****

# 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)-5

[2. Certificate Generation 5](#_Toc33111307)

[3. Deploy Cipher 5-6](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 6-11](#_Toc33111310)

[6. Functional Testing 11-12](#_Toc33111311)

[7. Summary 12-14](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/12/21** | **Mark Meyer** |  |

## 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

Mark Meyer

## 1. Algorithm Cipher

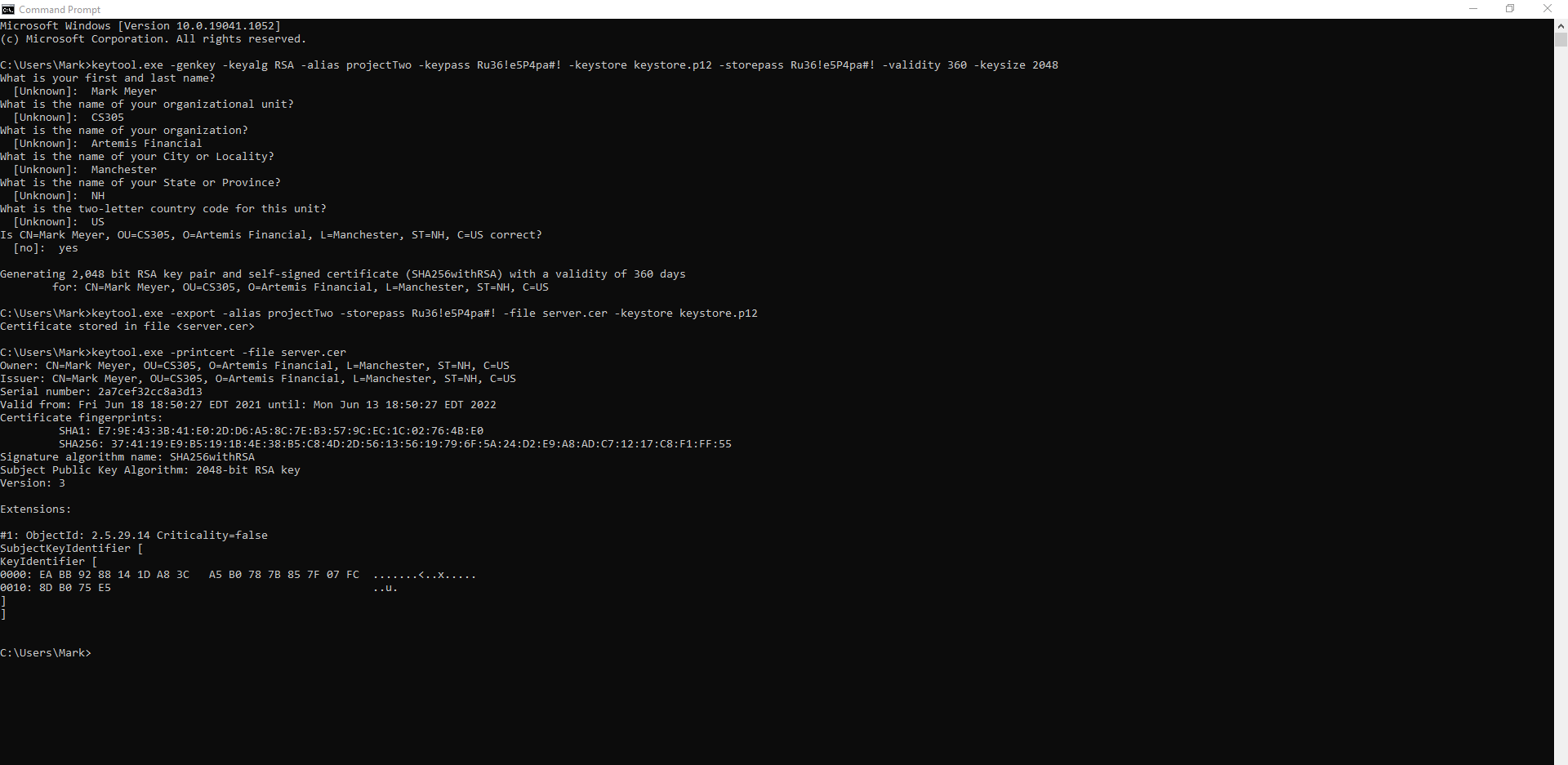
Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* 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.
* With the interest of protecting Artemis Financial’s software security vulnerabilities, I recommend using the 128-bit Advanced Encryption Standard (AES) encryption algorithm cipher with the SHA-256 hash algorithm to avoid hash collisions. The way it works is the encryption process jumbles the data into an unreadable form and assigned an encryption key—in this case, a symmetric key. This means the same key used to encrypt the file is also used to decrypt the file. Thus, the end point where the data is being transferred uses the same key to un-jumble the data back into a readable form. Furthermore, one false move while decrypting resets the key state such that any attacker would have to restart decryption from the very beginning.The AES encryption algorithm cipher is a US government security standard in protecting sensitive information. It would take hundreds of millions of years to break the encryption of files using this cipher.
* Hash functions are mathematical algorithms that take data of any size, then map (or hash) the data to an array of a fixed size. The bit level of cipher (i.e., 128-bit AES) refers to the fact that for any *n*-bit level, it would take a hacker 2n operations to break the security of that cipher. The reason you would not want a hash collision to occur is that once a hacker discovers a collision, they can obtain the hash value of that collision and use it to manifest a password that mimics the hash value of the actual password.
* High-quality random numbers are used to provide signatures for certificates, keys, authentication values, PINs, passwords, and other security mechanisms. The quality of the randomness of numbers determines the overall strength of the security mechanisms used. The difference between symmetric and non-symmetric keys is basically that a symmetric key is used to encrypt and decrypt the same file; whereas, a non-symmetric key uses separate keys to perform the encryption versus the decryption. Given the endless possibilities of key values, one can only imagine the difficulty of deciphering a high-quality randomly-generated key.
* The history of encryption algorithms is that technology is rapidly changing and so the methods of securing data transfers must always keep up with that technology. Attackers get more experience every day in finding new ways to hack systems, creating zero-day attacks, upgrading hardware and software to perform stronger attacks, and are generally educating themselves all the time on vulnerabilities. Right now, encryption algorithm ciphers provide security that some attackers cannot even fathom breaking in a million lifetimes; however, supercomputers advance at an impressive rate and, in the wrong hands, you never know what the future may bring for hackers. This is why it is so important to adopt the latest security protocols, encryption ciphers, and educate administrators and users alike on secure practices whether its being aware of shoulder surfing and social engineering or creating secure passwords and implementing strong encryption algorithm ciphers. The more layers of security you can provide at all levels, the more integrity your data will have overall.

## 2. Certificate Generation

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

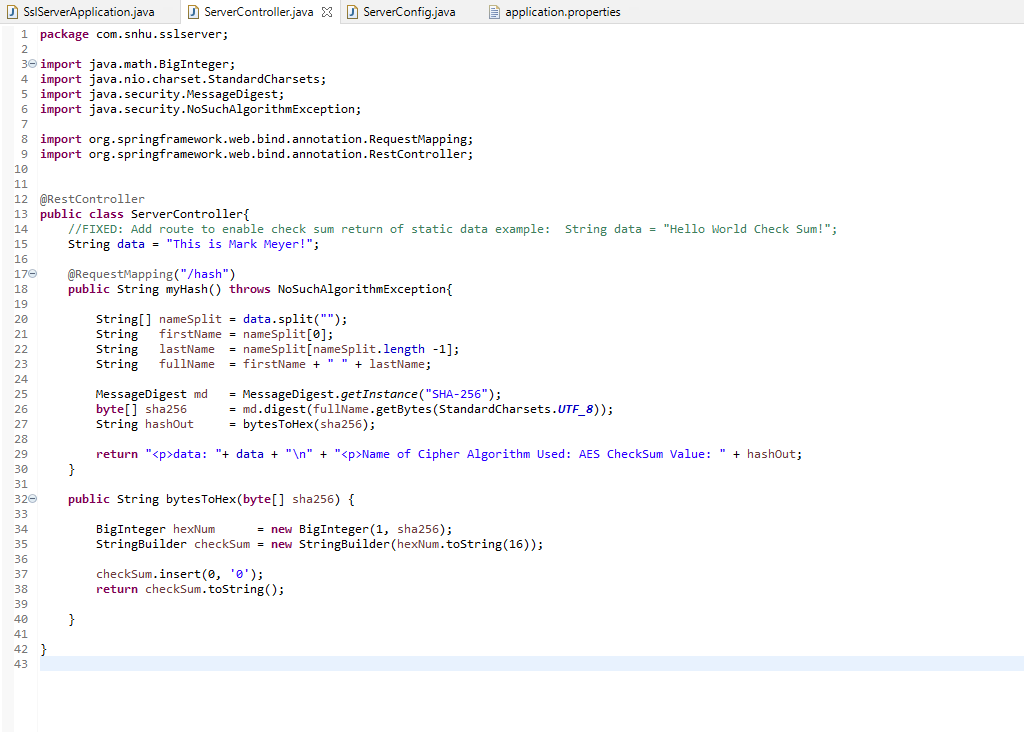
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

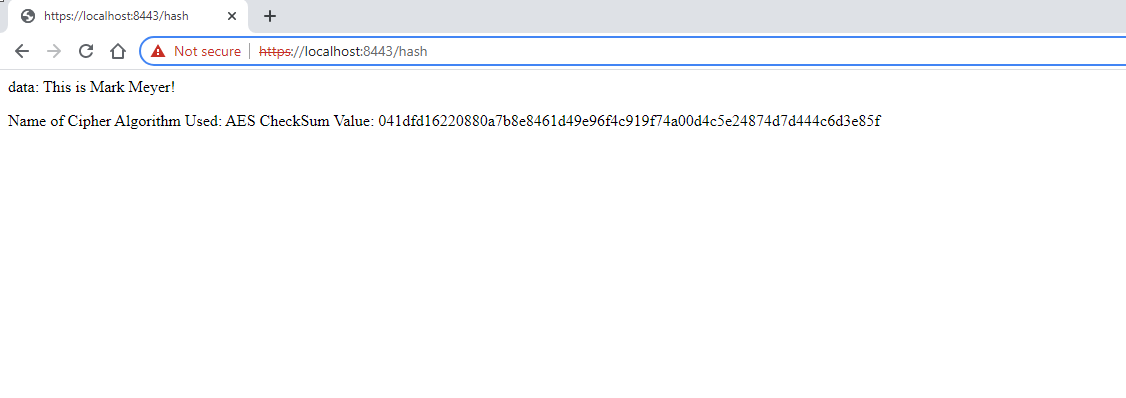
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

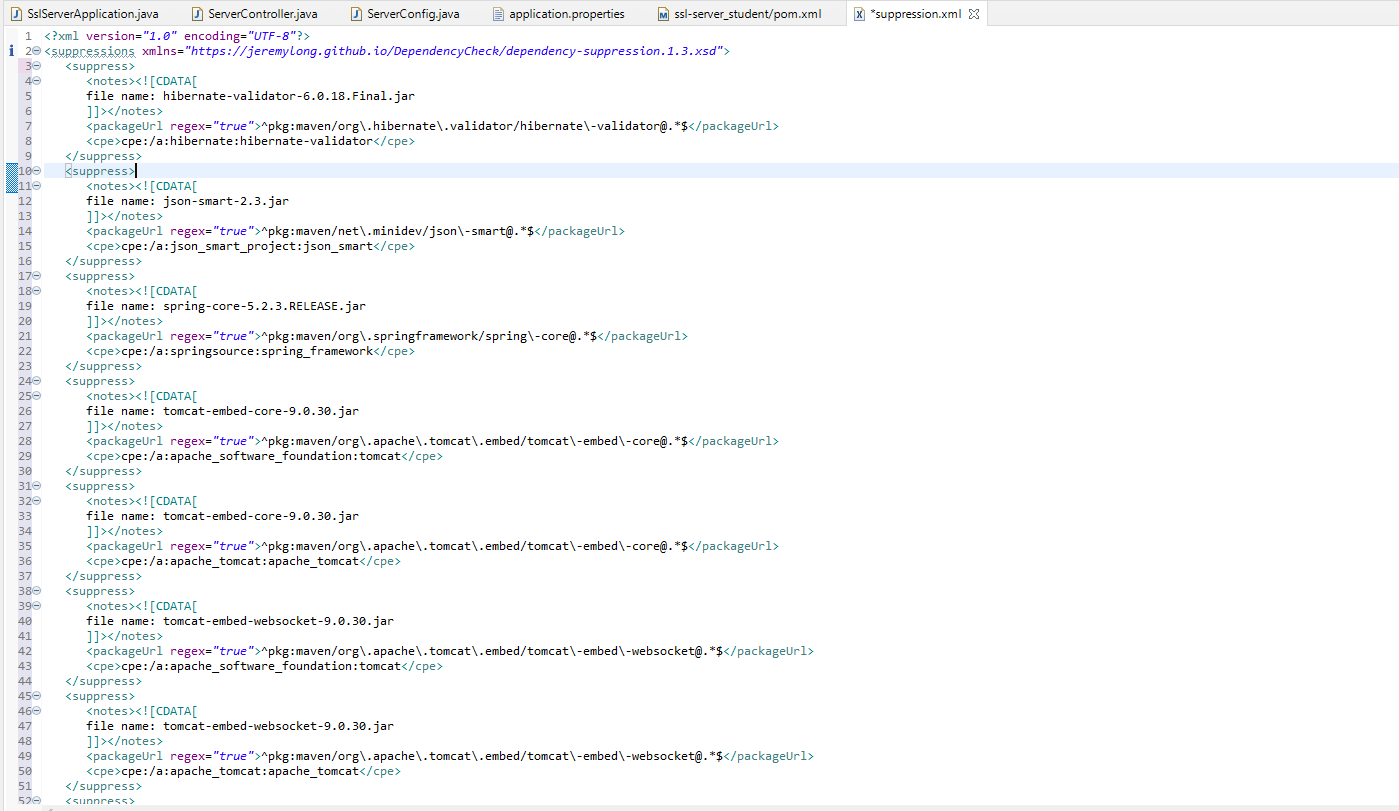
* Insert a screenshot below of the web browser that shows a secure webpage.

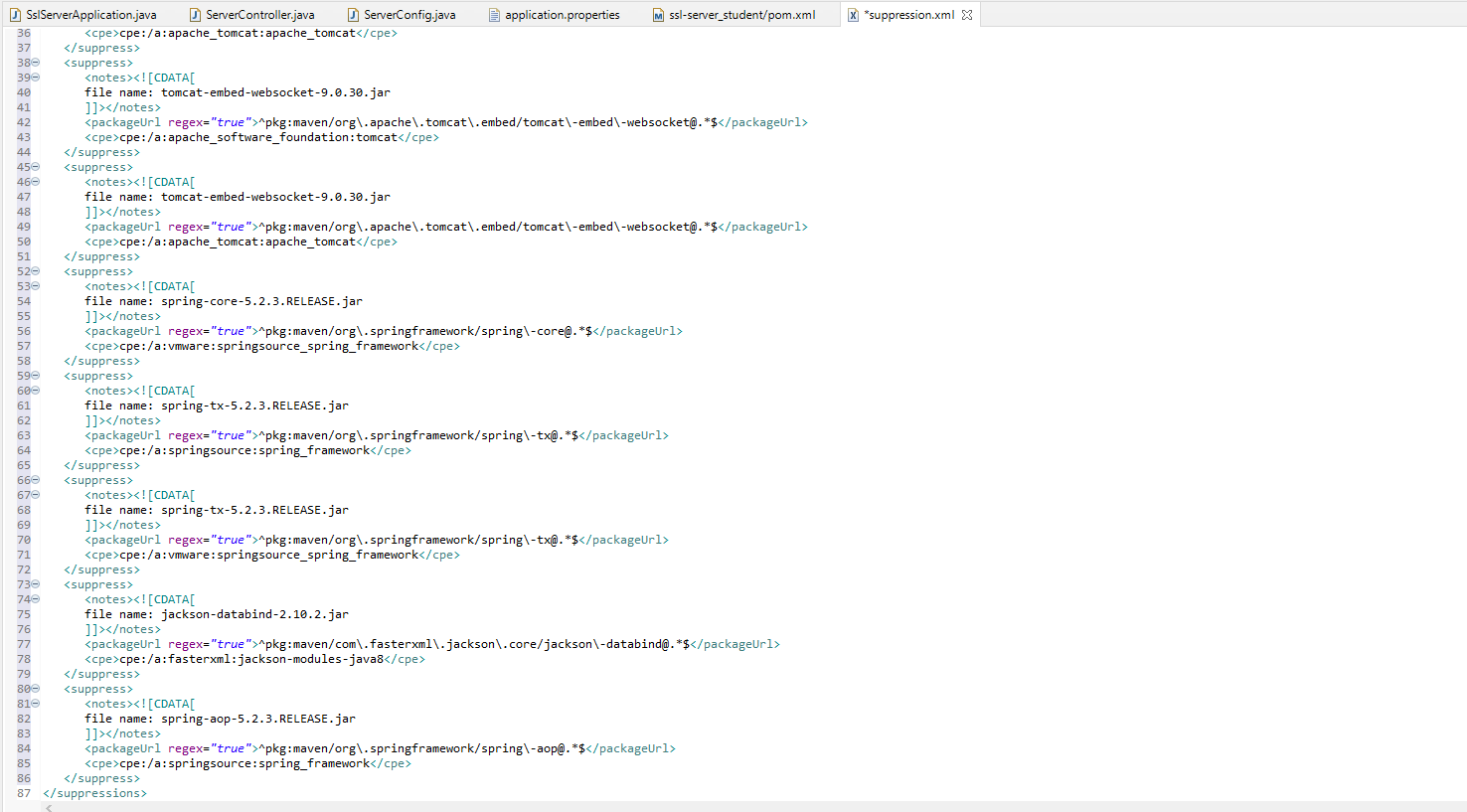


## 5. Secondary Testing

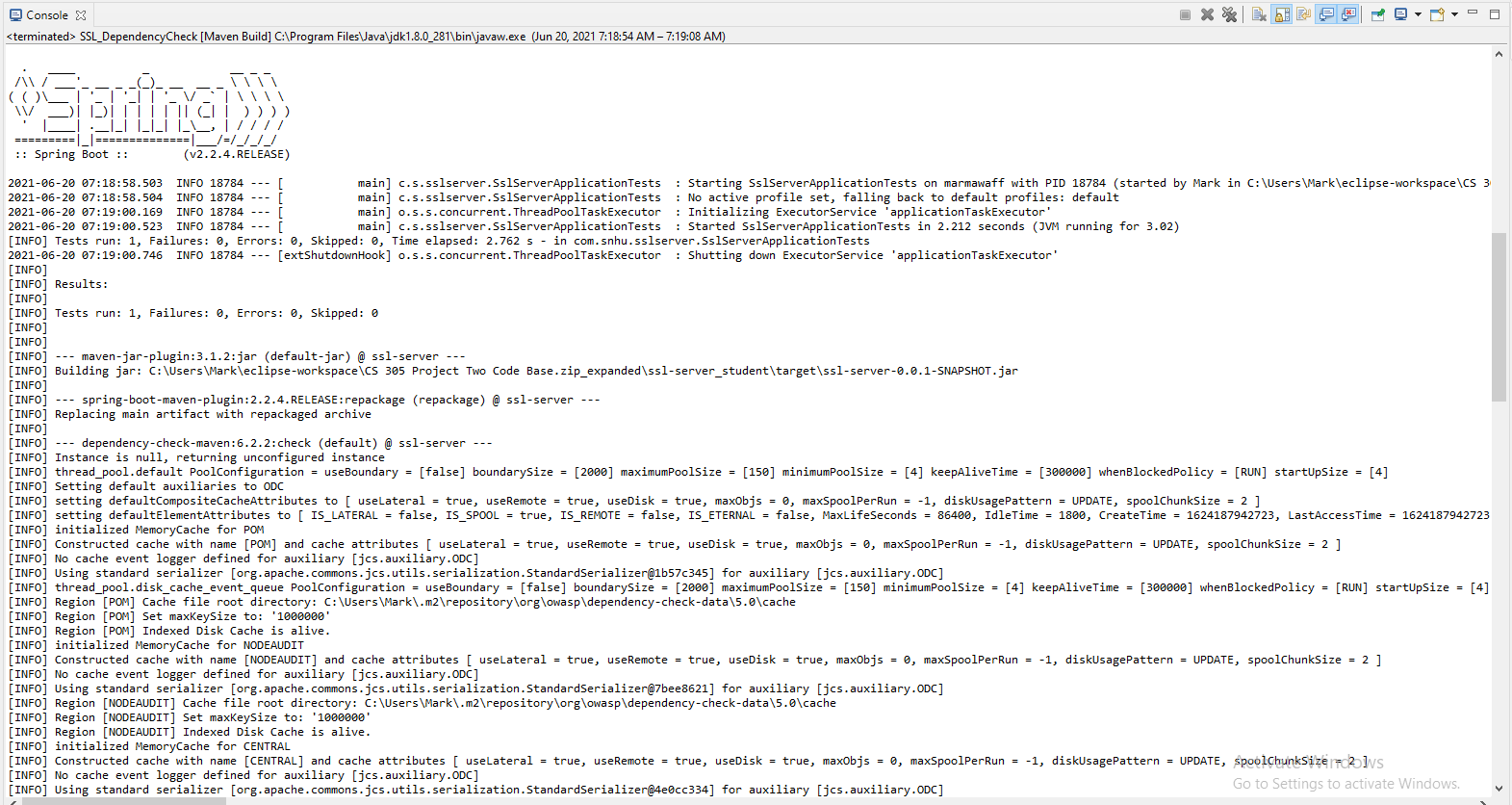
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not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

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









Graphical user interface, text, application

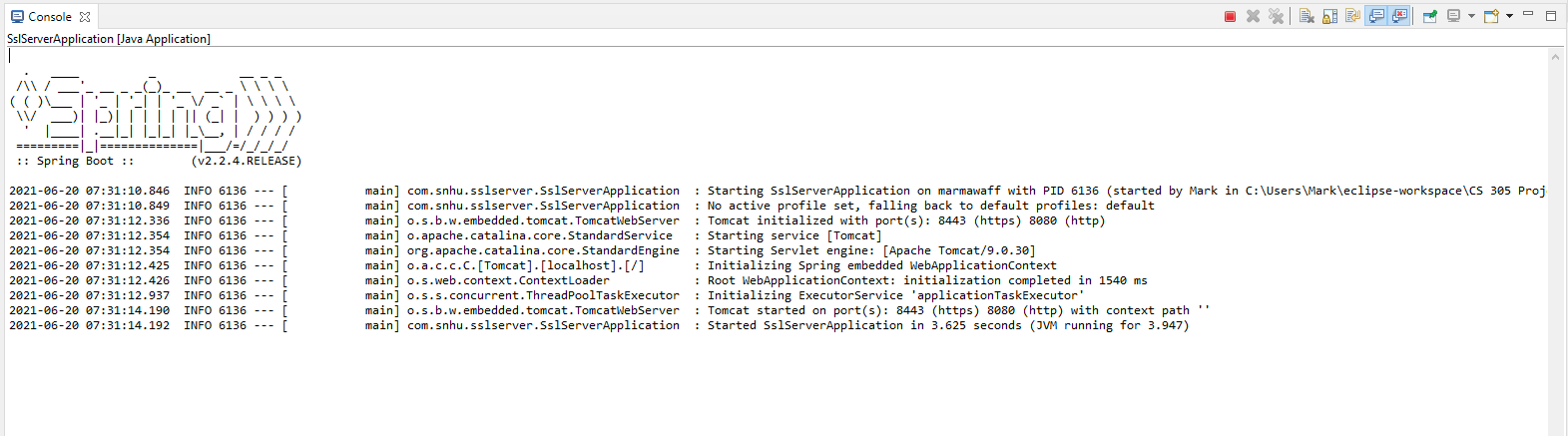
Description automatically generated

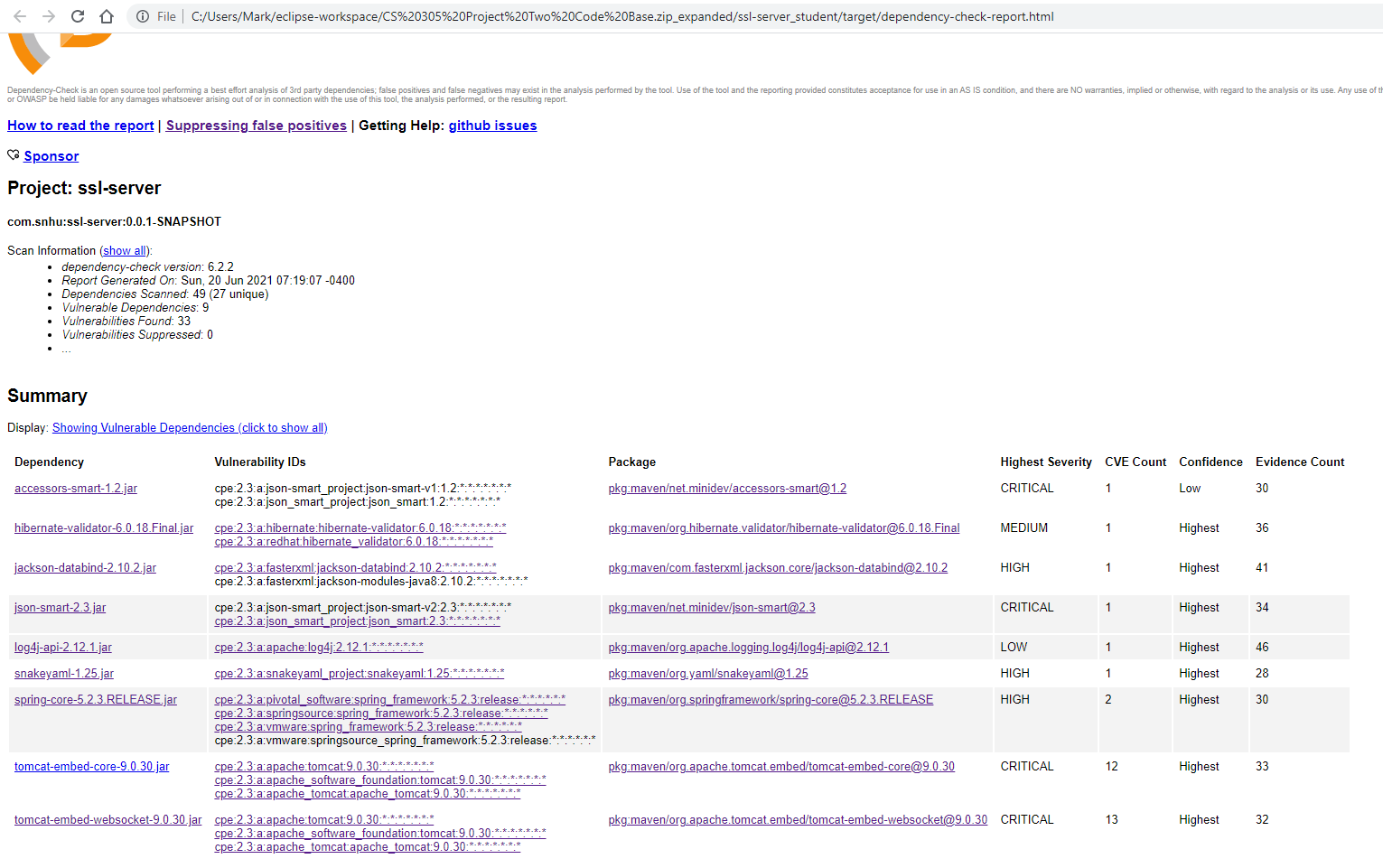
Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated



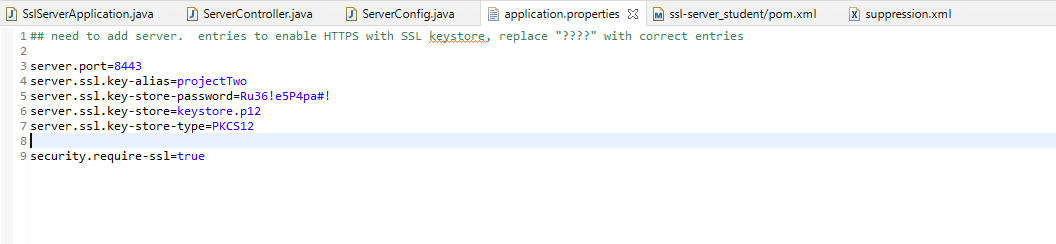


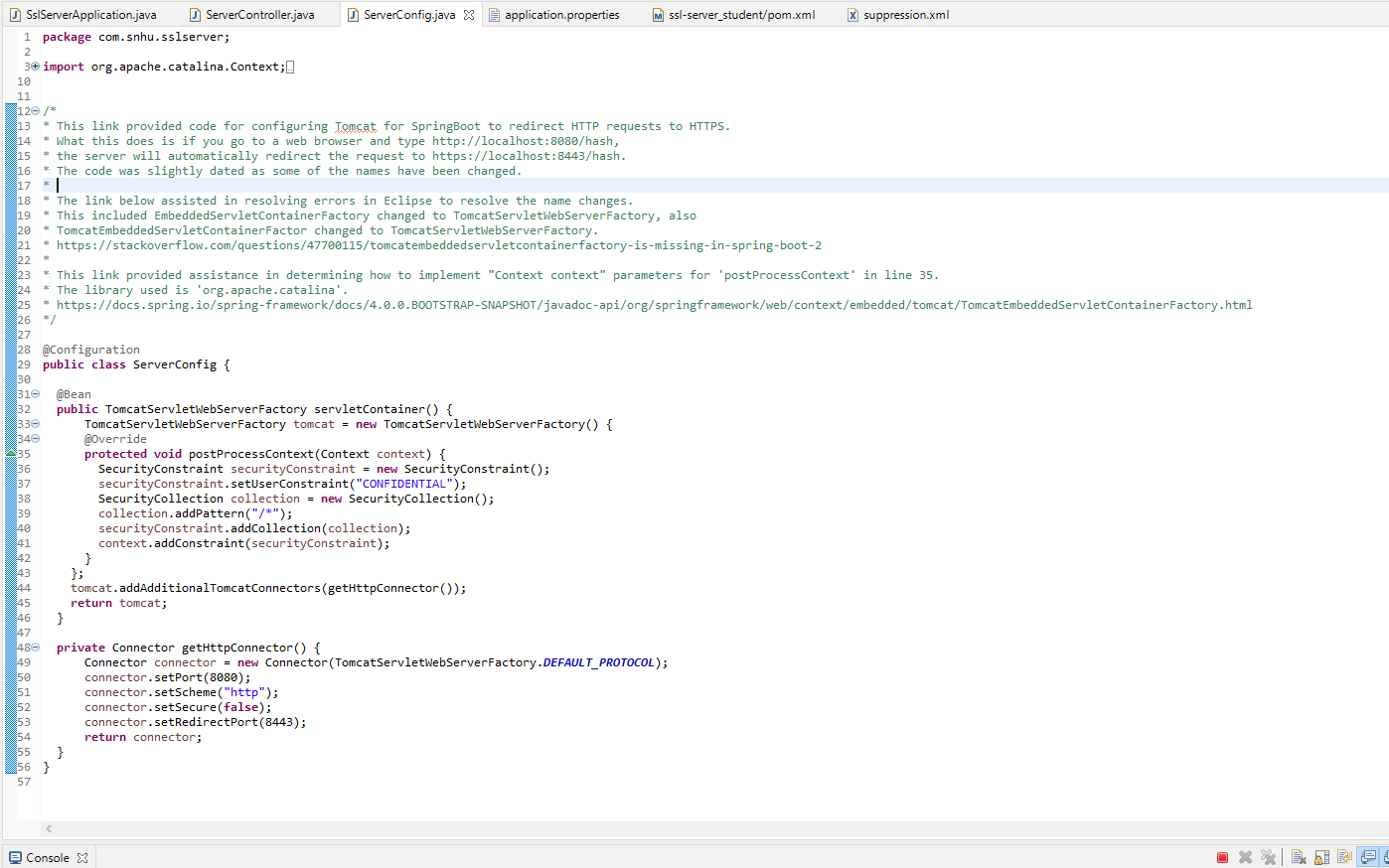
## 6. Functional Testing

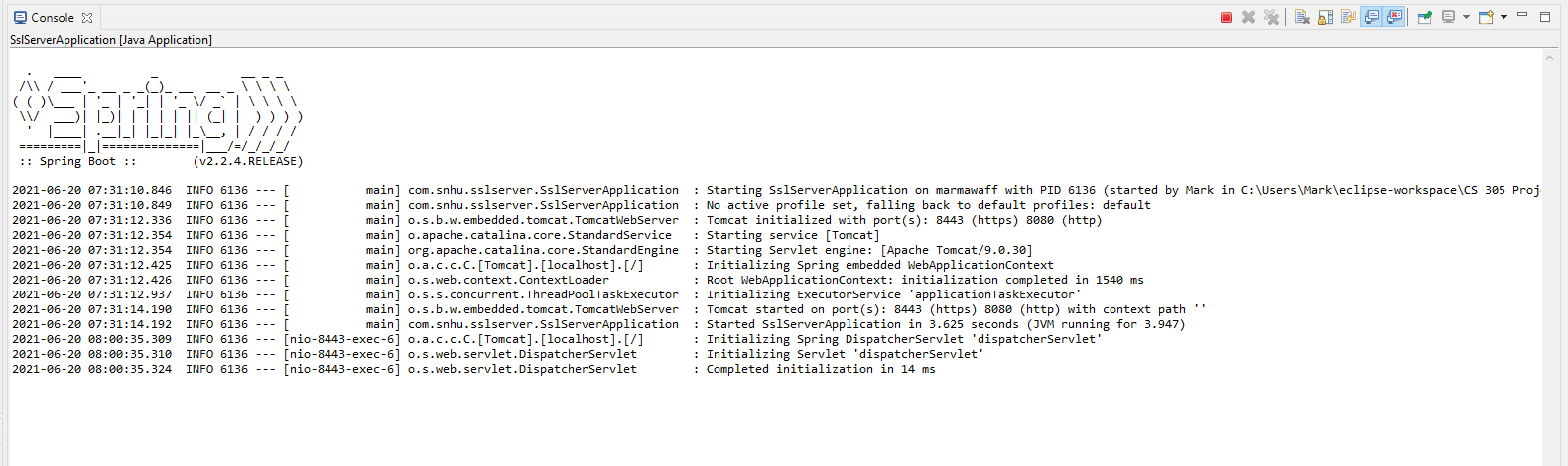
Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

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

Notice the server is accessed by ‘localhost’ successfully when the browser is opened using <https://localhost:8443/hash>. A class named “ServerConfig” was added to redirect HTTP requests to HTTPS. Thus, if you type <http://localhost:8080/hash> in a browser while the server is running, it will resolve to <https://localhost:8443/hash> automatically. In addition, the application.properties file was modified to add a secure SSL certificate to support HTTPS secure data transmissions.







## 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.

It should be noted that no suppressions were made for deployment because any and all suppressions made introduced an additional security vulnerability. Showing in the above dependency check report are 9 vulnerabilities, but the more suppressions were added, the more vulnerabilities were introduced. After reviewing the code repeatedly and suppressing a dozen CPEs, I was up to 11 vulnerabilities. Thus, I decided it was more secure to leave the dependency check unsuppressed. I’ve included a screenshot above in the “Secondary Testing” section of the suppressions I made, however these were all removed prior to code deployment. Now, referring to the Vulnerabity Assessment Process Flow Diagram, the areas of security I addressed by refactoring the code are as follows:

* **Input Validation:** added code to redirect HTTP requests to HTTPS. This will prevent user error of typing in the wrong URL, unintentional or otherwise, and becoming vulnerable to risky public data transmission.
* **APIs:** application utilizes the implementation of a RESTful API in granting access to system functionality which enables our API to be backwards compatible with legacy systems.
* **Cryptography:** application utilizes the Advanced Encryption Standard (AES) with SHA-256 hashing algorithm for secure data encryption, transmission, and decryption.
* **Client/Server:** application utilizes the RSA asymmetric encryption in the SSL certificate in implementing the HTTPS protocol, which is essentially an SSL encryption layer over an HTTP protocol. RSA uses a public key and private key in encrypting and decrypting.
* **Code Error:** the “NoSuchAlgorithmException” was added in the case that a certain algorithm is requested but not available in the environment, which could be a security vulnerability if not thrown/handled otherwise.
* **Code Quality:** standard code spacing, commenting, formatting, variable naming are all used to keep the code in such a way that is easy to read, understand, and maintain. Different functionalities are secluded to their own class to assist with this as well as promoting encapsulation.
* **Encapsulation:** It is important that critical data does not reveal weakness or vulnerabilities by leaking from the application. This is why it is good practice to utilize access identifiers such as ‘private’, ‘public’, or ‘protected’ to prevent unwanted access by unexpected data contents of a file being administered by the application.

My process for adding layers of security involved the following:

* **Added a file verification step by implement a checksum verification. This** takes a data string, for example, and encrypts the data with a private key into a jumbled, unreadable form of “ciphertext”. Then, the receiver of the data must use the exact same private key to decipher the text into a human-readable format.
* **Implemented the HTTPS protocol.** An SSL certificate was generated and implemented in the application by adding the key to the project and modifying the application.properties file to allow the server to perform the encryption/decryption.
* **Ran a dependency check to identify security vulnerabilities.** The dependency report was generated and the CPEs were analyzed to identify security vulnerabilities to try and mitigate any unwanted dependency reporting. When suppressions were added, additional vulnerabilities were introduced, so the suppressions were removed to minimize vulnerabilities.
* **Implemented redirect of HTTP requests to HTTPS protocol.** This enables the application to implement the secure HTTPS protocol whether trying to access the application securely or not. The server explicitly redirects all HTTP attempts to HTTPS.

The best practices for maintaining the current security of the software application is to adopt an overall awareness of security vulnerabilities—from the physical level all the way down to the development level—and educate those invested in security implementation as well as users as to how to be vigilant in identifying and mitigating risky behavior. You can only do so much with code if you have a mole on the inside feeding vulnerabilities to an attacker. Therefore, it is very important to also look at things such as perimeter security of data centers, biometrics used in system entry, addressing shoulder surfing and social engineering and tailgating, educating employees on running antivirus and malware scans on their systems, using strong passwords, and avoiding stepping away from their computer without at least some sort of password-protected screensaver. As you can see security is not just addressed in the development environment, but rather, it is an attitude and desirable approach to encompass all your investments and assets with proper guidance and protection.