# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07-16-2024** | **Eric L Foster** |  |

## Client



## Developer

Eric L Foster.

**1. Interpreting Client Needs**

Artemis Financial is an organization seeking to modernize its operations while keeping security as the key facet of its modernization plan. Artemis currently uses a RESTful API built within [Maven](https://www.browserstack.com/guide/what-is-maven-in-java#:~:text=Understanding%20the%20Maven%20framework,-1.&text=A%20POM%20is%20the%20basement,project%20ID)%2C%20and%20version.) ( a build automation framework) with the intent of shielding itself from nefarious actors and sleep deprived developers.

One of the biggest hurdles that Artemis must overcome is to establish and maintain secure communications. This is critical for Artemis as financial transactions and customer data must be as secure in transit as they are at rest. Keeping security in mind is a preeminent factor in maintaining client trust and ensuring compliance with regulatory requirements.

With Artemis Financial being a global company international transactions are surely going to be commonplace. As such, Artemis must consider varying data protection regulations, such as the [GDPR](https://gdpr-info.eu/) in Europe, which has in place strict mandates on data handling and protection measures. Once again, Secure communications play a critical role here to protect against data breaches and unauthorized access during these transactions.

As such there exist governmental restrictions regarding secure communications that Artemis needs to consider. For example, compliance with regulations such as [PCI-DSS](https://www.pcisecuritystandards.org/) for handling payment card information, [SOX](https://sarbanes-oxley-act.com/) for financial reporting, and [GLBA](https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act) for financial institutions.

Considering the nature of the data being handled along with the number of regulations involved external threats are certain to be varied and numerous; listed below are some of the horrors that await Artemis if it is lackadaisical in its development practices.

*Phishing and Social Engineering* with the purpose of gaining unauthorized access to sensitive information.

*Distributed Denial of Service (DDoS)* that can cause significant disruptions to the availability of the application.

*Man-in-the-Middle (MitM)* whereby a nefarious actor can intercept and alter communications between the client and the server.

*Injection Attacks (e.g., SQL Injection, XSS)* will result in integrity loss of the application and potentially expose sensitive data.

*Advanced Persistent Threats (APTs)* are targeted attacks aimed at persistent data breaches.

To modernize and secure Artemis operations the following requirements should be considered.

The evaluation and regular updates to open-source libraries to help mitigate known vulnerabilities. Implement automated tools to scan for vulnerabilities within the dependencies that were created when these libraries were adopted into application code. Be vigilant in implementing secure coding practices; keep up with the constantly evolving advancements in web application security technologies such as [Content Security Policy (CSP)](https://www.imperva.com/learn/application-security/content-security-policy-csp-header/#:~:text=A%20Content%20Security%20Policy%20(CSP,and%20others)%20can%20be%20loaded.), [Subresource Integrity (SRI)](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity), [HTTP Strict Transport Security (HSTS)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security), and [Natural Language Processing (NLP)](https://www.ibm.com/products/natural-language-processing?utm_content=SRCWW&p1=Search&p4=43700079447578567&p5=e&p9=58700008668116504&gclid=CjwKCAjw1920BhA3EiwAJT3lSRDs1pBiSN3I0X4TqWOt94UPVkbuBWEziXRU4x_mCYUEPaUhluBWYBoCul8QAvD_BwE&gclsrc=aw.ds) which can be used to analyze a commit for risk level among other tasks.

**2. Areas of Security**

The below identified areas of security will help ensure the protection of sensitive financial data and compliance with various regulations.

The web application heavily relies on a RESTful API for operations making *Secure API Interactions* one of the most critical aspects of protecting the Artemis application from unauthorized access, data breaches, and many other threats all a different flavor. Having secure API practices, such as strong authentication, authorization, and HTTPS encryption helps to mitigate the risks associated with API vulnerabilities. Secure API interactions are the central tenets for establishing and maintaining secure communications while protecting customer information and sensitive financial transactions.

Once we have established secure communications the next layer of defense *Cryptography* ensures that financial transactions and customer data are securely transmitted and stored. This ensures that data is encrypted both in transit and at rest. Implementing strong cryptographic algorithms and key management practices is essential for compliance with regulations like [PCI-DSS](https://www.pcisecuritystandards.org/) and [GDPR](https://gdpr-info.eu/). Cryptography will help to protect Artemis against data breaches and unauthorized access, ensuring the integrity and confidentiality of sensitive information.

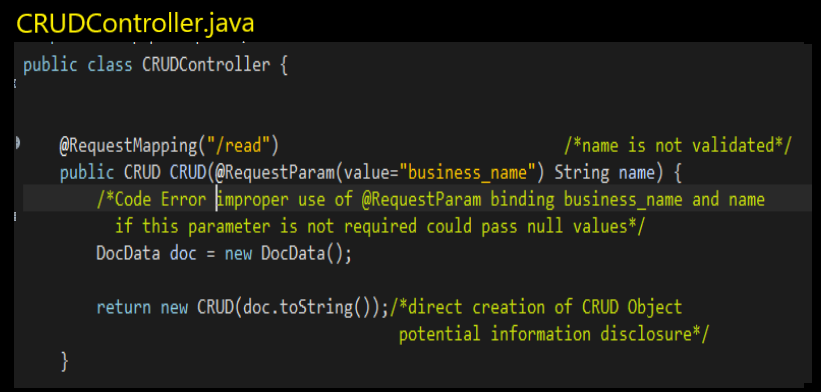
Having secure *Client-Server* interactions is a final critical step for protecting data exchanges and ensuring secure communications. The use of secure protocols like [TLS](https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/#:~:text=Transport%20Layer%20Security%2C%20or%20TLS,web%20browsers%20loading%20a%20website.) and implementing security controls within the server will help to mitigate risks from MitM attacks and other vulnerabilities. Regular security assessments should be conducted to ensure the readiness of client-server interactions.

Once we have secure and reliable communications we then need to focus on aspects that can make a functioning application go off the rails in a rather small time frame. As such, proper *Input validation* becomes crucial to protect the Artemis application from injection attacks and other forms of input manipulation. Having rigorous input validation, sanitization, and the use of parameterized queries prevent SQL Injection, XSS, and other vulnerabilities. This ensures that only valid, safe, and friendly data is fed to the application.

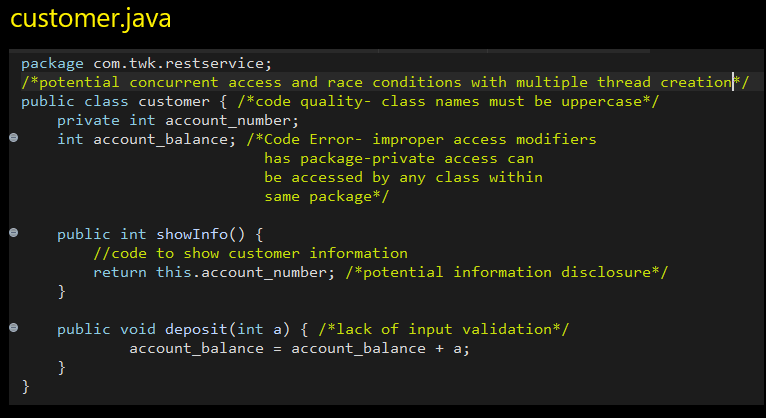
Ensuring high *Code Quality* is often the key first step -and often skipped- to prevent security flaws and vulnerabilities in the application. To identify and fix potential security issues Artemis needs to implement secure coding practices, regular code reviews, and automated vulnerability scanning from the beginning of the project, not as an afterthought. This is since maintaining high code quality reduces the likelihood of introducing security flaws during development.

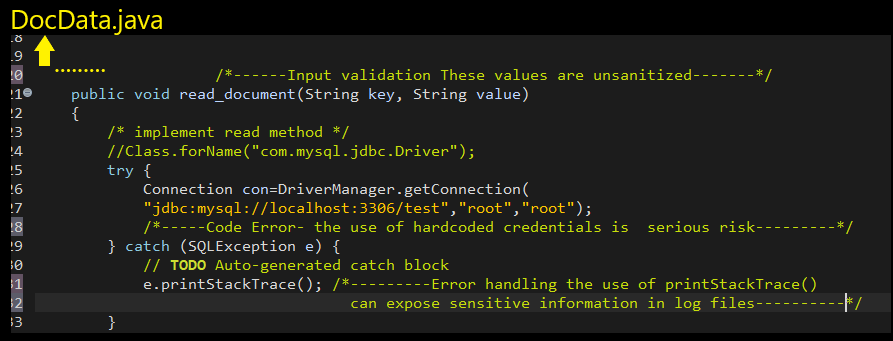
**3. Manual Review**

There was a total of eight files within the rest service package that was provided for Artemis, and they are littered with errors. There were over thirty security issues found within the code base. Provided below are some of these errors.

**A screen shot of a computer program

Description automatically generated

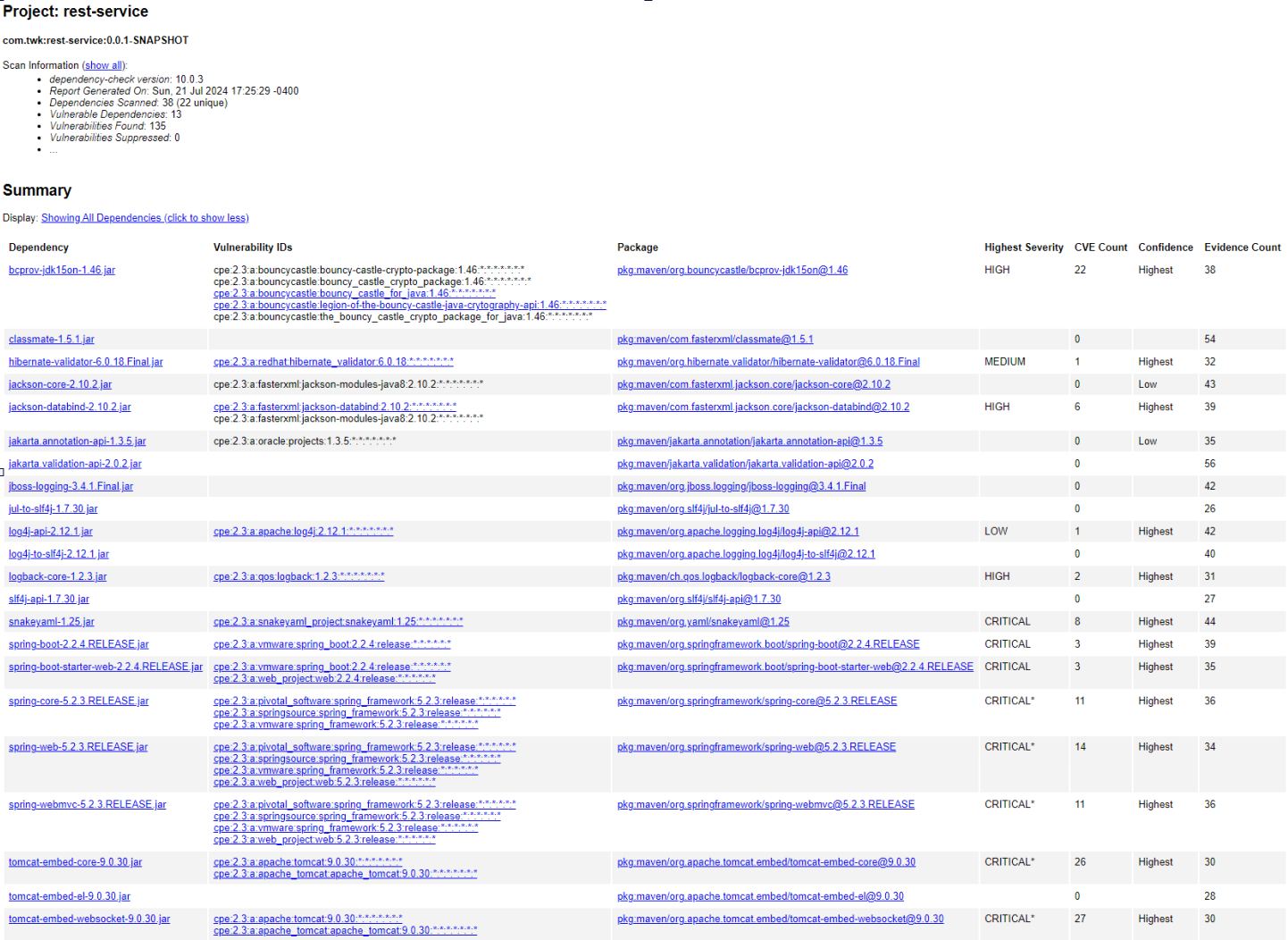
**

**

*A screen shot of a computer program

Description automatically generated*

**4. Static Testing**



This report reveals several critical vulnerabilities across multiple dependencies, particularly within the Spring framework and Apache Tomcat. Notable critical vulnerabilities include the snakeyaml-1.25.jar with eight CVEs and multiple Spring components such as spring-boot, spring-core, spring-web, and spring-webmvc, with each having a significant number of CVEs. The Tomcat embedded core and websocket components also have relatively high counts of critical vulnerabilities. These critical issues should be addressed in all haste; performing actions to update or replace the affected libraries and mitigate security risks.

High-severity vulnerabilities were also identified in dependencies such as bcprov-jdk15on-1.46.jar, jackson-databind-2.10.2.jar, and logback-core-1.2.3.jar. Medium and low-severity vulnerabilities were found peppered within other dependencies, including hibernate-validator and jackson-core. There were a few dependencies found to be safe such as classmate, jakarta.validation-api, and jboss-loggin, and showed no known vulnerabilities.

To maintain the security of the Artemis application the most pertinent thing to do regarding this report is to promptly address the critical and high vulnerabilities while continuously monitoring and planning for updates to medium and low vulnerabilities. Performing regular updates and using dependency management tools will help in containing these risks.

**5. Mitigation Plan**

To address the vulnerabilities identified in the various components of the Artemis application, a multi-layered approach is necessary to improve its overall security.

First, the implementation of rigorous input validation and sanitization is crucial. All user inputs, such as those accepted in the Crud and Customer classes, must be meticulously validated and sanitized to remove any potentially malicious content. For instance, the use of [regex patterns](https://support.insycle.com/hc/en-us/articles/10543578530967-Use-Regular-Expressions-Regex-for-Advanced-Data-Filtering-and-Manipulation) to filter out harmful characters and ensuring that numeric values are within expected bounds. Additionally, adopting a library like [OWASP’s ESAPI](https://owasp.org/www-project-enterprise-security-api/) (Enterprise Security API) for Java can provide built-in security controls for input validation and sanitation.

Second, it is essential to secure application configurations and enact data management protocols. Hardcoded credentials, as was seen in the DocData class, must be eliminated or secured using vault services like [AWS Secrets Manager](https://aws.amazon.com/secrets-manager/) or [HashiCorp Vault](https://www.hashicorp.com/products/vault). This ensures that sensitive information is not exposed in the source code. The configuration of [Spring Boot's security settings](https://reflectoring.io/spring-security/) to disable default error messages, and restriction of access to management and actuator endpoints will drastically reduce the risk of unauthorized access. Implementing Spring Security to enforce authentication and authorization will help to fortify the Artemis application against nefarious actors.

Lastly, the application should follow secure coding practices and regular security reviews. To ensure proper encapsulation classes like myDateTime should have their fields as private. The proper implementation of methods, and adherence to Java naming conventions greatly enhances application security and maintainability. Proactively conducting regular code reviews, static code analysis using tools like [SonarQube](https://docs.sonarsource.com/sonarqube/latest/?_gl=1*f8pq3u*_gcl_aw*R0NMLjE3MjE2MDAyODMuQ2p3S0NBanc0X0swQmhCc0Vpd0FmVlZaXzVvTXl3c0ZFa1hKNU1LaExLZEVINFN4NFk3ZFNIYnpESXdTSkxFSWRzUG9YR01Vcm01Q2hCb0NORVVRQXZEX0J3RQ..*_gcl_au*MjA1NzkwMDYzNS4xNzIxNjAwMjgz*_up*MQ..*_ga*MTkwNDEzNjk1OC4xNzIxNjAwMjgz*_ga_9JZ0GZ5TC6*MTcyMTYwMDI4My4xLjAuMTcyMTYwMDI4My42MC4wLjA.&gclid=CjwKCAjw4_K0BhBsEiwAfVVZ_5oMywsFEkXJ5MKhLKdEH4Sx4Y7dSHbzDIwSJLEIdsPoXGMUrm5ChBoCNEUQAvD_BwE), and dynamic security testing will help to identify and mitigate vulnerabilities.

References

<https://www.browserstack.com/guide/what-is-maven-in-java#:~:text=Understanding%20the%20Maven%20framework,-1.&text=A%20POM%20is%20the%20basement,project%20ID)%2C%20and%20version.>

<https://gdpr-info.eu/>

<https://www.pcisecuritystandards.org/>

<https://sarbanes-oxley-act.com/>

<https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act>

<https://www.imperva.com/learn/application-security/content-security-policy-csp-header/#:~:text=A%20Content%20Security%20Policy%20(CSP,and%20others)%20can%20be%20loaded.>

<https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

<https://www.ibm.com/products/natural-language-processing?utm_content=SRCWW&p1=Search&p4=43700079447578567&p5=e&p9=58700008668116504&gclid=CjwKCAjw1920BhA3EiwAJT3lSRDs1pBiSN3I0X4TqWOt94UPVkbuBWEziXRU4x_mCYUEPaUhluBWYBoCul8QAvD_BwE&gclsrc=aw.ds>

<https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/#:~:text=Transport%20Layer%20Security%2C%20or%20TLS,web%20browsers%20loading%20a%20website.>

<https://www.baeldung.com/spring-request-param>

<https://support.insycle.com/hc/en-us/articles/10543578530967-Use-Regular-Expressions-Regex-for-Advanced-Data-Filtering-and-Manipulation>

<https://owasp.org/www-project-enterprise-security-api/>

<https://aws.amazon.com/secrets-manager/>

<https://reflectoring.io/spring-security/>

[https://www.sonarsource.com/](https://docs.sonarsource.com/sonarqube/latest/?_gl=1*a05feq*_gcl_aw*R0NMLjE3MjE2MDAzMzcuQ2p3S0NBanc0X0swQmhCc0Vpd0FmVlZaXzVvTXl3c0ZFa1hKNU1LaExLZEVINFN4NFk3ZFNIYnpESXdTSkxFSWRzUG9YR01Vcm01Q2hCb0NORVVRQXZEX0J3RQ..*_gcl_au*MjA1NzkwMDYzNS4xNzIxNjAwMjgz*_up*MQ..*_ga*MTkwNDEzNjk1OC4xNzIxNjAwMjgz*_ga_9JZ0GZ5TC6*MTcyMTYwMDI4My4xLjEuMTcyMTYwMDM0NS42MC4wLjA.&gclid=CjwKCAjw4_K0BhBsEiwAfVVZ_5oMywsFEkXJ5MKhLKdEH4Sx4Y7dSHbzDIwSJLEIdsPoXGMUrm5ChBoCNEUQAvD_BwE)