****

# 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** | **8-14-2021** | **Jacob Bayer** |  |

## 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

Jacob Bayer

## 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.

To fulfil Artemis’s needs, I have chosen to use the RSA encryption algorithm. The RSA algorithm is an asymmetric encryption algorithm, meaning that There will be two keys generated by the server, a public key and a private key. The private key will be kept secret by Artemis and the private key can be sent to any of their customers. The same will also happen on the customer side. To facilitate secure communication, any files sent to the customer will be encrypted using the customer’s public key. Once the customer receives the file, they can decrypt the information using their private key. If the customer sends Artemis a file, the same process happens with Artemis’s key pair.

This asymmetric algorithm type is also useful as it can be used to sign the files that are sent to ensure that the file came from the expected party and that it has not been tampered with. This is typically done by hashing the file contents (so that the entire contents of the file are not sent twice), and then encrypting the hash with our private key to form the signature. This is packaged along with the original file and the whole package is encrypted normally. Once the customer receives the file, they can decrypt the file normally, and then decrypt the signature using Artemis’s public key. If it decrypts successfully then it proves the file came from Artemis, as Artemis’s private key is the only thing that could have encrypted the data, and Artemis should be the only entity to have their private key. They can also compare the signature hash to the hash of the file they received. If the hash values match, it means the file wasn’t tampered with in transit.

## 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.

A picture containing text, screenshot, computer

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, Word

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.

A screenshot of a computer

Description automatically generated

## 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

Graphical user interface, text

Description automatically generated

A picture containing text, screenshot, computer

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.

A picture containing text, computer, indoor, screenshot

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 designing the security infrastructure for Artemis Financial, I have addressed several security vulnerabilities from the Vulnerability Assessment Flow Diagram. The API I designed is secured by a certificate and a checksum. There are several areas protected by cryptographic functions that include the certificate and the checksum, and file transfer will occur using asymmetric public/private key encryption. Client/server interactions are secured by the aforementioned cryptographic security protocols. Code error and code quality have been addressed by running a dependency check and doing functional testing and refactoring code to remove any vulnerabilities found.

When adding security to Artemis’s application, I started by determining an algorithm cipher to use with their system. As what I was working on was a file transfer system, I settled on the asymmetric RSA algorithm, as this generated public and private keys that can be used to secure two-way file transfer. I then generated a certificate for the. I used this certificate to refactor the system too use the HTTPS protocol site so that the clients’ devices know that when they connect to the system, they are in actuality connecting to our system, and not being sent to a malicious URL or being sent malicious files. After that I inspected the code manually and with a dependency checker to ensure that no vulnerabilities were introduced that would affect the system I had created.

There are several things to consider in maintaining the security of the site. Make sure that the private key is kept in a secure location and is only accessible by people or processes that absolutely require it. Another consideration is to keep on eye on any security vulnerabilities with the implemented algorithms. For example, if it is found that the SHA-256 Hash algorithm starts to have collisions, it may be prudent to implement SHA-512 instead. Any new code added to this module should be fully inspected for errors and have a dependency check run on it to ensure any vulnerabilities would not affect the code.