# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **01/23/2025** | **Daniel Schween** |  |

## 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

Daniel Schween

**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 develops individualized financial plans for its customers. These financial plans may include savings, retirement, investments, and insurance. Considering these financial plans, Artemis Financial will be collecting a lot of sensitive information from their clients. It is extremely important to protect their client’s bank details and identity. If their customer’s information ends up in the wrong hands, such as a malicious attacker, it can be life-ruining and financially devastating. Global Rain will further develop the API to identify any security vulnerabilities and protect the code base from external threats.

Artemis Financial does not specifically address whether they only work within the country or international. However, web applications are available internationally, and it would be best to assume international security. Clients can download the application internationally and seek consultation for a plan. If customers can access the API, malicious attackers or unauthorized users can also access the API. Therefore, we must address vulnerabilities that can happen across the globe.

The government requires companies to uphold consumer privacy. Sharing sensitive information such as social security numbers and financial information must be limited and protected for the consumer’s well-being. Therefore, it is the company or organization’s responsibility to secure sensitive data. On the other hand, data is certainly sold and shared. This information is more general, such as location, gender, age, etc. The company must disclose whether they are sharing a user’s data with other parties by having them read and sign a terms and agreements.

Some external threats to consider include malware, phishing scams, brute-force hacking, and SQL injection. Denial of service attacks can happen if our resources are overloaded, which can shut down the entire application. Hackers and malware can target the applications cryptography, input validation, databases, and servers to gain unauthorized access.

To modernize our security, we can use web application technologies, such as open-source libraries. A reliable and well-maintained library can accelerate development while minimizing risks. Frequent updates to these libraries as well as our API will patch new vulnerabilities and access new features. We can also adopt cloud-native architecture and microservices to support our API. Managing our data through the cloud allows us to monitor and update how we store our data much more easily and reliably.

**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.

Input validation would be a top priority to secure the application. Our team must ensure every keystroke is required when providing credentials. If the code in our application does not detect “ghost inputs” or invalid strings, we will be vulnerable to brute force attacks. We must also ensure our services can handle the data flow and restrict boundaries when necessary to prevent denial of service.

Cryptography would be equally as important as input validation. The application is exposed to several vulnerabilities, including private key validation, extra injection elements in a sequence, and exposure to sensitive information to an unauthorized user. The API is also vulnerable to Dos attacks.

Code error is another area of security to concern. How our team writes the code matters. It must be functional, but at the same time it must be concise and secure. Our code should be reviewed and tested by our peers to ensure we are protecting our client’s information, controlling database leaks, and constantly updating our software as we learn about new attacks.

**3. Manual Review**

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

1. Improper Verification of Credentials – Customer.java does not secure the client’s username and password. With no boundaries set, the input strings can be manipulated. For example, an external threat can inject extra elements creating “invisible” data which can make it possible to bypass authorization.
2. Incomplete API - The RESTful Application isn’t fully built and updated. Although, the Application may not run properly, data and input would still exist. Credentials, interface, and data would still be accessible, and an external threat can access these without our knowledge.
3. DoS Attack – restServiceApplication.java does not consider the resources and parameters of the application. If left without parameters, the resources our application offer can be bombarded with unnecessary data. In return, this will clog up our resources and shut down our services.
4. Timing/Validation Issues - myDateTime.java outputs the user’s date and time, but it does not consider the timing it takes to run the script. Timing-based leakage may occur in RSA based handshakes because of exception processing. The exceptions and timing must be clear so the application can decrypt and authenticate properly.
5. Timing Easily Observed – In myDateTime.java the user’s time is easily accessible. This can be used against the cryptography of our program. It is possible to identify when the decryption is failing due to padding, and this is unwanted information an external threat should be able to access.
6. Cross-Site Scripting – When typing or accessing our HTML, there is an improper neutralization of input during web page generation. This means that our html access is unprotected, and a window of opportunity is left open to gain authorization into our data and services. We mut encode the protection of our HTML string into our script by setting character boundaries.
7. Library/Package – We must use the leading libraires available and protect the packages being used in our code. A timing issue in one of the libraires being used can expose information about the private key when an attacker is able to observe timing. Cryptography goes together with this concept and updating our libraries with the latest version can patch these 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 (Cryptographic issues):**

CVE-2024-34447: 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.

[CVE-2016-1000338](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1000338): Improper verification of cryptographic signature. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.

* **hibernate-validator-6.0.18.Final.jar & jackson-databind-2.10.2.jar:**

CVE-2021-26877: Allocation of resources without limits or throttling. Some versions of Jackson-databind allows attackers to cause a denial of service.

CVE-2022-42004: Deserialization of untrusted data. Resource exhaustion can occur because of a lack of a check in BeanDeserializer.

CVE-2020-9488: Improper Certificate Validation. Improper validation with host mismatch could leak information. CVE-2020-25649: Improper restriction of XML external entity reference. 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 (Input Validation):**

CVE-2020-9488: Impropoer certificate validation

* **Logback-classic-1.2.3.jar:**

CVE-2023-6378: Deserialization of untrusted data. 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.

* **spring-boot-2.2.4.RELEASE.jar**

CVE-2022-27772: Exposure of resources to wrong sphere. Outdated versions of springboot are vulnerable to temporary directory hijacking.

* **spring-core-5.2.3.RELEASE.jar**

CWE-94: Improper control of generation of code (code injection)

* **tomcat-embed-core-9.0.30.jar**

CVE-2020-13935: CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop'). Invalid payload lengths could trigger an infinite loop. Multiple requests with invalid payload lengths could lead to a denial of service.

In general, there are three main areas of concern. Number one being cryptography and sensitive information. Bouncy Castle is a program used to handle the encryption and decryption of signatures as well as user credentials. In our report, it was found that an external threat can inject extra elements into the sequence creating ‘invisible’ data into a signed structure. This vulnerability can allow the external threat to override and gain access to personal accounts and sensitive information. If an out-of-date version of the software is used, the software can generate a weak private key during encryption. This allows easier access to cryptography.

Secondly, Input and resource validation was a common vulnerability in the dependency check report. Deserialization of Untrusted Data and improper input validation were common vulnerabilities. If programs such as springboot or hibernate-validator is run on an outdated version, it could lead to brute-force attacks or malware hijacking personal accounts and level of authorization. Moreover, if the applications’ resources are left without bounds or limits, a denial-of-service attack is likely.

Lastly, our databases are left at risk. For example, FasterXML in Jackson Databind did not have entity expansion secured properly, allowing vulnerability to external entity attacks. Additionally, SQL injection is possible if commands are left too general and strings aren’t properly confined. Many of the vulnerabilities found in the report can be fixed by maintaining the latest version of programs being used in the application. The latest versions address these vulnerabilities, and it is important to ensure we are up to date. Another solution is developing code that is strict to boundaries in a concise language. While coding it is important to keep in mind any back doors or hole in the language structure.

The Dependency Check report contains data retrieved from the National Vulnerability Database, CISA Known Exploited Vulnerability Catalog, GitHub Advisory Database, RetireJS, and Sonatype OSS Index.

**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 reviewing the dependency check report, many of the vulnerabilities can be addressed by updating the embedded programs in the API to the current versions. The latest versions were patched to make the software more secure. Additionally, our team will continue to develop the code to establish stricter boundaries and prevent external threats from finding any gaps in the language. We will use a system of checks and a concise language to prevent unnecessary lines of code that can be manipulated.