# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/17/2025** | **Jael Ortiz** |  |

## 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

Jael Ortiz

**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 communication is very important for Artemis Financial to protect customer data and keep accounts safe. Without strong security, the company could face data breaches, fraud, and loss of customer trust. Since Artemis Financial handles international transactions, it must follow security rules to protect financial information worldwide.

Government security rules may also apply, especially if the company works with government employees or agencies. These accounts may need extra security measures, like strong passwords and encrypted data storage.

Possible security threats include hackers trying to steal login details, banking information, or personal data through fake websites and scams.

To improve security, Artemis Financial should use stronger login protection, like sending a text message code for extra verification. The company should also encrypt customer data so it stays safe even if hackers try to access it. Another important step is teaching customers how to spot scams that try to trick them into giving away their passwords or financial details.

As part of modernization, Artemis Financial should use safe and updated software tools to reduce security risks. Since technology is always changing, the company should also stay up to date with new security improvements, like better encryption and fraud detection.

**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.

Artemis Financial’s web application needs protection in several important areas. Input Validation is needed to make sure users can’t enter harmful or incorrect data, like attacks that could damage the system. By checking the input, the system can prevent these kinds of attacks and keep the data safe.

Artemis Financial uses a RESTful web API to communicate between systems. APIs can be a target for attackers who want to steal financial data. Protecting the API is essential to keep customer information and transactions secure.

Cryptography helps protect sensitive data, like banking details and transactions, by encrypting it. Without encryption, attackers could steal or change the data, causing major problems for both the company and customers. The client and server both need to be secure to prevent attackers from gaining unauthorized access. This protection helps stop issues like stealing personal data or taking control of accounts.

Lastly, code quality is important to avoid errors in the software that attackers might exploit. If the code isn’t written well, it could contain bugs that allow attackers to cause harm. Keeping the code clean and secure helps prevent these issues.

By focusing on these areas, Artemis Financial can better protect its customers' sensitive information from outside threats.

**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. **Input Validation Issue in CRUD.java**
   1. In the CRUD.java class, there is no validation to check if the input data is correct or safe before using it. This can be risky because bad or harmful data can be entered into the system, leading to problems like data corruption or malicious attacks. Validation is important to make sure only valid and safe data is used.
2. **API Security Risk in CRUDController.java**
   1. The CRUDController.java class takes user input directly from the URL and passes it to other parts of the system without checking if it's safe. This is risky because hackers can send harmful data through the API, which could lead to attacks like injecting harmful code into a website or data manipulation. The API needs protection to ensure only safe and expected data is used.
3. **Encapsulation Issue in customer.java**
   1. In the customer.java class, the account\_balance field is not hidden properly. It's exposed directly, and anyone can change the balance without any checks. This is a security problem because unauthorized users or attackers could modify customer account balances. The field should be private and accessed only through methods that control how the data is modified.
4. **Code Error Risk in DocData.java**
   1. In the DocData.java class, there is a method that connects to a database, but the actual SQL query is commented out. If this code were enabled, it would be vulnerable to hackers sending harmful information that changes the database query and could let them steal or change sensitive information. This is a common problem, so it's important to protect against it by using safer coding practices like making sure user input is treated as data, not as part of the query to keep the database secure.
5. **Code Quality Risk in Greeting.java**
   1. In the Greeting.java class, the content data is handled without any checks or encryption. This is not a big issue with non-sensitive data, but it can lead to problems if sensitive data is ever handled in a similar way. Data should be treated securely, with encryption and validation to protect against leaks or tampering.
6. **Code Quality Risk in GreetingController.java**
   1. In the GreetingController.java class, there's a counter that generates unique IDs. This counter is meant to be used by many users at the same time, but when there’s a lot of users, there could be a problem called a race condition. A race condition happens when two or more things try to use the counter at the same time, which can cause errors or unpredictable results. Special techniques should be used to make sure only one thing can use the counter at a time, even when there are lots of users.
7. **Input Validation Issue in myDateTime.java**
   1. In the myDateTime.java class, the method setMyDateTime() accepts time values (seconds, minutes, and hours) but doesn't check if they are within the correct ranges. Without this validation, the system might accept invalid values, leading to errors or incorrect data. It's important to ensure that the time values are correct before using them.

**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

**bcprov-jdk15on-1.46.jar (Bouncy Castle Crypto Package)**

CVE-2024-34447 – Certificate Validation Issue

* Problem: The software doesn't check if the certificate belongs to the right host, which can lead to security problems.
* Solution: Update to the latest version of Bouncy Castle where this problem is fixed.
* Source: [Bouncy Castle Release Notes](https://www.bouncycastle.org/releasenotes.html)

**hibernate-validator-6.0.18.Final.jar (Hibernate Validator)**

CVE-2023-1932 – Cross-Site Scripting (XSS) Risk

* Problem: There is a flaw in the isValid method that allows attackers to bypass security by not properly closing HTML tags. This can lead to HTML injection or Cross-Site Scripting (XSS) attacks.
* Solution: Update to the latest version of Hibernate Validator to resolve this vulnerability.
* Source: [NVD - CVE-2023-1932](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-1932)

**jackson-databind-2.10.2.jar (Jackson Databind)**

CVE-2020-25649 – XML External Entity (XXE) Attack Risk

* Problem: Jackson Databind has a security issue where it doesn’t properly handle certain XML data, which can allow attackers to access sensitive information.
* Solution: Update to the latest version of Jackson Databind to fix this security issue.
* Source: [NVD - CVE-2020-25649](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649)

**log4j-api-2.12.1.jar (Apache Log4j API)**

CVE-2020-9488 – Certificate Validation Issue

* Problem: Log4j doesn't check if the certificate matches the correct host. This can let attackers intercept emails and steal log data.
* Solution: Update Log4j to a newer version.
* Source: [Apache Security Advisory](https://security.apache.org)

**logback-classic-1.2.3.jar (Logback Classic Module)**

CVE-2023-6378 – Deserialization Vulnerability

* Problem: A vulnerability in Logback allows an attacker to send malicious data, which could lead to a Denial-Of-Service (DoS) attack.
* Solution: Update Logback to a higher version.
* Source: [GitHub Advisory](https://github.com/advisories/GHSA-vmq6-5m68-f53m)

**logback-core-1.2.3.jar (Logback Core Module)**

CVE-2023-6378 – Deserialization Vulnerability

* Problem: A vulnerability in Logback allows an attacker to send malicious data, which could lead to a Denial-Of-Service (DoS) attack.
* Solution: Update Logback to a higher version.
* Source: [GitHub Advisory](https://github.com/advisories/GHSA-vmq6-5m68-f53m)

**snakeyaml-1.25.jar (YAML Parser and Emitter for Java)**

CVE-2022-1471 – Remote Code Execution Vulnerability

* Problem: SnakeYaml allows unsafe deserialization, which could let an attacker run harmful code by sending malicious YAML data.
* Solution: Update SnakeYaml to a higher version or use SafeConstructor to limit deserialization.
* Source: [GitHub Advisory](https://github.com/google/security-research/security/advisories/GHSA-mjmj-j48q-9wg2)

**spring-boot-2.2.4.RELEASE.jar (Spring Boot)**

CVE-2023-20873 – Security Bypass Vulnerability

* Problem: A vulnerability allows attackers to bypass security in certain versions of Spring Boot deployed to Cloud Foundry.
* Solution: Upgrade to a higher version of Spring Boot.
* Source: VMware Advisory

**spring-boot-starter-web-2.2.4.RELEASE.jar (Spring Boot Starter Web)**

CVE-2023-20873 – Security Bypass Vulnerability

* Problem: A vulnerability in Spring Boot allows attackers to bypass security in certain versions deployed to Cloud Foundry.
* Solution: Upgrade to a higher version of Spring Boot.
* Source: VMware Advisory

**spring-core-5.2.3.RELEASE.jar (Spring Core)**

CVE-2022-22965 – Spring Framework JDK 9+ Remote Code Execution Vulnerability

* Problem: Spring MVC or Spring WebFlux apps running on JDK 9+ could allow attackers to execute malicious code when deployed on Tomcat as a WAR file.
* Solution: Upgrade to a higher version of Spring Framework.
* Source: VMware Advisory

**spring-expression-5.2.3.RELEASE.jar (Spring Expression Language)**

CVE-2022-22965 – Spring Framework JDK 9+ Remote Code Execution Vulnerability

* Problem: Spring MVC or Spring WebFlux apps running on JDK 9+ could allow attackers to execute malicious code when deployed on Tomcat as a WAR file.
* Solution: Upgrade to a higher version of Spring Framework.
* Source: VMware Advisory

**spring-web-5.2.3.RELEASE.jar (Spring Web)**

CVE-2016-1000027 – Remote Code Execution (RCE) Vulnerability in Spring Framework

* Problem: The Spring Framework has a security issue that could allow remote code execution if untrusted data is deserialized.
* Solution: Upgrade to a higher version of Spring Framework.
* Source: VMware Advisory

**spring-webmvc-5.2.3.RELEASE.jar (Spring Web MVC)**

CVE-2022-22965 – Spring Framework JDK 9+ Remote Code Execution Vulnerability

* Problem: Spring MVC or Spring WebFlux apps running on JDK 9+ could allow attackers to execute malicious code when deployed on Tomcat as a WAR file.
* Solution: Upgrade to a higher version of Spring Framework.
* Source: VMware Advisory

**tomcat-embed-core-9.0.30.jar (Tomcat Embed Core)**

CVE-2020-1938 – Apache Tomcat AJP Remote Code Execution Vulnerability

* Problem: Versions of Apache Tomcat before 9.0.31 have a security flaw that lets attackers run harmful code on the server if they can access the AJP connection.
* Solution: Upgrade to a higher version of Tomcat. If AJP is not needed, disable it.
* Source: NVD-CVE-2020-1938

**tomcat-embed-websocket-9.0.30.jar (Core Tomcat implementation)**

CVE-2020-1938 – Apache Tomcat AJP Remote Code Execution Vulnerability

* Problem: Versions of Apache Tomcat before 9.0.31 have a security flaw that lets attackers run harmful code on the server if they can access the AJP connection.
* Solution: Upgrade to a higher version of Tomcat. If AJP is not needed, disable it.
* Source: NVD-CVE-2020-1938

**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.

After reviewing the results from the manual and static testing reports, several security issues were identified that need to be addressed. One key issue is the lack of input validation in several parts of the code, such as in the CRUD and CRUDController classes. Without proper validation, user input may not be checked for correctness or security, which can lead to harmful data being processed. To mitigate this, input validation should be implemented across all parts of the system to ensure that only valid and safe data is used.

Another issue involves the protection of sensitive data, specifically in the customer class, where the account\_balance field is exposed directly. This presents a security risk, as unauthorized users or attackers could modify sensitive account information. To fix this, the account\_balance field should be made private, and changes to it should only be allowed through controlled methods that ensure secure updates.

Additionally, the static testing revealed that some libraries used in the system have known security vulnerabilities. These vulnerabilities could potentially be exploited if the libraries are not updated to newer, secure versions. To address this, it is important to regularly update any third-party libraries and ensure that all known security patches are applied. Keeping dependencies up to date is important for maintaining the overall security of the system.

By implementing input validation, securing sensitive data, and updating vulnerable libraries, we can mitigate the identified security risks and improve the system’s overall safety.