****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_1fob9te)

[Client 3](#_3znysh7)

[Instructions 3](#_2et92p0)

[Developer 4](#_tyjcwt)

[1. Algorithm Cipher 4](#_3dy6vkm)

[2. Certificate Generation 4](#_1t3h5sf)

[3. Deploy Cipher 4](#_4d34og8)

[4. Secure Communications 4](#_2s8eyo1)

[5. Secondary Testing 4](#_17dp8vu)

[6. Functional Testing 5](#_3rdcrjn)

[7. Summary 5](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **Mitchell Lynds** |  |

## 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

Mitchell Lynds

## 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.

For this application we will be using SHA-256 encryption. SHA-256 is a commonly used and widely trusted algorithm. SHA-1 and MD-5 have known vulnerabilities and are no longer recommended for use. SHA-256 uses a 256 bit hash value as compared to 120 for MD-5 or 160 for SHA-1. SHA-256 is currently recommended by the National Institute of Standards and Technology.

Hash functions allow a cipher to take an input of any length and produce a unique hash value of set length. Different “bit levels” of a cipher refer to the length of the keys. Longer keys are less susceptible to brute force attacks. SHA-256

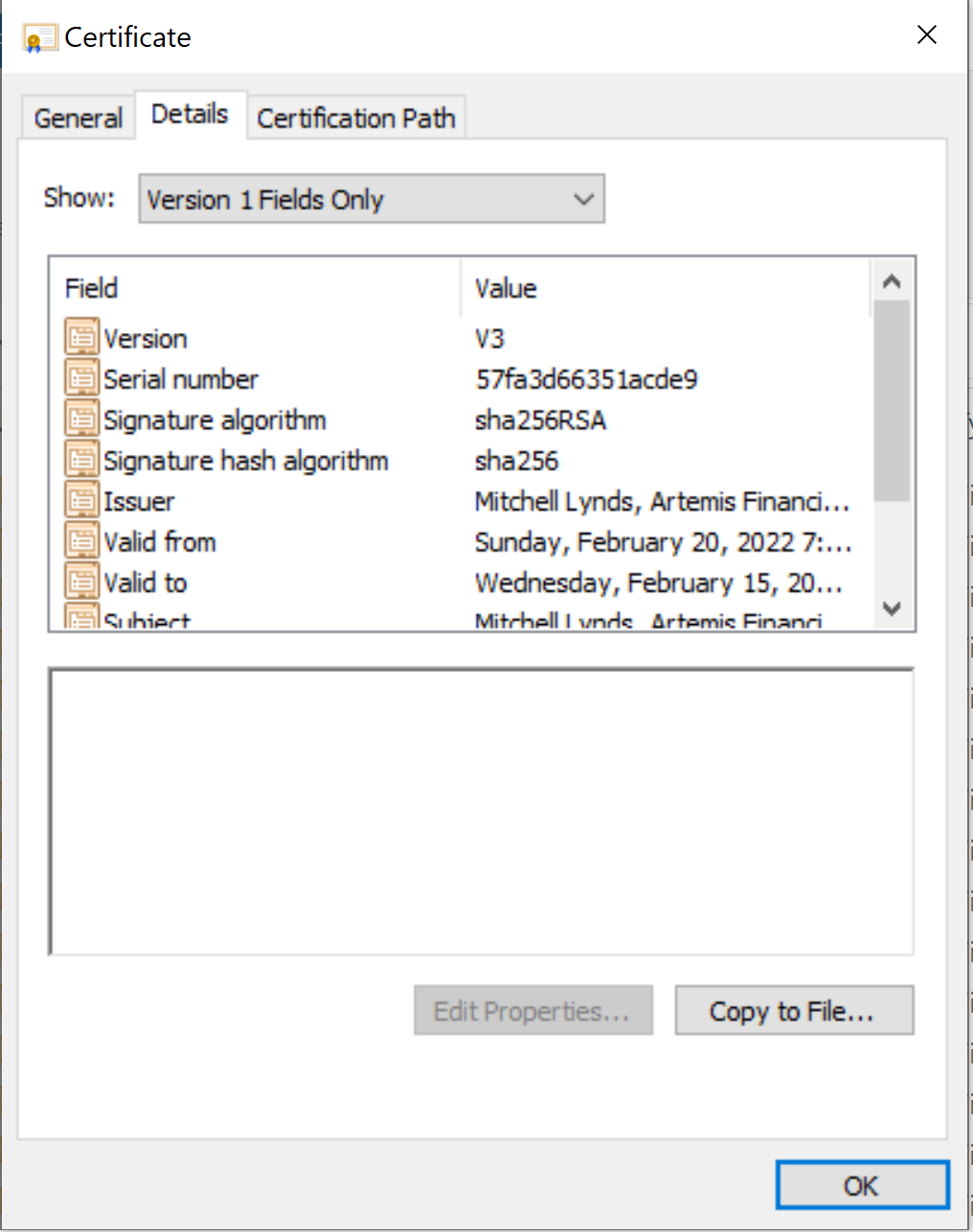
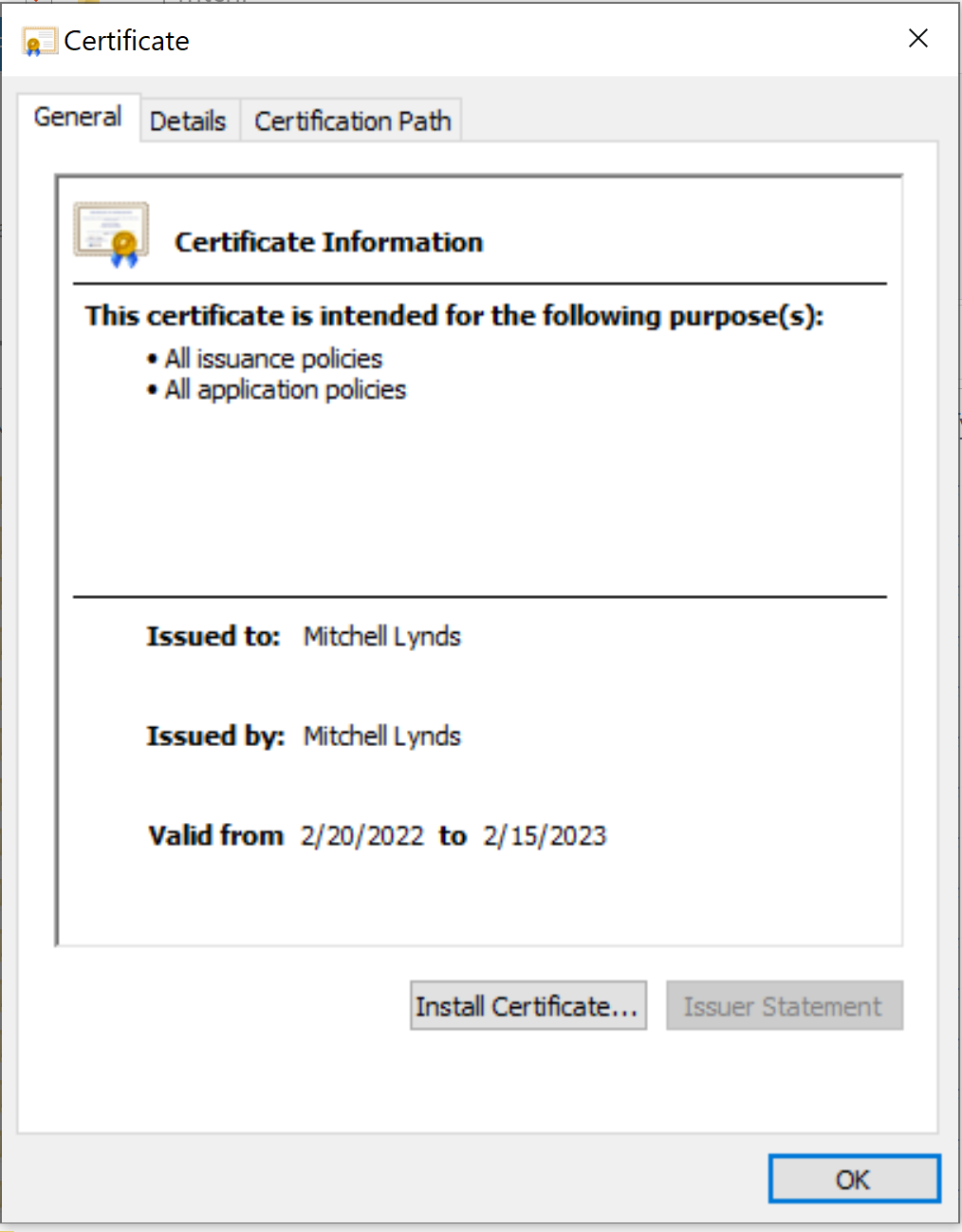
Encryption ciphers use random number generation to create keys. Symmetric keys use the same key for both encryption and decryption. Asymmetric encryption uses one key for encryption and another for decryption. Asymmetric encryption is good for data transfer because the encryption key can be made “public” and the decryption key kept “secret”. If, for example, Artemis Financial had an application for clients to submit tax documents, the users’ documents would be encrypted before transfer and only Artemis Financial would have the key needed to read them. In symmetric encryption the key must be kept secret. Symmetric encryption is appropriate for file storage because the key does not need to be transferred to a second party.

Encryption has been in use in some form or another for hundreds of years. Encryption went into widespread use for computer data in the 1970’s. Over time older encryption algorithms have become obsolete due to advances in computing power. Modern algorithms in use would take many years to crack using brute force methods on today’s hardware but advances mean that nothing is secure forever.

## 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.



## 

## 

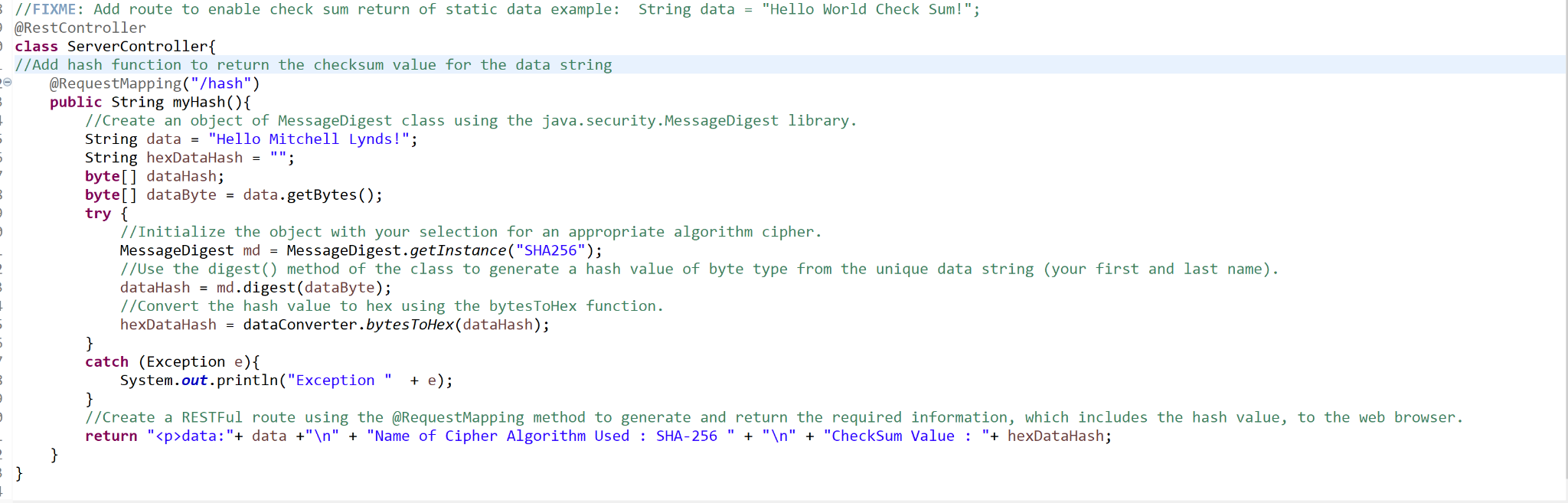
## 

## 

## 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.

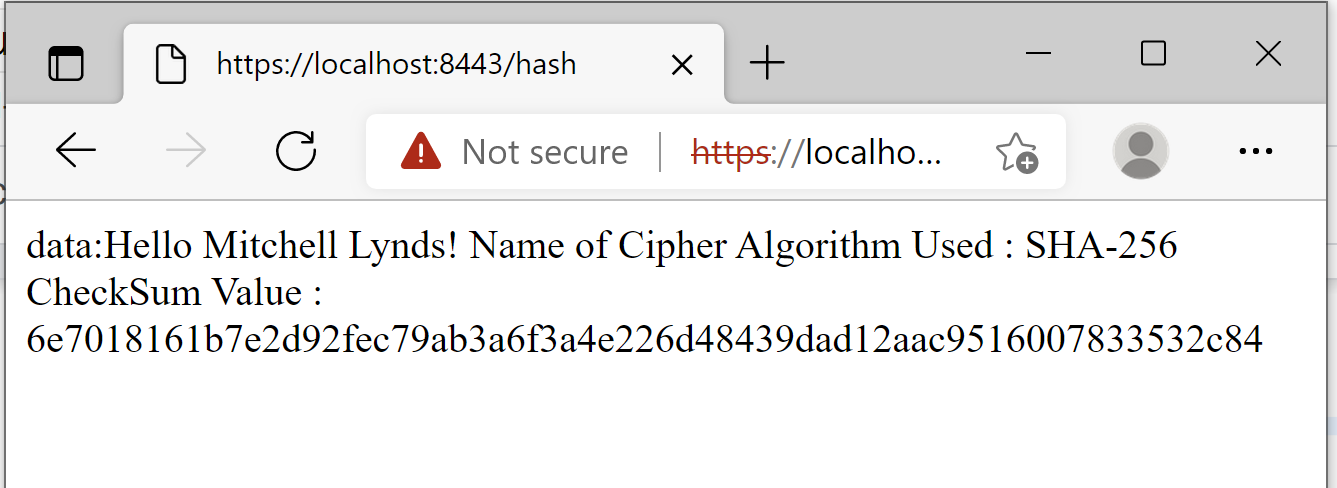
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



## 

## 

## 

## 

## 

## 

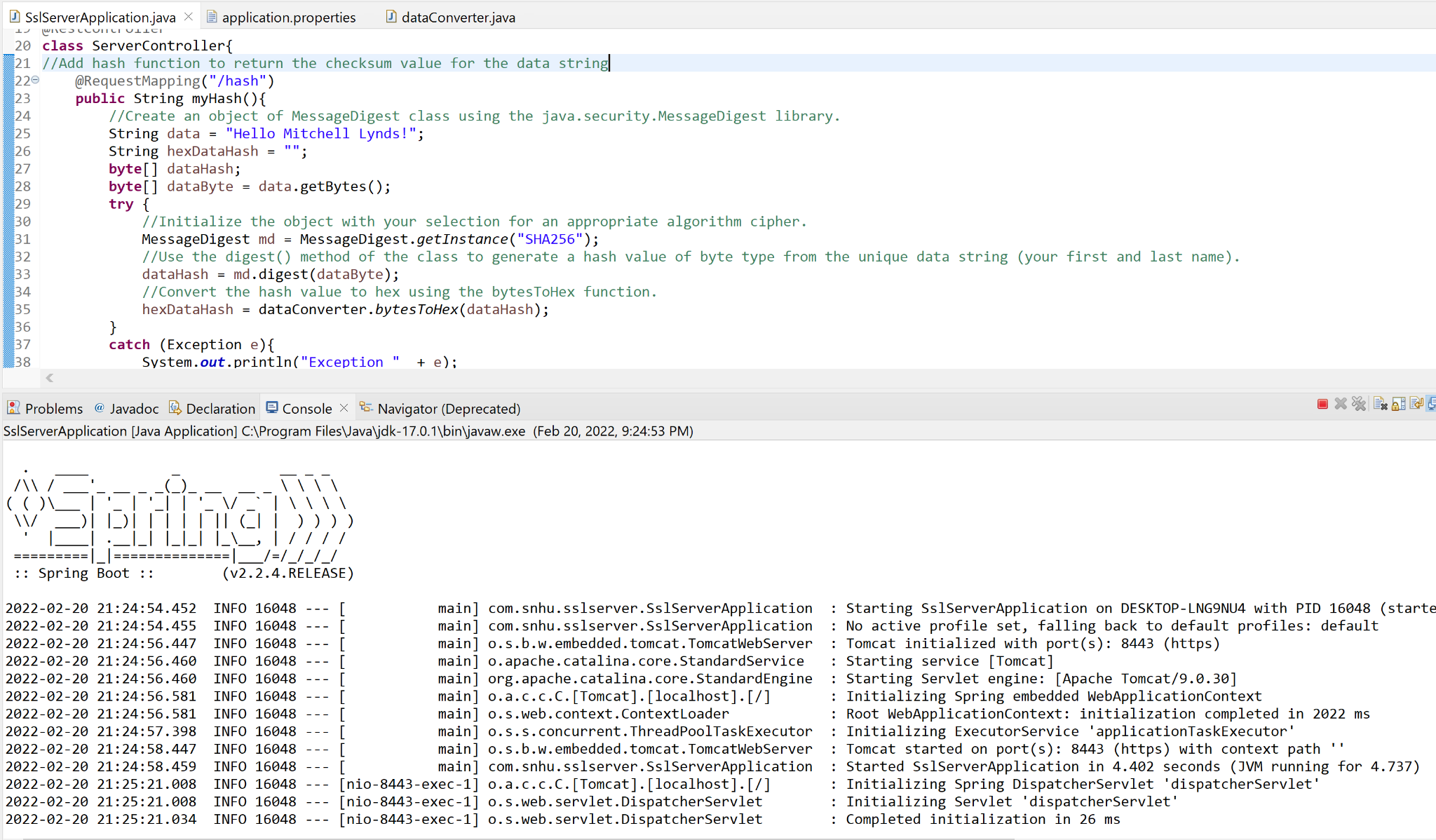
## 

## 

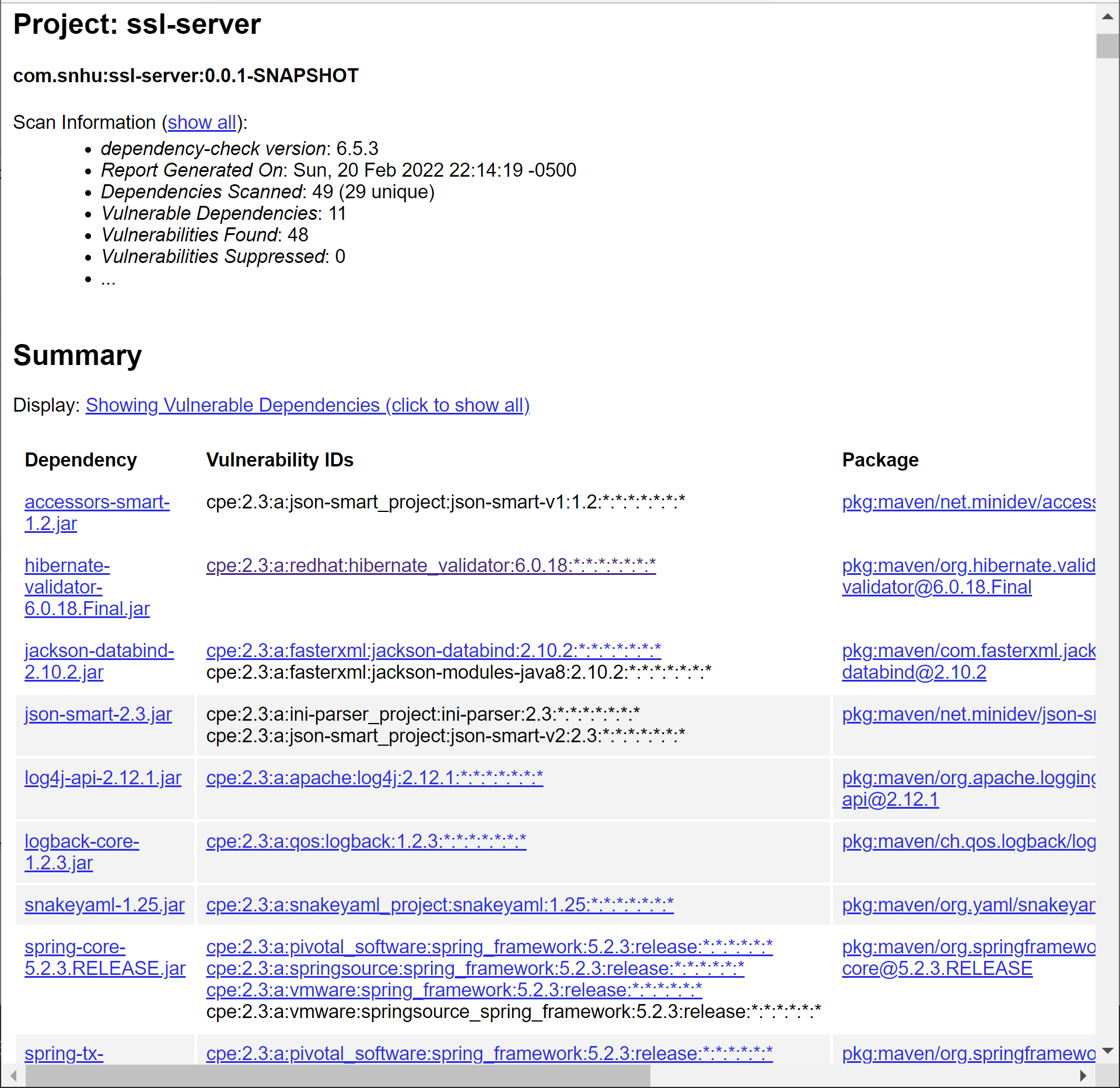
## 5. Secondary Testing

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



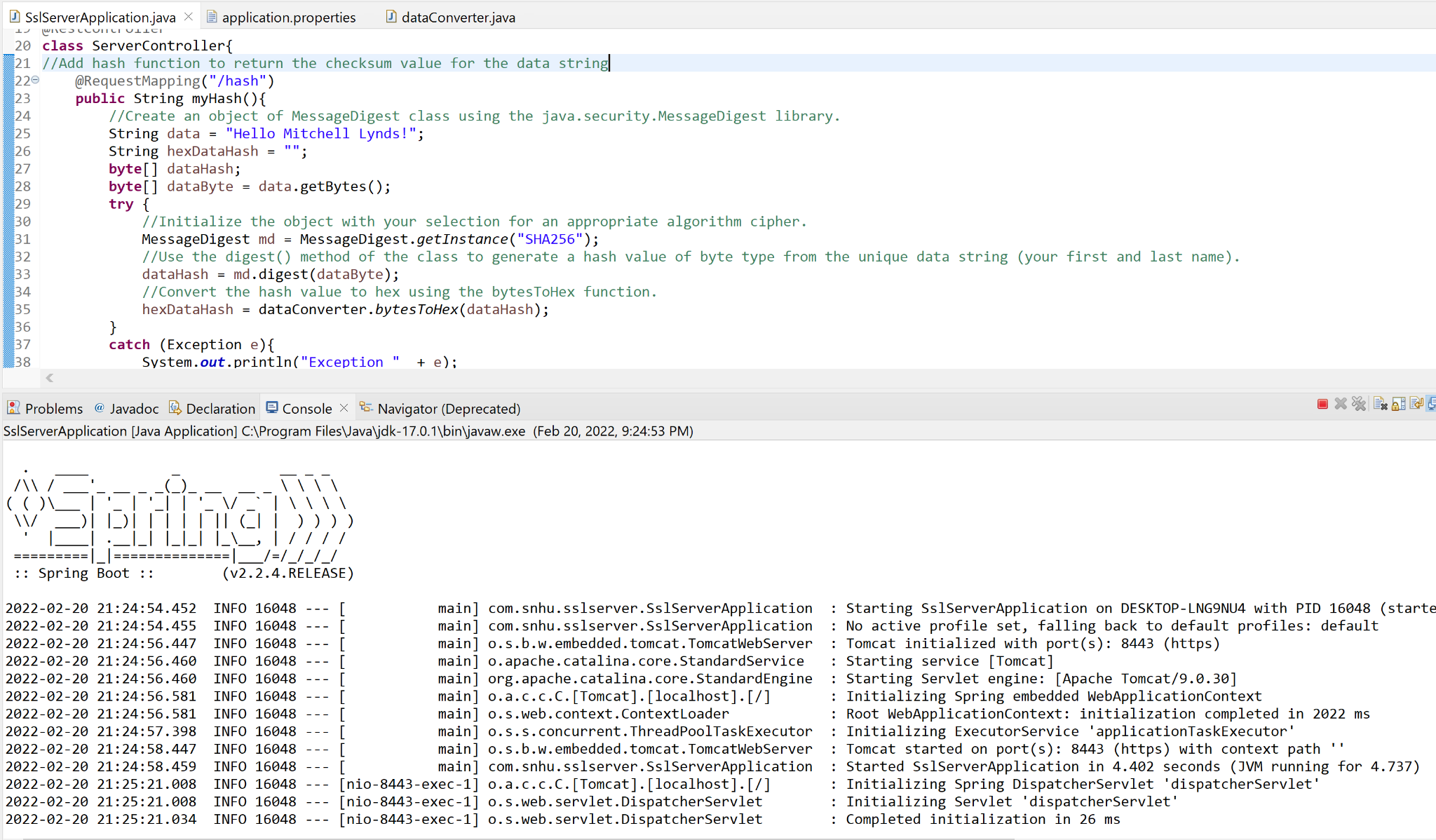
* + A screenshot of the dependency check report



## 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 process for ensuring the security of the software application involves review of seven main aspects of the application. These are: Secure Input and Representations, Secure API Interactions, Encryption Use and Vulnerabilities, Secure Distributed Composing, Secure Error Handling, Secure Coding Patterns and Secure Data Structures. Any security breach can be extremely costly financially for a company. A company may have their data ransomed, receive bad publicity and loss of customers, or loss of productivity while a system is down. Ensuring that each of these seven areas of security are implemented can help protect the application from the majority of vulnerabilities.

In refactoring the code for this application the primary areas of security that were addressed were Encryption Use and Secure Error Handling. An encryption algorithm was implemented and error handling that does not reveal information to the user was added.

There are several practices necessary for maintaining the security of this software application. First and foremost periodically running dependency checks will help to identify any new vulnerabilities that are discovered within the application. Most vulnerabilities can be mitigated by simply keeping the dependencies updated. In addition to this, it is also worthwhile to periodically check to make sure that our chosen encryption algorithm of SHA-256 is still considered secure.