****

# 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 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/15/23** | **Zach Beachy** |  |

## 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.

## Developer

Zach Beachy

## Algorithm Cipher

I believe that Artemis financials’ needs would be satisfied by the utilization of SHA-256. The algorithm here is a 256-digit value that can take lower-case letters or numbers from 0-9, which means that each digit has 36 possible values. To find the total possible combinations, one would take 36^256, which equates to approximately 1.172119 \* 10^402 different possible values. This function and its bit levels are created randomly, and the input values are compressed before being utilized. The hash value becomes the name of the compressed input.

In terms of encryption keys, there are two categories, symmetric and asymmetric. Symmetric keys, the more simple of the two, use plain text and a key to encrypt the provided data. Asymmetric keys, on the other hand, are typically considered more secure, as the key itself changes as well. This is more secure but potentially more computationally heavy, and with the large number of possible options with SHA-256, I do not believe that there is significant concern for alarm. Encryption has existed in some form or another likely since the beginning of written communication, however, the current methods and strategies from the perspective of software engineering were developed in the 1970s by IBM.

## Certificate Generation

Text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

Graphical user interface

Description automatically generated with low confidence

## Secure Communications

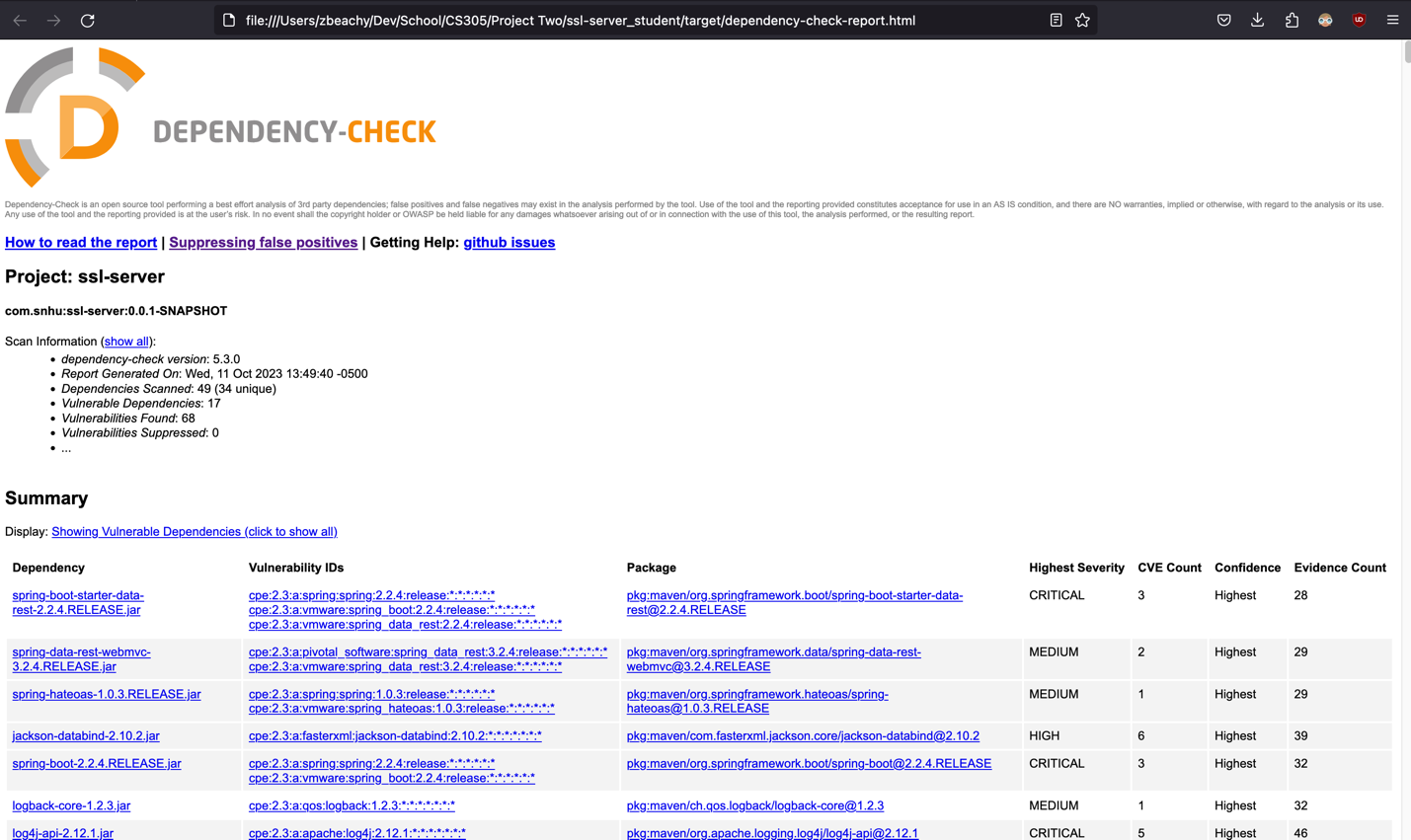
Insert a screenshot below of the web browser that shows a secure webpage.

Graphical user interface

Description automatically generated with low confidence

## Secondary Testing

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



Text

Description automatically generated

Graphical user interface, text

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

## Summary

While refactoring the code, I added a RestController to SslServerApplication.java which provides functionality for secure control of the hash RESTful endpoint. This addressed the Client/ Server aspect of the vulnerability assessment process flow. Additionally, it satisfies code error and code quality concerns by handling exceptions properly and executing without obvious exploits or weak spots. This refactor also includes the logic for the SHA-256 hashing cipher that I implemented. This creates a simple yet difficult to crack cipher for the data that is being pushed/received to/from the server, which addresses the cryptography aspect of the vulnerability process flow.

## Industry Standard Best Practices

In the process of evaluating and refactoring the code base, it was essential to apply industry-standards and best practices for secure code. There are aspects of industry standards that I described in the refactor that I implemented to target specific vulnerability areas, such as using a proven cipher, but also achieved this by utilizing standard java security dependencies, which ensures that I have well documented and time-tested processes to use. The standardization also ensures that there are consistent updates to patch out found vulnerabilities. This includes using the OWASP Dependency-Check to evaluate the dependencies that I used. It is pertinent for the company to use industry standard best practices for secure coding not only to ensure that the code developed by the company is up to snuff with competitors in terms of security, but also to protect the customer and users from attacks and information leaks. These sorts of insurances can bolster the company’s overall reputation and public reception.

**References:**

Dworkin, M. , Barker, E. , Nechvatal, J. , Foti, J. , Bassham, L. , Roback, E. and Dray, J. (2001), Advanced Encryption Standard (AES), Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.FIPS.197 (Accessed September 24, 2023)

GeeksforGeeks. (2021, April 12). *What is java AES encryption and decryption?*. GeeksforGeeks. https://www.geeksforgeeks.org/what-is-java-aes-encryption-and-decryption/#