****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 5](#_Toc102040759)

[3. Deploy Cipher 5](#_Toc102040760)

[4. Secure Communications 6](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 14](#_Toc102040763)

[7. Summary 17](#_Toc102040764)

[8. Industry Standard Best Practices 18](#_Toc102040765)

**General Hints if you have issue with refactoring from one of your peers:**

**Justin Starr** [**justin.starr@snhu.edu**](mailto:justin.starr@snhu.edu)

I do not believe you need to update the spring-boot version. The assignment is focused on whether or not your refactored code produces additional vulnerabilities. The only thing that I updated on the .pom file was the OWASP plugin version from 5.3.0 to 8.4.0. I did a dependency-check-report before and after refactoring the code.

The assignment is focused on whether or not your refactored code produces additional vulnerabilities. The only thing that I updated on the .pom file was the OWASP plugin version from 5.3.0 to 8.4.0. I did a dependency-check-report before and after refactoring the code.

However, concerning the .pom file, I will say that for the sake of trying, I updated the spring-boot version to 3.1.4 and received the same error you did. I did also clear my system cache and continued to receive the same error. I played around with it and found that I was able to successfully update the version to 3.0.8 without error. This does significantly reduce the number of vulnerabilities.

When I update the .pom file with spring-boot version 3.1.4, I am still able to run the SslServerApplication. It does load Spring Boot version 3.1.4. This only does not work for me when I try to get a dependency check report.

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **xxx** | **xxx** | **username: xxx**  **password: xxx** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Algorithm Cipher

The client, Artemis Financial, requires a form of cryptography that can validate the integrity of the data being transferred. This can be achieved using a checksum. A checksum, or hash, is an algorithm that computes a unique hash value to verify the integrity of data while stored or in transit (Siebert, 2021). Commonly, cipher algorithms are thought to be encoded and decoded as seen on television. These types of algorithms typically use random numbers inserted into the data to increase the encryption strength of the cipher. To encode and decode data, these algorithms use keys of both symmetric and non-symmetric types. Symmetric keys use the same key to encode and decode the data. Non-symmetric keys, by contrast, use one key to encrypt data and a separate key to decrypt data. This is often referred to as a public-private key pair. However, the algorithm used for a checksum is a hashing algorithm, which is a one-way conversion of data into a random bit string. The hashing algorithm chosen must be secure by regulatory standards to produce an uncompromisable checksum. To be considered secure, the algorithm must not allow a message to be identified from only a checksum and must produce a unique checksum for every message (National Institute of Standards and Technology [NIST], 2015, August). The latter property is also known as the collision resistance of an algorithm. This property is important because it is wholly responsible for verifying data integrity. Without collision resistance, it would be impossible to verify the integrity of data from a checksum. The recommended algorithm for the functionality required is the Secure Hashing Algorithm (SHA), specifically SHA-256. SHA-256 is part of a family of hashing algorithms called SHA-2 with algorithms being separated by bit level. The bit level, represented by 256 in SHA-256, quantifies the length of the hash by the number of bits. These algorithms use the Boolean Choose function and Majority function in addition to a circular rotation of the bits to create secure hashes. The original hash function SHA-1, created in 1995, is no longer secure due to the increased processing power of modern computers and should be eliminated by 2031 (NIST, 2022, December 15). SHA-256 and other members of the SHA-2 family of algorithms are still secure and sanctioned for use by the NIST. Therefore, the ideal algorithm for this functionality is SHA-256.

## Certificate Generation

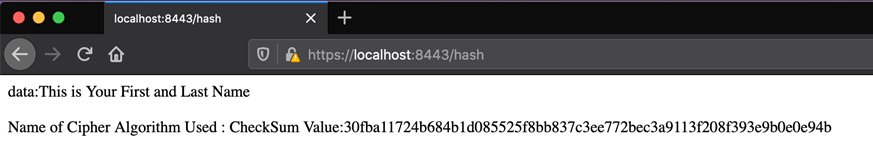
Insert a screenshot below of the CER file.

Text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A picture containing text

Description automatically generated

The browser will continue to say not secure as long as a self-signed certificate is used. This will be removed once the certificate used is from a CA.

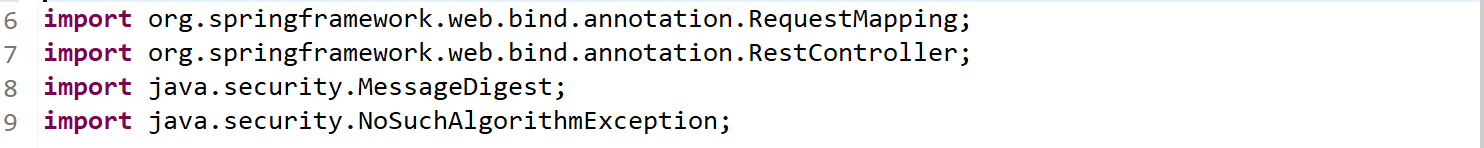
Graphical user interface, text, application

Description automatically generated

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

**SslServerApplication.java**

****

Text

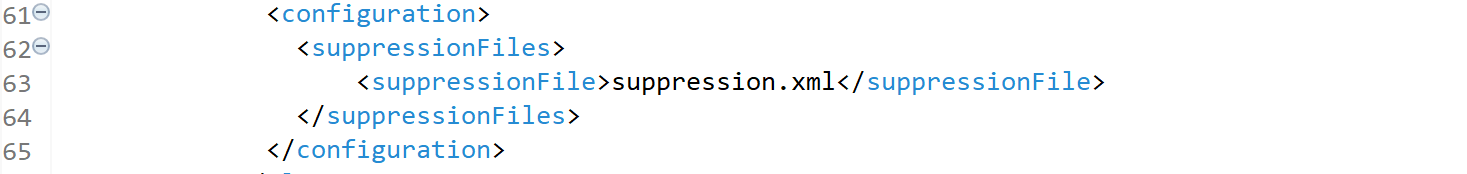
Description automatically generated

**application.properties**

**Graphical user interface, application

Description automatically generated**

**pom.xml**

****

**suppression.xml**

Suppresses vulnerabilities in the original code before refactoring began.

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Text

Description automatically generated**

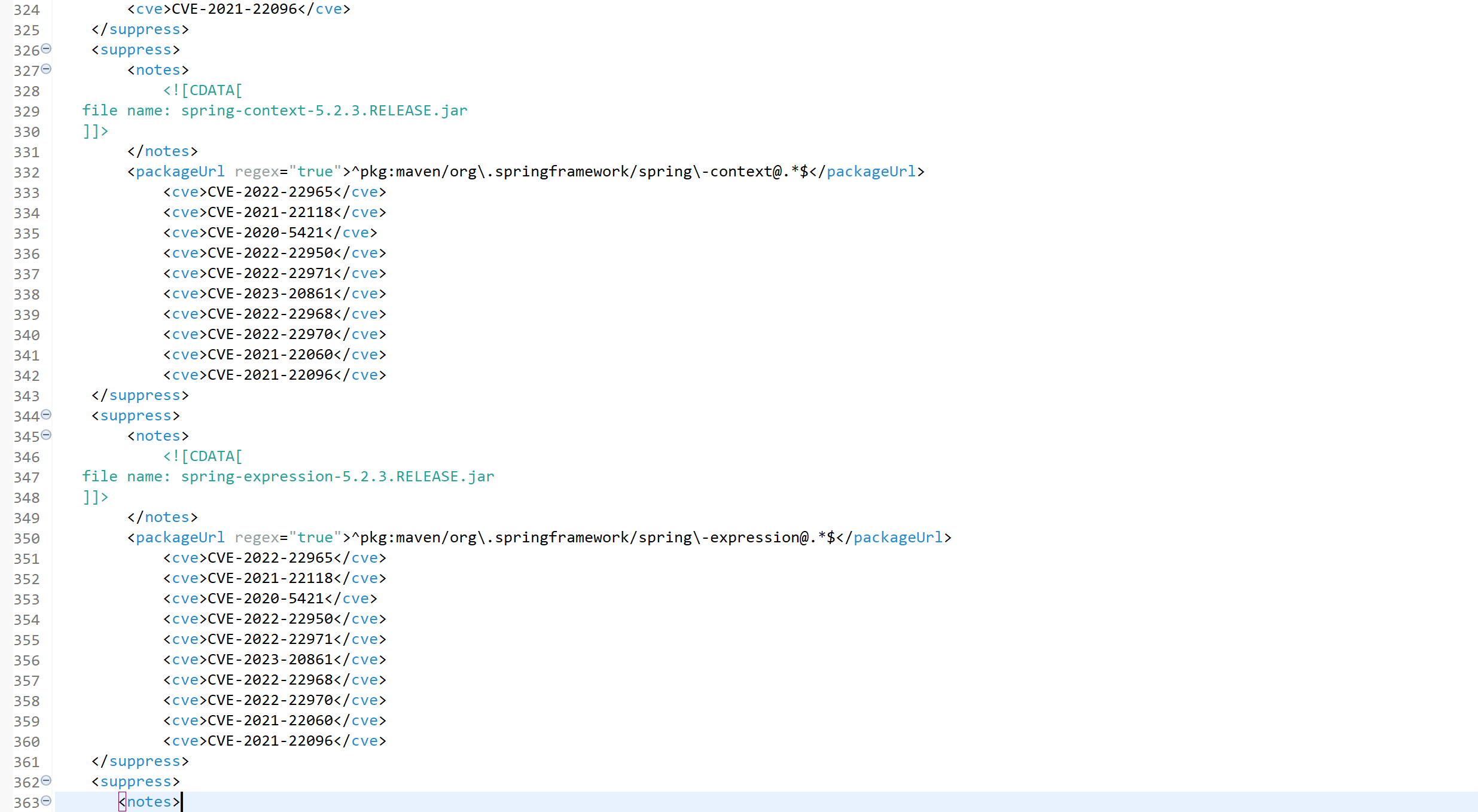
**Graphical user interface, text

Description automatically generatedGraphical user interface, text, email

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedScatter chart

Description automatically generated with medium confidence**

**Dependency Test**

**Graphical user interface, text, application, email

Description automatically generated**

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

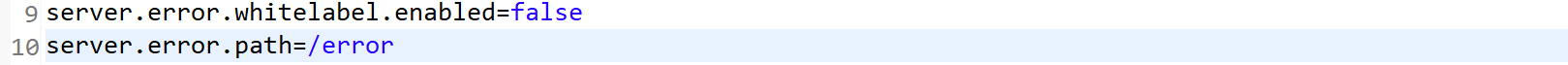
**SslServerApplication.java**

****

**Text

Description automatically generated**

**Application.properties**

****

**Pom.xml**

**Graphical user interface, text, application, email

Description automatically generated**

**Background pattern

Description automatically generated with low confidence**

**SecurityConfig.java**

**Graphical user interface, text, application, email

Description automatically generated**

**ErrController.java**

**Graphical user interface, application

Description automatically generated**

**Suppression.xml**

A false positive was detected in a subsequent dependency check. It is a confirmed false positive because the dependency version used was not in the versions listed in the vulnerability. Therefore, it was added to the suppression.xml.

**Graphical user interface

Description automatically generated with low confidence**

A new vulnerability was detected in one of the initial dependencies on April 13, 2023.

****

**Final Dependency Report**

**Graphical user interface, text, application, email

Description automatically generated**

## Summary

The initial area of security improved upon was the Cryptography of the application. This was addressing the specific needs and desires of the client Artemis Financial. The first method used to increase Cryptography security was the implementation of a checksum. This was achieved using an instance of the MessageDigest class that hashed the data utilizing the SHA-256 algorithm. After the original data was hashed using the MessageDigest instance, it was converted into a hexadecimal string via the BytesToHex function. This hexadecimal string was the checksum used to validate the data integrity of the given static data. The second method used to increase Cryptographic security was ensuring the website only used HTTPS. To do this, the application.properties file was modified to only allow traffic through port 8443 which requires HTTPS connections, and a certificate was added and implemented. However, because the certificate was self-signed, it is still regarded by browsers as insecure. This would change in production when a certificate from a certificate authority is used. After refactoring this code, a dependency report was conducted which used a suppression.xml file. The suppression.xml file was used to suppress any vulnerabilities in the application before the refactoring process began. It allowed me to analyze only the portions of code that I was refactoring and prevent the inclusion of new vulnerabilities.

The next area of security that was improved upon was the Code Quality of the application. This was strengthened in two ways the implementation of the SecurityConfig.java file created access control and the creation of a rate-limiting system using the bucket4j library. In the SecurityConfig.java file, A SecurityFilterChain method was created to funnel all HTTPS requests to a centralized login page. The userDetailsService method was created to handle the usernames and passwords of users. Currently, the username and password are hard-coded in plain text as “user1” and “user1Pass” respectively. This should be changed during production to values stored in an encrypted database. However, during development, I did not have access to the correct database and went with the hard-coded values. The passwordEncoder method is used to encode the password. Rate limiting was added in the SslServerApplication.java file via the bucket4j library. The ServerController constructor method creates a bucket instance that prevents more than 20 requests per minute. A check is then added to the checksum page to ensure the user has not exceeded the limit. To implement these features, the libraries spring-boot-starter-security, spring-security-web, spring-security-core, spring-security-config, spring-security-crypto, and spring-core were either added or updated to more current versions to fix bugs and resolve vulnerabilities. Additionally, a main page was added to hide system information because certain properties were being displayed in place of a main page when navigating to https::/localhost:8443/.

The final area improved in the application is Code Error. This was done by creating the ErrController.java file and modifying the application.properties file. The ErrController file creates a controller to handle HTTP errors and deploy a custom web page to the user. The server.error.whitelabel.enabled and the server.error.path properties were added to prevent the server’s generic 404 error page from displaying. In addition, the thymeleaf and thymeleaf-spring5 dependencies were added to enhance resource management which would allow the application to find the appropriate resource.

## Industry Standard Best Practices

Using a checksum allows the verification of data integrity. This ensures that users can be assured the data they expect is the data they get. It also reduces the risk of malware by preventing files and data that appear similar to the user but would have devastating consequences on the system. This was employed to assure the users that the files given to them are what they seem. Errors are being handled properly to prevent data from leaking to the user. This ensures the confidentiality of system information and often provides a better user experience. To increase the application’s defense against DOS attacks, I used rate limiting which prevents the over-expenditure of system resources by creating resource usage limits within a certain amount of time. This prevents users from crashing the application due to an overuse of the API. Access control was the final industry standard principle implemented. The importance of proper access control cannot be understated. Without it, any user of your site can make changes or access confidential parts of the system. This principle should not only be applied to essential aspects of the application, but it should remain an integral part of the overall security plan. This principle should begin with the initial contact made between the user and the application, which is why I felt the need to include an access control mechanism in the application.

The success of a business is largely dependent on the level of trust it has with its users. Cyber security is an important aspect of maintaining that trust in a digital world. If a breach were to occur, there may be a leak of financial data, user information, or system information, but those specific things can be regained to an extent. The biggest issue is the users’ trust because it is the hardest to regain and the quickest to fall. Maintaining a secure system and following industry standards is one of the best ways to ensure the continued trust of your users. Even if the business’s systems do not experience a breach, the company could still lose face if they appear susceptible to attacks. This is especially true in the world of social media where the news of a systems breach or system vulnerabilities travels like wildfire. If a business’s system does not appear secure, it can greatly harm the trust of its users even if its system is not breached.

**References**

National Institute of Standards and Technology. (2015, August). *Secure Hash Standard (SHS)* <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>

National Institute of Standards and Technology. (2022, Dcemeber 15). *NIST Retires SHA-1 Cryptographic Algorithm*. <https://www.nist.gov/news-events/news/2022/12/nist-retires-sha-1-cryptographic-algorithm#:~:text=As%20attacks%20on%20SHA%2D1,31%2C%202030>

Siebert, C. (2021). *Highly Scalable Parallel Checksums*. 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), Parallel and Distributed Systems (ICPADS), 2021 IEEE 27th International Conference on, ICPADS, 812–818. <https://doi-org.ezproxy.snhu.edu/10.1109/ICPADS53394.2021.00107>