# lCS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **May 20, 2024** | **Zachary Hancock** |  |

## 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

Zachary Hancock

**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 is a consulting company that develops individualized financial plans for customers. The value of secure communications for this company is of the highest importance, as financial information is some of the most sought-after sensitive information for intruders. With the possibility of international transactions for the company procedures, it is integral to keep security tight as different countries have different security requirements. In this scenario, that company must adhere to the stricter side of the rules to comply with all compliance requirements. External threats that may be seen for the client in the current scope would be through possible API attacks since the client is currently running on a RESTful web application. Through the use of injection attacks or improper input validation standards the client will be at risk. Modernizing the system will be crucial in ensuring the client’s system remains state of the art, starting with the simple step of ensuring that the web technologies being utilized are constantly updated. This will ensure the proper functionality and level of security that the application is required to be running.

**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.

In order to the vulnerability assessment process flow the five main concerns that I have for the client's system would be input validation, APIs, cryptography, client/server, code error, code quality, and encapsulation.

* Input validation: validating ownership/access of information for the financial system will be crucial in providing the best level of protection for the user's information.
* APIs: the use of a RESTful API the concern would involve injection attacks, and with the use of secure protocols and input validation this would be snuffed out.
* Cryptography: The data that the client will be handling, sending, and receiving is highly sensitive information so it is crucial to apply the proper levels of encryption so this data can be read by just anyone
* Client/server: The use of secure communication between the client and server utilizing proper session management to protect user data and prevent unauthorized access
* Code error: This is a concern because holes in the code can lead to holes in the system so the integration of proper error handling, without error handling the exception and errors can open up unauthorized access to the system.
* Code quality: This is the base of all proper/secure applications. Following a strict secure coding technique with regular code reviews and testing is crucial to upholding the safety of the user.
* Encapsulation: Along with code quality comes the OOP coding technique of encapsulation, which will be important to hide important background functions of the system from the user, if not these functions can be exploited and taken advantage of by malicious attackers

**3. Manual Review**

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

Following the vulnerability assessment process flow, I first tackled input validation.

* Looked for a validator in the POM.XML, but was not found
* In GreetingController there is no input validation for the requested parameters
* CRUDController has the same issue of no input validation for the requested parameters

Secondly API:

* Similar issue within the POM.XML earlier, lack of security dependencies being used such as Spring boot starter security
* Lack of HTTPS in RestServiceApplication to help protect traffic
* Lack of authentication to verify and ensure proper use of the system

Code Errors:

* Lack of error handling and validation in constructing the Greeting object

**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:

bcprov-jdk15on-1.46.jar: To mitigate these vulnerabilities it would be best to update to the latest version of bcprov

* 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:bouncy\_castle\_for\_java:1.46:\*:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:bouncycastle:the\_bouncy\_castle\_crypto\_package\_for\_java:1.46:\*:\*:\*:\*:\*:\*:\*
* bouncycastle - Improper Validation of Certificate with Host Mismatch: The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host.

hibernate-validator-6.0.18.Final.jar: Update to the latest version

* cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*
* A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.

jackson-databind-2.10.2.jar: update to current version

* cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\*
* A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.

log4j-api-2.12.1.jar: update to at least 2.12.3 that fixes this issue

* cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*
* Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1

logback-core-1.2.3.jar: update to current version

* cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\*
* A serialization vulnerability in logback receiver component part of

logback version 1.4.11 allows an attacker to mount a Denial-Of-Service

attack by sending poisoned data.

snakeyaml-1.25.jar: Upgrade to version 2.0 and beyond

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

CWE-502 Deserialization of Untrusted Data, CWE-20 Improper Input Validation

spring-boot-2.2.4.RELEASE.jar: Upgrade to 2.7.11 or newer

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

spring-boot-starter-web-2.2.4.RELEASE.jar: Upgrade to 2.7.11 or newer

* cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:web\_project:web:2.2.4:release:\*:\*:\*:\*:\*:\*
* 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+.

spring-core-5.2.3.RELEASE.jar: Apply current updates

* cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* 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.

spring-web-5.2.3.RELEASE.jar: Apply current update

* cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:web\_project:web:5.2.3:release:\*:\*:\*:\*:\*:\*
* 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.

spring-webmvc-5.2.3.RELEASE.jar: Apply current update

* cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*
* cpe:2.3:a:web\_project:web:5.2.3:release:\*:\*:\*:\*:\*:\*
* 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.

tomcat-embed-core-9.0.30.jar: Upgrade to at least 9.0.31

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

tomcat-embed-websocket-9.0.30.jar: Update to at least 9.0.31

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

**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 interpreting the results of the review and static testing report, a majority of these issues can be resolved by updating to the newer versions of the dependencies. This highlights the fact that it is highly important for a developer to stay on top of their system to ensure it’s the most up to date as malicious attackers will take advantage of known exploits in outdated systems. To mitigate the issues found in the manual review there will need to be an implementation of input validation to verify inputs into the application, as that will be taken advantage of with injection attacks. Secondly, the utilization of HTTPS will be needed to encrypt traffic for the RESTful API. This should be a baseline for a financial institution, since the customers of our client depend on privacy we must remedy these holes in the system by adding validation, secure connections, authentication, as well as up-to-date systems.