# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/26/2024** | **Dylan Bishop** |  |

## 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

Dylan Bishop

**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 specializes in creating financial plans for many individuals, covering areas such as savings, retirement, investments, and insurance. Secure communication is very crucial for the company due to the sensitive nature of all of the client information that they handle, which can include Social Security numbers, tax details, and many more. Although there is no specific information indicating Artemis Financial operates solely within the US, it is reasonable to assume that they handle international transactions as well. A key governmental restriction Artemis Financial must navigate is protecting trade secrets. With the critical need to protect all different information types, the main external threat involves the attempts to access client data, necessitating strong encryption to protect against unauthorized access. Regular maintenance checks to address any bugs and security vulnerabilities are important for maintaining the modern security standards at Artemis Financial.

**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.

* **Input Validation:** Artemis Financial employs input validation, especially for verifying the identity of information owners. This validation, typically done using string manipulation, adds a layer of protection for users.
* **Code Quality and Access Control:** Artemis Financial focuses on maintaining high code quality, which includes controlling access to methods based on user privileges. For instance, users are restricted to accessing only their own information, making sure that there is strong privacy and security.
* **API Development:** Creating APIs is very important for Artemis Financial because they operate both internally and externally. APIs enable controlled access to specific data to make sure that only authorized entities can retrieve or modify any information.
* **Error Handling:** Artemis Financial emphasizes strong error handling to identify and resolve problems within their APIs. This proactive approach minimizes the risk of exposing or compromising user information due to any type of system errors.
* **Cryptography:** Implementing strong cryptography is crucial for protecting any user data, especially in an international context where different currencies and regulations come into play. Cryptographic techniques make sure that any type of sensitive information remains secure even across diverse geographical regions.

**3. Manual Review**

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

After conducting a Vulnerability Assessment by reviewing the codebase, I focused on the POM.XML file and the Greeting Controller. Here are my findings:

* **POM.XML Analysis:** I wanted to confirm the presence of an Apache Validator within the XML file, which is very crucial for input validation and guarantee data integrity.
* **Greeting Controller Analysis:** I observed a lack of input validation in the Greeting Controller. This is a noteworthy issue that needs to be addressed in a future update to upgrade security and prevent any type of vulnerabilities.
* **Code Quality Analysis:** The overall code quality was considered acceptable. However, a significant concern was identified regarding the error handling because there was an absence of strong error handling mechanisms.
* **API Analysis:** When I was inspecting the API there were several deficiencies. The API lacked proper security measures in terms of user input exposure due to the absence of using a POST method.
* **Cryptography Verification:** Despite my efforts to verify the presence of cryptography within the code base, there were no indications of cryptographic implementations. This is raising concerns about data security and protection.

Overall, the code quality was acceptable in certain aspects but there are critical areas such as input validation, error handling, API security, and cryptography that will require immediate attention to mitigate any potential vulnerabilities and guarantee a secure software environment.

**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:\*:\*:\*:\*:\*:\*:\* | The Apache Log4j SMTP appender's improper certificate validation can lead to man-in-the-middle attacks on SMTPS connections, potentially leaking log messages. It's critical to fix this issue by ensuring proper certificate validation to prevent unauthorized access and data exposure. | Upgrade to version **2.23.1** of Apache Log4j to enable hostname verification for SMTPS connections. For the previous versions, set the system property **mail.smtp.ssl.checkserveridentity** to **true** to globally enable the hostname verification. |
| tomcat-embed-core-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:a  pache\_tomcat:9.0.30:\*:\*:\*  :\*:\*:\*:\* | Versions of Apache Tomcat from 10.0.0-M1 to 10.0.6, 9.0.0.M1 to 9.0.46, and 8.5.0 to 8.5.66 were vulnerable to HTTP request smuggling due to issues with parsing transfer-encoding headers. This could occur when used with a reverse proxy. Updating to a fixed version or applying patches is important to mitigate this vulnerability. | Upgrade to Apache Tomcat 10.1  or a later version. |
| 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:a  pache\_tomcat:9.0.30:\*:\*:\*  :\*:\*:\*:\* | Versions of Apache Tomcat from 10.0.0-M1 to 10.0.6, 9.0.0.M1 to 9.0.46, and 8.5.0 to 8.5.66 were vulnerable to HTTP request smuggling due to issues with parsing transfer-encoding headers. This could occur when used with a reverse proxy. Updating to a fixed version or applying patches is important to mitigate this vulnerability. | Upgrade to Apache Tomcat 10.1  or a later version. |
| bcprov-  jdk15on-  1.46.jar | 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:legi  on-of-the-bouncy-castle-  java-crytography-  api:1.46:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:bouncycastle:the  \_bouncy\_castle\_crypto\_pa  ckage\_for\_java:1.46:\*:\*:\*:  \*:\*:\*:\* | The Legion of the Bouncy Castle Java Cryptography APIs version 1.58 up to 1.60 (excluding 1.60) contains a vulnerability (CWE-470) known as 'Unsafe Reflection' in XMSS/XMSS^MT private key deserialization. This flaw can lead to the execution of unexpected code when deserializing an XMSS/XMSS^MT private key. Attackers can exploit this by crafting a private key with references to unexpected classes, which will be loaded from the class path of the application. The vulnerability has been fixed in version 1.60 and later. | Update bouncycastle to Version 1.78.1 |
| jackson-  databind-  2.10.2.jar | cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\* | A vulnerability was discovered in FasterXML Jackson Databind, where entity expansion was not securely handled. This flaw exposes the software to XML external entity (XXE) attacks, posing a significant risk to data integrity. | Update to the Latest Version |
| hibernate-  validator-  6.0.18.Final.j  ar | cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\* | A vulnerability was discovered in Hibernate Validator version 6.1.2.Final. This flaw is in the message interpolation processor allows invalid Expression Language (EL) expressions to be treated as valid. As a result, attackers can bypass input sanitation controls such as escaping or stripping, which developers use to secure user-controlled data within error messages. | Upgrade to the latests version of the hibernate-validator |
| snakeyaml-  1.25.jar | cpe:2.3:a:snakeyaml\_proje  ct:snakeyaml:1.25:\*:\*:\*:\*:\*  :\*:\* | 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.  Published December 1, 2022 ; 6:15:10 AM -0500 | Upgrade to version 2.0 or later |

**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.

Implement input validation in the Greeting Controller to prevent any malicious input. Enhance error handling with try-catch blocks and meaningful error messages. Secure the API endpoints by using POST methods and implementing authentication. Integrate cryptographic techniques, such as using Bouncy Castle, to encrypt sensitive data. Update dependencies like Snakeyaml, Hibernate Validator, Apache Tomcat, and Bouncy Castle to their latest versions for the security patches.