\*:\*:\*  
[cpe:2.3:a:bouncycastle:bouncy\_castle\_for\_java:1.46:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Abouncy_castle_for_java&cpe_version=cpe%3A%2F%3Abouncycastle%3Abouncy_castle_for_java%3A1.46)  
[cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api&cpe_version=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api%3A1.46)  
cpe:2.3:a:bouncycastle:the\_bouncy\_castle\_crypto\_package\_for\_java:1.46:\*:\*:\*:\*:\*:\*:\*

* **Description**:

Outdated Bouncy Castle library and has known vulnerabilities like potential security flaws in cryptographic functions and multiple others.

* **Recommended Solution**:  
  Upgrade bcprov-jdk15on library to the latest stable version and check the website of Bouncy Castle for the latest updates.

**2. Dependency: hibernate-validator-6.0.18.Final.jar**

* **Vulnerabilities**:

[cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aredhat&cpe_product=cpe%3A%2F%3Aredhat%3Ahibernate_validator&cpe_version=cpe%3A%2F%3Aredhat%3Ahibernate_validator%3A6.0.18)

* **Description**:  
  Known vulnerabilities with the Hibernate Validator library related to improper validation of user inputs, at risk for injection attacks.
* **Recommended Solution**:  
  Refer to Hibernate Validator’s release notes and upgrade to a compatible secure version.

**3. Dependency: jackson-databind-2.10.2.jar**

* **Vulnerabilities**:

[cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Afasterxml&cpe_product=cpe%3A%2F%3Afasterxml%3Ajackson-databind&cpe_version=cpe%3A%2F%3Afasterxml%3Ajackson-databind%3A2.10.2)  
cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\*

* **Description**:  
  The Jackson Databind library will allow remote code execution from its known deserialization vulnerabilities.
* **Recommended Solution**:  
  Ensure CVEs are addressed, avoid insecure deserialization practices and upgrade to version 2.13.x or higher.

**4. Dependency: log4j-api-2.12.1.jar**

* **Vulnerabilities**:

[cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Alog4j&cpe_version=cpe%3A%2F%3Aapache%3Alog4j%3A2.12.1)

* **Description**:  
  Outdated versions of Log4j are vulnerable to logging vulnerabilities, but this specific version might have lower severity issues.
* **Recommended Solution**:  
  For vulnerability patch upgrade to Log4j 2.17.x.

**5. Dependency: snakeyaml-1.25.jar**

* **Vulnerabilities**:

[cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Asnakeyaml_project&cpe_product=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml&cpe_version=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml%3A1.25)

* **Description**:
* Remote code execution risk due to SnakeYAML library having vulnerabilities in YAML parsing.
* **Recommended Solution**:  
  Configure safe YAML parsing settings and upgrade to version 1.30 or newer.

**6. Spring Framework and Dependencies**

* **Dependencies**:
  + spring-boot-2.2.4.RELEASE.jar
  + spring-core-5.2.3.RELEASE.jar
  + spring-web-5.2.3.RELEASE.jar
* **Vulnerabilities**:

|  |  |
| --- | --- |
| spring-boot-starter-web-2.2.4.RELEASE.jar  spring-core-5.2.3.RELEASE.jar  spring-web-5.2.3.RELEASE.jar | [cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_boot&cpe_version=cpe%3A%2F%3Avmware%3Aspring_boot%3A2.2.4) [cpe:2.3:a:web\_project:web:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aweb_project&cpe_product=cpe%3A%2F%3Aweb_project%3Aweb&cpe_version=cpe%3A%2F%3Aweb_project%3Aweb%3A2.2.4)  [cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Apivotal_software&cpe_product=cpe%3A%2F%3Apivotal_software%3Aspring_framework&cpe_version=cpe%3A%2F%3Apivotal_software%3Aspring_framework%3A5.2.3) [cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aspringsource&cpe_product=cpe%3A%2F%3Aspringsource%3Aspring_framework&cpe_version=cpe%3A%2F%3Aspringsource%3Aspring_framework%3A5.2.3) [cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A5.2.3)  [cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Apivotal_software&cpe_product=cpe%3A%2F%3Apivotal_software%3Aspring_framework&cpe_version=cpe%3A%2F%3Apivotal_software%3Aspring_framework%3A5.2.3) [cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aspringsource&cpe_product=cpe%3A%2F%3Aspringsource%3Aspring_framework&cpe_version=cpe%3A%2F%3Aspringsource%3Aspring_framework%3A5.2.3) [cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A5.2.3) [cpe:2.3:a:web\_project:web:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aweb_project&cpe_product=cpe%3A%2F%3Aweb_project%3Aweb&cpe_version=cpe%3A%2F%3Aweb_project%3Aweb%3A5.2.3) |

* **Description**:  
  Both Spring Framework and Spring Boot have critical vulnerabilities with Remote Code Execution and deserialization flaws.
* **Recommended Solution**:

Follow the official Spring security advisories and update to the latest versions and its related libraries.

**7. Tomcat Embedded Libraries**

* **Dependencies**:
  + tomcat-embed-core-9.0.30.jar
  + tomcat-embed-websocket-9.0.30.jar
* **Vulnerabilities**:

tomcat-embed-core-9.0.30.jar cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*

cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*

tomcat-embed-websocket-9.0.30.ja cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*

cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*

* **Description**:

Outdated versions of Apache Tomcat have vulnerabilities associated to WebSocket handling and input validation

* **Recommended Solution**:  
  Address all CVEs and update to the latest stable version.
* Upgrade to the latest stable version, addressing all known CVEs.

**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 Financials’ software application.

**1. Secure Configuration Management**

Hardcoded database credentials in the code like in the DocData.java file causes a security risk. To lessen this risk, remove the hardcode credentials and use environment variables. Another way to lessen the risk is to use secure configuration tools like AWS Secerts Manager or HashiCorp Vault in order to store sensitive information in a secure manner. Also, to protect data in transit, ensure that database access uses encrypted communications.

**2. Input Validation and Sanitization**

Files like GreetController.java and CRUDController.java have improper input validation and GreetController.java also has insecure default values. Using server-side frameworks like Hibernate Validator can correct these issues by validating all user inputs. This can help against injections attack, especially when inputs are sanitized with OWASP Java encoder. Lastly, utilize validation annotations like @Size and @NotNull whenever possible and validate default values and escape before use.

**3. Enhanced Error Handling**

The lack of exception handling in DocData.java potentially leads to information leaks and unhandled errors. To secure logging use error-handling mechanisms and frameworks like Logback and SLF4J. Secure logging exceptions and incorporate retry mechanisms or failover procedures to manage critical database error to avoid displaying stack traces to users.

**4. Proper Encapsulation**

Encapsulation is compromised by files customer.java and myDateTime.java because they contain class fields without access modifiers. Change class fields to private and provide controlled access through getter and setter method to improve security and maintainability. To stop unintended modifications to sensitive data, implement immutable practices when possible.

**5. Secure Database Queries**

DocData.java file is prone to SQL injection attacks due to it containing raw SQL queries. Use PreparedStatement to replace raw SQL queries with parameterized queries. To avoid common SQL vulnerabilities, use ORM frameworks like JPA or Hibernate to manage database interaction in a safe manner.

**6. Code Redundancy Cleanup**

CRUD.java has redundant code that can cause unexpected behavior and unclear logic. Restructure the code to reduce redundancies to increase clarity and maintainability. Implement conduct code reviews to ensure best practices are used and to prevent issues from spreading throughout the code.

**7. Dependency Management**

Outdated dependencies like Bouncy Castle, Hibernate Validator, Jackson Databind, Log4j, SnakeYAML, Spring Framework, and Tomcat have been identified through static testing. The best course of action will be to update these dependencies to their newest version to address known vulnerabilities. To keep the application up to date with security, schedule audits to occur regularly with OWSAP Dependency-Check to help monitor for new vulnerabilities and patch them in a timely manner.

**8. Secure Development Practices**

Following guidelines from OWASP and SEI CERT will help keep the code secure. Additionally, static and dynamic code review needs to be implemented to identify vulnerabilities early in the development process. Keep developers trained in the best coding practices to ensure they are up to date with the newest security challenges and techniques.

**9. Testing and Validation**

Verify vulnerabilities have been resolved by conducting testing after introducing mitigation steps. Automated testing tools should be used to validate database query safety and input sanitization. Error-handling mechanisms should be tested systematically to ensure they support security policies and protection is strong against attacks.

**10. Documentation and Monitoring**

There should be detailed documentation of all mitigations for audit and compliance reasons. Have a real time monitoring system to detect and respond to vulnerabilities. Use tools like Splunk for action to be taken in a prompt manner against potential threats and to log aggregation and security event monitoring to enhance system visibility.