# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/26/2024** | **Courtney Warner** |  |

## 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

Courtney Warner

**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?

**1.** Dealing with sensitive and valuable information like clients’ savings, retirement, investments, and insurance makes security paramount for Artemis Financial. A security deficit could mean a breach costing the company time, money, and the client's trust, and costing the customer time, money, and privacy.

**2.** While there are no explicit mentions of international transactions, modernizing the company and updating security would involve projecting growth and future security issues. It will be necessary for Artemis to abide by national and international laws and regulations and utilize secure protocols like HTTPS and VPNs.

**3.** Since Artemis deals withretirement plans and insurance abiding by regulations like HIPAA (Health Insurance Portability and Accountability) will need to be followed when dealing with clients' medical information, protecting persons' identity, and personal data information.

**4.** External threats could include:

* Cyberattacks – such as DDoS, SQL injections, and cross-site scripting (XSS)
* Data Breaches—Unauthorized access to sensitive data
* Malware and Ransomware—Malicious software made to disrupt, damage, or gain unauthorized access

**5.** Several modernization requirements should be considered**.** Open-source libraries are cost-effective and innovative solutions. While innovative they also can carry security risks. These libraries should be evaluated, managed, and updated to mitigate risk and vulnerabilities. Additionally, when considering evolving web application technologies Artemis should stay updated by implementing security features like secure authentication mechanisms, input validation, encryption, and access control.

**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.

Areas of security applied to Artemis Financial:

**Input Validation:** Ensuring all inputs are validated to prevent common vulnerabilities like SQL injections.

**Justification:** Secure input handling is significant for protecting against malicious data.

**API’s:**

Securing the API interactions to prevent unauthorized access and data breaches is necessary. **Justification:** Artemis uses RESTful APIs securing the endpoints is imperative for secure data confidentiality.

**Cryptography:** Encryption for data at rest and in transit to protect sensitive information.

**Justification:** Encryption ensures that if data is intercepted/stolen it can't be read or altered.

**Client/Server:** Ensuring secure communication and interactions between the client and the server.

**Justification:** Securing communication channels through things like HTTPS protects data from tampering or leaks during transmission.

**Code Error:** Identifying and handling code errors will help prevent information leakage or crashes.

**Justification:** Secure code handling helps prevent attackers from exploiting errors.

**Code Quality:** Securing code by using common practices to avoid introducing vulnerabilities.

**Justification:** Ensuring best practices with coding and testing reduces vulnerabilities.

**Encapsulation:** Secure data structures to encapsulate data and logic.

**Justification:** Proper encapsulation prevents unauthorized access or manipulation of data.

**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. Class myDateTime- lacks **input validation**- there are no checks for valid input ranges
2. Class myDateTime - lacks **encapsulation-** attributes mySecond, myMinute, and myHour should be private
3. Class CRUD- lacks **input validation** – there is no validation if the input is null
4. Class CRUDcontroller- lacks **input validation**- there is no input validation for business\_name
5. Class CRUDcontroller- lacks **quality coding** – CRUD CRUD method should be renamed to something more descriptive
6. Class customer – lacks **quality coding** – in Java it is convention to capitalize class names
7. Class customer – lacks **encapsulation –** int account\_balance should be made private

**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

The following is a list of the vulnerabilities and brief descriptions and solutions are provided by the National Vulnerability Database:

**cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\***Bouncy Castle Java Cryptography API

Update to Version 1.56 or Later:

CVE-2016-1000352

CVE-2016-1000346

CVE-2016-1000345

CVE-2016-1000344

CVE-2016-1000343

CVE-2016-1000342

CVE-2016-1000341

CVE-2016-1000339

CVE-2016-1000338

Ensure BKS Keystore is Generated with 1.47 or Later:

CVE-2018-5382

Upgrade to Version 1.0.3 or Later for TLS Configurations:

CVE-2017-13098

Upgrade to 1.48 or Later:

CVE-2013-1624

**cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\***

Spring Boot

Upgrade to Recommended Versions:

CVE-2023-20883: Update to 3.0.6+, 2.7.11+, 2.6.14+, or the latest version if using older unsupported versions.

CVE-2023-20873: Same recommendation as above.

CVE-2022-27772: Update to at least 2.2.11.RELEASE or later.

**cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\***

Logback

Upgrade to Secure Versions:

CVE-2023-6378: Ensure the latest secure version is in use.

CVE-2021-42550: Update to a version that patches this issue, typically 1.2.7 or later.

**cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\***

Apache Log4j

Upgrade to Patched Versions:

CVE-2021-44832: Update to 2.17.1, 2.12.4, or later.

CVE-2021-45105: Update to 2.17.0, 2.12.3, or later.

CVE-2021-45046: Ensure the use of 2.16.0, 2.12.2, or later.

CVE-2021-44228: Update to 2.15.0 or later.

CVE-2020-9488: Ensure a version that addresses this issue, 2.12.3 or later.

**cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\***

SnakeYAML

Update to Version 2.0 or Later:

CVE-2022-1471: Use SnakeYaml's SafeConstructor.

CVE-2022-41854: Ensure secure parsing with version 2.0 or later.

CVE-2022-38752: Same as above.

CVE-2022-38751: Same as above.

CVE-2022-38750: Same as above.

CVE-2022-38749: Same as above.

CVE-2022-25857: Update to 1.31 or later.

CVE-2017-18640: Ensure the use of version 1.26 or later.

**cpe:2.3:a:yaml\_project:yaml:1.25:\*:\*:\*:\*:\*:\*:\***

YAML Project

Update to the Latest Secure Version:

CVE-2022-3064: Ensure that the latest version is used.

CVE-2021-4235: Same as above.

**cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\***

FasterXML Jackson Databind

Upgrade to Latest Versions:

CVE-2023-35116: Update to versions that address cyclic dependencies issues.

CVE-2021-46877: Ensure a version that resolves heap usage issues.

CVE-2022-42004: Update to versions that handle deeply nested arrays.

CVE-2022-42003: Ensure proper resource management in updated versions.

CVE-2020-36518: Update to at least version 2.13.0.

CVE-2020-25649: Secure against XXE attacks by updating.

**cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\***

Apache Tomcat

Upgrade to the Latest Secure Versions:

CVE-2024-21733: Update to 8.5.64 or 9.0.44 onwards.

CVE-2023-46589: Update to versions fixing HTTP trailer headers parsing.

CVE-2023-45648: Same as above.

CVE-2023-42795: Ensure recycling process security with latest versions.

CVE-2023-44487: Mitigate DoS risks by updating.

CVE-2023-41080: Update to address open redirect vulnerability.

CVE-2023-28708: Secure session cookies with updates.

CVE-2022-42252: Ensure proper header validation with updated versions.

CVE-2021-43980: Fix simplified blocking read/write implementations.

CVE-2022-34305: Prevent XSS vulnerabilities with secure versions.

CVE-2022-29885: Secure clustering over untrusted networks.

CVE-2021-41079: Fix infinite loop issue in TLS packets parsing.

CVE-2021-33037: Address HTTP transfer-encoding header parsing.

CVE-2021-30640: Secure JNDI Realm authentication.

CVE-2021-25329: Complete the fix for CVE-2020-9484.

CVE-2021-25122: Prevent header duplication in HTTP/2.

CVE-2021-24122: Secure JSP source code disclosure.

CVE-2020-17527: Prevent request header reuse in HTTP/2.

**cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\***

Hibernate Validator

Upgrade to the Latest Secure Versions:

CVE-2020-10693: Update to a version later than 6.1.2.Final to mitigate the flaw where a bug in the message interpolation processor allows invalid EL expressions to be evaluated as valid.

**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.

Through static testing, it was clear that the vulnerabilities are associated with outdated insecure dependencies. These pose a significant risk to the security and integrity of the application. An update is necessary to increase security. The manual review revealed that most vulnerabilities identified were input validation quality coding and encapsulation issues. Overall, both the static testing and manual review revealed the importance of security measures in software development. Moving forward, the dependency check version needs to be updated. Then, the code should be updated with the adjustments noted in the manual review, focusing on encapsulation, input validation, and quality coding methods.