# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/19/25** | **Tristin Raymond** |  |

## 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

Tristin Raymond

**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?

To a company, secure communication can mean everything. Secure communication plays an important role, as it makes sure that the data being communicated is protected from third-party interference when it is moving from a server to a client. The use of encryption and authentication plays a huge role in this area. In the case of Artemis Financial, a misstep in security could mean the loss of retirement accounts, or a risk of fraud. Global Rain is a company that engages in international business. The custom software design and development can be contracted by anyone throughout the world. While there may not be any governmental restrictions on secure communications that Global Rain needs to consider, they do however need to consider the Gramm-Leach-Bliley Act. This act requires financial institutions, such as Artemis Financial, to explain their information-sharing practices to their clients. This means that Global Rain would need to take this into consideration as the information sharing practices directly affect the software product that is being designed. In the present and the immediate future, we may see financial institutions become targets for hackers due to the amount of sensitive information that they hold in their databases. If a hacker is able to interfere with secure communications, they may be able to get ahold of sensitive information that could affect the livelihood of customers in an extreme way. In terms of modernization requirements, making sure the system is updated regularly, as well as the platform the system is running on is up to date will be crucial for staying on top of security.

**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.

When we take a look at the vulnerability Assessment Diagram, the first thing we see is input validation. This step is very relevant and crucial to the current task, as a financial institution will be utilizing inputs quite frequently. On top of that, customers would need to log in to a website to look at their financials, so ensuring that input attacks do not jeopardize the system will be very crucial. The second area of security that applies here is the API. We can see that Artemis Financial uses a RESTful web API, which needs to be taken into consideration. Since this is an API that is typically used for communication between application types, we must make sure that it is secure and validated so no unauthorized access into the system is permitted. This is where cryptography can become useful, as it would allow us to make secure communications even more encrypted and secure. Our next area we need to look at on the chart is the client/server side. Making sure that both the client and the server are secure is necessary because we definitely don't want an attack on either end of the spectrum here. Because we are using an API as well, we must make sure that we have a proper certificate in place to secure the data on both sides. Code error is the next stop on our flowchart, as we need a way to handle an error correctly and in a way that does not lead to further exploitation of the system. Without proper handling, sensitive information could end up being displayed and then we have an even bigger problem on our hands.

**3. Manual Review**

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

* There is currently no catch block on the DocData file, meaning we may not have a way to handle exceptions that could be thrown.
* I don’t see a place where input is handled for the customer, let alone a check for input validation.
* We have a connection to a localhost that doesn't seem to have any added security outside of the dependencies security alone.
* Test database username and password are both ‘root’.
* We are missing accessors for retrieving and setting the date and time, while not crucial, could be used as a vulnerability.
* The value for CRUDcontroller is the business name, which is not only an easy guess at accessing the system, but could lead to potential vulnerabilities.

**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

CVE-2013-1624

TLS implementation in the bouncy castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, a related issue to CVE-2013-0169.

Solution: update to the most updated dependency

spring-boot-2.2.4.RELEASE.jar

CVE-2023-20883

In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache. CWE-400 Uncontrolled Resource Consumption ('Resource Exhaustion')

Solution: update to newest, supported version

logback-classic-1.2.3.jar

CVE-2023-6378

A serialization vulnerability in the logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data.

Solution: Update to the newer dependency

logback-core-1.2.3.jar

CVE-2024-12801

Server-Side Request Forgery (SSRF) in SaxEventRecorder by QOS.CH logback version 0.1 to 1.3.14 and 1.4.0 to 1.5.12 on the Java platform, allows an attacker to forge requests by compromising logback configuration files in XML.

Solution: Update to the newest dependency

log4j-api-2.12.1.jar

CVE-2021045105

Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3 and 2.3.1) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0, 2.12.3, and 2.3.1.

Solution: Update dependency to newer, mentioned above.

snakeyaml-1.25.jar

CVE-2022-41854

Those using Snakeyaml to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stack overflow. This effect may support a denial of service attack.

Solution: Upgrade to version 2.0 or higher.

jackson-databind-2.10.2.jar

CVE-2023-35116

jackson-databind through 2.15.2 allows attackers to cause a denial of service or other unspecified impact via a crafted object that uses cyclic dependencies. NOTE: the vendor's perspective is that this is not a valid vulnerability report, because the steps of constructing a cyclic data structure and trying to serialize it cannot be achieved by an external attacker.

Solution: Add the depth variable to record the current parsing depth. If depth exceeds a certain threshold, an exception is thrown. <https://github.com/FasterXML/jackson-databind/issues/3972>

tomcat-embed-core-9.0.30.jar

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.

Solution: Upgrade to newer fixed version.

hibernate-validator-6.0.18.Final.jar

CVE-2023-1932

A flaw was found in hibernate-validator's 'isValid' method in the org.hibernate.validator.internal.constraintvalidators.hv.SafeHtmlValidator class, which can be bypassed by omitting the tag ending in a less-than character. Browsers may render an invalid html, allowing HTML injection or Cross-Site-Scripting (XSS) attacks.

Solution: Upgrade to 6.0.20 or higher.

spring-web-5.2.3.RELEASE.jar

CVE-2024-38828

Spring MVC controller methods with an @RequestBody byte[] method parameter are vulnerable to a DoS attack.

Solution: Update dependency

spring-webmvc-5.2.3.RELEASE.jar

CVE-2024-38816

Applications serving static resources through the functional web frameworks WebMvc.fn or WebFlux.fn are vulnerable to path traversal attacks. An attacker can craft malicious HTTP requests and obtain any file on the file system that is also accessible to the process in which the Spring application is running.

Specifically, an application is vulnerable when both of the following are true:

\* the web application uses RouterFunctions to serve static resources

\* resource handling is explicitly configured with a FileSystemResource location

However, malicious requests are blocked and rejected when any of the following is true:

\* the Spring Security HTTP Firewall https://docs.spring.io/spring-security/reference/servlet/exploits/firewall.html is in use

\* the application runs on Tomcat or Jetty

Solution: Implement Spring Security HTTP Firewall, or update dependency.

spring-context-5.2.3.RELEASE.jar

CVE-2025-22233

CVE-2024-38820 ensured Locale-independent, lowercase conversion for both the configured disallowedFields patterns and for request parameter names. However, there are still cases where it is possible to bypass the disallowedFields checks.

Solution: Upgrade to 6.2.7 or higher.

spring-expression-5.2.3.RELEASE.jar

CVE-2024-38808

In Spring Framework versions 5.3.0 - 5.3.38 and older unsupported versions, it is possible for a user to provide a specially crafted Spring Expression Language (SpEL) expression that may cause a denial of service (DoS) condition.

Solution: Update the dependency

spring-core-5.2.3.RELEASE.jar

CVE-2023-20863

In spring framework versions prior to 5.2.24 release+ ,5.3.27+ and 6.0.8+ , it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition.

Solution: Update dependency.

**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.

A large majority of the issues found in the dependency check could be solved just by updating the dependencies to the newer versions. By doing this, we can already feel more secure about the product we are building for Artemis Financial. On top of that, we can then run a further dependency check on the updated dependencies to ensure that we have taken care of the issues at hand. I would then focus heavily on making sure the code base is secure and issue free. One major issue that I saw was the username and password both being ‘root’. This is a simple fix and it would also be possible to integrate Two-Factor Authentication to make sure things are even more secure when logging in. I then would turn my attention to making sure that input validation is handled appropriately. I didn’t see any input methods or checks in the code base. After this, I would continue to address the possible issues outlined in the code base and have them checked by a second set of eyes. This redundancy is important, as it allows for mistakes to be caught easier and before a product is finalized for consumer use.