# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/22/2025** | **Jeremiah Havener** |  |

## 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

Jeremiah Havener

**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 communications should be extremely valuable to the company. It is necessary to ensure that customer information stays secure as it is being stored in their database or during transit as money is being deposited into an account. Due to Artemis Financial’s financial plans, it’s entirely possible that they only do national transactions; however, in the event they do international transactions, they would need to encrypt and protect their data that much more as it crosses international borders. This type of large, widespread transaction could have weaknesses sending and being received.

There may be government restrictions on data, however these are generally just standards to be met or restrictions on the type of data being sent and to where it’s being sent. These restrictions are easily complied with and should not be of serious concern, assuming everything is above board. External threats will be present with any sort of transactional business. Using external sources for financial planning introduces threats, but the real threat is the possibility that a phishing email or some other form of unsecure content is introduced inadvertently. Modernization requirements are necessary to ensure the system grows and maintains a strong sense of security. These can include evolving web applications, because an outdated application could work on the surface but if a weakness was identified, it could be exploited in the future.

**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.

For Artemis Financial, there a few vulnerabilities that stand out to me specifically. The most important of which is input validation. I believe this is the most important because properly managing user input is crucial, it’s one of the few things that is otherwise outside of a business’ control. That especially makes user input dangerous if they’re not validating inputs properly.

Additionally, incorporating APIs could introduce unique concerns. APIs need a strong security or unauthorized users can access data that is meant to be secure. There is also a large need for client/server security as working between financial institutions could create multiple weak points. In theory, this should be secure; however, there is a high risk when data is in transit and it is important to ensure it stays safe.

**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. Using /read in the CRUDController class does not properly validate user input and exposes the system to an attack.
2. In DocData, the function read\_document includes a URL rather than calling on a []
3. Another issue with read\_document is that the connection should be closed after being opened.
4. In the customer class, there is an encapsulation weakness as the account number and balance are public variables.
5. Additionally, the deposit method doesn’t validate the amount, allowing any amount to be entered, including negative numbers.
6. Using e.printStackTrace in DocData could lead to code error handling problems, as it could give out sensitive information of the program.
7. Finally, the time in myDateTime class is declared as public which can lead to an attacker abusing the clock, similar to video game hacks back in the day to alter the clock instead of waiting.

**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

|  |  |  |
| --- | --- | --- |
| **Vulnerability Code** | **Description** | **Attribution** |
| CVE-2023-33202 | Bouncy Castle for Java before 1.73 contains a potential Denial of Service (DoS) issue within the Bouncy Castle org.bouncycastle.openssl.PEMParser class. This class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN.1 data through the PEMParser causes an OutOfMemoryError, which can enable a denial of service attack. (For users of the FIPS Java API: BC-FJA 1.0.2.3 and earlier are affected; BC-FJA 1.0.2.4 is fixed.) | **Weakness Enumeration:** CWE-400 | Uncontrolled Resource Consumption | NIST  **Known Affected Software Configurations:**  **1.** cpe:2.3:a:bouncycastle: bouncy\_castle\_for\_java:\*:\*:\*:\*:\*:\*:\*:\*  **2.** cpe:2.3:a:bouncycastle: fips\_java\_api:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2016-1000352 | In the Bouncy Castle JCE Provider version 1.55 and earlier the ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider. | **Weakness Enumeration:** CWE-310 | Cryptographic Issues | NIST  **Known Affected Software Configurations:** cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2020-10693 | A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages. | **Weakness Enumeration:** CWE-20 | Improper Input Validation | NIST   Red Hat, Inc.  **Known Affected Software Configurations:**  **1.** cpe:2.3:a:redhat: hibernate\_validator:\*:\*:\*:\*:\*:\*:\*:\*  **2.** cpe:2.3:a:redhat: hibernate\_validator:\*:\*:\*:\*:\*:\*:\*:\*  **3.** cpe:2.3:a:redhat:hibernate \_validator:7.0.0:alpha1:\*:\*:\*:\*:\*:\*  **4.** cpe:2.3:a:ibm:websphere\_application \_server: \*:\*:\*:\*:liberty:\*:\*:\*  **5.** cpe:2.3:a:redhat:jboss\_enterprise\_application \_platform:7.2.0:\*:\*:\*:\*:\*:\*:\*  **6.** cpe:2.3:a:redhat:jboss\_enterprise\_application \_platform:7.3.0:\*:\*:\*:\*:\*:\*:\*  **7.** cpe:2.3:o:redhat:enterprise\_linux: 6.0:\*:\*:\*:\*:\*:\*:\*  **8.**cpe:2.3:o:redhat:enterprise\_linux: 7.0:\*:\*:\*:\*:\*:\*:\*  **9.** cpe:2.3:o:redhat:enterprise\_linux: 8.0:\*:\*:\*:\*:\*:\*:\*  **10.** cpe:2.3:a:redhat:satellite:6.8:\*:\*:\*:\*:\*:\*:\*  **11.** cpe:2.3:a:redhat:satellite\_capsule :6.8:\*:\*:\*:\*:\*:\*:\*  **12.** cpe:2.3:a:quarkus:quarkus :\*:\*:\*:\*:\*:\*:\*:\*  **13.** cpe:2.3:a:oracle:weblogic\_server: 14.1.1.0.0:\*:\*:\*:\*:\*:\*:\* |
| CVE-2023-35116 | jackson-databind through 2.15.2 allows attackers to cause a denial of service or other unspecified impact via a crafted object that uses cyclic dependencies. NOTE: the vendor's perspective is that this is not a valid vulnerability report, because the steps of constructing a cyclic data structure and trying to serialize it cannot be achieved by an external attacker. | **Weakness Enumeration:** CWE-770 | Allocation of Resources Without Limits or Throttling | NIST  **Known Affected Software Configurations:**  cpe:2.3:a:fasterxml:jackson-databind:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2021-44832 | Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2. | **Weakness Enumeration:**   1. CWE-20 | Improper Input Validation | NIST   Apache Software Foundation 2. CWE-74 | Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') | Apache Software Foundation   **Known Affected Software Configurations:**  cpe:2.3:a:apache:log4j:2.0:-:\*:\*:\*:\*:\*:\* +69 more |
| CVE-2023-6378 | A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. | **Weakness Enumeration:**  CWE-502 | Deserialization of Untrusted Data | NIST  **Known Affected Software Configurations:**  **1.** cpe:2.3:a:qos:logback:\*:\*:\*:\*:\*:\*:\*:\*  **2.** cpe:2.3:a:qos:logback:\*:\*:\*:\*:\*:\*:\*:\*  **3.** cpe:2.3:a:qos:logback:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2022-1471 | SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond. | **Weakness Enumeration:**  CWE-502 | Deserialization of Untrusted Data | NIST  CWE-20 | Improper Input Validation | Google Inc.  **Known Affected Software Configurations:**  **1.** cpe:2.3:a:snakeyaml\_project: snakeyaml:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2023-20883 | In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache. | **Weakness Enumeration:**  CWE-400 | Uncontrolled Resource Consumption | NIST VMware  **Known Affected Software Configurations: 1.** cpe:2.3:a:vmware:spring\_ boot:\*:\*:\*:\*:\*:\*:\*:\*  **2.** cpe:2.3:a:vmware:spring\_ boot:\*:\*:\*:\*:\*:\*:\*:\*  **3.** cpe:2.3:a:vmware:spring\_ boot:\*:\*:\*:\*:\*:\*:\*:\*  **4.** cpe:2.3:a:vmware:spring\_ boot:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2023-20863 | In spring framework versions prior to 5.2.24 release+ ,5.3.27+ and 6.0.8+ , it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition. | **Weakness Enumeration:**  CWE-917 | Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression La | NIST  CWE-400 | Uncontrolled Resource Consumption | VMware CISA-ADP  **Known Affected Software Configurations:**  **1.** cpe:2.3:a:vmware:spring\_ framework:\*:\*:\*:\*:\*:\*:\*:\*  **2.** cpe:2.3:a:vmware:spring\_ framework:\*:\*:\*:\*:\*:\*:\*:\*  **3.** cpe:2.3:a:vmware:spring\_ framework:\*:\*:\*:\*:\*:\*:\*:\* |
| CVE-2025-24813 | Path Equivalence: 'file.Name' (Internal Dot) leading to Remote Code Execution and/or Information disclosure and/or malicious content added to uploaded files via write enabled Default Servlet in Apache Tomcat. This issue affects Apache Tomcat: from 11.0.0-M1 through 11.0.2, from 10.1.0-M1 through 10.1.34, from 9.0.0.M1 through 9.0.98. If all of the following were true, a malicious user was able to view security sensitive files and/or inject content into those files: - writes enabled for the default servlet (disabled by default) - support for partial PUT (enabled by default) - a target URL for security sensitive uploads that was a sub-directory of a target URL for public uploads - attacker knowledge of the names of security sensitive files being uploaded - the security sensitive files also being uploaded via partial PUT If all of the following were true, a malicious user was able to perform remote code execution: - writes enabled for the default servlet (disabled by default) - support for partial PUT (enabled by default) - application was using Tomcat's file based session persistence with the default storage location - application included a library that may be leveraged in a deserialization attack Users are recommended to upgrade to version 11.0.3, 10.1.35 or 9.0.99, which fixes the issue. | **Weakness Enumeration:**  CWE-502 | Deserialization of Untrusted Data | NIST Apache Software Foundation  CWE-706 | Use of Incorrectly-Resolved Name or Reference | NIST  CWE-44 | Path Equivalence: 'file.name' (Internal Dot) | Apache Software Foundation  **Known Affected Software Configurations:** cpe:2.3:a:apache:tomcat:9.0.0:milestone1:\*:\*:\*:\*:\*:\* +74 more |

**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.

From the results, what stands out to me is the need to update the dependencies to a newer version. This type of mitigation should be done frequently to always minimize the risk of vulnerabilities. Additionally, do what can be done to reduce the number of false positives so time is not being wasted on unnecessary vulnerabilities. These vulnerabilities are always being identified, so stay on top of frequent dependency checks and implement changes quickly for the best results.