****

# 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** | **08/09/22** | **Brandon Coulter** |  |

## 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

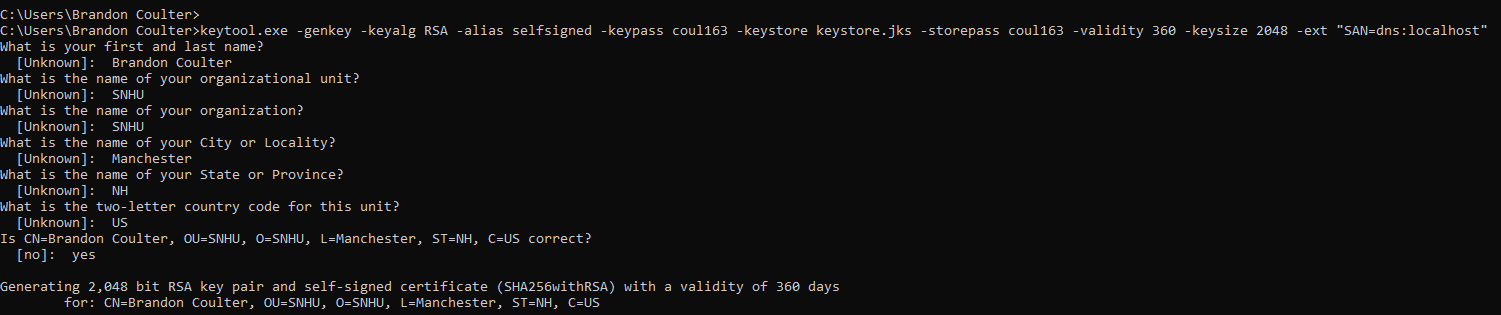
Brandon Coulter

## 1. Algorithm Cipher

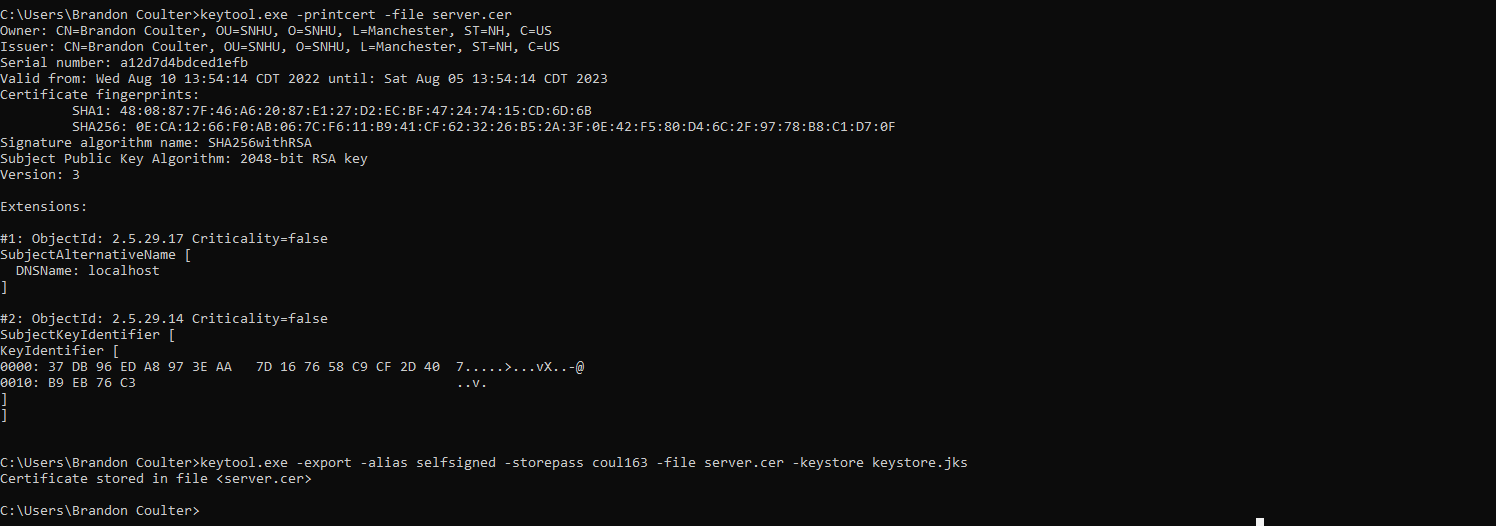
Encryption algorithms are vital to the secure data transfer of the internet. For example, a checksum value generated by an algorithm can be used to verify the data is not manipulated to be malicious. Given Artemis Financial seeks to add this verification process to their application, the suggested method of choice would be a SHA-256 algorithm. This algorithm is highly secure and is widely utilized for data verification. (Manico et al., 2014) The SHA-256 algorithm, as described by the National Institute of Standards and Technology (NIST), “algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a message digest” (National Institute of Standards and Technology, 2015) These message digests can then be used in the verification process of the data that is being received. A comparison of the digests should provide a means of verification such that the data requested is secure and the data received was intended not malicious or erroneous. Because of the high likelihood that even a slightly altered message would produce a completely different digest, the SHA-256 algorithm is exceptionally secure. (National Institute of Standards and Technology, 2015) This hash function is 256 bits meaning it outputs a value of 256 bits after the hash. Because of the bit level of this algorithm it is nearly impossible to crack with current technology. (National Institute of Standards and Technology, 2015) Additionally, cryptography algorithms implement random numbers to intensify the security of the algorithm. However, these numbers are not exactly random as the computer can only generate pseudorandom numbers which are based on a seed or key typically of something truly random such as system time. (Manico et al., 2014) As well as the use of random numbers, cipher differ based on the key structure, take for example the SHA-256 algorithm, which is a non-symmetric one way key. In other words, the same key is not utilized to get the original message as it is to encrypt the digest. This is different from a cipher that utilizes the same key to encrypt and decrypt a message such as the AES-256 cipher. (Manico et al., 2014) As time goes on, ciphers develop and increase in encryption strength; nevertheless, malicious hackers also grow their arsenal of hacks to bust these encryptions. For example, encryptions such as the DES cipher are no longer safe encryptions to utilize. This has led to stronger ciphers such as the AES-256CBC which is currently in use by the U.S government and has proven to be exceptionally strong given the current computational power of computers.

## 2. Certificate Generation

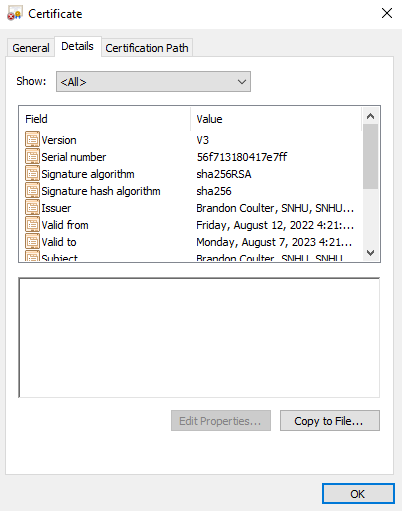
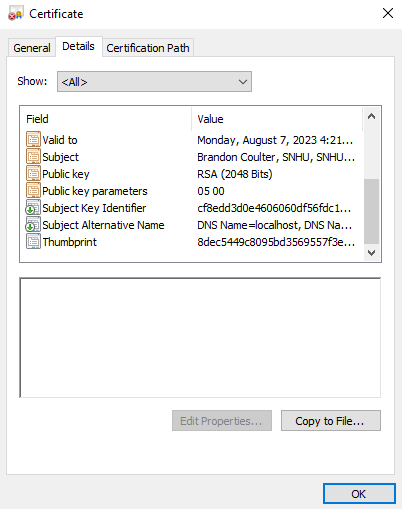
**Certificate Generation Information:**

****

**Certificate Generation Complete:**

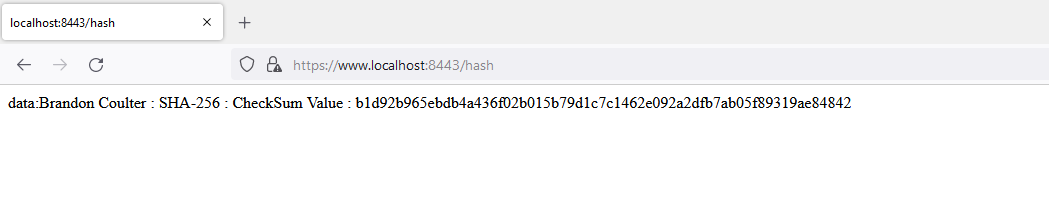
****

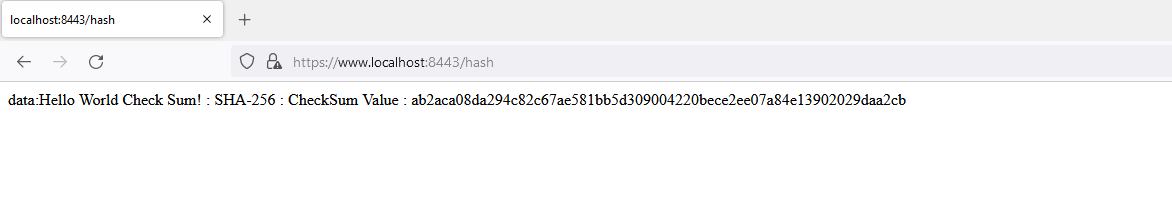
**Server.cer File:**

** **

## 3. Deploy Cipher

## As seen, the deployment of the SHA-256 cipher has resulted in a checksum verification that has been routed using the RESTful API and is visible in the web browser.

**Cipher Deployment With Name:**

**Cipher Deployment With Hello World Example:**

## 4. Secure Communications

## Indicative of the of the lock displayed on the browser, and the https protocol that is visible in the address bar of the screenshot below, the HTTPS protocol and certificate have been utilized successfully within the web application.

**Secure Communications Verification:**Graphical user interface, text, application, email

Description automatically generated

## 5. Secondary Testing

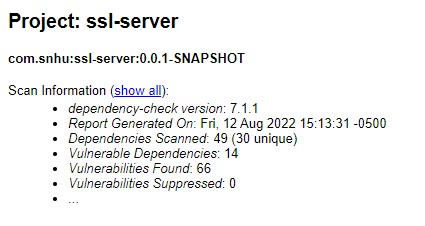
Upon further review as demonstrated by the screen shot held below, both the pre and post dependency check reports indicate the same dependencies are vulnerable and that no additional vulnerabilities have been identified upon refactoring of the code base. Additionally, the final screenshot demonstrates the proper execution of the code without errors.

**Pre-Refactored Dependency Report:**

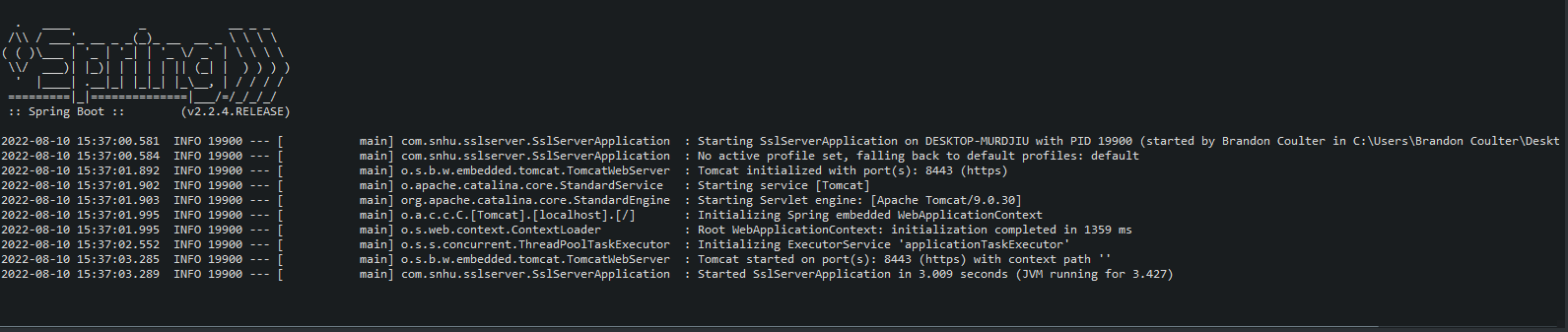
**Text

Description automatically generated**

**Post-Refactored Dependency Report:**

****

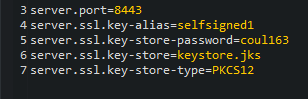
**Code Execution Verification:**

****

## 6. Functional Testing

With a web based application such as this, conducting a manual review following the vulnerability assessment process flow chart is as always necessary. Regions of the application such as APIs, cryptography, client/server, code error, and code quality were of most concern. Following a manual code review, it was identified that this web application should use a RESTful API to transfer data securely. Prior to refactorization of the code, a RESTful API was not implemented for safe data transfer. The rest controller is missing in the SslServerApplication.java file. APIs must be secure, of which importance is describe by RedHat, a major open source software distribution company, as “[Businesses use APIs to connect services and to transfer data](https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces). Broken, exposed, or hacked APIs are behind major data breaches. They expose sensitive medical, financial, and personal data for public consumption.” (*What is API security,?* 2019) The RESTful API should implement a secure connection via the HTTPS protocol. Additionally, the client/server relationship is secured by utilizing signed certificates for the server of which there were none. It is necessary to generate a certificate utilizing the Java keytool program. The keytool program is a program used to create keys for both keystores and truststores on client and server machines. (Manico et al., 2014) Although they are self-signed, the connection is still secure. Typically, a certificate is signed by a third party certificate authority thus making it a trustable certificate, however this certificate is valid as well. Implementation of this can be found in the application.properties file in the resources folder. By changing the settings as shown in the screenshot below titled “Application Properties”, the application will utilize the self-signed certificate on the server. Because of the nature of this program, it is important as per usual, to ensure a high standard for code quality is met to guarantee the best practices are maintained. In doing so, not only will the quality be exceptional, but the security of the application will be less vulnerable. Take for example, lines 26→33 of the SslServerApplication.java file, which creates a message digest utilizing known and trusted built in java functions. Moreover, secure practices such as the static dependency check were implemented in the pom.xml file on line 50→61 to confirm the quality of the code and some if not most of the vulnerabilities it faces. The most recent version of this check, 7.1.1 provides a report of all known vulnerabilities based on the National Vulnerability Database, with relation to the dependencies with the project.

**Application Properties:**



**Code Execution Verification:**

A computer screen capture

Description automatically generated with medium confidence

## 7. Summary

Following the refactoring of the code, features such as the RESTful API and client/server certificates were added to complete this project for Artemis Financial. For example, in in refactored code in SslServerApplication.java line 22→57 this section of code implements the rest controller which is the portion of the RESTful API. As mentioned above, this RESTful API utilizes the HTTPS protocol which is the secure version of HTTP protocol. HTTPS protocol is used to protect information that is in transit from client to server or vis versa. (Manico et al., 2014) Furthermore to secure the client server connection, a generated certificate was added to the project. Utilizing the keytool command line program, the certificate necessary to secure the client server portion of the project were generated and implemented into the application through the application.properties file via the method describe earlier. This adds the layer of security such that the server can be verified and trusted because of this certificate. All areas mentioned above such as API, Cryptography, client/server, and code quality were addressed in the refactoring of the code. Cryptography for instance can be seen in the file SslServerApplication on line 26→33 which implements a secure SHA-256 encryption and creates a checksum verification to be viewed in the web application. Lastly, code quality was maintained throughout the refactorization which can be verified by reviewing code and checking an updated vulnerability report which indicated no additional vulnerabilities were detected following the changes. In conclusion, to maintain security, it is vitally important to implement all practices to secure the application, this may include fixing non-secure features, adding more updated security features and lastly updating all dependency libraries.

**Reference:**

Manico, J., Detlefsen, A., & Kenan, K. (2014). *Iron-clad Java*. O'Reilly Online Learning. Retrieved July 28, 2022, from <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch01.html#ch01lev2sec3>

National Institute of Standards and Technology. (2015, August 4). *Secure hash standard (SHS)*. CSRC. Retrieved July 28, 2022, from <https://csrc.nist.gov/publications/detail/fips/180/4/final>

What is API security? (2019, January 8). Retrieved August 10, 2022, from <https://www.redhat.com/en/topics/security/api-security#:~:text=API%20owner's%20manual-,Why%20is%20API%20security%20important%3F,personal%20data%20for%20public%20consumption>.