****

# 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** | **3/25/2021** | **Victor Feight** | **Revision 1** |

## 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

Victor Feight

## 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.

AES\_128 is “Advanced Encryption Standard”, capable of handling 128-bit block ciphers.

According to Oracle documentation, AES cipher can be used with one valid key size using AES\_<n> format, where n is 128, 192, 256-bit sized keys. Higher bit keys require more processing power and higher latency, though it takes longer to guess. This makes AES\_128 a great choice for storing archives of large sizes over long periods.

Creating strong keys requires pseudo-random generation techniques, as generation of the key for a symmetric algorithm requires a random string of 128 bits.

AES is a symmetric key algorithm, in which the key is used for both encrypting and decrypting data.

AES was developed as a mathematically efficient symmetric block cipher, and the standard replaced the slower DES in 2000.

Extra:

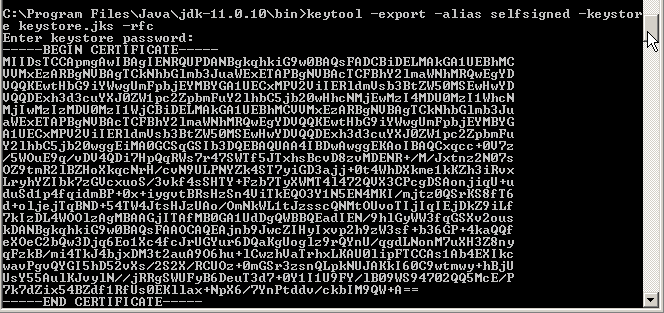
AES Operates in CBC Mode, which requires IV to make each message unique.

## 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.

We require a trusted **SSL certificate** for securing our program, for the purpose of distributing our public key and identifying our server. We use keytool to generate a keystore.jks.



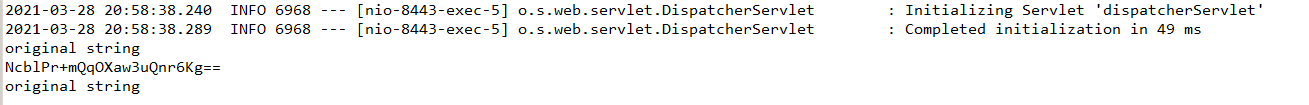


## 3. Deploy Cipher

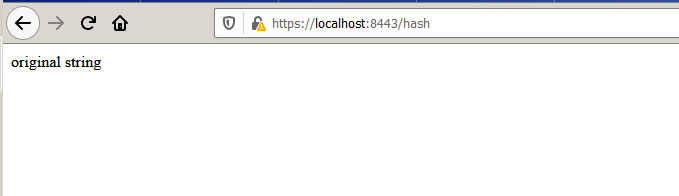
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

Eclipse output:



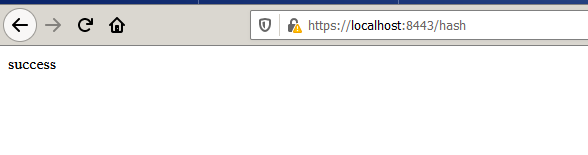
Output from checksum GET response:



## 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 following settings are in application.properties to enable HTTPS:

• server.port=8443

• server.ssl.key-alias=selfsigned

• server.ssl.key-store-password=password

• server.ssl.key-store=keystore.jks

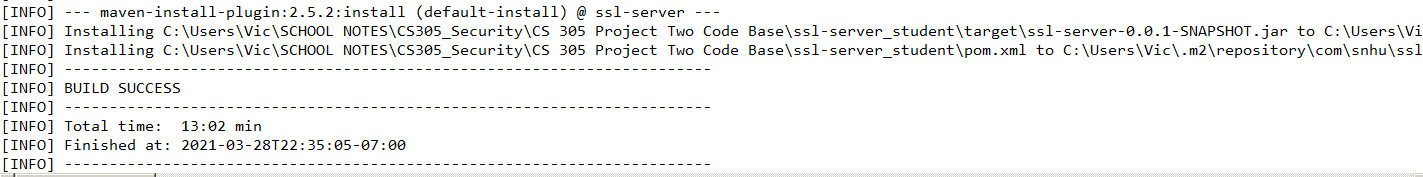
• server.ssl.key-store-type=JKS

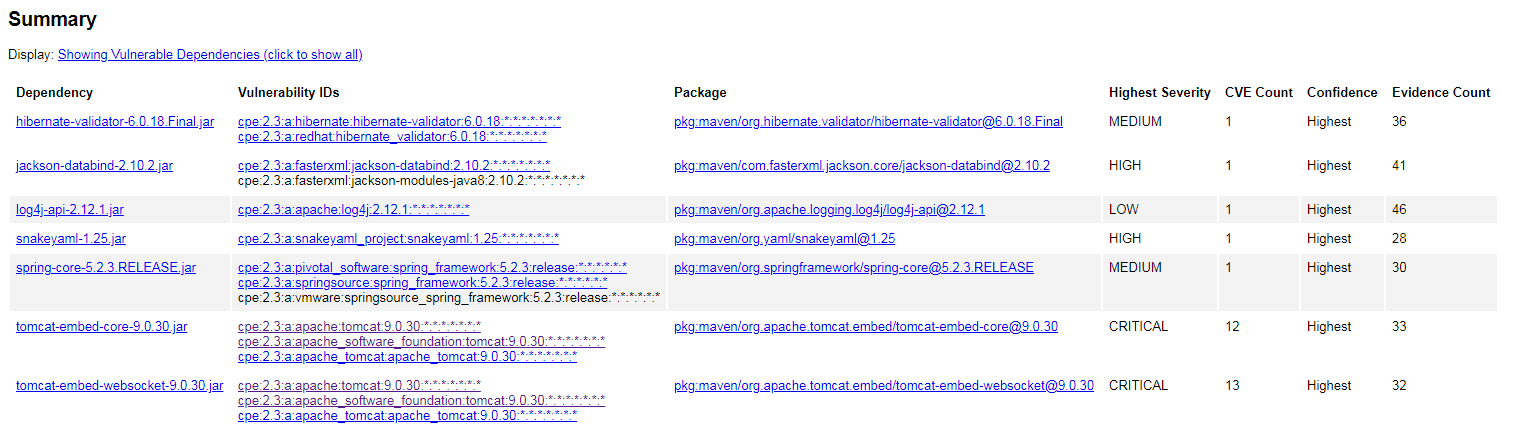
## 5. Secondary Testing

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

Notes: Updated dependency-check-maven to version 6.1.3 as per requirements.





No additional vulnerabilities were introduced, however there is a critical vulnerability listed for our tomcat version when using Apache JServ Protocal (AJP), additional hardening configuration needed as well as recommended upgrade to 9.0.31.

## 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.

Cross-Site scripting is listed as a vulnerability as well as data injection attacks. Configure application to return a Content-Security-Policy header, with associated HTML tag. Enable CSP header in Spring Boot app to help mitigate these attacks.

Consider preventing unauthorized users from accessing certain pages such as /home or /hash by configuring Spring Security in the application.

We can add spring-boot-starter-security to our pom.xml and utilize WebSecurityConfigurerAdapter to add auth details, with @Configuration @EnableWebSecurity annotation to secure endpoints in a separate class.

Ensure server side code is protected, do not store passwords/connection strings/other sensitive info.

Log all exceptions, ensure all resources closed.

## 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.

We are highlighting security enhancements in **API / Secure API interactions** as we have focused on creating a secure RESTful api using Spring boot and maven.

We have focused on secure **Cryptography / Encryption** by adding a getHash that demonstrates our cipher string encrypt/decrypt functionality with AES.

We have focused on securing our **Client / Server –** By ensuring HTTPS (Hypertext Transfer Protocol Secure) encryption is used between client and server during transfer requests. Additionally by ensuring our self-signed certificate used for HTTP is valid, and the information in the certificate is correct to prevent man-in-the-middle attacks.

Finally, we focused on **Encapsulation / Secure Data Structures** by utilizing principle of least privilege to our accessor methods, and making our static instance data as private.

Adding additional security layers involves researching best practices and libraries, not reinventing the wheel, ensuring things fail securely, and using regression testing to ensure no creeping bugs or security issues have come up.

The choice of our encryption algorithm matters less than how we choose to implement it, how we choose to distribute and store the key, which libraries and services we will use for authentication, which OS for the server hosting the service, and the chosen hardware and configuration are all important details for maintaining security.