****

# 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 5](#_Toc33111306)

[2. Certificate Generation 5](#_Toc33111307)

[3. Deploy Cipher 6](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 6](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 8](#_Toc33111312)

8.Works Cited………………………………………………………………………………………………………………………………………10

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **04/12/2021** | **Craig Harrigan** | **Initial Draft** |

## 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

Craig Harrigan

## 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.

My recommendation for Artemis Financial is to adopt the SHA\_512bit encryption cipher. The government regulated minimum for AES is 128bit (Probasco, 2017). However, the integrated Java tool for cryptography is natively capable (and mostly used with) SHA. It is for this reason that we recommend stepping away from the usual AES encryption for this use case, to save overall time and cost while still maintaining the highest level of security.

Some things we need to ensure we’re protected against is the communication eavesdropping, impersonation, password cracking/compromise, and data extraction (Manico, 2014). For communication, TLS should be instituted for all communication lines and web portal connections. This is not difficult to implement with the Java Secure Sockets Extension (JSSE) and is worth having for a financial institution (Manico, 2014). It is also necessary to make sure our data encryption keys are not compromised.

This encryption needs to be implemented at the application level, giving us control over how the data is encrypted and decrypted. This is widely considered the most secure way to protect the data so this is what should be used for a financial institution (Probasco, 2017). Going with the 512 bit SHA is a strong compromise between great security and reasonable cost, making it an ideal solution vs instating overkill, massive levels of encryption.

Use of symmetric vs non-symmetric keys is very simple – it comes down to use case. The symmetric cryptography has the same key for both encrypting and decrypting, thus being useful within a secured system. Non-symmetric keys have two keys – one public and one private. This is used for an unsecured system like a communication channel (Manico, 2014).

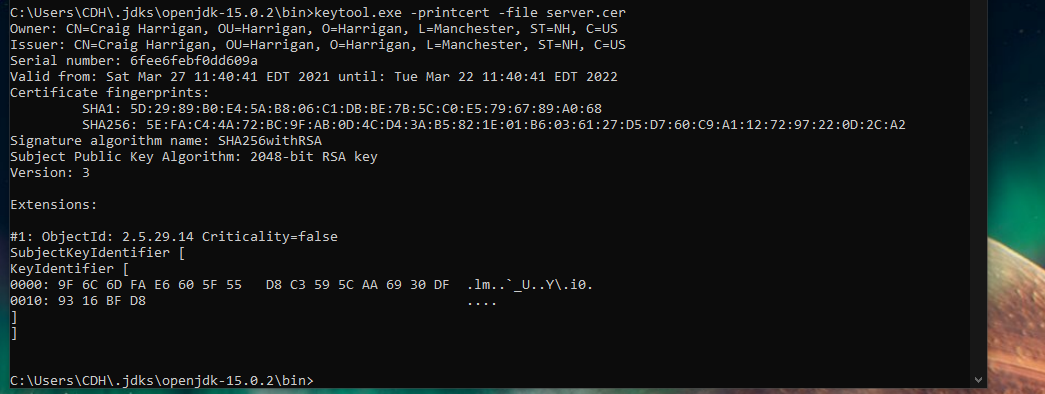
Specifically when discussing public keys, hash functions are used to help the client machines “trust” the key. They will output a bit level prescribed by the key hasher, and a secure enough bit level cipher should always be used – such as SHA-256 (Manico, 2014).

Encryption protocols to avoid include RC4, DES, and 3DES, so those weren’t ever considered. 128bit minimum key size for AES (Manico, 2014), which also happens to be the minimum government standard for the financial industry as mentioned above. The use of RSA encryption for the data (also a common standard) is significantly slower and more computationally heavy than using AES (Manico, 2014). Therefore, AES emerges as the best possible choice for this use-case.

## 2. Certificate Generation

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

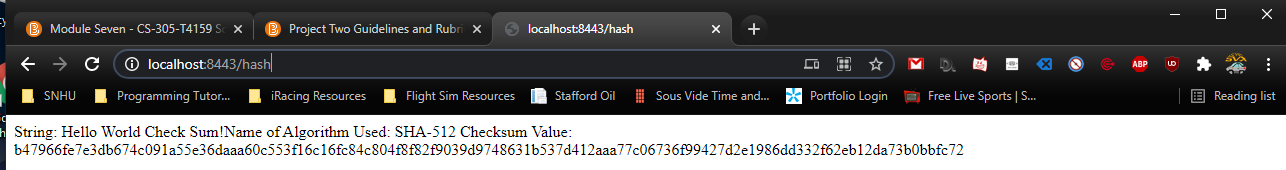
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

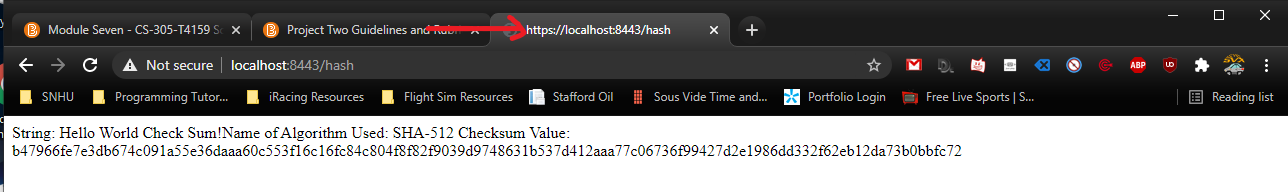
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

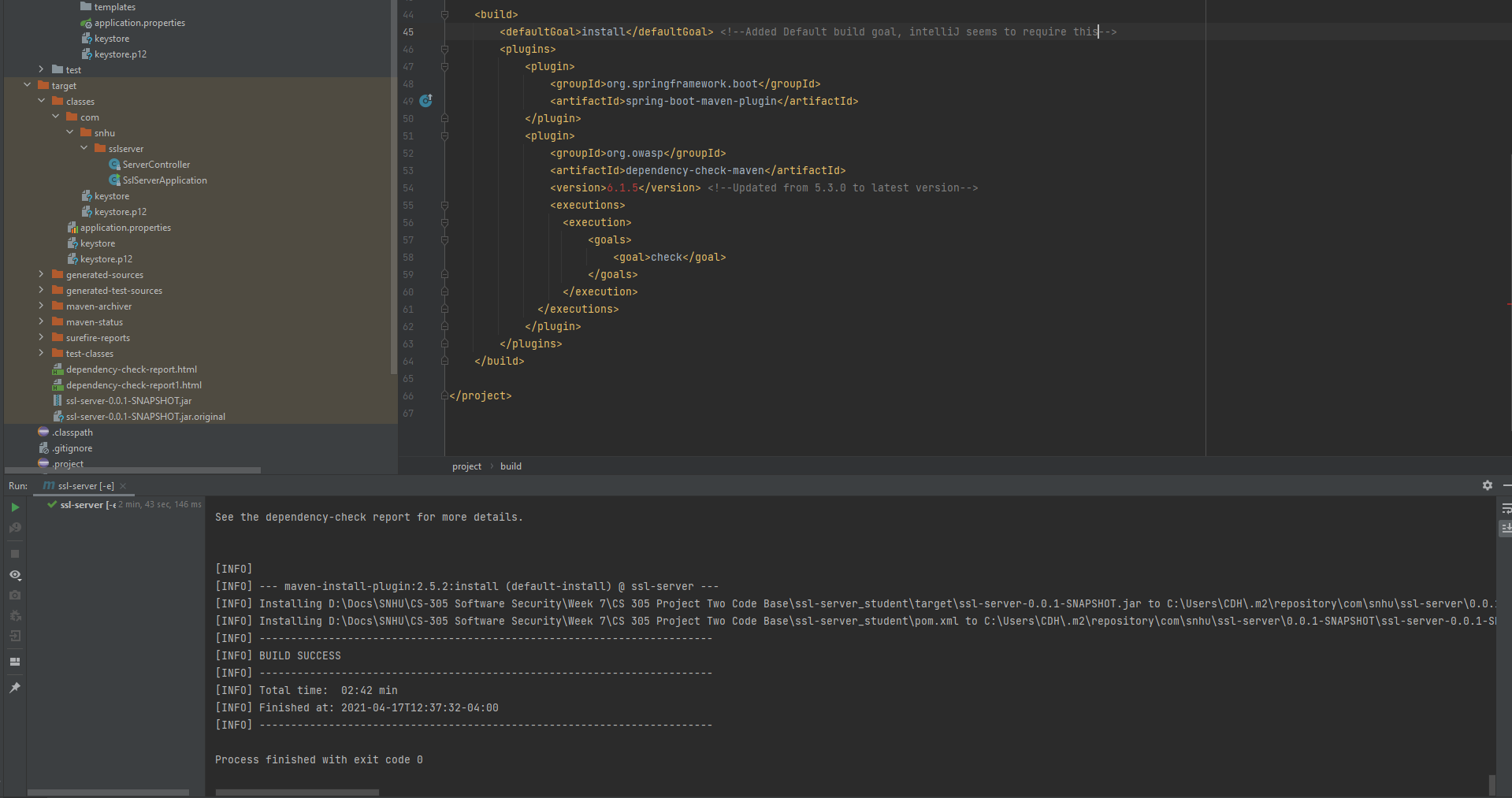
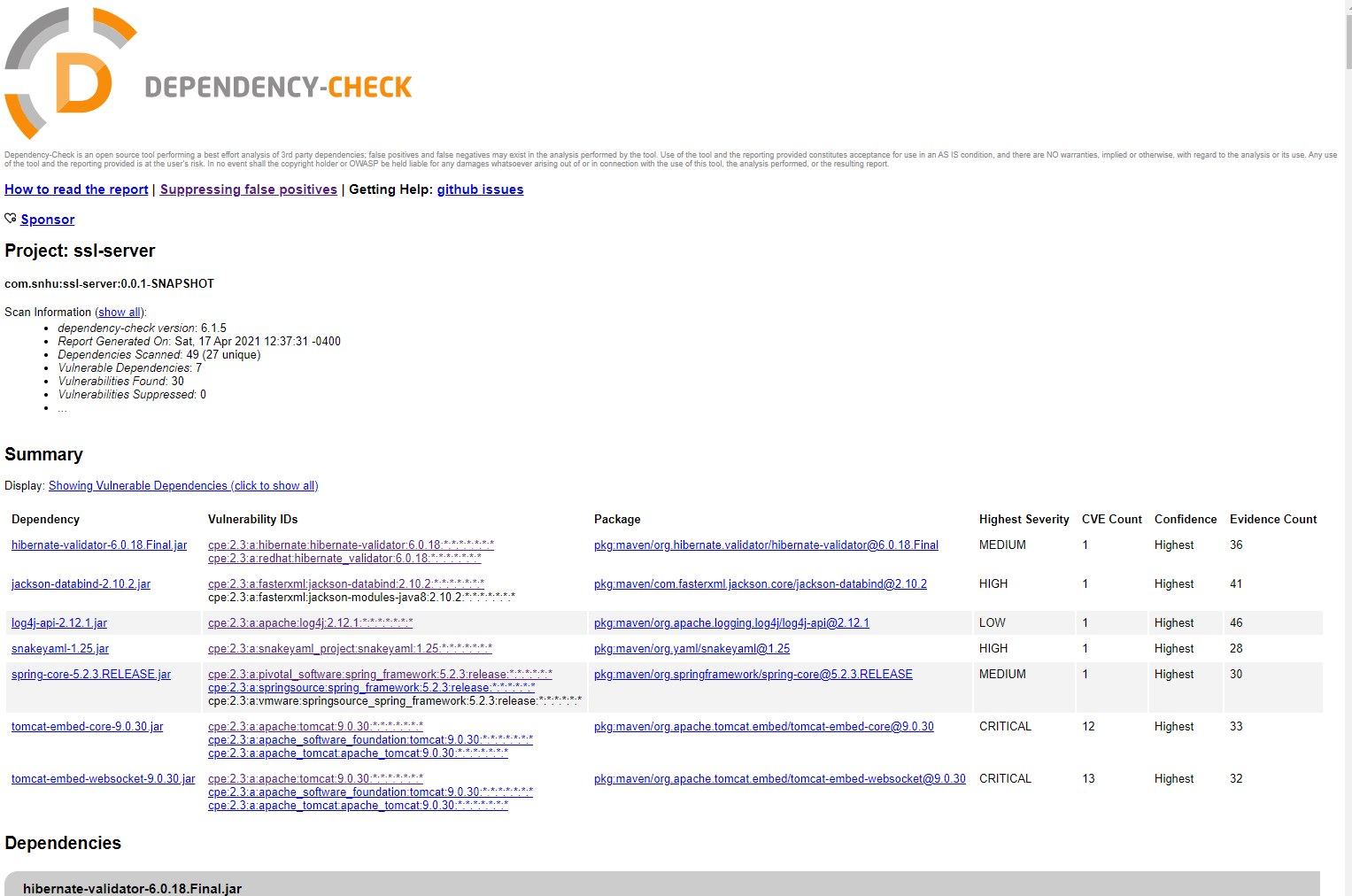


## 

## 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.

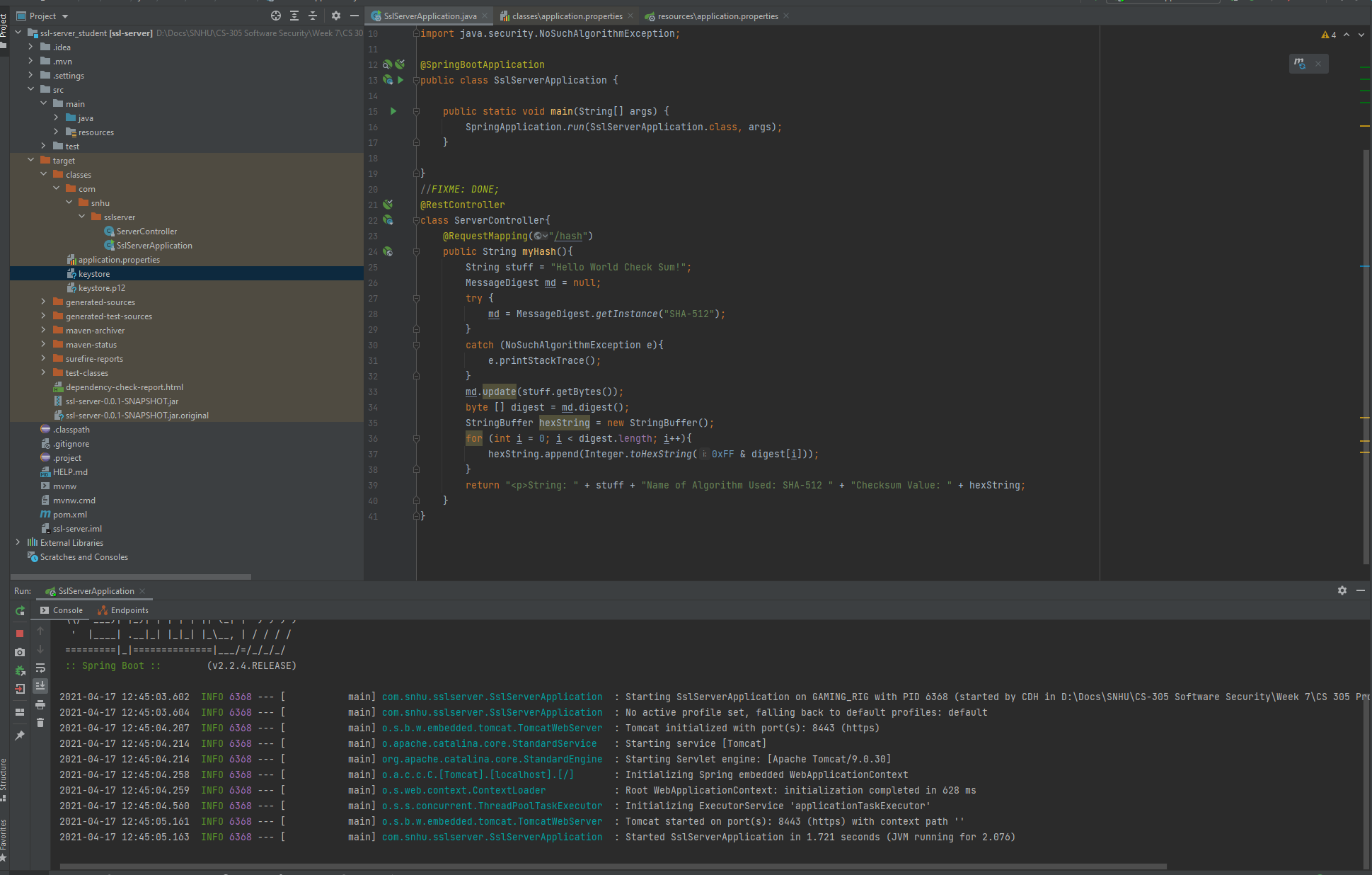
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



## 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.

Within the refactoring of this codebase, the areas of the Vulnerability Assessment Process Flow that were addressed included the Cryptography, Client/Server, Code Error and Code Quality points. First, cryptography needed to be added to the ServerController. This implementation had to be mapped to the client/server connection by updating the properties of the connection to include the SSL keystore making it an HTTPS connection. Code Errors needed to be utilized in this implementation through a “try/catch loop” and Code Quality is always a point to keep in mind when writing new code.

Adding these layers of security to this application bringing untold value to it. Within the banking industry, this code was previously unusable as it not only wasn’t secure but also didn’t meet government regulations. With the changes made however, this code is now deployable to the field following sufficient QA testing. This security will also measurably save the well-being of the company in the coming future, knowing that they’ve secured against the inevitable attacks that will be coming their way once this is deployed out to the public.

The current security level and measures are only as effective those using it from day-to-day. Going forward, the practice of least privilege needs to be upheld not just among the public but within their organization. That means the IT technician that needs to be able to reset users’ passwords *does not* need nor should have access to deeper systems and the source code.

For full details on the changes made beyond the screenshots provided, please see the attached source code: [Project Two Code Base Refactored.zip](CS%20305%20Project%20Two%20Code%20Base%20Refactored.zip) (Note: you will need your device configured with a Java IDE as well as the JDK)

**Works Cited**

Manico, Jim. (2014). Iron-Clad Java. Oracle Press. Source: https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/

Probasco, L. (2017, April 25). Encryption requirements for banks & financial services. Retrieved March 18, 2021, from https://info.townsendsecurity.com/encryption-requirements-for-banks-financial-services

Unknown, U. (2015). Secure Hash Standard (SHS) (pp. 1-36, Rep. No. FIPS PUB 180-4). Gaithersburg, MD: Information Technology Laboratory. doi:https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf