# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11-12-24** | **Geslie Benjamin** |  |

## 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

**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?

In the fast-paced financial world of today, Artemis Financial recognizes how important it is to keep up a competitive, modern organization. Their proprietary software solution, tailored to their unique requirements, is the cornerstone of their strategy. To safeguard their sensitive data and business activities, companies are aware of the pressing need for robust software security measures, though. One of the main components of Artemis Financial's technological infrastructure is a REST-based online application programming interface (API). The company is anxious to take advantage of Global Rain's cybersecurity knowledge to fortify the protection of its clients' private financial information from outside threats. By collaborating with Global Rain, Artemis Financial hopes to guarantee the availability, confidentiality, and integrity of its systems, boosting stakeholder and customer confidence.

**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.

There are multiple areas of security are found:

* APIs : The web application makes use of API calls. We need to ensure a secured configuration for the APIs to limit the potential of compromised security between the system and the API connection.
* Input Validation and Output Encoding: Proper input validation should be implemented to prevent common vulnerabilities like SQL injection and cross-site scripting (XSS). Output encoding should be applied to protect against XSS attacks when displaying user-generated content. This is one of the critical steps that ensure Artemis Financials’ software is functioning in its expected nature. This validation of the right types of data are being utilized to reduce the risks of vulnerabilities.
* Code Quality- Use of the best and most recently updates software, and coding practices ensures software will function at its best. The better the software, the lesser the chance of potential vulnerabilities.
* Session Management: The application should properly manage user sessions, including secure session creation, maintenance, and termination, to prevent session-related attacks such as session hijacking or session fixation.
* Data Protection: Sensitive data, such as customer financial information, should be properly encrypted both during transit and at rest to prevent unauthorized access or data breaches.
* Secure Communication: The application should utilize secure communication protocols such as HTTPS to protect data transmission between clients and servers, ensuring confidentiality and integrity.

**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 analyzed both POM.XML and the Greeting Controller. From the XML file, I wanted to verify if I could find an Apache Validator. From the greeting Controller, I noticed there was a lack of input validation, which would be something I would take note for the feature. Code quality was acceptable; however, there was an issue with the error handling because it wasn't there. Moving on to the API. From the API, I noticed that it lacked in a lot of things. There was a breach that could expose the user input because it wasn't written through a POST method. Lastly, I tried verifying signs of cryptography; however, I was unable to find it. I was able to find a potential vulnerability in the CRUDController.java file where the application could potentially expose the internal objects. They are able to be passed and accessed through injections of code. When the value “business\_name” is passed through the CRUD method it may expose the DocData object database access vulnerability. I was also able to find a vulnerability when it comes to data access. In the DocData.java file, in the method to access data which involves the definition of the location of the database, the username and password have some potential vulnerabilities. The username and password are both root which is not recommended as the username and password combination could be easily guessed by anyone. This could allow unauthorized users to easily use a brute force attack to gain access to the system.

**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

There are five different dependencies that this software has. First, I will go over each item, then I will show the description that was included in the report, and finally, I will provide the code that corresponds to that description. The Bouncy Castle Crypto Package, which can be found in the bcprov-jdk15on-1.46.jar file, is the principal component. Several of these were present.   
  
There are fifteen vulnerabilities for Bouncy Castle, one of which is designated as CVE-2015-6644. This vulnerability has the potential to cause information to be leaked. A malicious application could be able to access the sensitive data that is recorded in the database as a result of this. For the purpose of preventing a breach in the application's security, it is recommended that the program and the operating system that it is based on be upgraded on a constant basis. (2016) According to the source. The entity that comes after it. The Apache Log4j Application Programming Interface (log4j-api-2.12.1.jar). This has a vulnerability known as CVE-2020-9488, which indicates that faulty certificate validation may allow man-in-the-middle attacks to intercept SMTPS connections, which might possibly expose log messages that are delivered via the appender. The recommended solution for this problem is to upgrade to the most recent version, which is 2.13.2 and has a verification function that is integrated within the program. 2019 (Apache, 2020). The third component was the YAML 1.1 parser and emitter for Java (snakeyaml-1.25.jar), which provided a significant amount of difficulty in terms of understanding. The CVE-2017-18640 issue allows for the development of entities while the program is being executed.A migration to SnakeYAML should, according to my research, be able to remedy the issue that you are now facing. The total number of aliases for collections will be reduced as a result of this. The system is able to either fail or restrict the amount of RAM that is assigned to each parser as a result of this. The fourth component was called Starter, and it was responsible for reading and writing JSON (spring-boot-starter-json-2.2.4.RELEASE.jar). Furthermore, it is responsible for facilitating the vulnerability known as CVE-2020-7712, which allows for the possibility of command injection through the parseLookup service. Through the implementation of this strategy, the syntax that is permitted for the bracketed components of the lookup strings is restricted, hence removing the requirement for eval during the processing of the text. Because of this, the vulnerability that had previously allowed command injections has been eliminated. For the purpose of ensuring the safety of the application, this is a very simple but extremely important component. The last dependant was the Core Tomcat implementation (tomcat-embed-core-9.0.30.jar), which had a number of vulnerabilities, specifically eight of them. The vast majority of vulnerabilities are vulnerabilities that are caused by excessive requests, which might potentially result in a denial of service or a server that is unavailable. This dependence has a number of vulnerabilities, and I have discovered that upgrading to a more recent version of tomcat-embed-core might be able to mitigate some of these issues. In light of the fact that high memory utilization might result in the failure of the system, this is a crucial cure. The utilization of an updated version of this dependence appears to be an effective strategy for mitigating the potential problems that are linked with it. Another Tomcat vulnerability, precisely CVE-2020-13935, is something I would like to solve on my own. The length of the payload is not verified well enough, which creates the possibility of a loop and makes it easier to launch a denial of service attack. Implementing sufficient validation for the WebSocket Framework is the solution to this problem with the architecture.

**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.

To make an effective mitigation plan for Artemis Financial’s software application. To start I would

address the data access username and password issue by creating a strong combination of

alphanumeric characters for both the username and password. This would mitigate the risk of

someone being able to simply brute force their way into the system. The next issue that I would

address would be to prevent the exploitation of vulnerability CVE-2020-9488 by updating the

Apache Server to the newer version of 2.13.2. This update will fix some of the vulnerabilities in

the previous version of the Apache Server. After this, I would go to the code review and

modification, here we should integrate secure coding practices, the quality of the code should be improved so proper authentication and error catching should occur. (Bhalla et al., 2020). This can

assist in the mitigation of authentication vulnerabilities. After this, I would want to address the

issues of certificate validation, the code should be properly sanitized to allow for proper validation and verification of all of the digital certificates for the application and the webserver.

This will help us to prevent the exploitation of vulnerability CVE-2020-9488 and CVE-2020-13935. Finally, I would address the TLS Certificate problem. We should implement a mutual check on both the client and server-side through pinned certificates.

In review of Manual inspection:

1.

Improve input validation.

by adding more string restrictions.

2.

Create an authenticator service with access controls.

3.

Integrate a working API.

4.

Utilize encryption services and switch to HTTPS protocols.

5.Implement more manual testing to verify if cade works are expected.

In review of our static testing for vulnerabilities:

1.

Update versions of all dependencies in the above.

2.

After all dependencies are update, run another maven check to see which vulnerabilities remain.

3.

If any vulnerabilities remain, inspect begin suppressing faulty ones once they are deemed a non-

threat