# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/20/25** | **Hailey Bondra** |  |

## 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

Hailey Bondra

**1. Interpreting Client Needs**

The value of secure communications to Artemis is essential as the company handles sensitive data such as personal financial details and accounts. Data is passed between the customer and the financial system, and it is important to maintain compliance with data protection rules and regulations.

There is nothing specific noting international transactions, but it is very possible that there could be customers who are overseas and would like to use Artemis Financial. For the company to have the ability to operate across different countries the company would need to be compliant with the rules and regulations specific to international data transfers. An example is the system would need to be compliant with the European General Data Protection Regulation (GDPR) to operate in select countries. The GDPR is run by the European Union designed to protect citizens from data compromises.

Artemis Financial must consider some U.S. government restrictions in handling sensitive data. An example is following the Sarbanes-Oxey act (SOX), SOX is a framework that ensures financial record accuracy by ensuring security practices are enforced. A few other restrictions that need to be considered is the Gramm-Leach-Biley Act (GLBA) for financial privacy and the Payment Card Industry (PCI) Data Security Standards (DSS) for protecting credit cardholders’ details.

Some external threats that are present now are injection attacks, phishing, ransomware, and DDoS attacks. Injection attacks happen when attackers send malicious code into a system to reveal sensitive information, allow unauthorized access, and manipulate the behavior of the system. Phishing is where attackers impersonate a real system and trick individuals whether it is a customer or an authorized employee to reveal sensitive information which is then manipulated and abused by the attacker. Ransomware happens when the attacker restricts access to a system in exchange for something in return, typically a ransom payment. DDoS attacks occur when the attacker floods the system with internet traffic to make the system inoperable to authorized users and potentially shut down the system due to so much traffic.

Some modernization requirements that need to be considered are some open-sourced libraries as they provide a template that has been tried and true by others to make the development and testing process more efficient. Some libraries that should be considered are Spring Boot and Front-end frameworks like Angular and Bootstrap. Some evolving web applications that should be considered as well are Single-page applications, Cloud and DevOps. Cloud and DevOps is a cloud platform that gives a place to perform DevOps practices and build, test, and deploy software efficiently. Single-page applications is a web application where an HTML page is operating and updates content as the user interacts with it instead of reloading the entire page.

**2. Areas of Security**

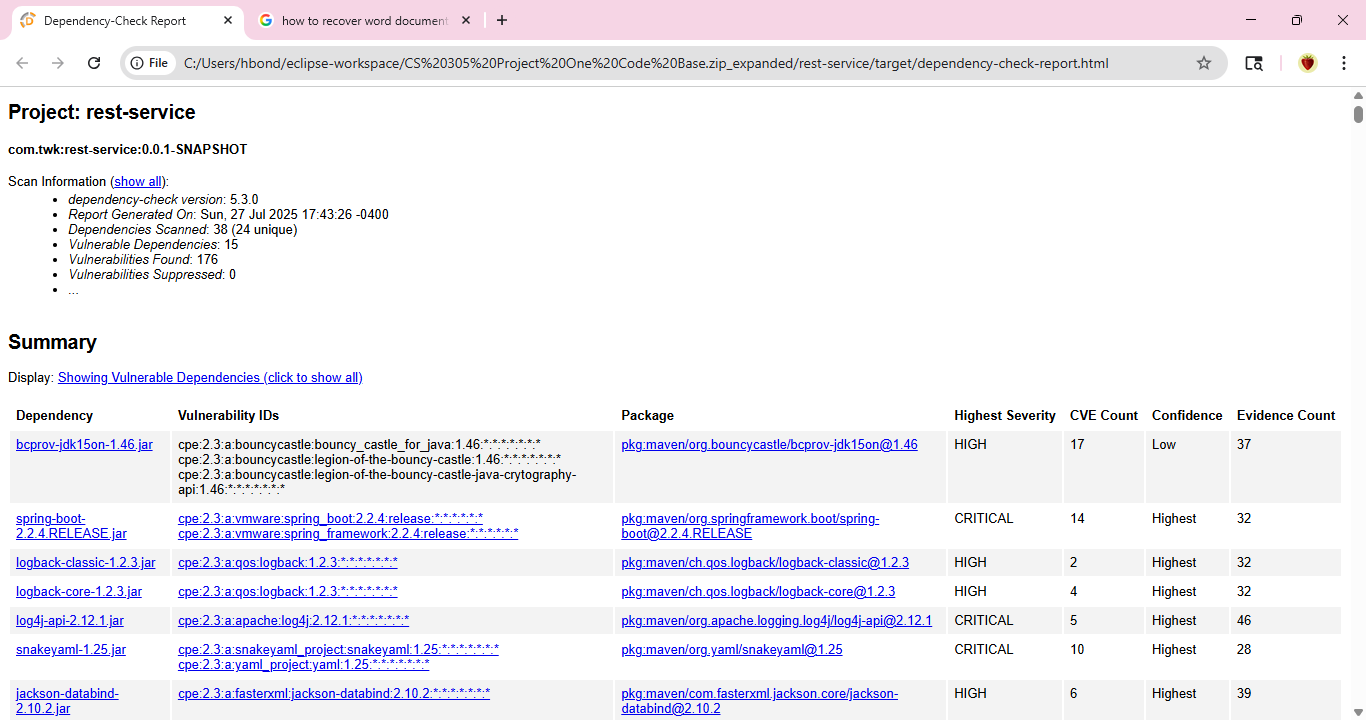
Input validation is relevant to the application as it is one of the most important areas to have as the financial system will constantly be taking in user’s personal financial information. Validating input will help prevent injection attacks when the financial system takes the user’s input, but additionally sanitizing and encoding the system properly will take it to the next level in security. APIs are relevant as the use of RESTful API exposes the backend services of the financial system to client applications. Making sure APIs are secure is important to prevent abuse of access and information leaks. Rate limiting is integrated into the system as well as authorization checks and end point authentications to make sure the APIs follow REST principles. Cryptography is relevant to the application as these practices will ensure that any personal financial information is handled properly and encrypted. This can be done by creating strong cryptographic algorithms that will maintain confidentiality of user information. Client/server model is important as the client interacts with the backend server using APIs. The model is structured to manage financial data and follow security protocols such as encryption and authentication measures. This is necessary in this application to ensure that the application meets not only compliance regulations but protects the integrity of the application from attackers. Code error is relevant as error handling not done properly can leak information to attackers. Small errors can lead to critical consequences, especially for a financial company that handles very sensitive and personal user information. Code quality is relevant as ensuring the code that makes up the system meets the standards for secure coding. Poor code quality leads to a program that does not work how it is supposed to or errors in the code that could potentially leak information. Having excellent code quality is essential for a financial system that would be handling user’s financial accounts as a logic error could have major effects on user financial accounts and the integrity of Artemis Financial. Encapsulation is not relevant to the application as the focus is not on internal data hiding and more on interface level security.

**3. Manual Review**

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

* Input Validation
  + GreetingController.java and CRUD.java files:
    - Both take inputs directly from parameter @RequestParam and are passed to functions without validation or sanitization. Any input or command from the user will be accepted and passed through right now.
* APIs
  + GreetingController.java and CRUDController.java files:
    - The parameter in CRUDController.java called @RequestParam(“business\_name”) is accepted but never used. This is an unnecessary API implementation and could become a security risk if misused by users.
    - Both files send back data objects directly to the client without wrapping them in a response format or status code. The API structure is inconsistent as it cannot tell the client if something went wrong like an error as the data object is always sent over. Clients could also have difficulty understanding responses and fail to properly handle errors.
* Cryptography
  + The entire project has no encryption for the sensitive data taken in. Without having sensitive data protected it is at high risk of an attack as it is currently stored in plain form and can be sent in plain form.
* Client/Server
  + The entire project has no session handling as it does not keep track of who is logged in and anyone could use the API. It also does not protect against any attacks on users who are logged in, so users could fall into an attacker’s trap by performing actions the user did not mean to do such as transferring money or changing their information.
* Code Error
  + myDateTime.java file:
    - The method setMyDateTime() has parameters but there is no functionality to it. The second method in the file retrieveDateTime() returns an empty array. Both of these functions are logic errors from the developer as the program would still run, but they have no purpose.
* Code Quality
  + Customer.java file:
    - The data inside the class is not protected and can be changed anytime. To improve code quality, implement sensitive data as private attributes and control access to these attributes using accessor and mutator functions.

**4. Static Testing**



A screenshot of a computer

AI-generated content may be incorrect.

* bcprov-jdk15on-1.46.jar
  + Is not the latest version and so encryption keys are more predictable, crashes or leaks are more prone to an older version
* logback-core-1.0.13.jar
  + This file is prone to injection attacks due to not being the latest version downloaded.
* logback-classic-1.0.13.jar
  + This file and logback-core-1.0.13.jar are the same vulnerabilities and the best solution is to update to the latest version
* commons-beanutils-1.8.3.jar
  + File is out of date and is at risk of classloader manipulation, where an attack would take control of the process during a Java application load and interpret code.
* xercesImpl-2.11.0.jar
  + An attacker could create an XML file that uses too much memory in order to slow down and potentially crash the system. Switching to a different and safer XML parser would reduce the risk.

**5. Mitigation Plan**

To address the vulnerabilities found in the testing of Artemis’ Financial application, the following steps need to be taken immediately to ensure the integrity of the application. The testing identified at least four outdated libraries used in the application. These libraries put the application at risk of external attacks such as injection attacks, Denial-of-service (DoS) attacks, and classloader manipulation. The remedy is to update the following libraries to the latest version: bcprov-jdk15on-1.46.jar, logback-core-1.0.13.jar , logback-classic-1.0.13.jar , and commons-beanutils-1.8.3.jar. If left unaddressed, attacks could lead to data breaches and loss, reputational and financial damage, remote code executions, and system compromise. The final vulnerability identified was the use of the Xerces XML parser. The current version allows attacks to create XML files and overload the system leading to a DoS attack. The best solution is to replace it with a more secure XML parser such as Java API for XML Processing (JAXP). Overall, implementing these steps will help build a secure application where Artemis Financial can be confident in strongly securing client data.