****

# 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** | **August 14, 2022** | **Ethan Smith** |  |

## 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

Ethan Smith

## 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.

An appropriate algorithm cypher that avoids collisions and meets the company’s requirement for data verification in the form of a checksum. To be clear this would be used with SHA256 as this method is used in many secure APIs. This algorithm first calculates a unique hash of the input data using SHA256 algorithm. The hash is then encrypted with a private key using the RSA algorithm. RSA is an asymmetric cryptography algorithm, meaning it uses a public key and a private key. This solves the company’s need for security. They keys are mathematically linked together which means that collisions are impossible if the keys are created correctly because the private key is basically the inversion of the public key algorithm (though it is a bit more complicated than that behind the scenes.) Furthermore, RSA is a standard encryption algorithm that is used by many large companies for identification and other purposes such as GitHub which uses it for identifying/authenticating users that want to modify code repositories that require permissions to modify.

## 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.

Text

Description automatically generated





Text

Description automatically generated

## 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.

Graphical user interface, text, application, email

Description automatically generated

## 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.

Graphical user interface, text, application, email, website

Description automatically generated

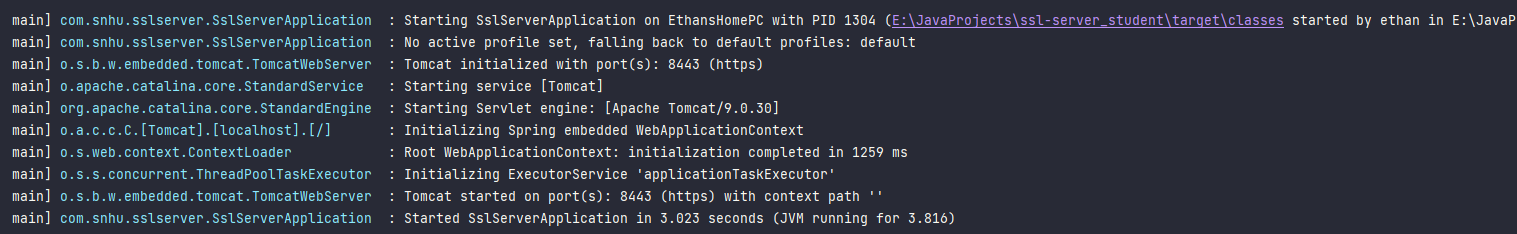
As you can see there is a padlock because it is a self-signed certificate there is an exclamation symbol, but I would have to make changes to my system to make that go away for a self-signed certificate on localhost.

## 5. Secondary Testing

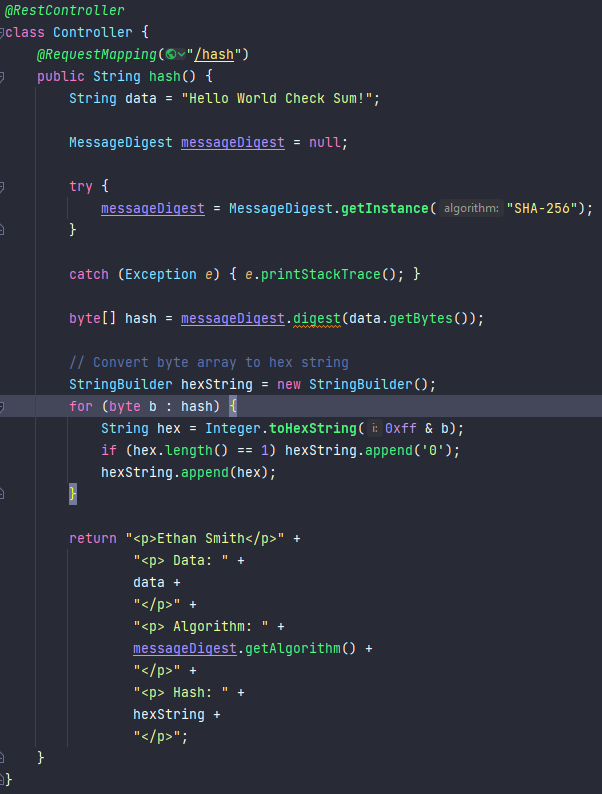
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not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

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

**Running Server**:



**Refactored Code**:



**Dependency Check Report**:

Graphical user interface, text, application

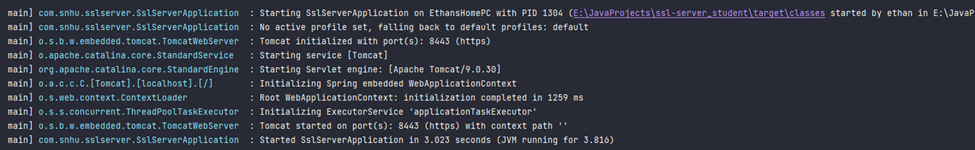
Description automatically generated

## 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.

**Running Server**:



**Refactored Code**:

Text

Description automatically generated

**Pom.xml Changes:**

I updated the dependency-check-maven plugin to 7.1.1 to make sure that all potential vulnerabilities with the code were found.

Text

Description automatically generated

## 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.

In this project I have addressed multiple areas of security in the Vulnerability Assessment Process Flow Diagram. I addressed Cryptography, Client / Server, and Code Quality. I used cryptography to generate certificates for a secure web connection and created a hashing algorithm for checksums that securely verified data. Data that could be secured could be financial records or miscellaneous documents that were of a sensitive nature. Client / Server was used because we were serving an HTTPS web server using a certificate. The web server sends data between a client and the server on a secure connection. Code Quality was addressed by account for security vulnerabilities in the code and using best practices in the code to improve performance and overall security for the application.

The first layer of security for the software was a certificate that was used to secure the traffic sent back and forth between the client and the server. This is of value to the company because it increases the level of trust that the users have in the product and keeps their very sensitive data such as passwords and financial information safe. On top of this I used SHA-256 to produce a hash that secures plain text data using a method in the code. A secure connection is good because it keeps information safe from malicious users trying to intercept traffic but once traffic is in the system it can be acquired in other ways such as stealing data from a database. Ensuring that important data such as passwords are hashed using a secure method (such as SHA-256) means that there is a much needed layer of security on persistent data. The final layer of security that was used for this software application was the dependency checks. This analyzes all of the dependencies used in the software and makes the developers aware of potential vulnerabilities. This is important for Artemis Financial because it ensures the developers are using code correctly as well as informing other teams such as sys admins to setup the environment the software runs on correctly, keeping all parts of the system as safe as possible.

Best practices for maintaining the current security of the software application are making sure that all dependencies are up to date as well as the dependency check plugin. A new dependency check should be run upon each dependency update as well as any update to the check tool itself. The certificate should be renewed prior to expiring but as soon as it expires at the very least. Any changes to the encryption methods used in the application should be tested thoroughly prior to release. If RSA/SHA-256 becomes obsolete, then the software should be updated to use the most secure algorithm at the time. Input validation should be used to make sure that data that could crash the system if not handled properly doesn’t make it to the logic of the business methods. APIs should be made to handle data neatly and securely and roles should be made to make sure that ever API user has the proper authorization to perform each function of the software.