# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/20/2025** | **Victor Bandish** | **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

[Insert your name here.]

**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?

Artemis Financial is a financial firm that manages both internal and external sensitive information for its clients, making secure communication crucial for the company. Given that Artemis Financial handles financial transactions from both domestic and international sources, securing communications both internally and externally is essential to prevent data breaches. Although there are no current state or federal restrictions on secure communications, Artemis Financial must still prioritize security to safeguard customer information. Since the company handles sensitive data such as biometric information, Social Security numbers, account details, and trade secrets, it is critical to mask this information during both storage and transmission. Additionally, to stay ahead of security threats and bugs, Artemis Financial must ensure that their libraries are kept up to date to implement the latest fixes.

**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.

After assessing the security of Artemis Financials’ system, several potential vulnerabilities have been identified. First, proper input validation is essential to prevent issues such as SQL injection and ensure that user inputs are safe. The application will also need a well-designed API, as it will be used both internally and externally, including by end users through web browsers. This API must define acceptable methods for accessing data and ensure secure interaction, especially if it integrates with third-party services. Additionally, strong cryptography is necessary to protect proprietary customer information during international transfers, ensuring compliance with regulations both in North America and the destination country. Effective error handling is crucial to prevent unauthorized access or privilege escalation, particularly when dealing with input validation and the API. Finally, maintaining high code quality is imperative to avoid unintentional data exposure and to ensure that methods are only accessible by authorized users, according to their specific user roles.

**3. Manual Review**

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

**I began the vulnerability assessment by reviewing the input validation of the system. My first step was to check the pom.xml file for any Apache validators but found none. I then examined the greeting controller, where the input was simply assumed to be a string without any validation. This allowed raw user input to be accepted without any checks, which is a significant security concern. I also searched for an API but was unable to locate one, even though the program could still access data in an unsecured manner. Instead of using the POST method, which would securely transmit data, the system retrieves data directly via the URL. This exposes sensitive information through browser history and increases the potential for exploitation. Even though no data was visibly displayed, the system still accepted unfiltered user input via the URL, which made it vulnerable to security risks. The lack of an API meant there was no clear method for end users to interact with the system unless they had access to the code, further complicating usability. For a system to be truly RESTful, an API needs to define clear interaction methods.**

**Moving on, I checked for cryptography to ensure the system met security standards. Unfortunately, I found no encryption methods in place. Given that Artemis Financial may need to handle sensitive international transactions, the absence of encryption is concerning. The company will need to implement encryption for both data storage and transactions to comply with global regulations and safeguard user data.**

**I then turned my attention to errors in handling. Upon reviewing the code, I found that the DocData.java class contained a try-catch block, but it lacked any actual error handling logic. No other parts of the code included any error handling mechanisms either, which left the system exposed to unexpected failures without providing any feedback or recovery options.**

**Finally, while the overall code quality was relatively high, the absence of an API and the lack of proper input validation made the program difficult to use and prone to security vulnerabilities. The system still accepted data via the URL, which could expose sensitive information, and there was no clear method for interacting with the system. Input validation should be implemented to ensure that data is processed securely, and the use of URLs for handling input should be replaced with the POST method to protect sensitive data. In conclusion, although the code quality was adequate in some areas, it requires significant improvements in API design, input validation, cryptography, and error handling to ensure the program is both functional and secure.**

**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

|  |  |  |  |
| --- | --- | --- | --- |
| Dependency | Vulnerability | Description | Solution |
| log4j-api-2.12.1.jar | |  |  | | --- | --- | |  | [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) | | 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. | Upgrade |
| tomcat-embed-core-9.0.30.jar | [cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](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%3Atomcat&cpe_version=cpe%3A%2F%3Aapache%3Atomcat%3A9.0.30) [cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache_tomcat&cpe_product=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat&cpe_version=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat%3A9.0.30) | 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. | Upgrade |
| tomcat-embed-websocket-9.0.30.jar | cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:apache\_software\_foundation:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\* | Apache Tomcat 10.0.0-M1 to 10.0.6, 9.0.0.M1 to 9.0.46 and 8.5.0 to 8.5.66 did not correctly parse the HTTP transfer-encoding request header in some circumstances leading to the possibility to request smuggling when used with a reverse proxy. Specifically: - Tomcat incorrectly ignored the transfer encoding header if the client declared it would only accept an HTTP/1.0 response; - Tomcat honoured the identify encoding; and - Tomcat did not ensure that, if present, the chunked encoding was the final encoding. | Upgrade |
| bcprov-jdk15on-1.46.jar | [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) | 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. | update |
| jackson-databind-2.10.2.jar | |  |  | | --- | --- | |  | [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) | | A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity. | Update to current version |
| spring-aop-5.2.3.RELEASE.jar | cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*  cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\* cpe:2.3:a:vmware:springsource\_spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\* | In Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versions, the protections against RFD attacks from CVE-2015-5211 may be bypassed depending on the browser used through the use of a jsessionid path parameter. | Upgrade to current version |
| hibernate-validator-6.0.18.Final.jar | [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) | 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. | Upgrade |
| snakeyaml-1.25.jar | [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) [cpe:2.3:a:yaml\_project:yaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Ayaml_project&cpe_product=cpe%3A%2F%3Ayaml_project%3Ayaml&cpe_version=cpe%3A%2F%3Ayaml_project%3Ayaml%3A1.25) | 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. | Use Snakeyamls safeconstructor |

**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.

The overall mitigation strategy aims to transition the current DevOps pipeline into a DevSecOps pipeline. Given that the project is in its early stages, it’s crucial to instill a security-focused mindset in the development team from the beginning. By doing so, we can ensure that security considerations are embedded into the development process, addressing concerns like input validation early on, as highlighted by the NIST, before the project goes into production. The identified threats and their specific mitigation measures are outlined below. First, to mitigate the vulnerability in **bcprov-jdk15on-1.46.jar**, we will upgrade to version 1.56 or later, which validates key parameters during agreement calculation, ensuring cryptographic keys are properly secured and validated before data exchange. For **jackson-databind-2.10.2.jar**, we will configure the XML parser and validator to disable external entity expansion, preventing attackers from injecting altered DTD files. To mitigate the **log4j-api-2.12.1.jar** issue, we will ensure that certificate properties, particularly the hostname, are properly validated before certificate pinning, avoiding false security for end users. For **snakeyaml-1.25.jar**, we will disable the use of DTDs or employ a parser that restricts recursive definition expansion, preventing excessive recursion and reducing the risk of denial-of-service (DoS) attacks. Regarding **spring-core-5.2.3.RELEASE.jar**, while no specific mitigation is listed, upgrading to the latest version, 5.3.1, will ensure that the latest security patches are applied, although we will weigh the risk since newer versions may not have been fully tested in real-world conditions. Finally, for **tomcat-embed-core-9.0.30.jar** and **tomcat-embed-websocket-9.0.30.jar**, applying the principle of least privilege and implementing secure zones will help mitigate the risk by restricting access to sensitive data and ensuring that it remains protected with proper authorization. By implementing these mitigations early in the development process, we will significantly reduce the risk of security vulnerabilities and ensure the system is built with security in mind.