****

# 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** | **10/18/2020** | **Tyler Thomas** |  |

## 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

Tyler Thomas

## 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.

**Algorithm Cipher**: Advanced Encryption Standard (AES)

There are various cipher algorithms to choose from that could potentially meets the needs of our client Artemis Financial. I have decided that Advanced Encryption Standard is the best suited for my client’s needs. It is one of the most widely use standards by similar financial companies and is trusted as the standard by the U.S. Government and many other organizations.

To protect assets and vital information stored in a system and be able to share/transfer that information over the internet requires security protection. Some common actions attackers may attempt are impersonations such as guessing a password for a user login to gain access to their online bank account. This can happen when passwords are not securely stored to the database and can be guessed easily by using a password generator.

Another common attempt from an attack may be to crack a password file due to lack of proper encryption. Some of the best ways to protect sensitive data is use methods of encryption. Sensitive data stored and transferred within an application and over the internet can be encrypted requiring decryption using the same method of encryption to see the information.

When it comes to using AES, there are not many risks to consider since it uses varying key sizes for encryption (current 256-bit is the highest key size). The time to crack on a 256-bit key is years, which is quite frankly, a very long time. The most common attack again an AES encrypted application is a brute force attack which requires guessing all possible combinations.

However, AES only uses one key. For example, if a message is encrypted using the AES standard, it would generate a 128-bit key which would then be sent over the internet and made publicly available, but it only can be used by the recipient to decrypt the message. This causes a potential issue with key management and could cause leaks. This is by far one of the biggest drawbacks to AES.

To date, the AES algorithm has not been cracked and is consider one of the most secure encryptions due to the high bit level of encryptions. Atermis Financial can use this encryption standard to ensure their archive files are secured within the system. It is said that a hack would need to use quantum computing machines to break a 128- or 256-bit AES encryption key.

**Justification**

The Advances Encryption Standard is one of the most widely used encryptions standards to date. AES is used for encrypting and decrypting messages, the cipher will encode messages in blocks of a chosen bit size (commonly 128-but, or 256-bit). This process will result in the creation of a 128- or 256-bit encryption key, which is symmetric meaning that the sender and receiver of the message, file, etc. must both know the key. The key is used to encrypt and decrypt the message.

The advantage to using the AES cipher is that the encryption also known as “secret keys” at the high bit levels of 128 or 256 will require the use of a brute force algorithm and require hackers to use supercomputing to guess the number of combinations and break the key. It is also widely accepted and use by many other financial companies and social media companies like Twitter. This standard is used for storing data at rest which will be great for Artemis financial to store their archive files.

Some disadvantages to using AES is the fact that it only uses one key, and that key is public. However, being that the keys are so difficult to crack it remains something that be overlooked. Other algorithms such as RSA can be implemented along with AES, RSA is great because it uses 2 keys when data is sent and received this can combat the short comings of AES.

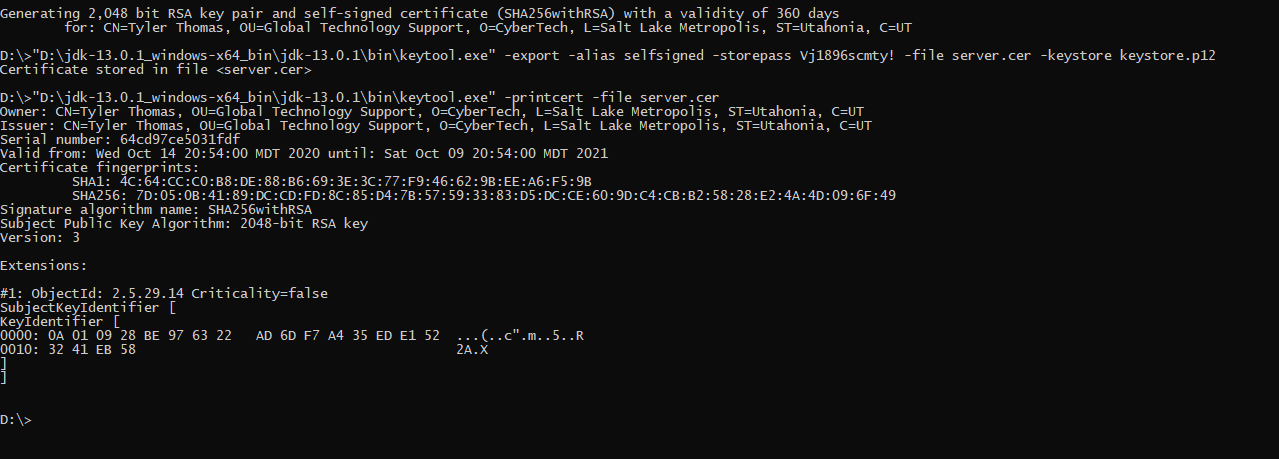
As a look into the history of encryption algorithms, encryption has been around since the early 1900s. Since inception, encryption was typically and most used by military and government to secure messages being sent. Some of the more common encryptions standards we are familiar with today started in the early 1970s with IBM. “IBM realized that their customers were demanding some form of encryption, so they formed a "crypto group" headed by Horst-Feistel”- (Sidhpurwala, 2013). This group designed an algorithm called Lucifer. In 1973 the National Bureau of Standards (now called NIST), in the US requested that the block cipher become a national standard.

Lucifer went on to be called Data Encryption Standard until 1997 when it was broken, and later moved on to the current Advanced Encryption Standard (AES).

## 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.

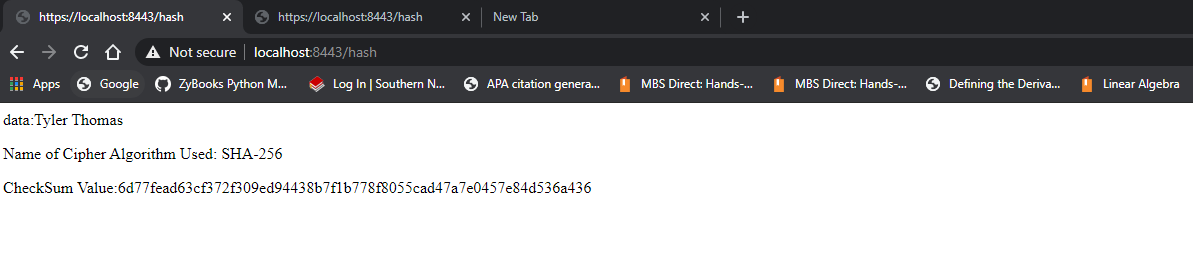


The screenshot above displays the output of the CER file. Certificates will ensure that users are verified using trusted keys, as opposed to public keys.

## 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.

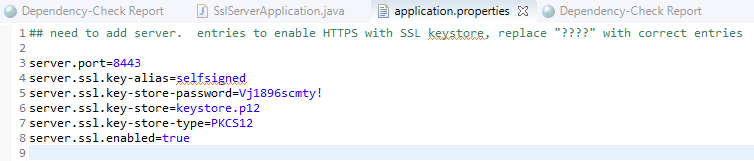


The screen capture above displays use of the SHA-256 message digest algorithm. It takes a data string, and then the digest algorithm takes the data string and outputs

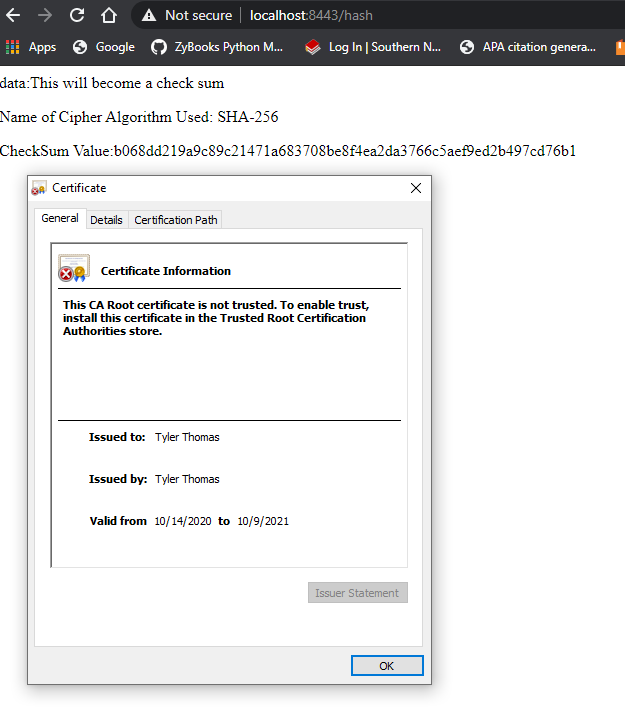
## 4. Secure Communications

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.



The screenshot above is the refactored application properties files, I generated the self-signed authority using the p12 keystore.



The screenshot above is displaying the self-signed certificate information that was generated in the CER file output in the screenshot above.

## 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.

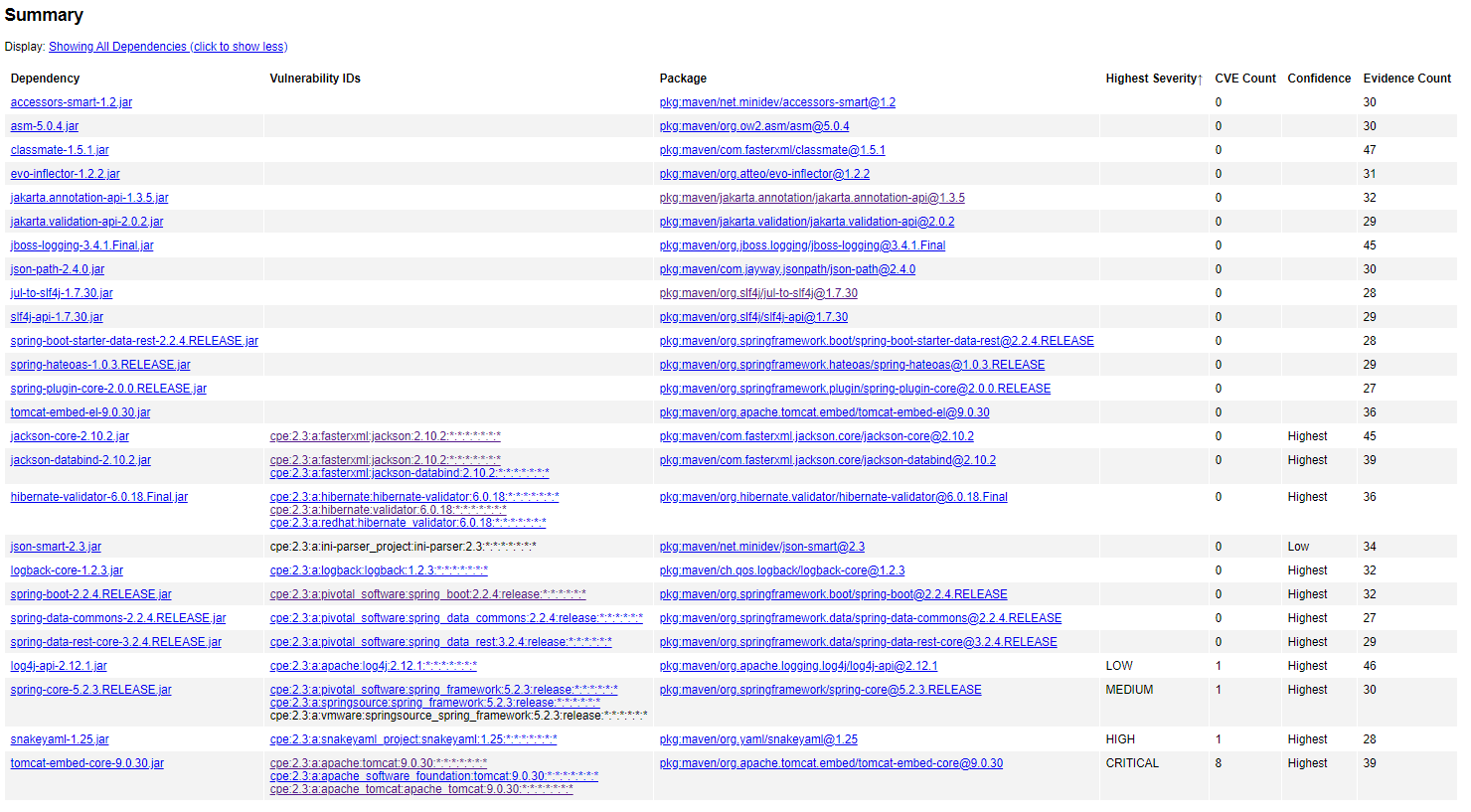
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



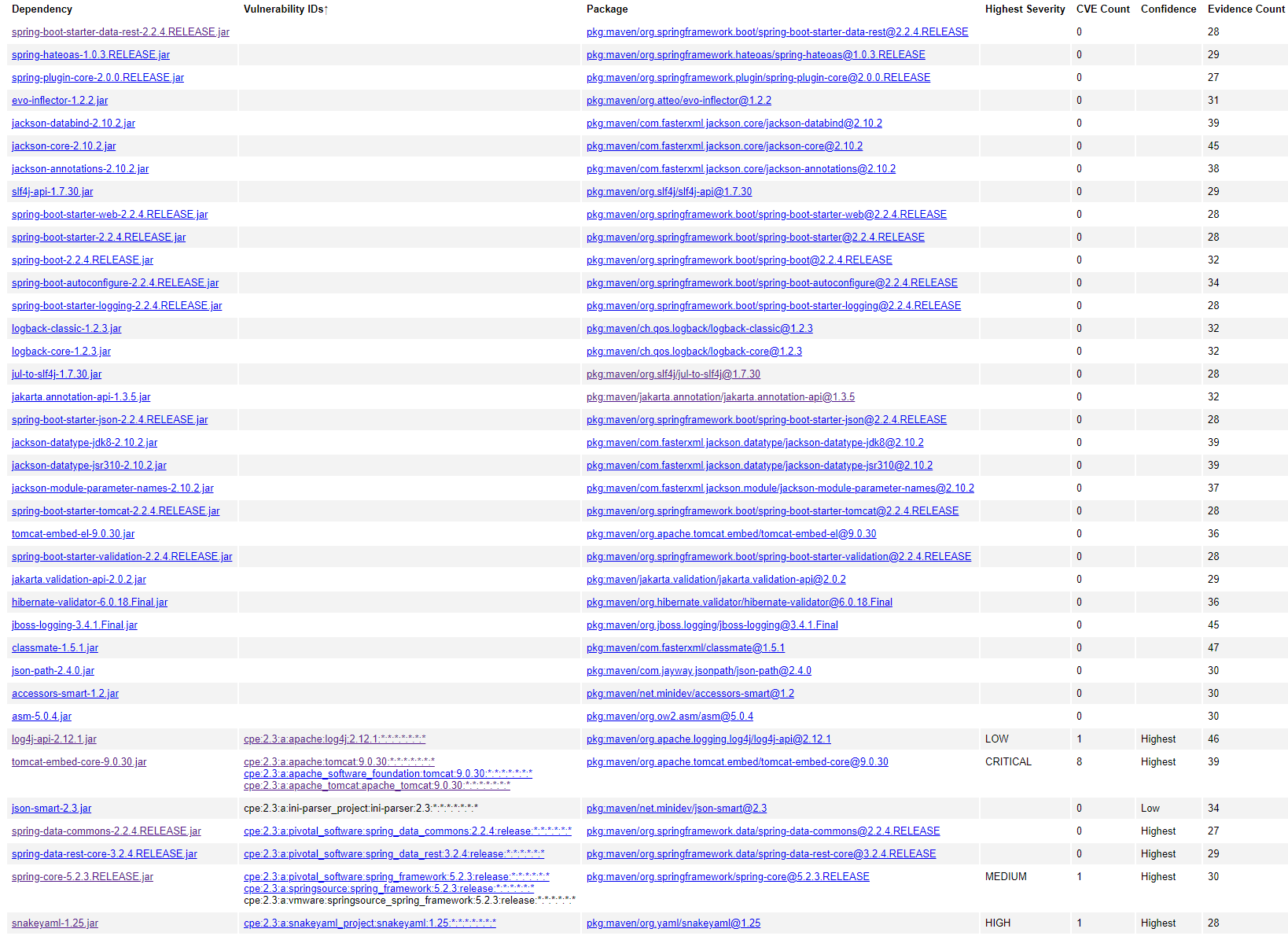
Above is a screenshot of the code executed without any errors, this was done by running the maven compile.



The screenshot above shows the Rest application running successfully with the refactored code.



The screenshot above is the dependency check ran **before** the refactoring of the code to add the message digest and application properties file to enable HTTPS with SSL keystore.

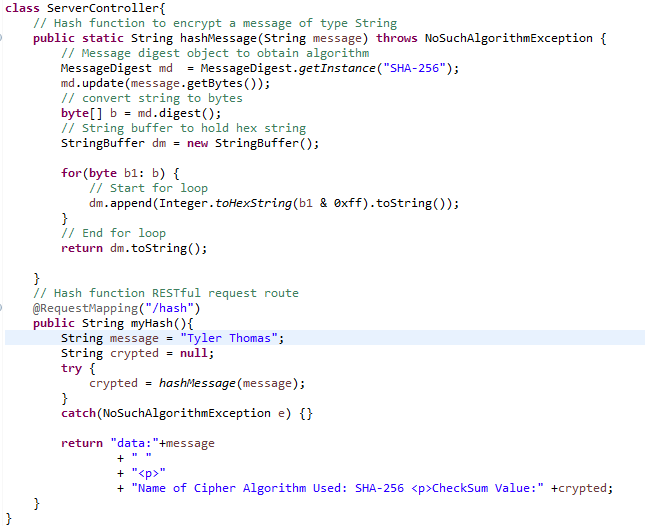


The screenshot above is a run of the dependency check **post** refactoring of the code to add the checksum verification and to enable HTTPS with SSL keystore. I also suppressed false positives which did not appear in the code base.

## 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.



Here is the refactored code that was added to enable the checksum verification to demonstrate the chosen algorithm message digest to hash a message and display the unique data string. The function reads a String to then pass into the message digest algorithm and display the unique data string. This is more secure as opposed to evaluating a passed in expression which could potentially be modified and exploited.



This screen demonstrated a successful compile of the application.



This screen demonstrates a successful application run.

## 7. Summary

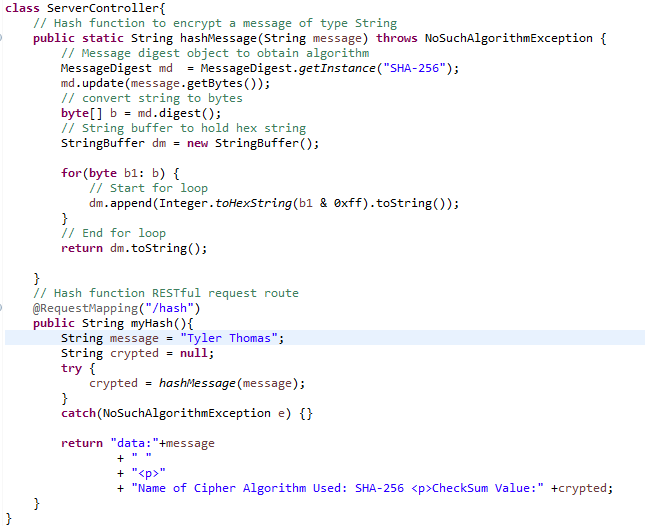
Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* 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.

**Vulnerability Assessment Process Flow:**

These are the areas of the VAPFD that focused on when refactoring the code:

**Cryptography** – By adding the cryptographic hashing algorithm called message digest, I was able to take a data string and output a hashed 256-bit data string. The message digest algorithm is powerful for ensure communication are secured.



**Client / Server** – In order to ensure that communications between the client and server are secure, I refactored the code to only accept HTTPS request, and enabled SSL KeyStore for certificate authorization.

**Code Error** – I ensured proper error handling was implemented because the message digest algorithm requires a try/catch block to compile. This will ensure that exceptions are handled when the hash function is called to digest the data string.

**Code Quality** – I ensured that my code was concise, and well commented to ensure that it can be worked on by other developers. I avoided adding unnecessary classes or methods that could cause errors or vulnerabilities.

**Input Validation** – This code did not require user input to run successfully. Input validation not a highlighted area for this application.

**APIs** – By ensuring that vulnerabilities in the application were identified, I will be able to make the necessary updates to the dependencies outlines in the dependency check report. This will ensure the security of the API for Global Rain.

The process I followed to add layers of security this application starting by assessing the Vulnerability Assessment Flow Diagram. This allowed me to pinpoint areas that needed additional security layers added for the client. I needed to ensure that communicated were hashed by using the message digest algorithm, generated self-signed certificates, and enable HTTPS only requesting to ensure that communications between the client and API were secure.

After refactoring the code and ensuring that it works as intended, I made certain that no additional vulnerabilities were detected in the Maven Dependency Check. I was able successfully refactor the code and not add any additional dependencies when running the dependency check. The final step I took was to manually review the code and inspect for vulnerabilities, I highlighted that the function and API route I added did not evaluate input, and thus making it more secure and less likely to be vulnerable to manipulation.

To maintain current security of a software application, it is important to ensure that no new vulnerabilities are introduced when changes are made to the code base. Using different plug-ins, platforms, etc. can lead to additional vulnerabilities. Maintaining latest security patches for the platforms you use is crucial, as many vulnerabilities are cause by exploits in software that are fixed with security patches. If you do not update when necessary, this could lead to major security breached. Security should be considered in all phases of software development, if security is not factored in until after the application has been built this could lead to major technical challenges. We must avoid the principle of security through obscurity, taking even the most basic security approaches to ensure that the platform and its users are safe from attackers.

Following guidelines such as the vulnerability assessment flow diagram will allow for a focused approach to adding security to a software application such as an API. It serves as a guideline for an assessment of each layer of the application. This allows for me as a developer to implement software security practices such as certificates, cryptography, and secure communication via web protocols.

References:

Oracle. (2017). *Java security standard algorithm names*. Moved. <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>

Elsevier B.V. (2020). *Hash collision*. ScienceDirect.com | Science, health and medical journals, full text articles and books. <https://www.sciencedirect.com/topics/computer-science/hash-collision>

SSH Communications Security Corp. (2007). *Advantages and disadvantages of certificate authentication*. SSH.com. <https://www.ssh.com/manuals/server-zos-product/55/ch06s03s05.html>

SNHU. (2020). *Vulnerability Assessment Process Flow*. Southern New Hampshire University. <https://learn.snhu.edu/content/enforced/525459-CS-305-Q1514-OL-TRAD-UG.20EW1/course_documents/CS%20305%20Vulnerability%20Assessment%20Process%20Flow%20Diagram.pdf?_&d2lSessionVal=lFET3KFi0BDY1y2bXxpM8Sv8B&ou=525459>