****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/12/2022** | **Jonathan Pun** | **This documents the security of the application and the refactoring of the code.** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Jonathan Pun

## Algorithm Cipher

I would highly recommend using an AES encryption algorithm to meet Artemis

Financial’s needs. AES is the newest, strongest, and industry standard encryption algorithm

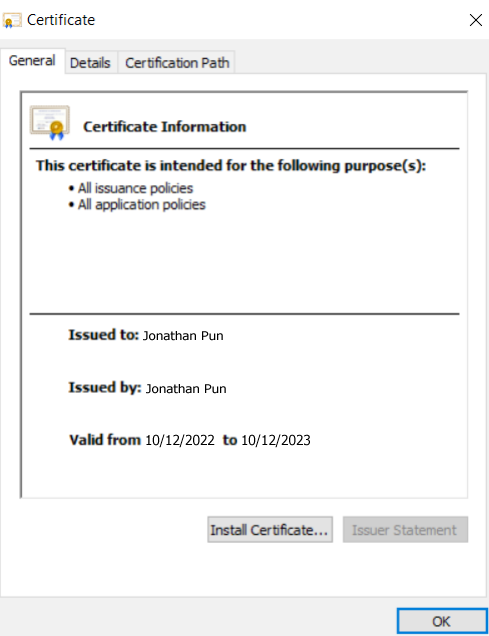
currently available to the public. It was designed by the National Security Agency to protect

highly classified documents. AES256 encryption is extremely secure, using 1.1 x 1077 possible combinations of keys that it would take in order 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 risks are nearly void, there is nothing but security and protection in mind when using this algorithmic encryption key; but the question is, how is AES encryption used? First, there’s a file of plain text or data, this data becomes encrypted, this encrypted data is now in jumbled ciphered text, hiding the original text from anyone who isn’t authorized to view it with the decryption key. When the sender and the receiver share the same key, then the ciphered text/file becomes its original state once again, showing the initial plain text/data of the file that was previously encrypted. As stated, considering the near endless number of possibilities of key variants, both parties must have the same exact key to unlock the file. This is quite beneficial for

a financial institution when it comes to the sending and receiving of sensitive financial information. This can block out anyone trying to snoop through, perhaps, a supply chain attack. Sure, an outside attacker during a supply chain attack could slip in something like a compromised .dll file to disrupt the entire file network, but ultimately as long as the files within are protected by AES256 encryption, the attacker will never be able to access the sensitive information without having access to the key which they would have no possession of in the first place. When it comes to its hash functions, it also works somewhat similarly all around; basically it takes a bit string as an input, then returns a fixed 256 length string as output and if this input is tampered with in any way by some malicious intent or accident, the output then becomes unclear and ciphered. This is why AES encryption is the best, it’s secure, it’s modern, it’s standard, it’s military grade and publicly shared for all institutions that want security at their forefront. Ultimately, this is my justification.

## Certificate Generation

Insert a screenshot below of the CER file.



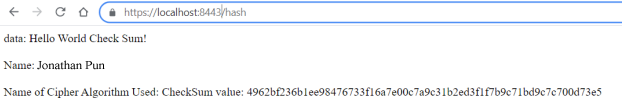
## Deploy Cipher

Insert a screenshot below of the checksum verification.

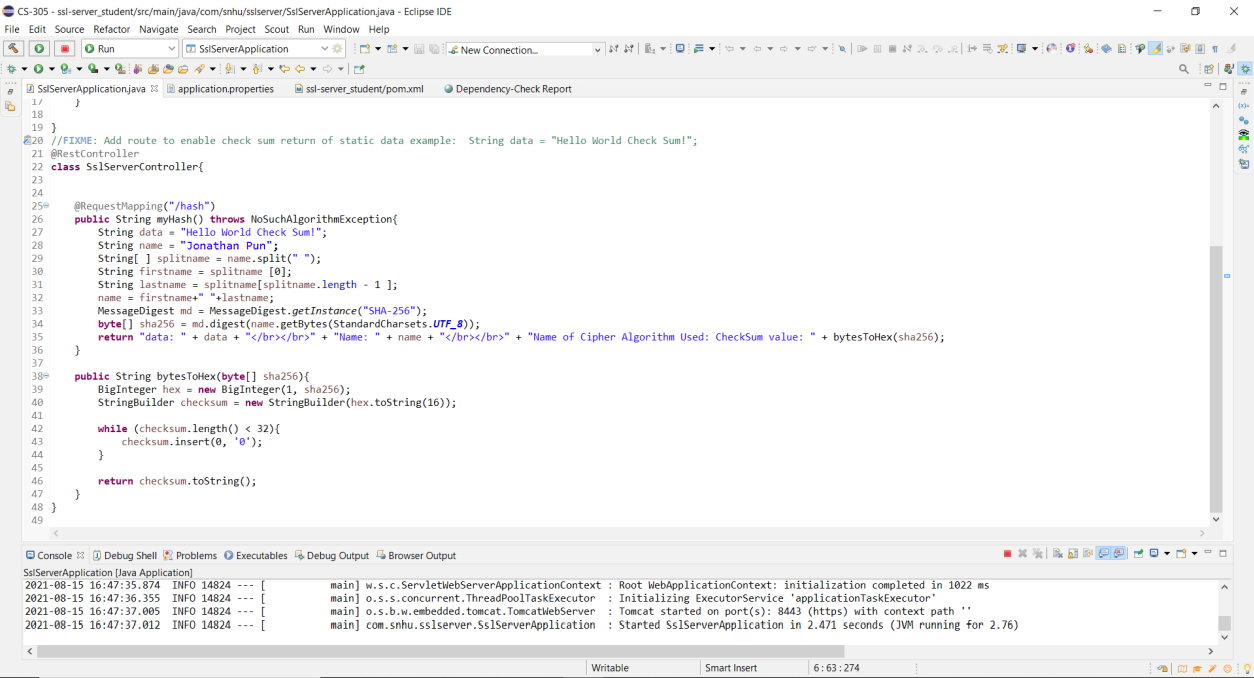


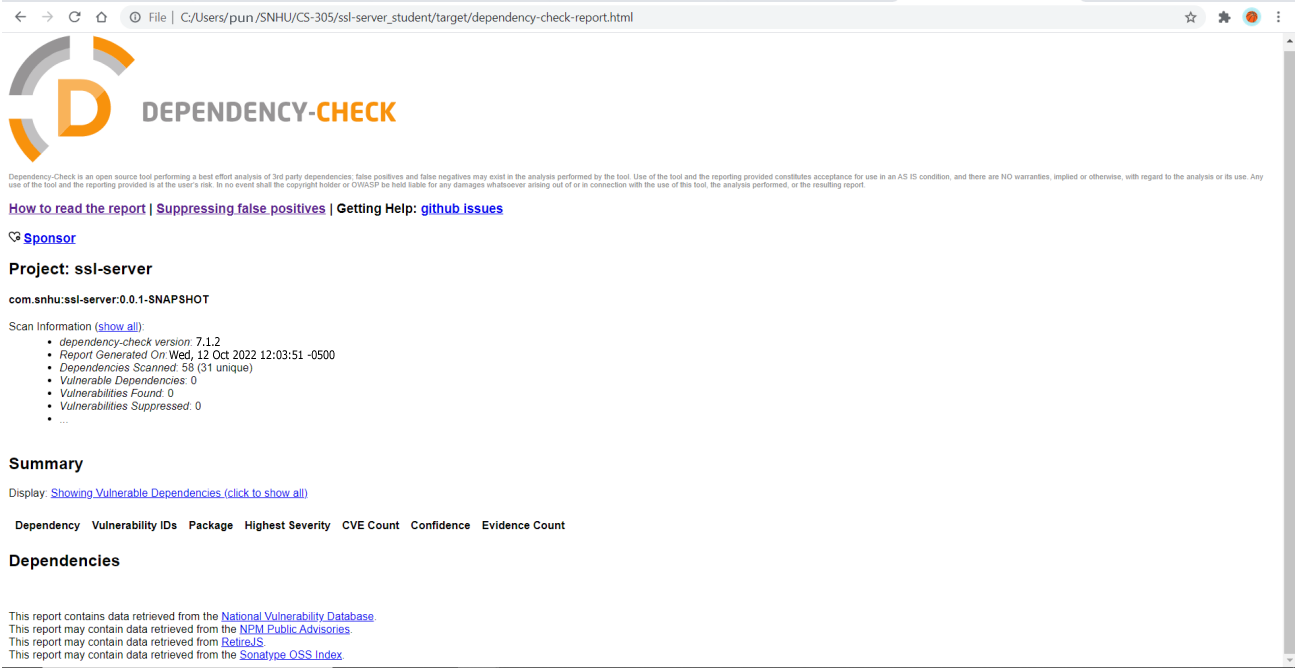
## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.



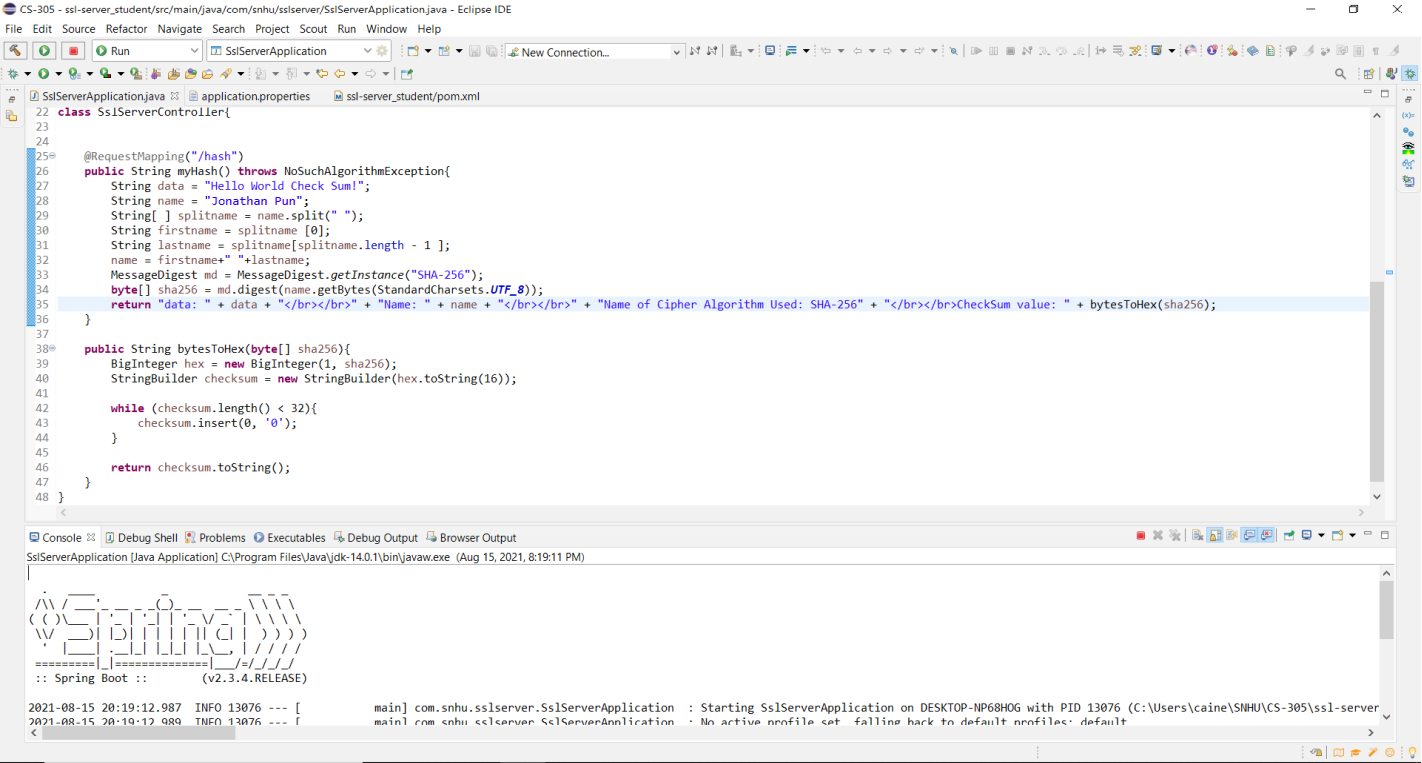
## Secondary Testing

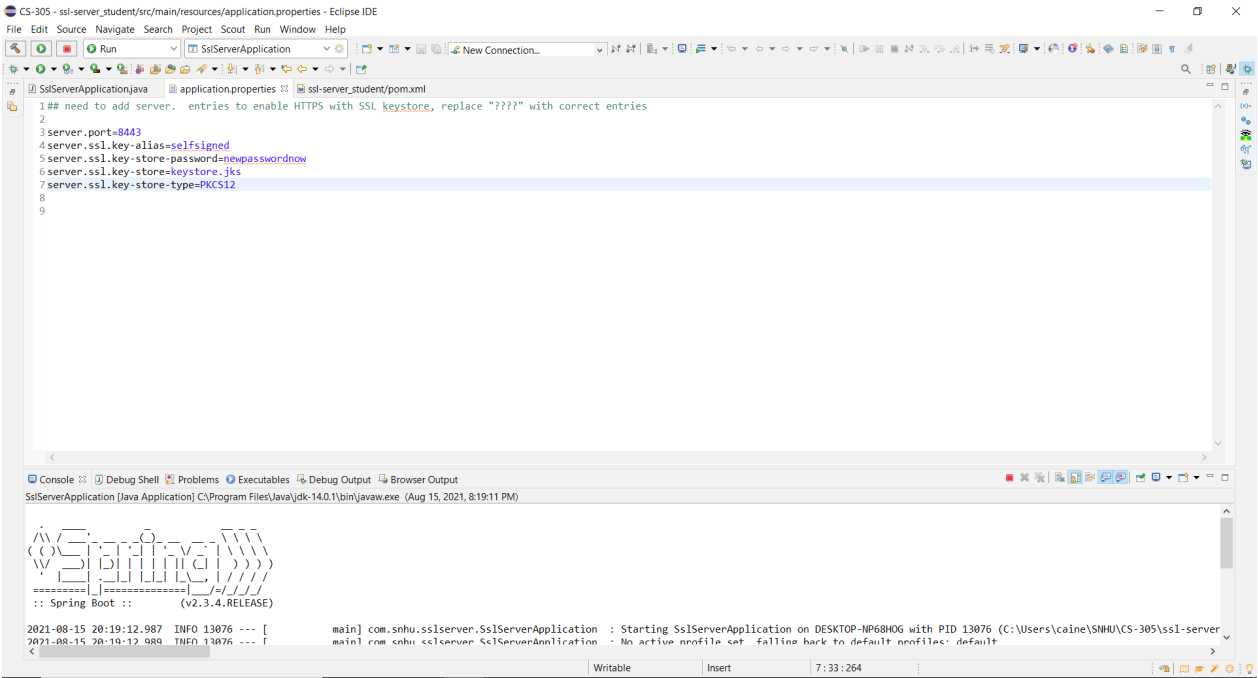
Insert screenshots below of the refactored code executed without errors and the dependency-check report.



## Functional Testing

Insert a screenshot below of the refactored code executed without errors.





## 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.

1. APIs: APIs were addressed with the implementation of HTTPS and the creation of a browser interface that a user can view
2. Cryptography: Cryptography was addressed with encryption algorithm ciphers and hash functions, along with checksum verification.
3. Client/Server: Client/Server was addressed, even though the client and server were the same at least in the testing phase of this application, as information was sent from the client to the server to be displayed.
4. Code Error: Code error was addressed by utilizing exceptions within our class, mainly a NoSuchAlgorithm exception within the myHash method.
5. Code Quality: Code quality was addressed as the code was reviewed to ensure functionality and readability.

The main security that was added to our application was self-signed certificates that allowed for HTTPS to be used. We also refactored the pom.xml file to ensure that all vulnerabilities that were discovered within the dependency check were resolved. The first step in my process was ensuring that the certificates were made correctly so that we would be able to utilize HTTPS once our application was up and running. This security adds to our company’s wellbeing by ensuring that our website is secure, and users can be assured that they are dealing with us and not a person trying to impersonate another to gain access. The next step was making sure that our hashing function worked properly and verifying this with the checksum. This security helps our company’s well-being by letting us be well assured that our users’ data is being hashed properly and not easily retrievable. The final step was making sure that all vulnerabilities are patched up. Having that security ensures that we as a company have everything covered and we can be certain that the application is up to date and working as intended.

## Industry Standard Best Practices

* 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 least privilege is important as well. 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.