****

# 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** | **8/20/2022** | **Jordan Barnes** |  |

## 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

Jordan Barnes

## 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.

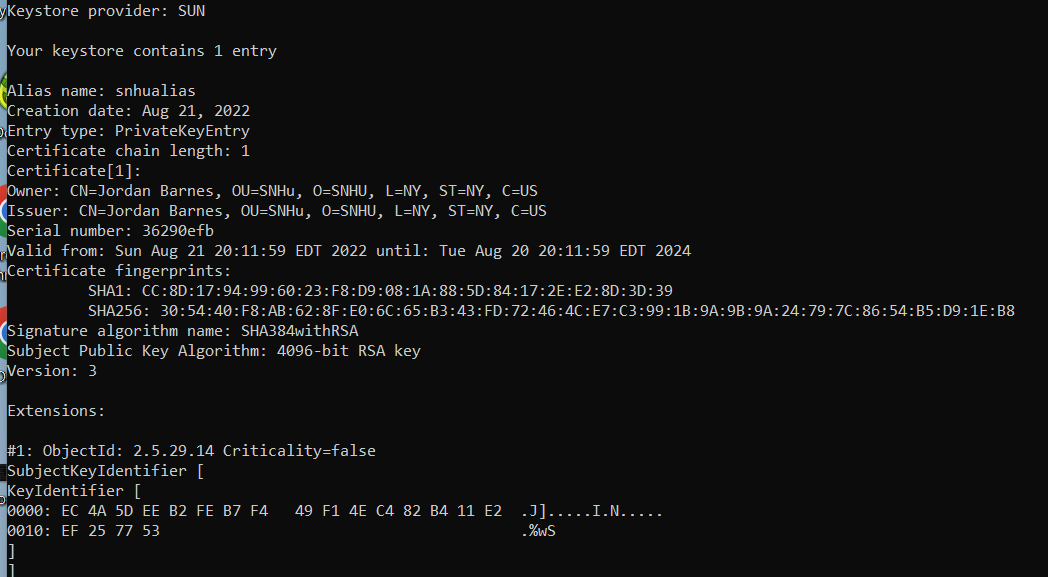
Before transmission of data occurs, all components of the data can be assigned a checksum value after running a cryptographic hash function. In essence, a CheckSum is just a value that represents the number of bits in the message being transmitted. SHA-256 CheckSum will aid in verifying the integrity of data as it is moved between endpoints, thus, making it more secure. There are a variety of cryptographic functions that can be used to secure stored and transmitted data. There are the widely used and known ciphers like AES and DES. A state-of-the-art AES hardware implementation uses 2400 GE and is widely used to benchmark new ciphers. (Sugawara, 2020, p. 1) Researchers and data scientists are constantly developing new ways to implement ciphers. For key encryption to work properly, it must have some way to verify with its recipient that it is what it claims to be. SSL provides this proof by requiring one or more of the parties to provide a digital certificate for the connection to occur, prior to any transmission of encrypted data. This is commonly referred to as the “handshake” (Sugawara, 2020, p. 8)

Bluntly, symmetric encryption uses the **same** key to encrypt and decrypt data. Asymmetric encryption (public-key cryptography) uses public and private key pairs to encrypt and decrypt.

## 2. Certificate Generation

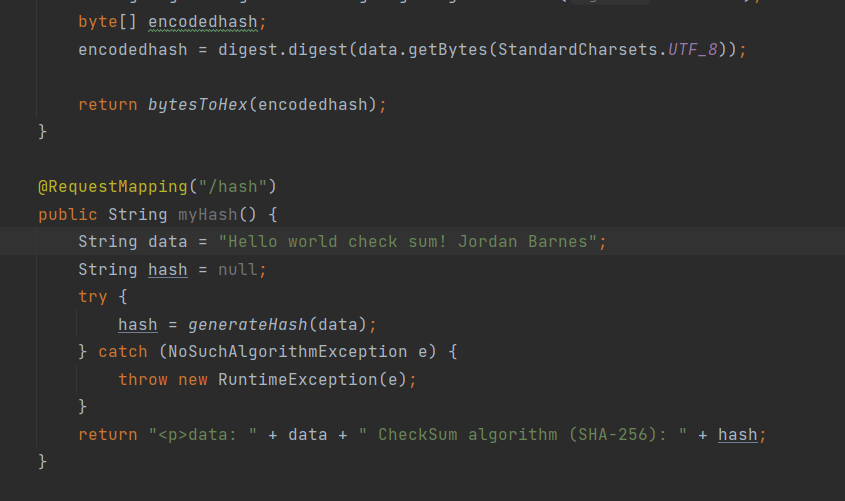
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* 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

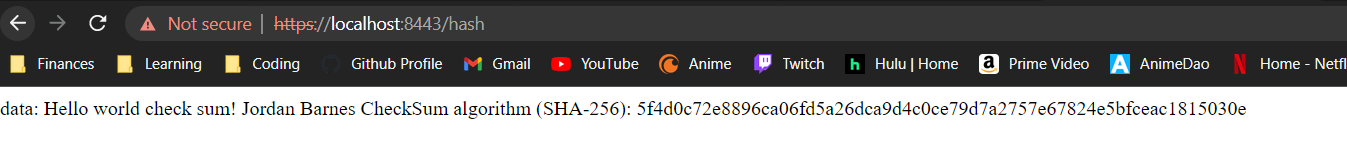
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string **(in the code you specifically asked for “Hello world check sum!” so I provided both.)** 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.

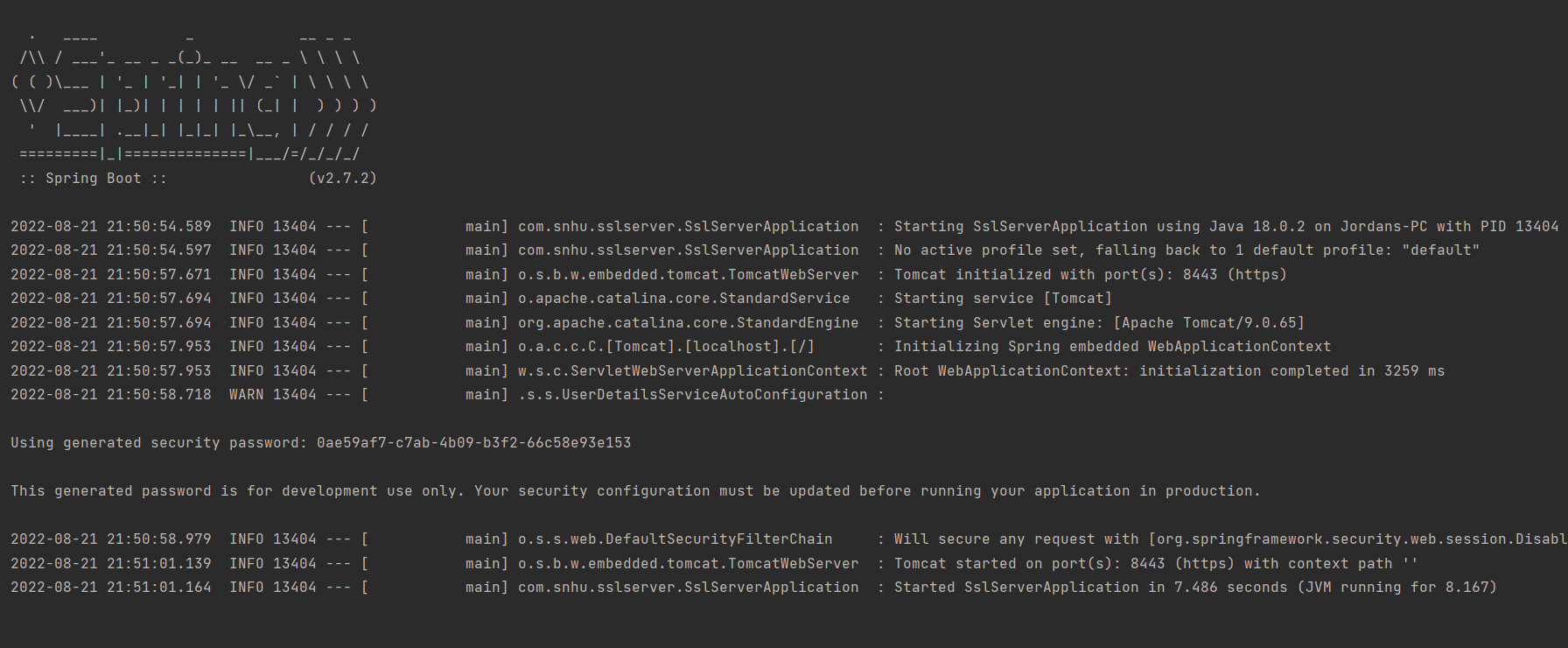
* Insert a screenshot below of the web browser that shows a secure webpage.

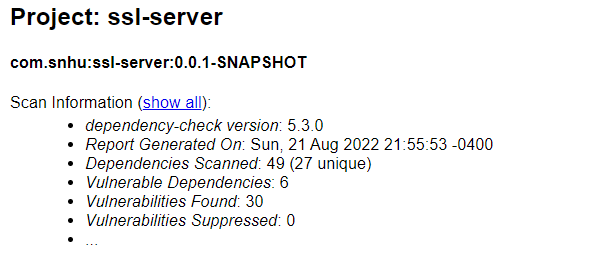
(**You did not specify in the instructions to import the certificate to the CA root trust store, so the browser does not recognize the self signed cert as secure.)**

## 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

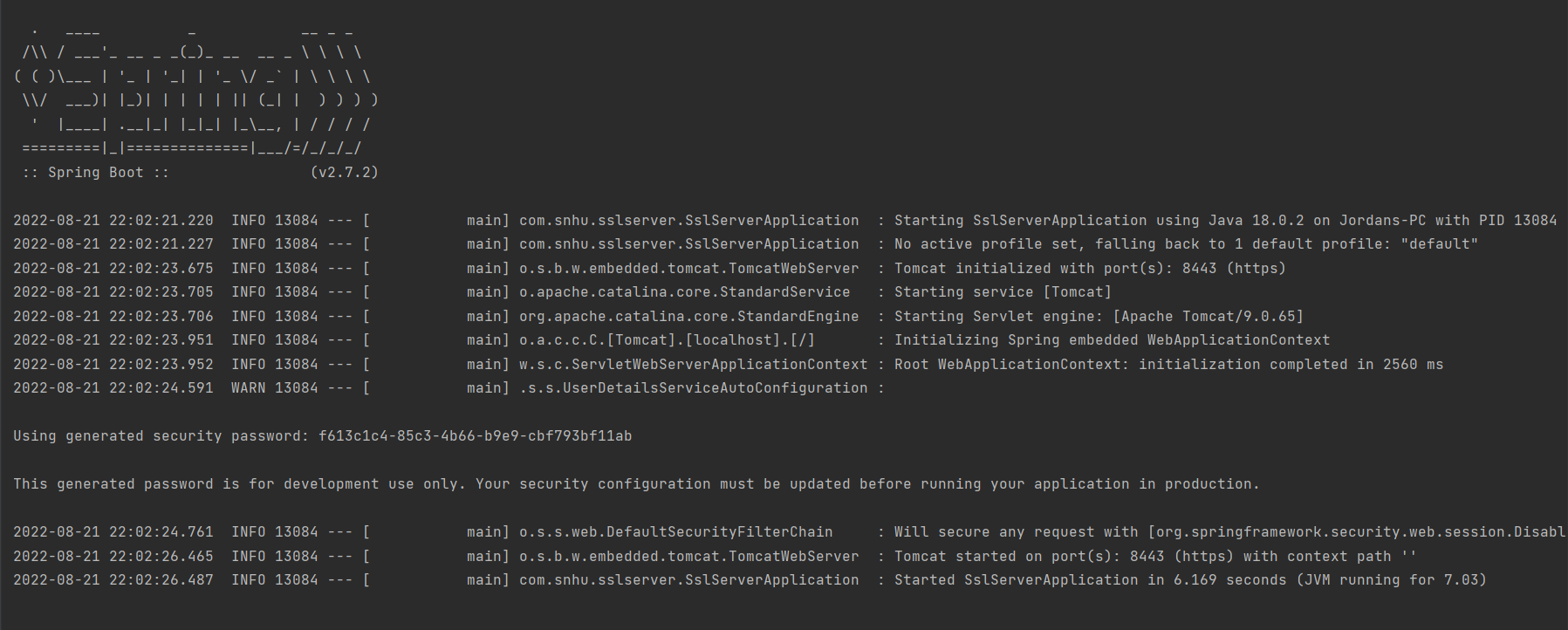




## 6. Functional Testing

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.
* Input Validation: We implemented a checksum verification to confirm data integrity of our string input
* Cryptography: We implemented a SHA-256 encryption algorithm to encrypt data. We also implemented SSL socketing which provides us with web key encryption capabilities.
* Code Error: Reviewed and fixed all syntactical errors and applied a secondary security layer using spring-boot-security functionality.
* Client/Server: We will need to ensure that we purchase a CA certificate and confirm that it is used to guarantee data is reliable while transferring via https requests.

The customer should seek to inspect any additional code implemented. They should confirm that data integrity remains in-tact and testing is added as code is implemented.

**Citations**

Sugawara, T. (2020). Hardware performance evaluation of authenticated encryption SAEAES with threshold implementation. *Cryptography*, *4*(3), 23. https://doi.org/10.3390/cryptography4030023