# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/25/24** | **Peter Krentz** |  |

## 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

Peter Krentz

**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?

All communications between the company and it’s clients, as well as internal discussions need to be secure so that their clients can have faith that the company will keep their information private. While it is not explicitly stated, the application does make transactions and keep a record of each client’s balance. Assuming this application is run by a US based company, there would be governmental restrictions in the form of the Securities and Exchange Commission (SEC) and depending on what some of these financial plans hold as assets, they may also be regulated by the Commodity Futures Trading Commission (CFTC).

When it comes to any software that deals with the financial sector, there will always be the threat of malicious actors. Regardless of the motives of these actors, extra precautions as to the safety, privacy, and handling of sensitive data regarding each transaction must be taken. In the modern age of software, it is imperative that no open-source software is used, or if it absolutely must be, a separate in-house branch of said open-source software is created and thoroughly tested and checked for potential malware injections. It has become common practice for pull requests to contain malicious code. If it were my company’s reputation, and my money on the line, I would not trust any open-source solution.

**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.

API’s are going to be a security concern. Any external interface will always be a potential security vulnerability. I see in the code a call to a SQL database. This raises concerns as to the security revolving around the client/server connection. Throughout the entire code base, I only found one error catching solution. This means there could be a potential vulnerability in the lack of error handling in the form of overflows or underflows.

**3. Manual Review**

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

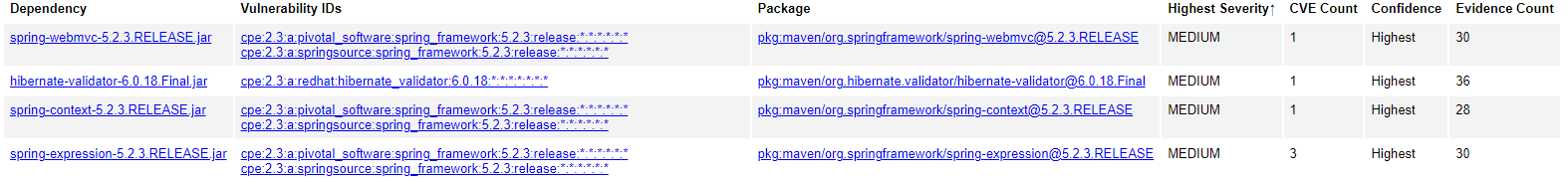
As I said above, there is a lack of error handling in the majority of the code. There is client/server connections to a database. There is a call to an external API. Customer account numbers are a private integer, but I do not see any encryption taking place here, so while private it is entirely possible that a malicious actor reading packets could access even a private variable value. There is also an immense amount of dependencies, which should raise a red flag when it comes to potential vulnerabilities.

**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

There are 13 potentially vulnerable dependencies. I will list here the medium severity dependencies, and then go into more detail for the high and critical entries.



HIGH Severity Dependencies

* **bcprov-jdk15on-1.46.jar** Bouncy Castle Crypto package. This package has been found to under specific circumstances allow remote attackers to conduct distinguishing and plaintext-recovery attacks.
* **Logback-core-1.2.3.jar** logback-core module. In previous versions, an attacker with the required privileges could craft a malicious configuration allowing the execution of arbitrary code.
* **Jackson-databind-2.10.2.jar** General data-binding functionality for Jackson: works on core streaming API. Insecure entity expansion, allowing for XML external entity attacks.
* **Spring-web-5.2.3.RELEASE.jar** Spring Web. While there exists the possibility of RCE, using trusted data mitigates this. However, there are several other exploits that malicious actors can use to overwrite files, perform SSRF attacks, and perform RFD attacks.
* **Spring-beans-5.2.3.RELEASE.jar** Spring Beans. There exists an RCE exploit via data binding. While it requires a specific setup it is possible that there are more ways to trigger this RCE exploit.
* **Log4j-api-2.12.1.jar** The Apache Log4j API. Susceptible to man-in-the-middle attacks. This is fixed in 2.12.3 and 2.13.1.
* **Spring-boot-2.2.4.RELEASE.jar** Spring Boot. Previous versions were vulnerable to temporary directory hijacking. It is recommended to update to the most recent version that is currently maintained.
* **Snakeyaml-1.25.jar** YAML 1.1 parser and emitter for Java. The Alias feature, prior to 1.26, allows entity expansion during a load operation.
* **Tomcat-embed-core-9.0.30.jar** Core Tomcat Implementation. Multiple potential exploits that can lead to Denial of Service or overloading the CPU usage causing the server to become unresponsive.

**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.

Many of the dependency issues can be resolved by updating to the latest maintained package. For those that cannot be solved by simply updating, I find it imperative to find a suitable replacement package, or spend some time writing in-house protections against potential vulnerabilities when using the library/package.