****

# 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** | **April 17, 2021** | **Shawn Whittaker** |  |

## 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

Shawn Whittaker

## 1. Algorithm Cipher

Algorithms are sets of instructions to follow for performing a procedure. A cipher algorithm is a set of instructions for a procedure to encrypt and decrypt data. These cipher algorithms act as a secret key used to encrypt the data, and then is sent along side with that encrypted data so the next machine that needs to read it can decrypt the information safely. This allows safe transit for the data but allows ease of use for those encrypting as well as those decrypting.

Brute force attacks use trial and error to guess login information, encryption keys, and even hidden web pages. Thinking of a simple 3-digit lock containing numbers 0 through 9, there are only 1,000 possible solutions. Although it might be time consuming to try each combination up to 1,000, if you had a program that could try 20 combinations a second, it would only take 50 seconds tops to crack the combination.

The AES encryption algorithm is the newest standard, as well as government regulation, for financial/banking transaction systems. AES offers a 128-bit and a 256-bit encryption key, which is its main benefit. The time required to crack an encryption algorithm is directly proportionate to the length of the key, as discusses previous. The previous banking standard, DES, only offered a 56-bit key so we can see why the offering of 128-bit or a 256-bit key is vital. AES is a symmetric cipher, which means the same key used to encrypt the data is used to decrypt the data. This makes the encryption easy to use, while remaining a very secretive. An asymmetric cipher uses one key to encrypt the data, and then two keys are used to decrypt the data, a public key and a private key.

AES is secure to the point hackers do not try to crack the algorithms, so the major risk of the encryption comes from side-channel attacks. Side-channel attacks gain information through the weakness of the implementation of the algorithm. These side-channel attacks collect information about what actions the computing device is performing while using the encryption algorithm to reverse-engineer the key.

The AES algorithm is widely used to protect data at rest, that is, data that is being stored on hard drives or servers. There is much financial information that must be stored, which makes AES a good fit for a storage algorithm.

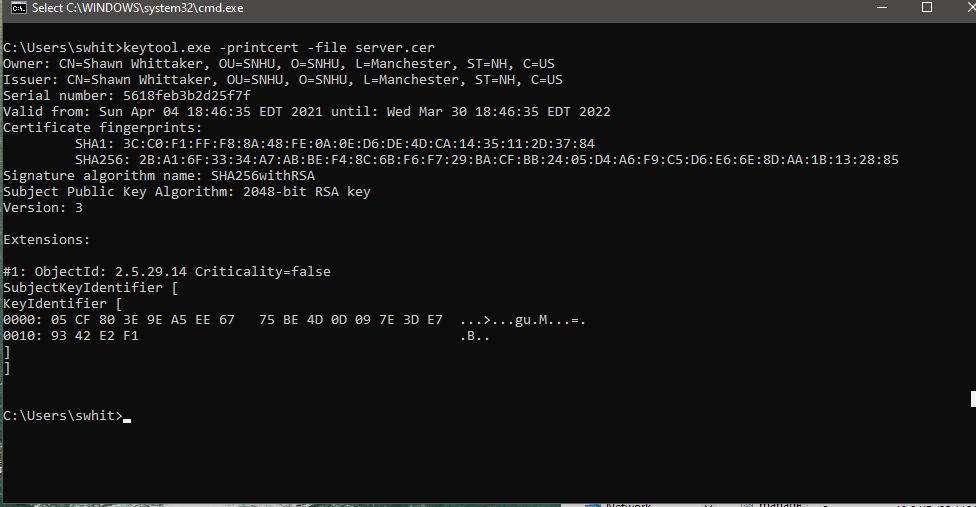
One reason the not choose the AES or the less-secure DES is for backwards compatibility. It can be a large task to change a large system to a different encryption system, so having DES implemented at least partially can allow backwards compatibility allowing the system to access older files and documents that may not be compatible with the new AES encryption.

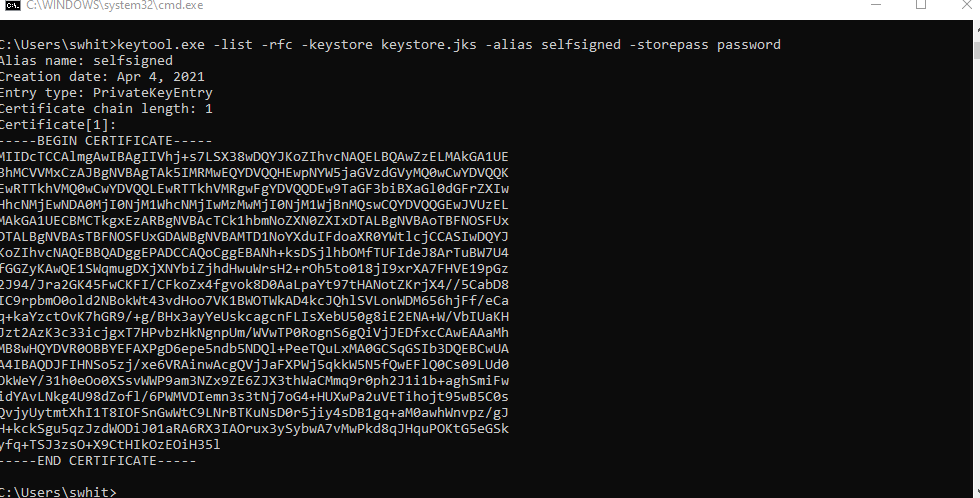
Hash functions map data of any length to a fixed-length output using an algorithm. The purpose of this data mapping is to allow authentication, allowing senders and recipients to ensure the data has not been altered. Bit levels are vital to encryption as, discussed earlier, the higher the bit, the harder it becomes to crack as the number of possibilities exponentiates.

Random numbers are required in cryptography to allow key generation; the key that is used to encrypt data is made of random numbers. Each data unit is given a random number or special character, so if by chance a hacker accesses the data, all they have is a bunch of jumbled up text.

## 2. Certificate Generation

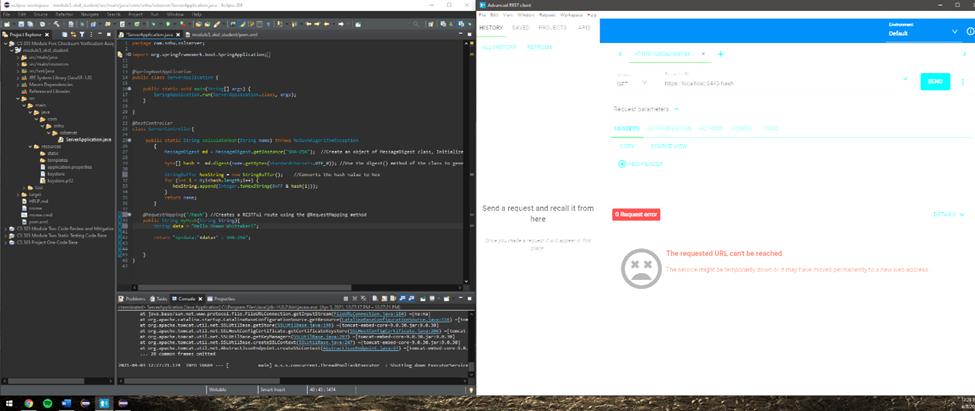
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.





## 3. Deploy Cipher

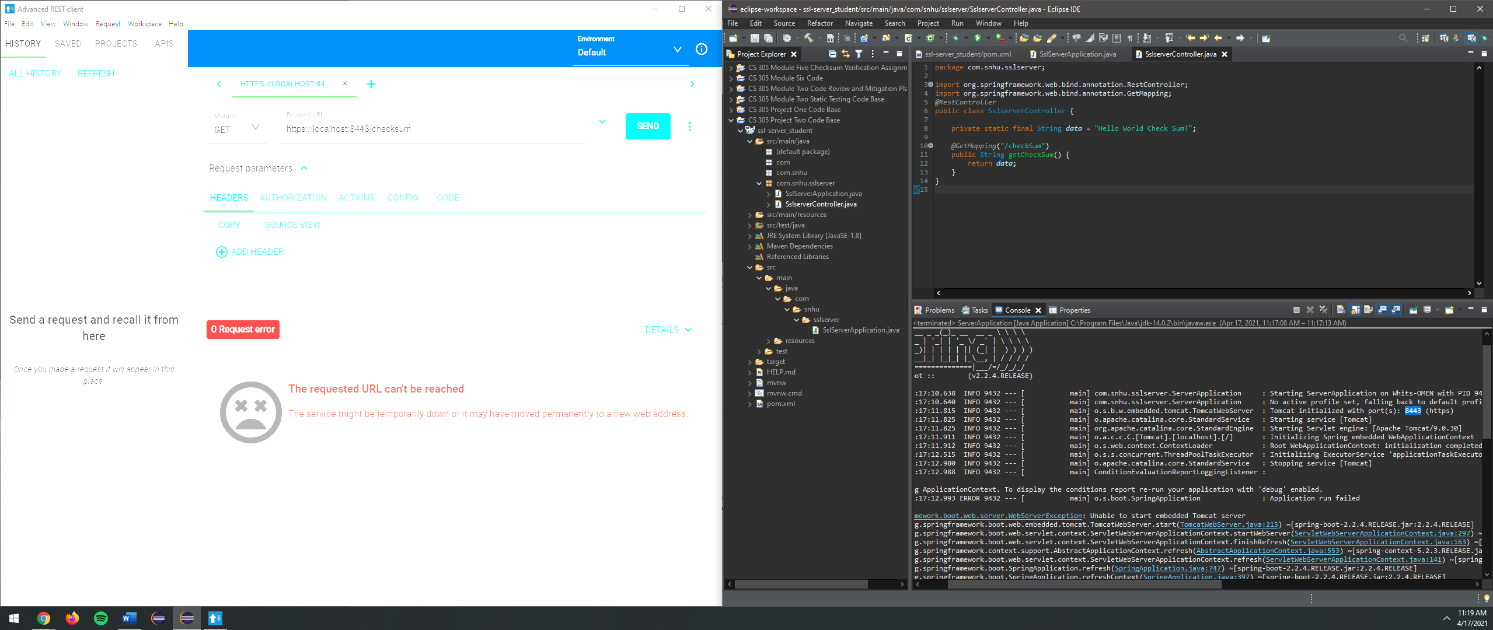
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.



\*I was never able to successfully deploy the cipher.

## 4. Secure Communications

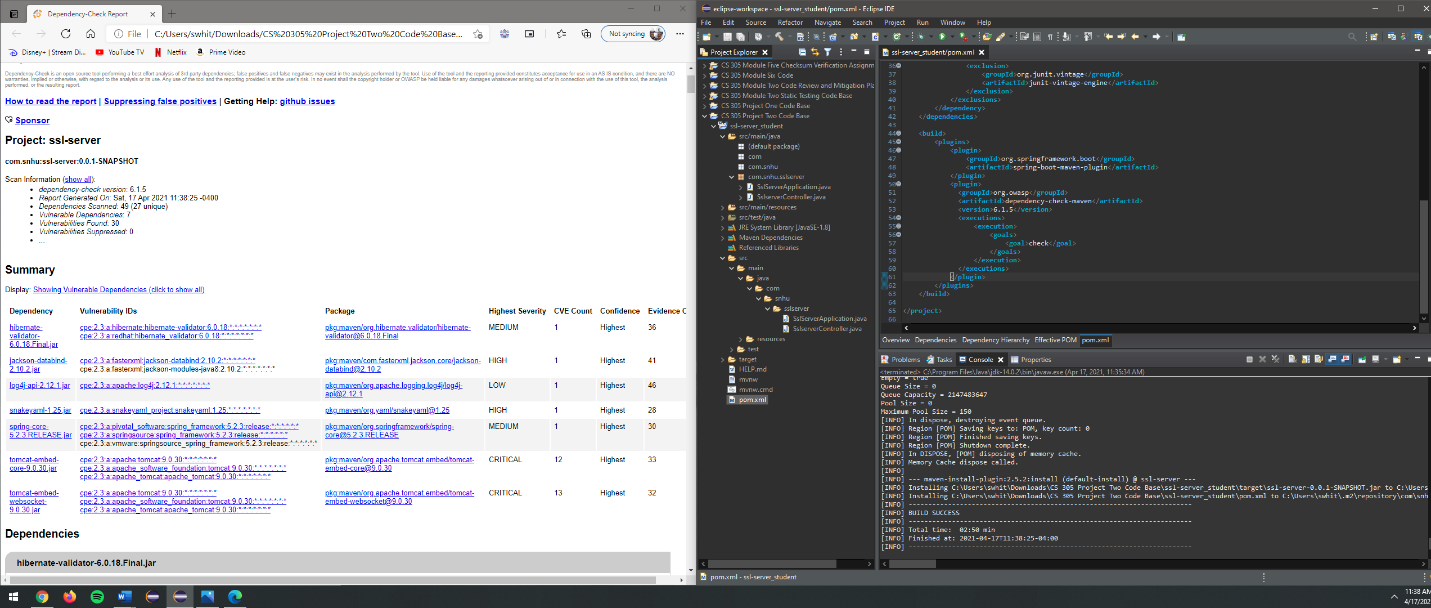
Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.



\*Again, could not successfully deploy the cipher, but added screenshot of refactored code.

## 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.

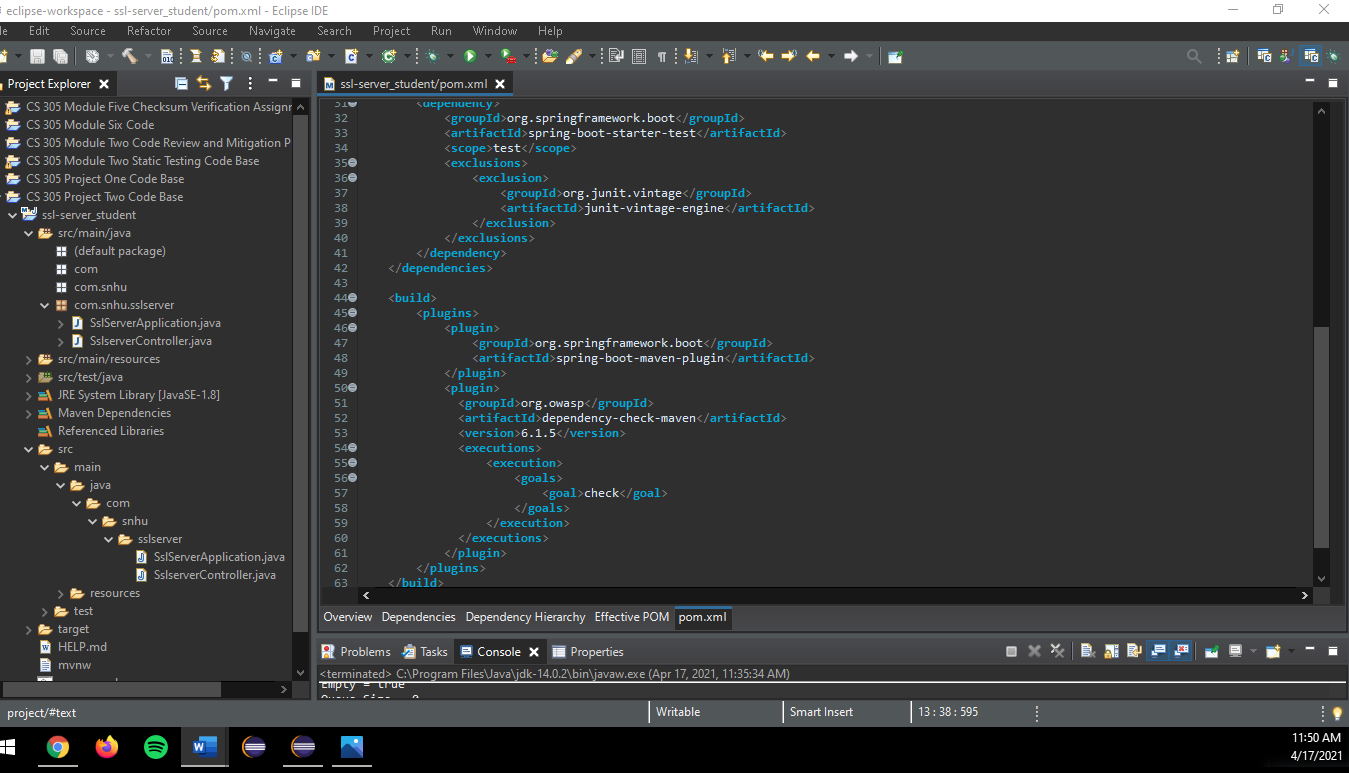


## 6. Functional Testing

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

In our pom.xml, we have a dependency on springframework.boot. The version we are dependent on is 2.2.4, which is out of date. The current live version is 2.4.5. We should upgrade to the latest version since the latest version of the dependency could have many security updates since 2.2.4.

Also, in the pom.xml, the maven dependency check was using an outdated version, which we have switched to 6.1.5, as many dependencies have become available in the timeframe between the updates.



## 7. Summary

We have refactored the code to implement the updated versions of the springframework.boot, as well as the maven plug in. Keeping the plug ins up to date is one of the best practices for maintaining the current security of any software application.

We also implemented algorithm ciphers, in this case the SHA-256 cipher, that can encrypt the data using 2256 characters, which could take thousands of years to brutal force guess the encryption.

We generated a certificate, which can be authenticated by a third-party service which can then the certificate authentication can tell other machines to trust my certificate and program.

We \*should\* have been able to deploy the cipher and use the checksum to add a secure HTTPS to our program, which would allow secure online access to our financial information. This would allow Artemis Financials’ customers to view their banking information securely online.

We then used secondary testing to run the maven dependency check to ensure the plug-ins and third-party dependencies we’ll be using are safe to use. Using the most up to date version of this dependency checker ensures we are getting the most secure data we can.

Using the Vulnerability Assessment Process Flow Diagram, we did the architecture review during the Functional testing review, where we manually reviewed the code for syntactical, logical, and security vulnerabilities. We used secure APIs using the RESTful API to deploy the cipher and enable secure communications. We used cryptography in our algorithm cipher. We also ensured Code Error handling and Code Quality were checked during out functional testing review.