****

# 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** | **2/20/2022** | **Matthew Wilt** |  |

## 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 (Kelly) Wilt

## 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.

I would recommend using an AES encryption algorithm to meet Artemis Financials’ needs. AES is the newest, strongest, and industry-standard encryption algorithm currently available to the public. It was designed by the National Security Agency to protect highly classified documents using the range from 1 to 256 bits of encryption keys. AES256 encryption is extremely secure, using 1.1 x 1077 possible combinations of keys that it would take for the encrypted file to be decrypted; one false step in the process of decryption too will reset the entire key state, thus causing anyone trying to decrypt an AES256 encrypted file to be forced to start from the very beginning. It would take hundreds of millions of years, yes literally, to break open this type of file, it is that secure.

The Secure Hashing Algorithms (SHA) are common hash functions. SHA can be used in conjunction with AES to ensure security, as SHA takes data input as plaintext and creates a ciphertext that is unlikely to be deciphered. The bit-level in security lets us know that an attacker would need to perform 2n operations (n being the bit level) to crack a cipher of that bit level. AES utilizes either 128, 192, or 256 bits, with AES-256 being the most secure of them.

AES is a symmetrical cipher, so only one key is used for encryption and decryption. This is safer for at-rest data. The processes for encryption and decryption are much faster than asymmetrical. This is useful when encrypting large amounts of data. Data that enters this algorithm is transformed under sets of rules over the course of 10 rounds eventually outputting as a Ciphertext. Once encrypted, the data is seen as a seemingly random combination of characters that can undergo a reverse transformation, or decryption, if the user has the “secret key” that was generated during the encryption process. The specific series of operations that takes data of an arbitrary length and creates a ciphertext of a fixed length is also known as the hash function. When AES is implemented properly, it is unbreakable. Now there are stats out there that it would take the most powerful computer in the world, 885 quadrillion years to brute force crack an AES-128 key. So, to reiterate, it’s unbreakable. The US government recommends AES for all sensitive data. AES-128 is great but the “gold standard” is AES-256 which is used for the government’s “Top Secret” files. With AES also being internationally certified, there should be no issues with Artemis’ clients around the globe.

With one key it is very important to make sure this key is secured. Secure coding comes into play here as there can be mistakes that aren’t caught and it will open you up to attack. File input/output verification comes to mind. These will need to be handled with a file type verification as well. Throughout my research, there are many things we need to be aware of and on the lookout for. Cryptography seems to be advancing daily, those with bad intentions are trying to find backdoors to every standard of security out there. How lucky were the allies to decrypt the Enigma Machine from the Nazis? Smart people, hard work, and maybe a little bit of luck. As of now, AES is an algorithm we as security developers can hang our hat on. As with the demise of DES, AES may suffer the same fate if it’s not continuously inspected and expanded on. AES will be around for at least a few quadrillion years, that’s why I support this algorithm for Atremis’ data archive.

## 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 generatedText

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.

A screenshot of a computer

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.

This is where I had problems. The web browser would not show the required information.

## 5. Secondary Testing

Complete 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

Text

Description automatically generated

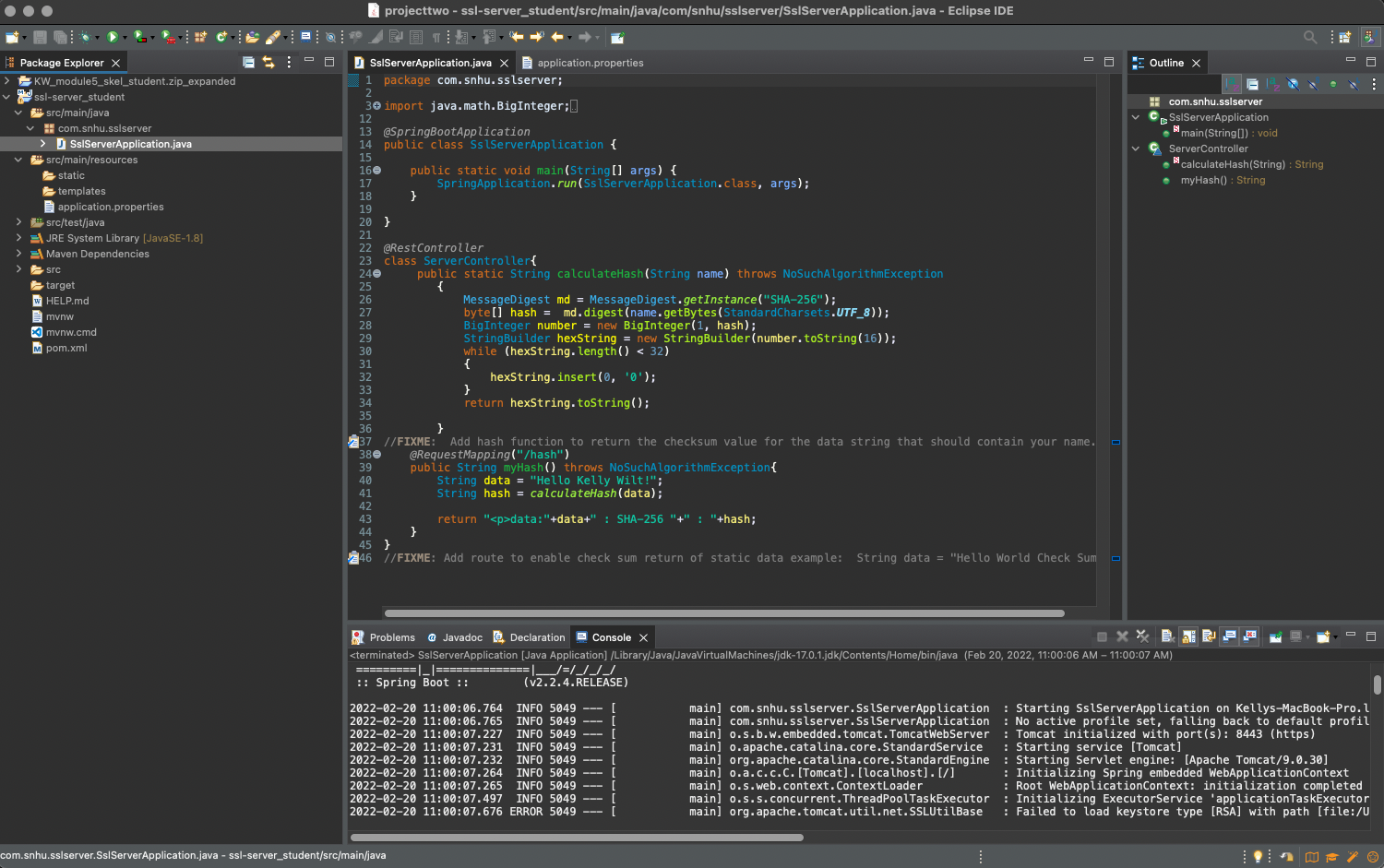
Graphical user interface, application

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.



## 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.

After reviewing the security vulnerability assessment process flow the following areas were identified as necessary for review are the APIs, Cryptography, Code Quality, and Input Validation.

Secure communications are very important to maintain in this application as unsecure communications can cause data to be seen by an attacker and could cause them to steal confidential or personal information. The results of a security breach can cause (but are not limited to) a loss of trust between the application owner and the consumer, financial loss due to fines incurred by governments requiring secure communications, and financial loss due to resources spent repairing/addressing the security issue. Maintaining security in the application will protect the company’s products and assets and keep their client’s trust in tact.

One best practice for maintaining our application’s security is patching our software and systems to ensure everything is up to date. This ensures that attackers cannot exploit out-of-date systems. Enforcing the least privilege is also crucial. While it is not in place with the current state of our application, ensuring users only have the access that they need rather than giving everyone access to everything protects the organization from attacks within the group.

References

Inc, O. (2017). *Java Security Standard Algorithm Names.* Retrieved September 27, 2020, from https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html

TechNadu. (2018, August 08). *What is the Strongest Encryption Today?* Retrieved September 27, 2020,rom <https://www.technadu.com/strongest-encryption/37596/>

Prabhu, R. (2018, September 27). *SHA-512 Hash In Java.* Retrieved October 04, 2020, from <https://www.geeksforgeeks.org/sha-512-hash-in-java/>

Shaikh, R. (2017, February 24). *Google Just Cracked SHA1, But There's No Need to Panic.* Retrieved October 18, 2020, from <https://wccftech.com/google-cracked-sha1/>

S. (na). *Vulnerability Assessment Process Flow* [PDF]. Manchester: SNHU.