****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **06/16/21** | **Kari L. Cheslock** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Kari L. Cheslock

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

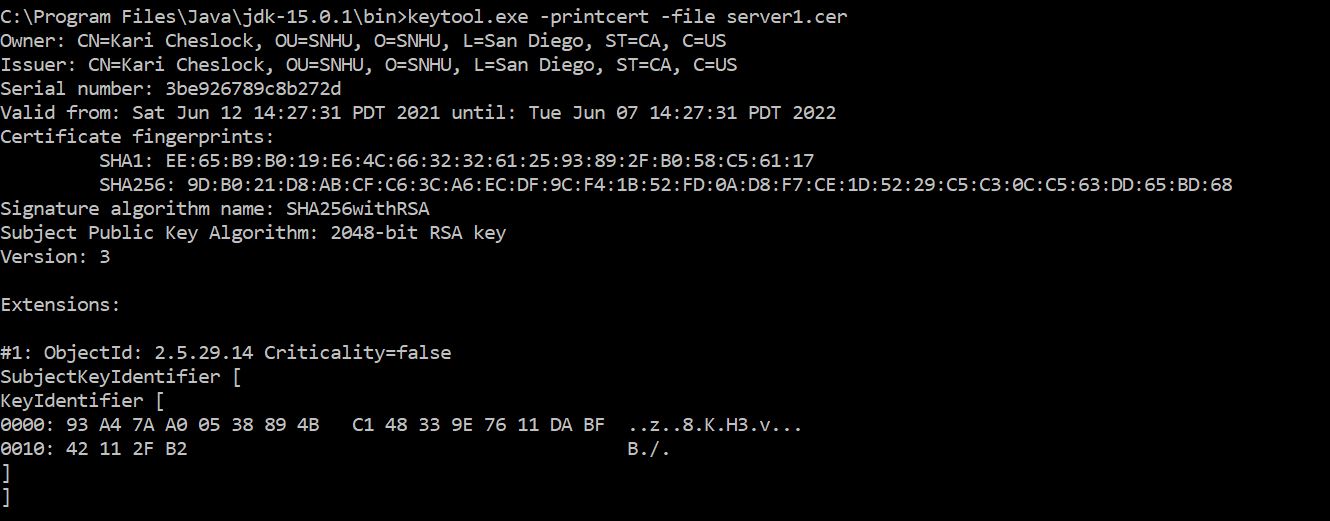
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

The recommended algorithm cipher for Artemis Financial is the Advanced Encryption Standard, or AES, cipher. This cipher is easy to implement and use and is particularly useful for file archiving. AES is a type of symmetric block cipher. It is symmetric in the sense that it uses the same key for encrypting and decrypting, as opposed to an asymmetric cipher which uses different keys for encrypting and decrypting. Keys can be of three different lengths, specifically 128-bits, 192-bits, and 256-bits. The 256-bit key length is recommended in this situation to provide the highest level of security. The block cipher used by AES encrypts data in blocks. This means that the hashing function transforms the data into blocks of data of a set size and encrypts each block of data. The size of the data blocks is determined by the key length. The key length also determines the number of rounds of encryption that each data block will go through, which will be 14 rounds in the case of a 256-bit key length. Overall, the AES-256 algorithm cipher will provide the best possible security for Artemis Financial’s sensitive information.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

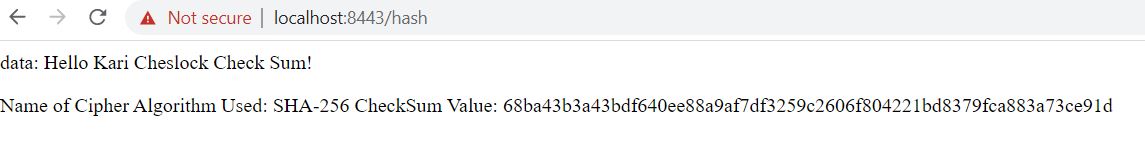
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

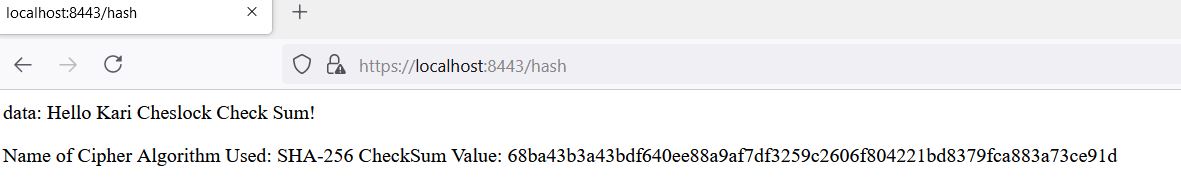
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

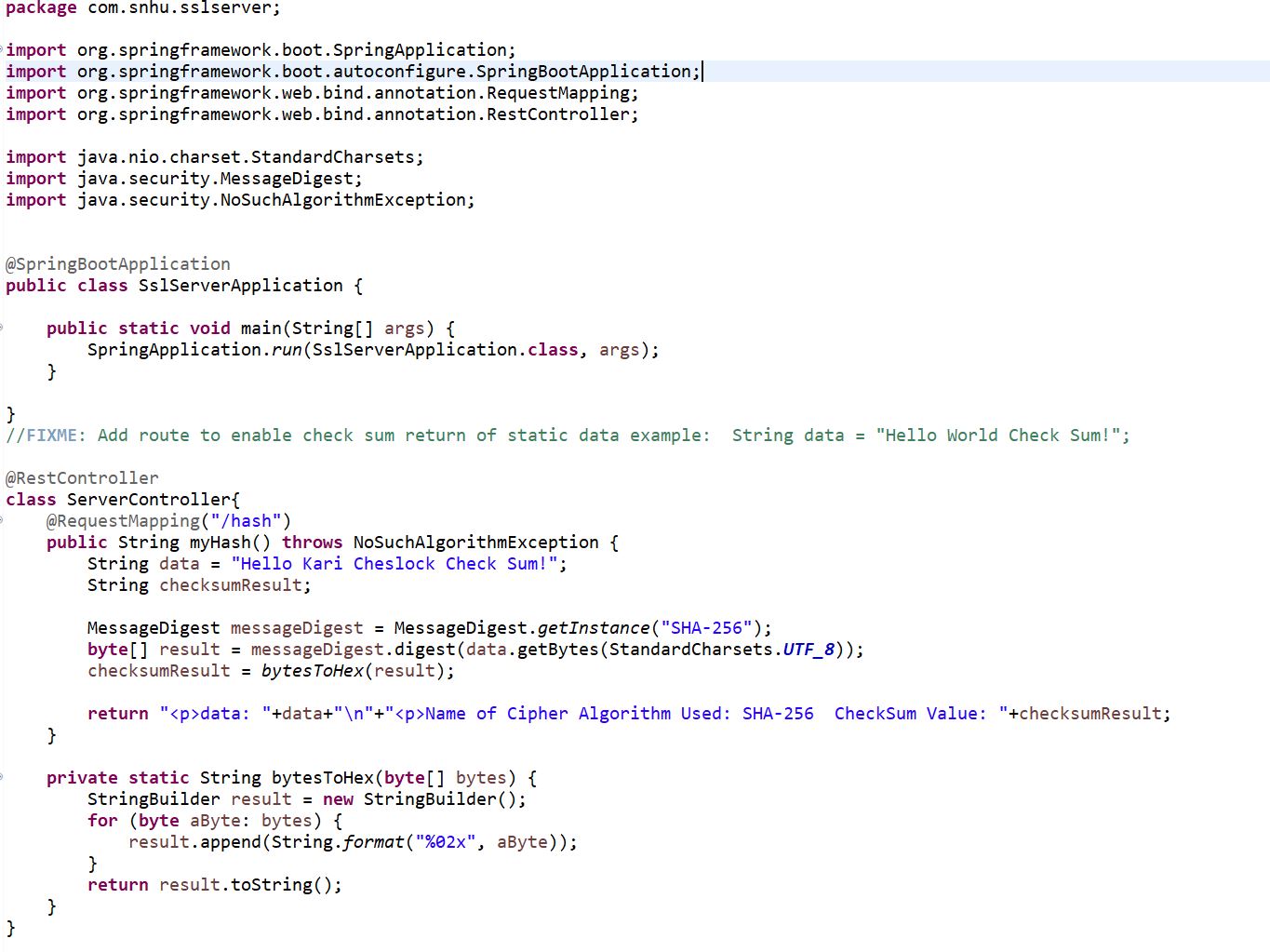
* Insert a screenshot below of the web browser that shows a secure webpage.

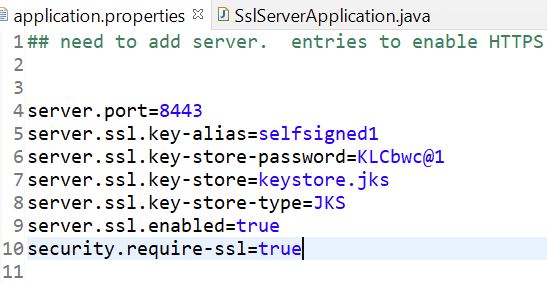


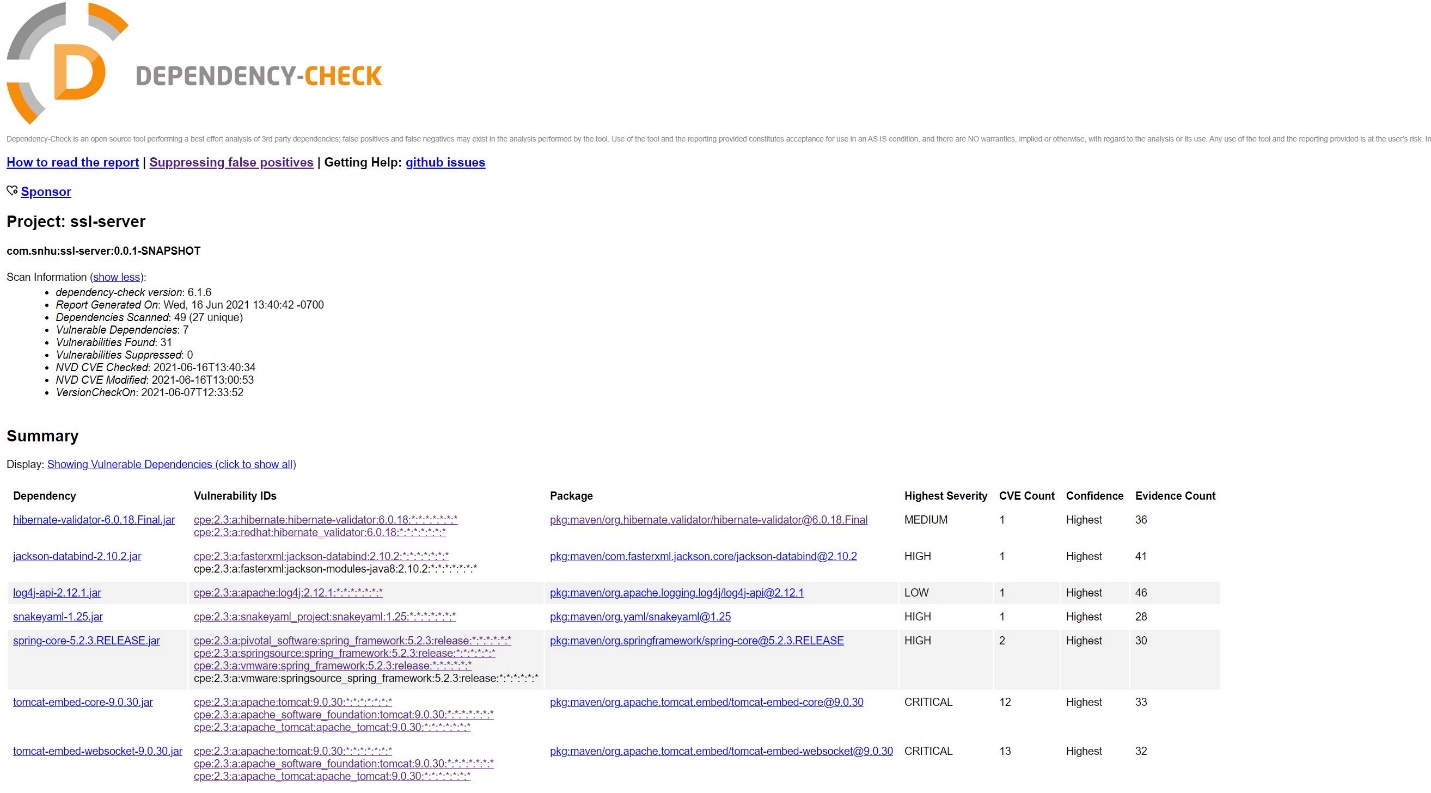
## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



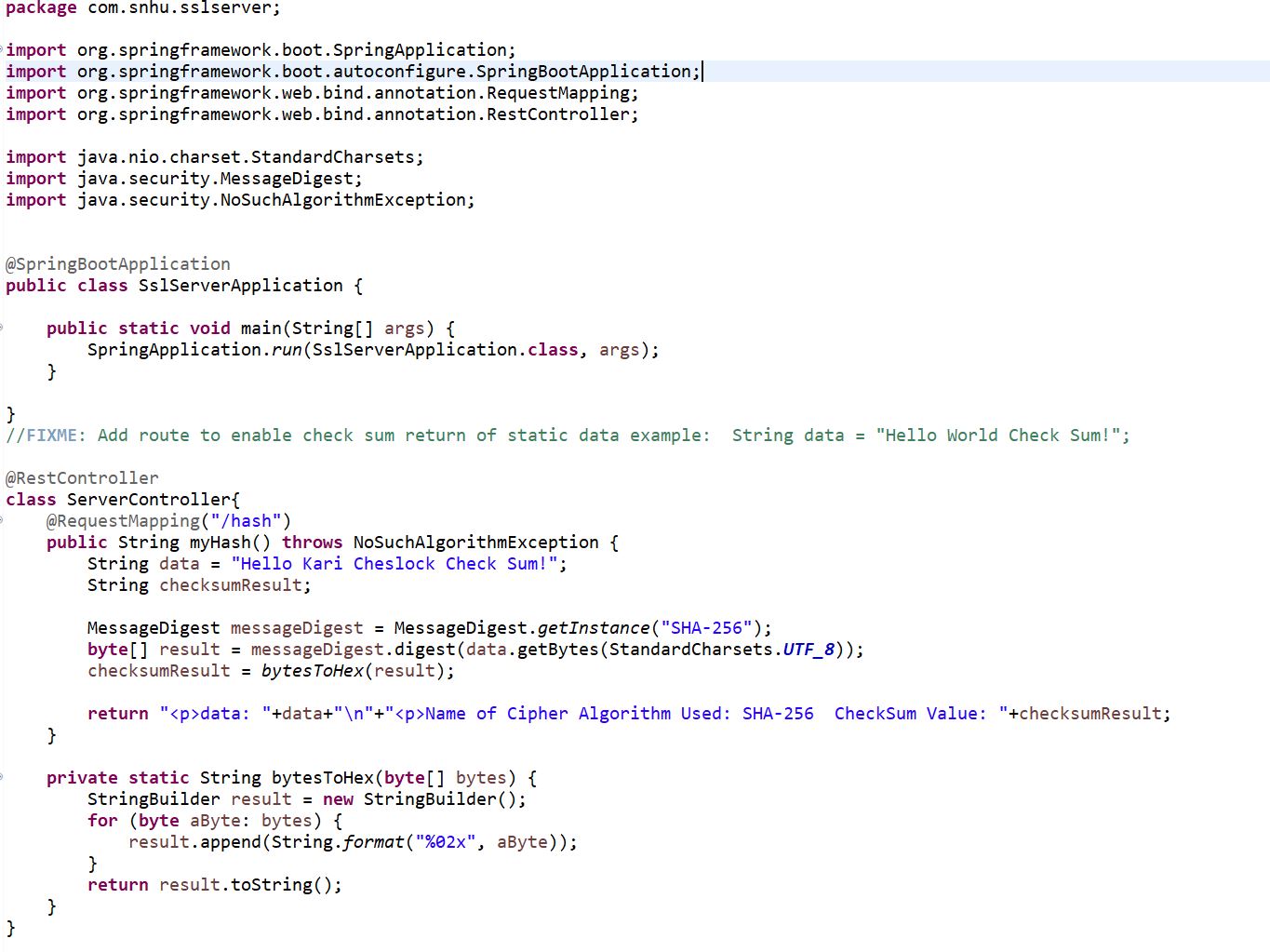


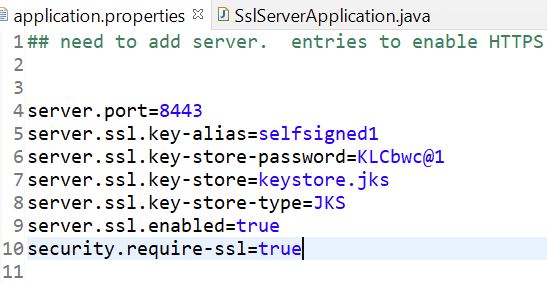


## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.





## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

The two main additional layers of security that were added to this code were the checksum verification for a string of data that was passed to the application and a certificate authority to allow the application to be accessed via the https protocol. The areas of security that were addressed with these added layers of security are Secure API Interactions and Encryption Use and Vulnerabilities based on the Vulnerability Assessment Process Flow Diagram. A checksum was added to the code by adding an instance of the MessageDigest class that used the SHA-256 algorithm to encrypt a simple string of data that was passed to the application. This checksum was then converted to hex and displayed on a localhost port. A self-signed certificate authority was generated using the Java keytool as shown in the application.properties file. Following the creation of this certificate, the application could be loaded using the https protocol which provided additional security for the application.

These security protocols exemplify best practices for securing web applications. Encryption allows for transactions of a sensitive nature to happen securely over the web. Certificate authorities ensure that a website is what it claims to be and is not an imposter website. The combination of these types of added security will be useful for any web application because it will give any user of the application the knowledge that the website is the website they are intending to use, and any transactions carried out through the website will be encrypted and secured. Without these security layers, the web application could be vulnerable to many types of attacks by hackers. For example, without a certificate authority a hacker could impersonate a website and steal sensitive information. Without encryption, sensitive information could be intercepted in transition and stolen by malicious parties. For these reasons, adding these types of security should be standard practice for any web application.

References

Crane, C. (August 11, 2020). What Is a Certificate Authority (CA) and What Do They Do? Security Bloggers Network. Retrieved from <https://securityboulevard.com/2020/08/what-is-a-certificate-authority-ca-and-what-do-they-do/>

Cobb, M. (April 2020). Advanced Encryption Standard (AES). Retrieved from https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard