****

# CS 305 Project Two

**Practices for Secure Software Report**

David Smith

Southern New Hampshire University: Department of Computer Science

CS-305: Software Security

Professor Sabahudin Tricic

June 20, 2021

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** | **[June 19, 2021]** | **David Smith** | **Initial version** |

## 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

David Smith

## 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 encryption algorithm cipher chosen for this checksum modification is SHA-256. SHA-256 is a variant of the SHA-2 encryption algorithm. It uses 256-bit encoding. SHA-256 is a block hashing cipher. This means that it converts input data into blocks of 256 bits length of encrypted data. As SHA-256 is encrypt only, the result gives a checksum that can be compared without giving out the unencrypted data. No collisions have been possible so far with SHA-256.

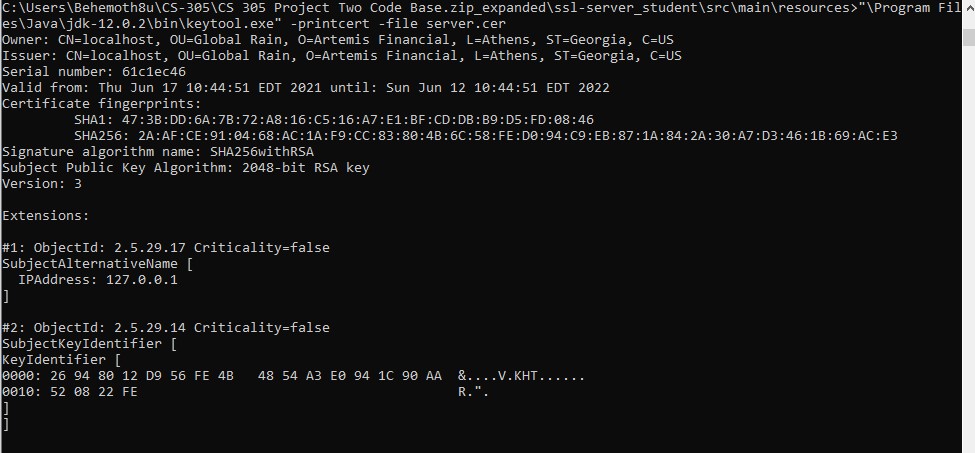
The encryption algorithm chosen for the certificate is RSA with a 2048 bit key. This is an asymmetric encryption algorithm. This means that one private key does the encryption, and a shared public key allows verification of the certificate. As this certificate is self-signed, it was important to add the Subject Alternative Name to allow certain browsers to be able to authenticate the connection.

A few encryption and hash algorithms have been cracked. However, the ones chosen for this project are considered beyond the capacity of current computational ability to break within such an extreme amount of time as to be considered impossible.

## 2. Certificate Generation

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

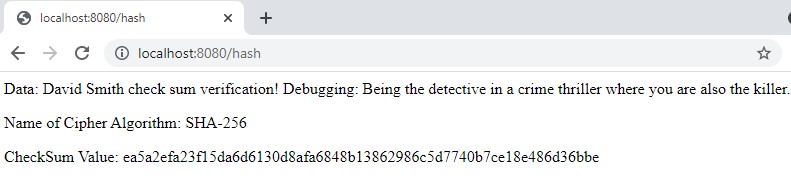
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

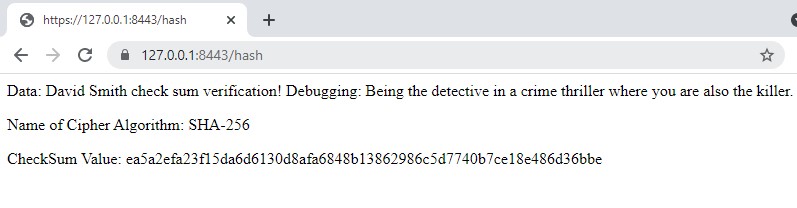
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

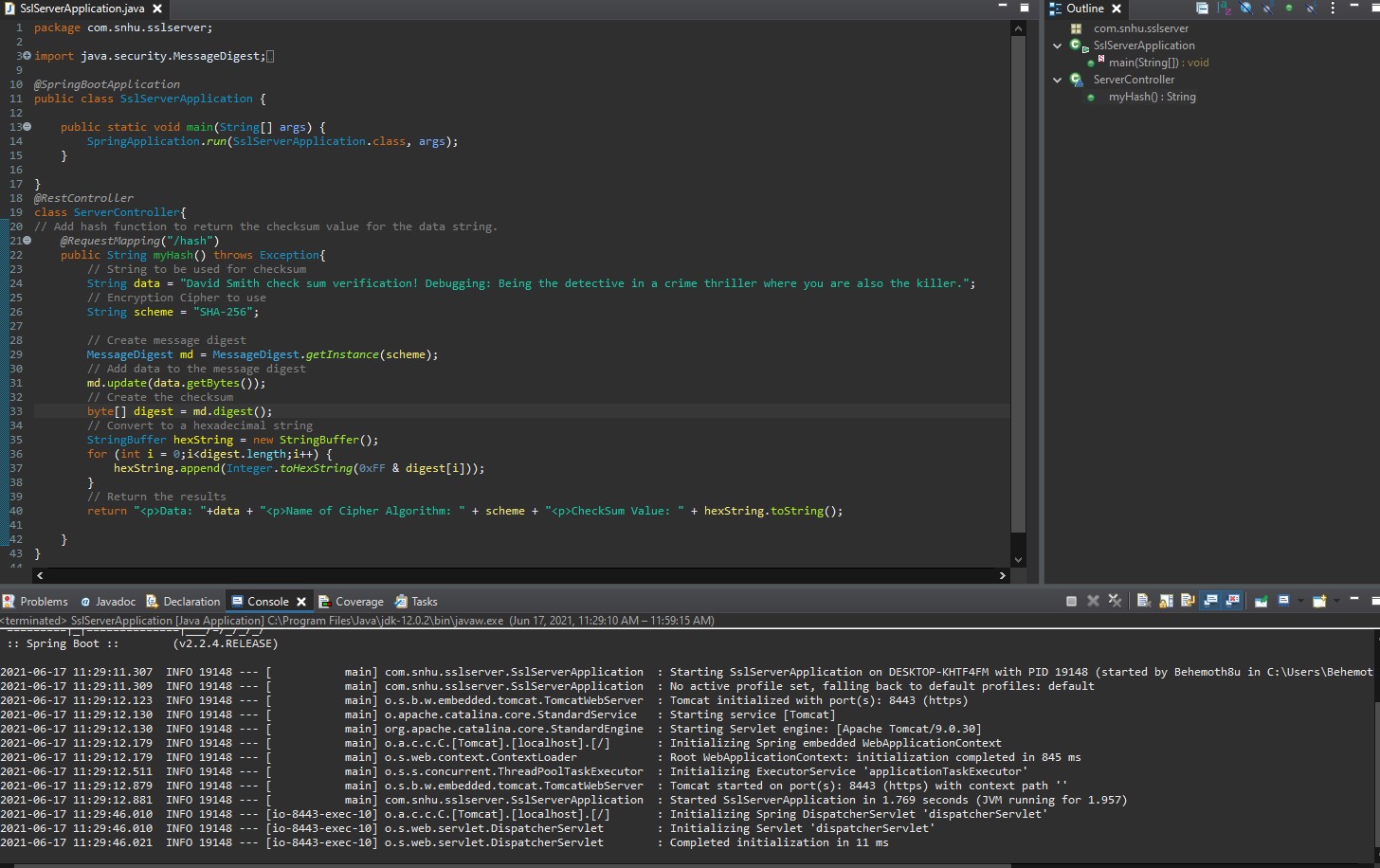
* Insert a screenshot below of the web browser that shows a secure webpage.

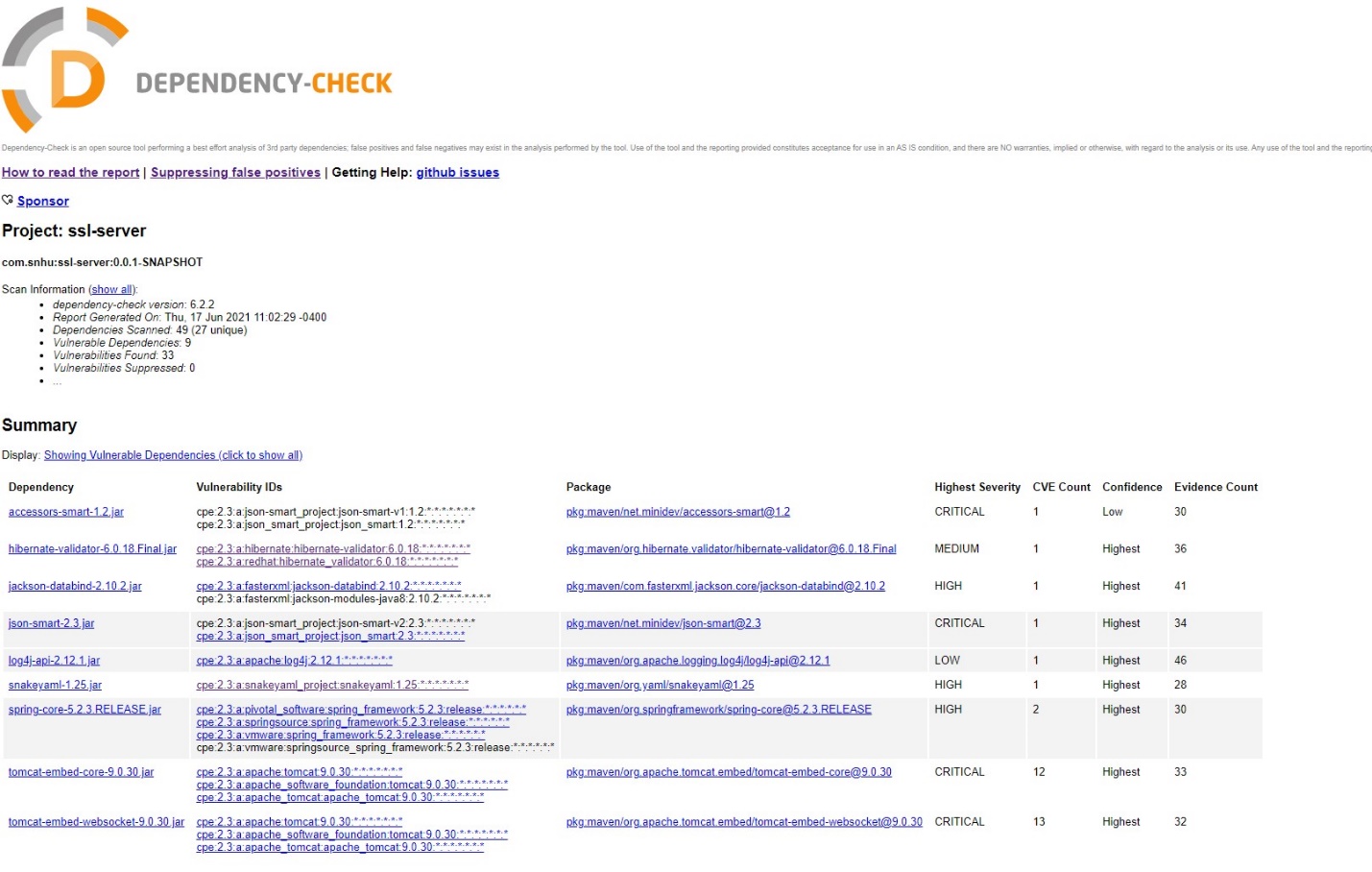


## 5. Secondary Testing

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

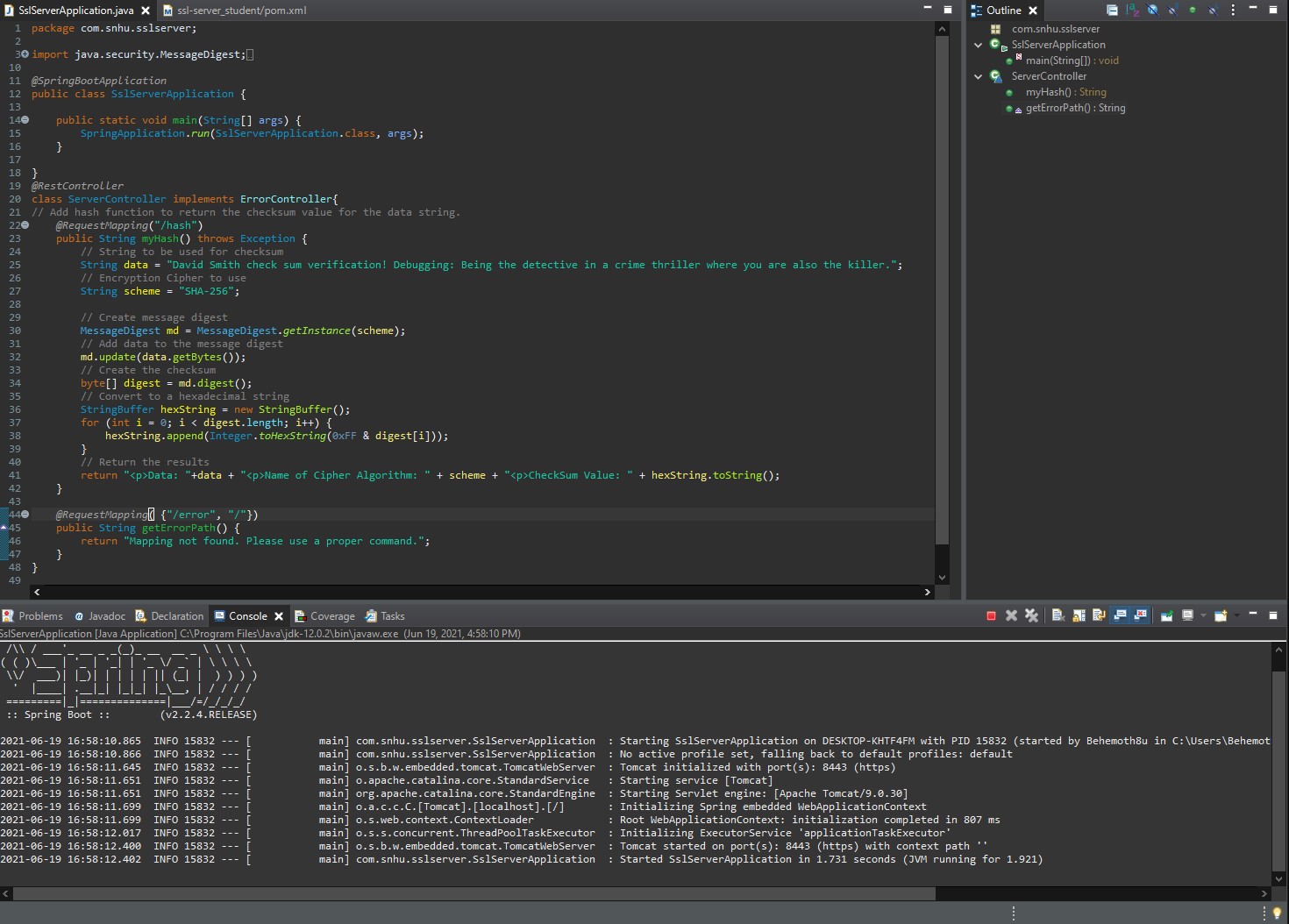




## 6. Functional Testing

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.

While examining the base functionality of the provided code, there were several areas of vulnerability that were noticed. These were APIs, Cryptography, Code Error, Code Quality, and Client/Server. To close these vulnerabilities, it was necessary to make some changes to the original code.

The provided reports for the dependencies show which APIs need to be updated to newer versions if possible. All modifications to the code were made without adding any additional dependency vulnerabilities.

Code Quality and Code Error were both modified by adding a code generated default error page. The base code did not have a mapping for errors, so it was using Tomcat’s built-in error page. This could lead to issues with error messages leaking information. For example, the “/” mapping was showing a code snippet. This was also mapped to the created error page.

Cryptography was addressed by adding a checksum verification to the application. This allows users to verify the data that they were given by comparing its checksum to the one provided. The checksum verification was created using the SHA-256 hashing cipher. If this is changed in a future update, it is important to ensure that the cipher used is secure.

Client/Server communications were addressed by adding a certificate, created using RSA, to verify the server’s identity. This certificate needed the Subject Alternative Name as mentioned earlier to make it work with modern browsers. To maintain security through this method, it is recommended that a trusted certificate authority be used to store a certificate as trusted proof of the server’s identity. This will allow the system’s identity to be verified by users and protect from imposters. The current certificate will only work on the local machine using the localhost or 127.0.0.1 loopback address.

By adding the certificate and using it within the application, SSL was enabled to require https protocol be used for accessing the REST API. This creates an encrypted network interface between the client and server, which also is part of the cryptography updates. Clients are no longer allowed to connect using an http connection. This will protect user data from being easily accessible.

The updated code is being submitted as an attachment to this report as David\_Smith\_CS-305\_Project\_Two\_Code\_Base.zip.