# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/22/24** | **Tyler Doupe** |  |

## 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

Tyler Doupe

**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?

When first interpreting Artemis Financial’s needs, we first need to understand the business and their goals. From the scenario, we are told that Artemis Financial is a consulting company that develops individualized financial plans for customers which include savings, retirement, investments and insurance. With these being four important financial areas in society, not to mention the blatant aspect that this is dealing with customer’s personal information, secure communication is a must. Maintaining secure communication, especially when handling personal information ensures confidentiality and integrity with all transactions and information being shared between the customers and Artemis Financial. A failure to maintain secure communication could quickly lead to public distrust with the company, the information they possess regarding their customers and ultimately could lead a company’s demise. From the information provided at this point to me, I am not aware of any active international transactions that Artemis Financial may be handling, however, Artemis Financial stated that their software is intended for entrepreneurs, businesses, and government agencies around the world, thus the idea of international transactions is not off the table. Due to the company’s desire to be used around the world, the company must keep in mind international transaction governing compliance. In addition to international compliance, Artemis Financial must also keep governmental restrictions in mind in regards to secure communication. The government also places very strict standards when it comes to handling customer’s personal information and data. When dealing with personal information and financial information, Artemis Financial will always need to keep in mind that external threats will always be lurking in the shadows. External threats in both the present and future will be looking to perform data breaches of the information possessed by the company, along with finding ways to take advantage of a man-in-the-middle attack and phishing. In addition, we must consider modernization requirements such as open-source libraries and evolving web application technologies. Open-source libraries can be extremely useful resources but they also require continuous monitoring for vulnerabilities as an inability to continually monitor these resources could lead to security vulnerabilities as external threats constantly search for gaps in these libraries that they can exploit. Evolving web application technologies also calls for continuous updates and new best-practices to be implemented into web applications.

**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.

Referring to the vulnerability assessment process flow diagram, there were a few areas of security that apply to Artemis Financial’s software application. To start, input validation is a must have in order to prevent injection attacks. All inputs should be validated on both the client and server side to ensure security. The software application uses APIs thus we need to lock down security by implementing authentication and encryption in order to prevent unauthorized access or data leaks. Cryptography is another area of security that mist be considered with the application as we are handling customer’s sensitive information. Due to the high sensitivity of the information being shared, discussed and handled, we must implement means of encryption to protect this information. Client / server interactions will be an integral part of this entire application thus we must ensure that these communications are secure and authentic by implementing encryption. Additionally, code quality is another area of security that must be considered when developing the software application as failure to ensure sound code quality could lead to improper error handling, improper overall functionality and ultimately introduce vulnerabilities. Another vulnerability I noticed was the lack of input validation

**3. Manual Review**

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

When manually reviewing the code, I noticed several vulnerabilities that should be addressed. First, I noticed that the database credentials are hardcoded into the application. The read document function hardcodes the database connection to the database which grants anyone with access to the source code access to the username and password credentials within the database. Another vulnerability that I noticed was insecure API endpoint which doesn’t enforce authentication thus any user can access the endpoint and retrieve the sensitive information. Another vulnerability I noticed was the lack of input validation as the greeting endpoint takes in a name value but does not require input validation which leaves the application susceptible to injection attacks. Additionally, I noticed that the account number and account balance values are stored as plain integers without any form of encryption. This leaves this sensitive information unprotected if it were to be stored or transmitted. I also noticed improper error handling implemented in the DocData file as the method catches SQLExceptions and prints the stack which could potentially expose internal information to an attacker. I noticed that there is a lack of proper access control for the CRUDController which could allows for any user, regardless of their role, to access the read functionality which could lead to a data leak. Additionally, I noticed a lack of data validation when pertaining to the customer. The customer can change the account number and account balance without any validation or restriction.

**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

|  |  |  |
| --- | --- | --- |
| Dependency | Description | Severity |
| bcprov-jdk15on-1.46.jar | 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. | High |
| hibernate-validator-6.0.18.Final.jar | 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. | Medium |
| jackson-databind-2.10.2.jar | 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. | High |
| log4j-api-2.12.1.jar | The Apache Log4j API | Low |
| logback-core-1.2.3.jar | 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. | High |
| snakeyaml-1.25.jar | 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. | Critical |
| spring-boot-2.2.4.RELEASE.jar | 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+ | Critical |
| spring-boot-starter-web-2.2.4.RELEASE.jar | 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+. | Critical |
| spring-core-5.2.3.RELEASE.jar | 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. | Critical |
| spring-expression-5.2.3.RELEASE.jar | 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. | Critical |
| spring-web-5.2.3.RELEASE.jar | 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. | Critical |
| spring-webmvc-5.2.3.RELEASE.jar | 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. | Critical |
| tomcat-embed-core-9.0.30.jar | 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. | Critical |
| tomcat-embed-websocket-9.0.30.jar | 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. | Critical |

**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.

Following the manual review and static testing report, there are several steps to mitigate the identified vulnerabilities for Artemis Financial’s software application. To start, it was identified that there are hard coded credentials in the DocData.java file. These hardcoded credentials should be removed and a configuration management tool should be used to load the credentials during the runtime. Regarding insecure API endpoints, authentication should be implemented to ensure endpoints are protected. For the lack of input validation, input validation should be integrated into the application. There was unencrypted sensitive data found in the customer.java file which should be encrypted when in rest and when being transmitted. This should be done with encryption algorithms and transmissions should occur via SSL. Improper error handling was another vulnerability identified that can be addressed with proper logging mechanisms being implemented and by making error messages more detailed rather than being very generic. With the static test, there were several outdated libraries identified which can be improved with updated libraries and frequent audits to ensure that the most up to date libraries are in place. Additionally, there is a lack of data authentication which can be mitigated with a requirement for authentication tokens to prevent unauthorized access to endpoints.