# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/20/2025** | **Eleanor Shamble** |  |

## 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

Eleanor Shamble

**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?

Artemis Financial dandles sensitive financial data about its customers. It is very important to handle sensitive information securely- malicious actors could do horrible things with the information that I know Artemis Financial handles. For example, if someone wanted to scam a client and obtained their financial information through Artemis Financial, they’d know exactly how much money they could squeeze out of them.

* Are there any international transactions that the company produces?

It is unclear whether there are any international transactions that the company produces. I would assume there are, if they take clients outside of the United States. If they do, everything the application does when interacting with an international client would be subject to the laws of both our country and theirs.

* Are there governmental restrictions on secure communications to consider?

Yes, there is the Financial Modernization Act of 1999, which requires companies handling financial data to have adequate safeguards for sensitive financial information (DAD, 2021). Being lax about security would be breaking the law.

* What external threats might be present now and in the immediate future?

Financial data is of great interest to scammers. In addition, any usernames and passwords for any website are always of interest, as many people use the same usernames and passwords on different websites.

* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

The application should have a modern look and feel to it, so using React or something similar may be a good idea for the front-end. Modern applications are supposed to be fast, so speed should be a consideration. WCAG accessibility guidelines should be adhered to. With open-source libraries, Artemis Financial should make sure that they stay consistently up to date when using them as dependencies, as security vulnerabilities are always being found and patched.

**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*- There are multiple mappings within the application that take parameters from the user. With any user input comes the risk of a user inputting data maliciously, so care should be taken to mitigate the risk by validating all of the user input.

*APIs*- There should be authentication for any APIs so users cannot input any malicious data.

*Cryptography*- Presumably, this application will have users who have passwords and sensitive information, even if the application isn’t in a state where it has this data yet. When it does have this data, it should be encrypted for security and user privacy.

*Client/Server*- Enforcing HTTPS protocol would make the application more secure.

*Code Error-* The application should have error handling so it doesn’t completely crash when something goes wrong. Errors should be avoided in general.

*Code Quality*- Better quality code is more secure and is easier to edit if a vulnerability is found. This is important for every application, so it’s also important here.

*Encapsulation*- Every edit to a variable should be authorized. If a variable isn’t edited through an authorized setter, then incorrect data could be added, which could trigger an error, or could be used maliciously.

**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 am looking at code that already exists, not any functions that need to be filled in later.

General issue

* The code lacks comments in some areas.

customer.java

* ENCAPSULATION: account\_balance is not a private variable, and it should be. It should not be able to be altered from outside of the class.
* CODE QUALITY: The name of the class is in lowercase, which is not standard for Java, and is not standard for this application. It should be Customer instead.
* CODE QUALITY: The parameter “a” in deposit could have a more descriptive name, like “amount”.

GreetingController.java

* INPUT VALIDATION: /greeting needs input validation- people can put anything for the name.
* CODE ERROR: /greeting needs to catch any errors coming from the input. Currently, if you don’t supply a parameter, it just leads to an error page.

myDateTime.java

* ENCAPSULATION: second, minute, and hour should be private. setMyDateTime exists, so there’s no reason why second, minute, and error should be altered from outside of the class.

CRUDController.java

* INPUT VALIDATION: Input validation for the parameter “name” is needed, because anything can be put there.
* CODE ERROR: There should be error handling in case count surpasses the maximum value of an AtomicLong. A malicious actor could reload the page again and again using a bot to try to trigger an error.

**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

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A group of blue text boxes

AI-generated content may be incorrect.**

There were 15 vulnerable dependencies found, which had 176 vulnerabilities total.

I will focus on the critical vulnerabilities:

[**CVE-2016-1000027**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1000027)

*“Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data.”*

*“CWE-502 Deserialization of Untrusted Data”*

Recommendation- Use input validation to make sure that all data the program handles is trusted.

[**CVE-2022-22965**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965)

*“A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.”*

*“CWE-94 Improper Control of Generation of Code ('Code Injection')”*

Recommendation- Do not run the application on Tomcat as a WAR.

[**CVE-2023-20873**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20873)

“In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.”

“NVD-CWE-noinfo”

Recommendation- Update to the newest version of Spring Boot.

[**CVE-2021-44228**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-44228)

*“Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.”*

*“CWE-400 Uncontrolled Resource Consumption ('Resource Exhaustion'), CWE-502 Deserialization of Untrusted Data, CWE-20 Improper Input Validation”*

Recommendation- Update to the newest version of Log4j.

[**CVE-2021-45046**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-45046)

*“It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments. Log4j 2.16.0 (Java 8) and 2.12.2 (Java 7) fix this issue by removing support for message lookup patterns and disabling JNDI functionality by default.”*

*“CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')”*

Recommendation- Update to the newest version of Log4j.

[**CVE-2022-1471**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=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.”*

*“CWE-502 Deserialization of Untrusted Data”*

Recommendation- Update to the newest version of SnakeYaml.

[**CVE-2020-1938**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-1938)

*“When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defence-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations.”*

*“NVD-CWE-Other”*

Recommendation- Update to the newest version of Apache Tomcat and make the necessary configurations changes.

[**CVE-2025-24813**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=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.”*

*“CWE-502 Deserialization of Untrusted Data, CWE-706 Use of Incorrectly-Resolved Name or Reference”*

Recommendation- Update to the newest version of Apache Tomcat.

[**CVE-2025-31651**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-31651)

*“Improper Neutralization of Escape, Meta, or Control Sequences vulnerability in Apache Tomcat. For a subset of unlikely rewrite rule configurations, it was possible*

*for a specially crafted request to bypass some rewrite rules. If those*

*rewrite rules effectively enforced security constraints, those*

*constraints could be bypassed.”*

*“This issue affects Apache Tomcat: from 11.0.0-M1 through 11.0.5, from 10.1.0-M1 through 10.1.39, from 9.0.0.M1 through 9.0.102.”*

*“Users are recommended to upgrade to version [FIXED\_VERSION], which fixes the issue.”*

Recommendation- Update to the newest version of Apache Tomcat.

**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.

MANUAL REVIEW

General issue

* The code lacks comments in some areas.
  + *Add comments to every function to explain what it does.*

customer.java

* ENCAPSULATION: account\_balance is not a private variable, and it should be. It should not be able to be altered from outside of the class.
  + *Add the private keyword to account\_balance’s variable declaration.*
* CODE QUALITY: The name of the class is in lowercase, which is not standard for Java, and is not standard for this application. It should be Customer instead.
  + *Change the name of the class to Customer.*
* CODE QUALITY: The parameter “a” in deposit could have a more descriptive name, like “amount”.
  + *Change the parameter name in deposit() from “a” to “amount”.*

GreetingController.java

* INPUT VALIDATION: /greeting needs input validation- people can put anything for the name.
  + *Use an if else loop to test the length of the string parameter, and to test for any injection attempts.*
* CODE ERROR: /greeting needs to catch any errors coming from the input. Currently, if you don’t supply a parameter, it just leads to an error page.
  + *Include a try catch loop around the line “****return******new*** *Greeting(counter.incrementAndGet(), String.format(****template****, name));”. If the parameter triggers an error, it can be handled that way.*

myDateTime.java

* ENCAPSULATION: second, minute, and hour should be private. setMyDateTime exists, so there’s no reason why second, minute, and error should be altered from outside of the class.
  + *Include private in the variable declarations for second, minute, and hour.*

CRUDController.java

* INPUT VALIDATION: Input validation for the parameter “name” is needed, because anything can be put there.
  + *Use an if else loop to test the length of the string parameter, and to test for any injection attempts.*

GreetingController.java

* CODE ERROR: There should be error handling in case counter surpasses the maximum value of an AtomicLong. A malicious actor could reload the page repeatedly using a bot to try to trigger an error.
  + *Include a try catch loop around the line “****return******new*** *Greeting(counter.incrementAndGet(), String.format(****template****, name));****”.*** *If the length of counter triggers an error, it can be handled that way.*

In general, more care could be shown when it comes to error handling, code quality, input validation, and encapsulation.

STATIC TESTING

Most of the critical vulnerabilities found in static testing (seven out of the nine) had to do with the dependencies not being up to date. Updating all the dependencies to their latest versions would likely cut the vulnerability list down a lot. It is important to keep dependencies up to date, because security vulnerabilities can be identified and patched at any time.

One of the other two ([**CVE-2022-22965**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965)) dealt with an irrelevant use case (this application will not be run in Tomcat). That one can be disregarded.

The last, [**CVE-2016-1000027**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1000027), could be mitigated by robust input validation. The risk is that untrusted data could cause problems during deserialization. If all data is screened before it’s assigned to any variables or used, then the risk of attacks is far lower.

References

DAD, T. K. (Acting, & Nguyen, S. T. (2021, July 16). *Financial privacy*. Federal Trade Commission. https://www.ftc.gov/news-events/topics/protecting-consumer-privacy-security/financial-privacy