****

# 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** | **13 Aug 21** | **Matthew Berdecia** | **Security Report** |

## 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

Matthew Berdecia

## 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.

As computer technology proliferates nearly every aspect of our lives, ensuring that the data that is transferred around the web is secure. And there are many cipher algorithms to choose from when creating a checksum value string, though currently the best overall security algorithm is the Advanced Encryption Standard (AES) cipher, it supports of a large variety of key sizes and is one of the best security standards in place today, being used by large Fortune 500 companies and Governments alike. The AES algorithm is largely considered impervious to all attacks, except for brute force and is highly efficient in 128-bit form. However, AES also can use keys of 192-bit and 256-bit for superior encryption purposes. with AES 256 your encryption key is 2 to the power of 256 which equals 1.1579209e+77 or in long form it is written out as a very large and almost incomprehensible number: 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936. This number represents the number of possible keys for AES 256 crypto which means it for all intents and purposes unhackable. Hash functions, from a crypto standpoint, are a unique identifier for any given piece of content, meaning that it ensures that all data sent reaches the intended recipient completely intact just the way you sent it. It also is a process that takes plaintext data of any size and, through a mathematical algorithm, converts the text it into a ciphertext of a set length.

Secure Hash Algorithms (SHA) gives us a flexible one-way crypto algorithm that is unique and is a fixed length, regardless of the length of the text. So, a SHA function takes a document and computes it into a "hash" of the input, though it is important to realize that this is a one-way process, so a hacker can't take the hash and recover the original document. So, using AES-256 encryption with SHA-512 Hash for data integrity makes your documents nearly unhackable. Symmetric encryption uses the same secret key to encrypt and decrypt the sensitive information whereas asymmetric encryption, uses mathematically linked public- and private-key pairs to encrypt and decrypt senders' and recipients' sensitive data. With RSA (Rivest–Shamir–Adleman) encryption, private keys and public keys are mathematically related. And should the key to any of the encrypted data be lost, the data is practically lost as well since there will be no way to decrypt that data without the key. With random numbers generators, use a sequence of numbers are used to gain proper access to web site or servers in tandem with login information, encrypt data and de-crypt the data when the correct sequence is used. The sequence is generated by a token which is generated from a seed number that is unique to the individual’s token. The number, along with the elapsed time to generate, are needed in order to generate the private-public key pair. Thus, making it very secure, as any hacker would need to know the sequence of random numbers, the seed number and algorithm used to create the number and the length of time to generate the sequence, to generate the proper key pair in order to find the public key and private key.

The history of cryptography has its roots going back to 600BC, up through the Roman Empire where Julius Caesar would use a plane text substitution cipher, letters in the alphabet are replaced with other letters. In World War 2, German engineer Arthur Scherbius’s Engima Machine which had disks that rotated at different speeds while typing, making the output letters into another substitution cipher, the key was generated in the initial setting of the spinning disks. But modern computer cryptography had its start in the early 1970’s at IBM. The “crypto group” at IBM created the Data Encryption Standard (DES) in 1973 which was used up until 1997, when it was hacked. In 2000 Advanced Encryption Standard (AES) replaced DES as the standard for encryption, and is used worldwide and is even approved for use to encrypt classified government data. And the newest encryption Elliptic Curve Cryptography (ECC) created in 2005. Is an advanced public-key cryptography scheme and allows shorter encryption keys by generating security key pairs for public key encryption by using the mathematics of elliptic curves vice more traditional linear mathematical equations.

## 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.

[Insert screenshot(s) here.]

## 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.

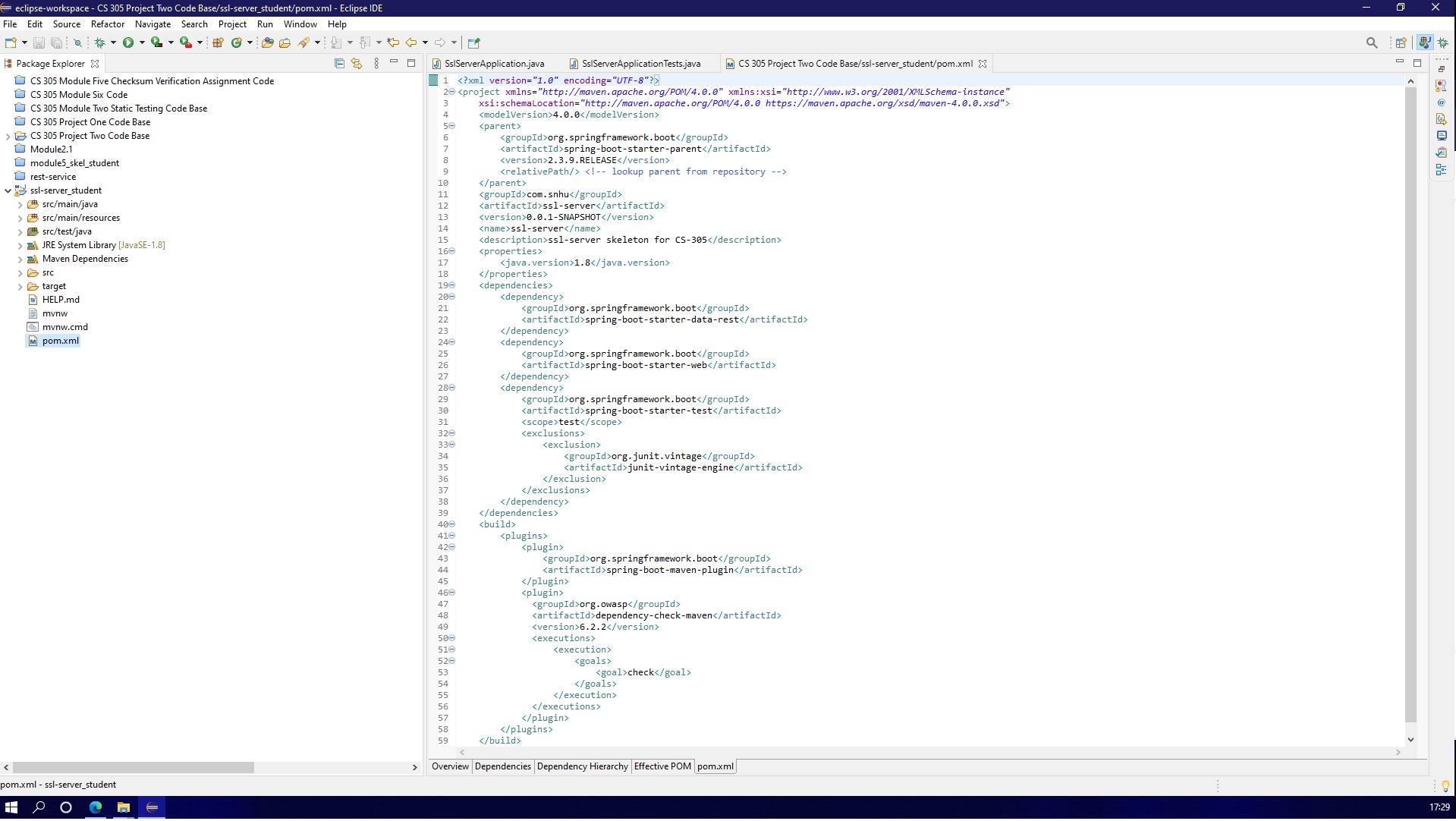
* Insert a screenshot below of the web browser that shows a secure webpage.

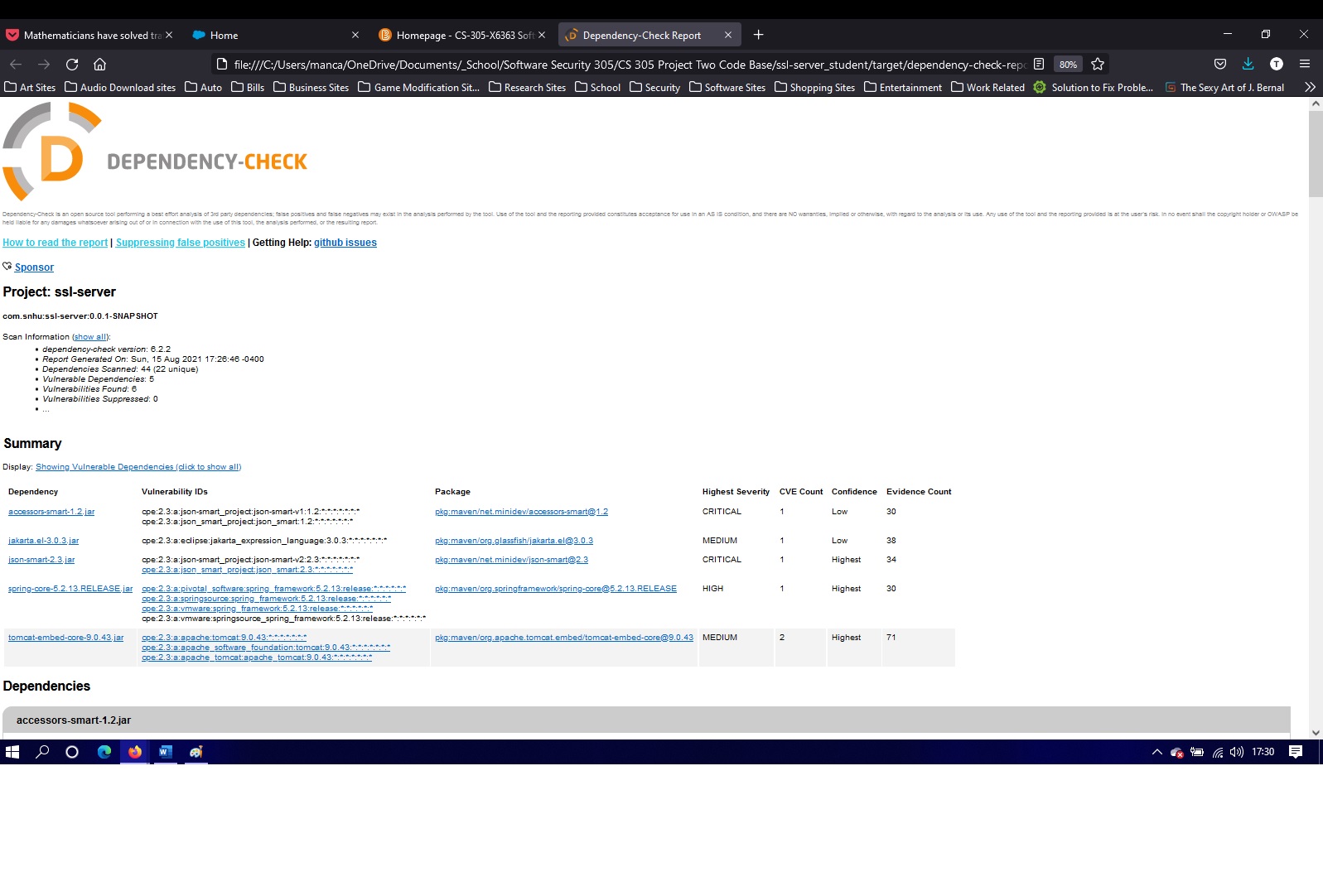
[Insert screenshot(s) here.]

## 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



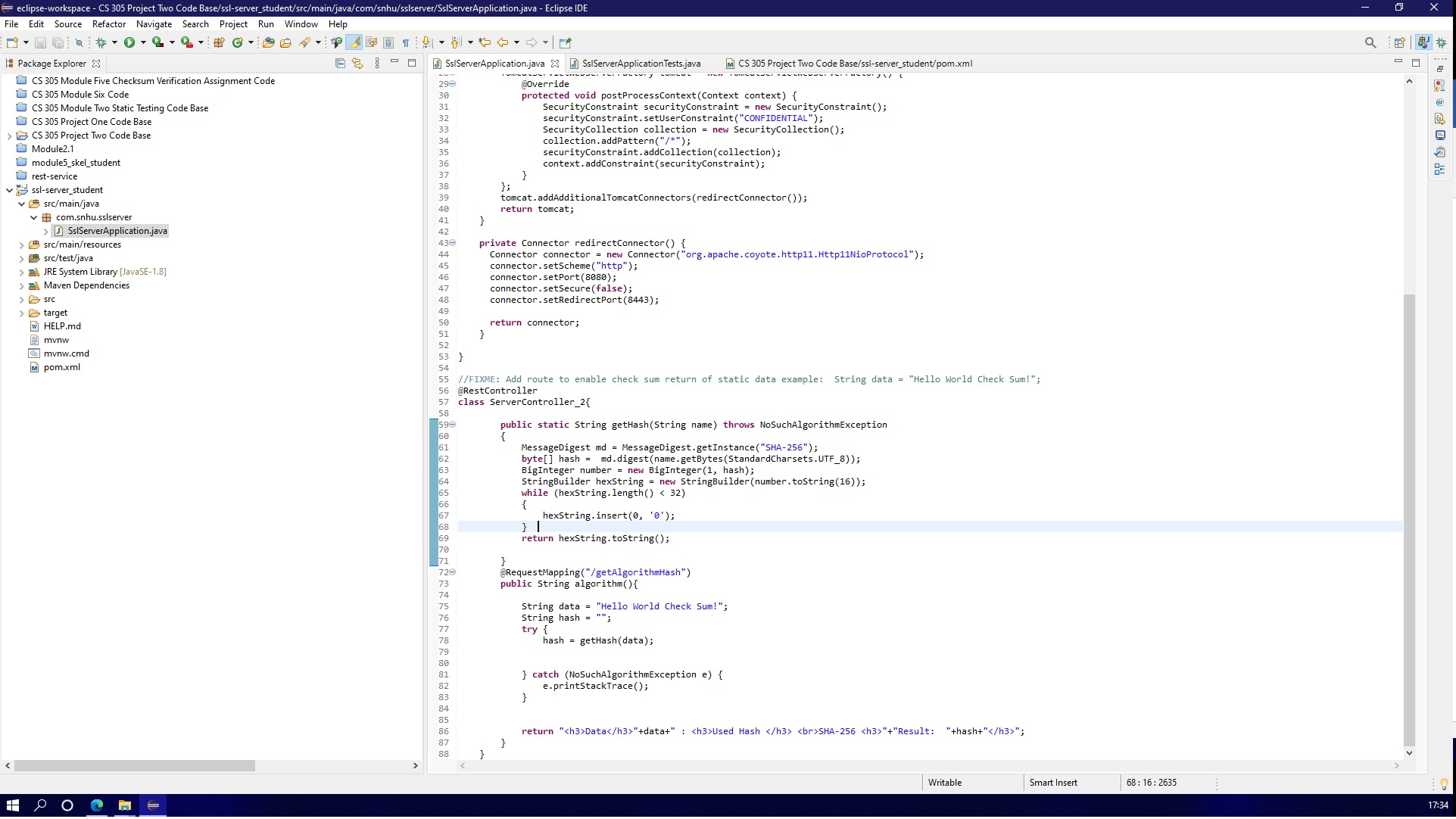


## 

## 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.

Securing all transactions and communications is crucial for any business, particularly for financial institutions. In reviewing the Vulnerability Assessment Process Flow Diagram, one of the initial and pivotal aspects, in my perspective, is code quality. Similar to constructing a house, establishing a robust foundation is paramount, and ensuring code quality serves as that foundation. High-quality code is easily understandable and navigable, facilitating future modifications or fixes. Proper indentation, comprehensive comments, clear notations, and effective communication among developers are essential components of quality code. This is imperative because the development team may not always be the same as the maintenance team.

Another significant aspect is validating all inputs to the system. In this instance, implementing a checksum for input validation and a self-signed certificate, enabling the use of HTTPS protocol for Artemis Financial, addresses API security. These measures enhance input security, along with updating the spring-boot version, fortifying the code against major threats. This seamlessly transitions into cryptography, arguably the cornerstone of security protocols. By generating signed certificates and keys for the application, we enabled RSA encryption using SHA2-256 hash functions, bolstering data security.