# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/18/24** | **Aashish Shenoy** |  |

## 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

Aashish Shenoy

**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?

The needs of the company from out viewpoint can be broken down into three main sections which are the management of the security aspect of its API, securing its existing financial data, and improving and bringing its day-to-day operations to the new age. Given Artemis Financial’s structure, their risks include vulnerabilities in their API, dangerous integration of third-party software by its users, ransomware attacks, denial of service attacks, phishing and other user-centric attacks, and finally data breaches.

Due to the nature and environment of Artemis Financial’s business they heavily rely on secure communication to not only perform secure transactions to avoid direct monetary losses, but also to build user trust and maintain good standing with both their clients as well as their respective governing laws. Since financial institutions would likely not purposefully limit themselves, they would most likely have international transactions. There are various governmental restrictions to consider for Artemis Financial to consider which include Sarbanes-Oxley Act, The Back Secrecy Act, Payment Card Industry Data Security Standards, The Gramm-Leach-Bliley Act, all of which primarily operate out of the US. As for worldwide and European services The Digital Operational Resilience Act, Payment Services Directive 2, and The EU and UK General Data Protection Regulations. External threats to consider would be issues or damages to the supply chain which disrupt workflow, consistent and targeted attacks from individuals or groups with a specific goal, and lastly zero-day vulnerabilities. Integration of up-to-date security standards such as the use of OAuth 2.0 to ensure secure API authentication, as well as incorporation of MFA for their clients and employees, along with the broad use of TLS for encrypted communications. Utilization of open-source libraries helps drive down the costs both financial and those related to time when it comes to development. Other areas of potential modernization include integration of serverless technologies and cloud-native system architecture. A more direct approach could include the additional of web application firewalls as well as DevSecOps for persistent involvement of risk assessment in the development process.

**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.

* Cryptography: Due to the sensitive nature of the business setting, it is crucial that the data is protected not only during normal flow but at rest as well. Having heavy encryption helps protect data if for some reason it does become intercepted and unauthorized access occurs.
* Input Validation: Essential to validate inputs to help prevent cross site scripting, cross site request forgeries, and SQL injections.
* Secure APIs: Having secure APIs is essential to ensure safe and secure communication between the systems involved. They can also help dictate that communication taking place happens only between authorized users.
* Secure Error Handling: Proper management of error handling is one of the most effective ways to ensure that system architecture and algorithms are not leaked should an error take place. It also helps prevent user data leaks as well as reduced the risk of possible attack avenues.
* Proper Coding Practices: Adhering to standard coding practices helps the application is crucial to help avoid some of the most common vulnerabilities while also reducing the number of flaws in the code, which attackers could use as potential exploits.

While encapsulating and secure distributed composing were left out, they do still play a large part in ensuring proper security, especially if this application were to expand and incorporate more factions.

**3. Manual Review**

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

CRUD:

* Assigning both content and content2 to content could lead to logical errors, especially since the constructor has only one argument.

CRUDController:

* No input validation and improper handling of data where the @RequestParam and the name parameter can lead to both injection attacks as well as cross site scripting.
* Another issue would be returning doc.toString() could possibly return an error which if not handled properly could reveal not only unnecessary information, but also possibly reveal information regarding code structure.

Customer:

* A large access control error is a possibility with the initialization of account\_balance. It is not set to private which allows other methods other than deposit() and showInfo() to access and even potentially modify account\_balance.
* In the deposit() method, there is no input validation of the parameter a which could lead to errors if the value passed to a is a negative value, thus resulting in lowering of account funds when making a deposit.

DocData:

* Hardcoding credentials when trying to access the database with the inclusion of both the username and password. This can easily lead to unauthorized access with the noted default credentials.
* The read\_document() method could also be at risk for injection attacks if its two parameters key and value are based on user inputs.
* While e.printStackTrace() is in an exception catching block, the function call itself will display unneeded information for most users who come across this error, which could then be used to expose vulnerabilities in the code.

myDateTime:

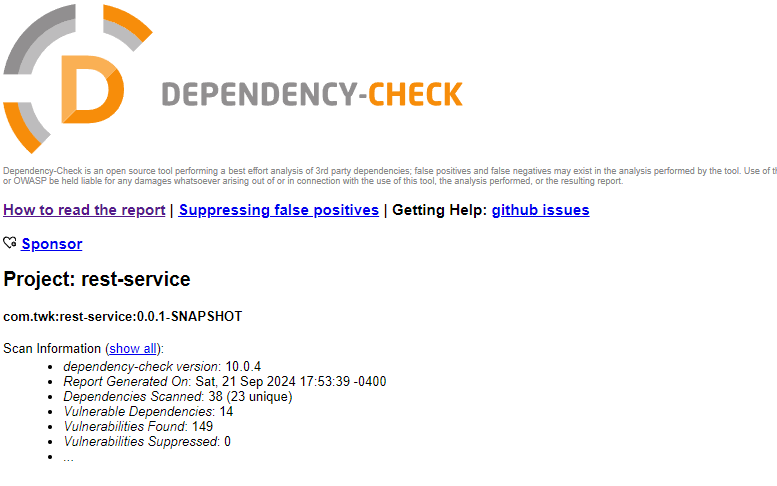
* If the inbuilt time functions are not being used, and the three initialized variables mySecond, myMinutes, and myHour are not set to private, then potential attacks could change the code behaviour by modigying these values outside of the intended methods retrieveDateTime() and setMyDateTime().
* Improper code such as return new int [3] can cause unexpected issues if the rest of the proposed code in that second is not written proper and the return item is not fixed, since currently it is not returning any specific array.
* Input validation is needed for the method setMyDateTime() if its three parameters are based off user inputs.

Greeting:

* No input validation for id and content, in conjunction with finalizing both elements could cause errors and result in data leak if exposed to injection type attacks.

**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:



|  |  |  |
| --- | --- | --- |
| **Name/Vulnerability Code** | **Description** | **Previous Identification/Documentation** |
| CVE-2024-34447 | 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. | <https://ossindex.sonatype.org/vulnerability/CVE-2024-34447?component-type=maven&component-name=org.bouncycastle%2Fbcprov-jdk15on&utm_source=dependency-check&utm_medium=integration&utm_content=10.0.4> |
| [CVE-2020-25649](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649) | 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 | https://ossindex.sonatype.org/vulnerability/CVE-2020-25649?component-type=maven&component-name=com.fasterxml.jackson.core%2Fjackson-databind&utm\_source=dependency-check&utm\_medium=integration&utm\_content=10.0.4 |
| [CVE-2023-6378](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-6378) | 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. | http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-6378 |
| [CVE-2022-1471](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-1471) | 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 | http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-1471 |
| [CVE-2023-20873](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20873) | 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. | https://security.netapp.com/advisory/ntap-20230601-0009/ |
| [CVE-2022-27772](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-27772) | Spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. | https://github.com/JLLeitschuh/security-research/security/advisories/GHSA-cm59-pr5q-cw85 |
| [CVE-2022-22965](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965) | 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. | https://packetstormsecurity.com/files/166713/Spring4Shell-Code-Execution.html |
| [CVE-2021-22118](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-22118) | In Spring Framework, versions 5.2.x prior to 5.2.15 and versions 5.3.x prior to 5.3.7, a WebFlux application is vulnerable to a privilege escalation: by (re)creating the temporary storage directory, a locally authenticated malicious user can read or modify files that have been uploaded to the WebFlux application, or overwrite arbitrary files with multipart request data. | https://security.netapp.com/advisory/ntap-20210713-0005/ |
| [CVE-2016-1000027](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1000027) | 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. | https://bugzilla.redhat.com/show\_bug.cgi?id=CVE-2016-1000027 |
| CVE-2024-38816 | Applications serving static resources through the functional web frameworks WebMvc.fn or WebFlux.fn are vulnerable to path traversal attacks. An attacker can craft malicious HTTP requests and obtain any file on the file system that is also accessible to the process in which the Spring application is running. | <https://spring.io/security/cve-2024-38816> |
| [CVE-2020-1938](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-1938) | 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. | https://issues.apache.org/jira/browse/OFBIZ-11847?page=com.atlassian.jira.plugin.system.issuetabpanels%3Aall-tabpanel |
| [CVE-2020-8022](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-8022) | This seems to be a SUSE specific security issue in the tomcat packaging. This allows a compromised tomcat group account to perform a full local root exploit. | https://bugzilla.suse.com/show\_bug.cgi?id=1172405 |

**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.

* Implement appropriate certificate validation which includes ensuring that it matches the given hostname, while also only accepting certificates from trusted and verified sources. Using the most update to date BouncyCastle library will aide with this step.
* To curtail the XML External Entity vulnerability, the best plan would be to severely limit or disable altogether the external entity expansion in the Jackson Databind while using a secure configuration and adequate input validation for XML inputs.
* Upgrading Logback to its latest version would help address the possible denial of service attacks. Rate limitation could also be an avenue that can be explored, however further testing would need to be done to ensure no loss of service to end users and clients.
* No object type restriction which could lead to unwanted code execution could be handled with upgrading SnakeYaml to its latest version while also using a safe constructor to limit the types of objects which could be created.
* To address another source of remote code execution we can upgrade the Spring Framework to its latest version in conjunction with applying strict data binding rules.
* A common vulnerability which was identified involved privilege escalation, which could be addressed by upgrading the Spring WebFlux framework. Other measures which can be taken include adding stricter access controls for users that have been authenticated along with file permission controls.
* Using appropriate validation for HTTPS requests to prevent unauthorized access to system architecture and other stored data.
* Updating Apache JServ Protocol and Tomcat to their latest versions.
* Proper error handling within the code for methods, specifically in classes CRUDController, DocData, and myDateTime.
* Add input validation for the Greeting, myDateTime, DocData, Customer, and CRUDController classes.
* Multiple objects across the classes that have been instantiated could be set to private to avoid unintended modifications.
* Using a limited privilege user based access to the SQL server instead of outright access by hardcoding credentials would also curtail some potential risks.

References

Kost, E. (2024, September 16). *Top 9 Cybersecurity Regulations for Financial Services*. RSS. https://www.upguard.com/blog/cybersecurity-regulations-financial-industry#:~:text=The%20Gramm%E2%80%93Leach%E2%80%93Bliley%20Act,threatening%20data%20integrity%20and%20safety.

Manico, J., & Detlefsen, A. (n.d.). SQL Injection and Other Injection Attacks. essay. Retrieved September 21, 2024, from https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch07.html.

Manico, J., & Detlefsen, A. (n.d.). Access Control. essay. Retrieved September 21, 2024, from https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch07.html.

IBM PTC Security. (2022, August 4). *XML security vulnerability series - part 1*. Medium. https://medium.com/@ibm\_ptc\_security/xml-security-vulnerabilities-series-850a9c19faf9

impreva. (2023, April 18). *API security: Best practices for SOAP and REST API: Imperva*. Learning Center. https://www.imperva.com/learn/application-security/api-security/

Manico, J., & Detlefsen, A. (n.d.). Cross Site Scripting Defense. essay. Retrieved September 21, 2024, from https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch07.html.