# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[7/19/2024]** | **[Natalia Santiago]** |  |

## 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

[Insert your name here.]

**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?

[Since Artemis is a company offering financial services, security will be at a premium since financial rewards are the key target of hackers. It will also be important for clients that their assets are not publicly disclosed for reasons of security and privacy. After all, some clients may find it quite embarrassing to disclose their assets or the lack thereof in a public forum.

As a financial firm, transfers to and from foreign nations are possible. This could subject Artemis to foreign laws–even those surrounding Personally Identifiable Information. Even further, different countries might have different rules regarding record retention, contract law, or even format rules, i.e., digital vs. hard copy records. This may also mean that there are applicable laws regarding secure communications.

For example, in the USA, ECPA provides for government access to digital communications such as email, social media messages, information on public cloud databases, and more with a subpoena. There is also CFAA, SOX, and Gramm-Leach-Bliley Act that may apply. Any outside council would be wise to verify full compliance with and applicability of any legal requirement.

Artemis having an online presence poses the risk of various cybercrimes. They may be as minor and annoying yet not significantly harmful as a denial-of-service DOS, or as harmful and grave as a breach resulting in loss.

A breach could mean financial loss to either the consumers or the company, which at that time may dent Artemis' reputation, resulting in loss of business. Currently, this would take about 15 years to break RSA-512—NIST recommends at least 112 bits—but given the shifting power of processing, even RSA-2048 is believed to fall by 2030. High level encryption should be placed upon all databases, even those thought not confidential since having only a small quantity of PII might make a phishing attack more successful or expose Artemis to regulatory scrutiny.

Of course, any implication of open-source code should be known to Artemis. First, there could be overhead with some of the open-source licenses like any of the copyleft licenses such as GNU GPL and others that require reciprocity. This openness could expose Artemis to scrutiny or even vulnerability if a weakness were identified in either code. Also, any library needs maintenance and should be updated as frequently as possible. For instance, the recent problems with Log4J have been traced to more than 93-million exploits already, and Apache had to release four separate patches to fix all the Common Vulnerabilities and Exposures identified.]

**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.

[Since this is a RESTful API, I would check for input validation, secure API interactions, code errors, and encapsulation.

The need to perform input validation is very important since you must "sanitize" the inputs from the user, even if he or she is trusted. One wrong input that is intentionally or unintentionally malformed could trigger errant behavior. Securing the API goes one step further in ensuring that the connection, and possibly the user, is trustworthy. Some code checking is already built into Spring, but this error handling needs to be generic and shall not provide any signature that a hacker could use to understand more about the code structure. And lastly, since we are using methods, the methods shall use encapsulation to protect the sensitive data from unwanted changes, i.e. set variables to private and use Get/Set functions. Also important is the aspect of cryptography, for all flows of data should be encrypted. As previously mentioned, a strength of RSA-2048 should be considered for all databases and data transmission. The type and capabilities of encryption on every operating country ought to be checked since laws vary on cryptography.]

**3. Manual Review**

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

[Within static testing, it was observed that the POM.xml should be upgraded to a more recent release of Maven to 5.1.X versus 7.1.X. Older Static Cases Will Miss Some CVEs.

Some methods inside the code already have Get/Set functions available. The Greeting class, for instance, sets its parameters private. The GreetingController, however, does not use the available get/set functions.

User input to API is not sanitized. For example, inside the number method the value from array is passed via string splices: String message="Element in the given index is:: "+myArray[id] id is the raw variable obtained from user. Same goes for the CRUDcontroller's CRUD method. These could lead to injections.

All methods taking user input, such as the greeting method, need to be sanitized and checked for length, i.e., to prevent buffer overruns.

The customer class has the account\_balance as a public member. It should be set to private and have Get/Set methods used to control manipulation.

A few of the methods, such as setMyDateTime within the myDateTime class, are poorly coded because they are incomplete. In the class DocData, there is hard coding for human-readable username and password!! These should be obfuscated at least for testing and a lot harder to guess than "root" "root". In addition, all passwords need to be salted and hashed; none of that seems to be done here.

Also, error checking and messaging needs to be put in place. Only a few classes and methods have any user implemented error checking. As Artemis will not want to reveal more information than they intend to, e.g., "your password is incorrect" or "array out of bounds", such messages will need to be vetted and sanitized from "extra information" leaking to anyone attempting to find cracks in the code. For example, if the link to the SQL database fails in DocData --> read\_document, then a full stack trace is returned! This could easily lead to exploits.]

**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 | Vulnerability Code | Description | Mitigation |
| [bcprov-jdk15on-1.46.jar](file:///C:\Users\BTH\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\rest-service\target\dependency-check-report.html#l1_991c96a4e31e6c19e2b9136c8955bd423f2dc4c7) | 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:legion-of-the-bouncy-castle-java-crytographyapi:1.46  cpe:2.3:a:bouncycastle:the\_bouncy\_castle\_crypto\_package\_for\_java:1.46 | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms | Upgrade Bouncy castle to version 1.60 or higher |
| tomcat-embed-websocket-9.0.30.jar | cpe:2.3:a:apache:tomcat:9.0.30  cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30 | Core Tomcat implementation | Update Tomcat to the newest version |
| tomcat-embed-core-9.0.30.jar | cpe:2.3:a:apache:tomcat:9.0.30  cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30 | Core Tomcat implementation | Upgrade to newest Tomcat version |
| spring-core-5.2.3.RELEASE.jar | 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:vmware:springsource\_spring\_framework:5.2.3:release | Spring Core | Upgrade to the latest version of Spring |
| spring-aop-5.2.3.RELEASE.jar | 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 | Spring AOP | Upgrade to the latest version of Spring |
| spring-boot-2.2.4.RELEASE.jar | cpe:2.3:a:vmware:spring\_boot:2.2.4:release | Spring Boot | Only affects unsupported versions. Upgrade. |
| snakeyaml-1.25.jar | cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25 | YAML 1.1 parser and emitter for Java | Allows entity expansion. Upgrade. |
| jackson-databind-2.10.2.jar | cpe:2.3:a:fasterxml:jackson-databind:2.10.2  cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2 | General data-binding functionality for Jackson: works on core streaming API | Data integrity concerns. Upgrade to latest release. |
| logback-core-1.2.3.jar | cpe:2.3:a:qos:logback:1.2.3 | logback-core module | Arbitrary code execution. Upgrade to the latest version |
| hibernate-validator-6.0.18.Final.jar | cpe:2.3:a:redhat:hibernate\_validator:6.0.18 | Hibernate's Bean Validation (JSR-380) reference implementation | Input validation bypass issue. Upgrade to the latest version. |
| log4j-api-2.12.1.jar | cpe:2.3:a:apache:log4j:2.12.1 | The Apache Log4j API | Allows Man-in-the-middle attack on SMTP. Upgrade. |

]

**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.

[First of all, all dependent libraries need to be upgraded. This may introduce new code, so this is the first step that shall be taken.

Then, after the code recompiles and passes any internal tests, refactoring of the code would be necessary to introduce sanitized error messages, input validation/sanitization, security improvements such as salting and hashing.

There should be code reviews or peer programming; there are some large security issues within the code.]