****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **4-16-2023** | **Alex Mecum** |  |

## 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

Alex Mecum

## Algorithm Cipher

Given the nature of the type of data we are trying to protect, I would recommend the Advanced Encryption Cypher (AES). AES is often referred to as the gold standard for encrypting data and is the go-to encryption cipher of the US government. It has since been used for most industries and services, including being used for almost everything online. As this includes financial information, Artemis Financial will find AES to be a great choice for their purposes.

Cyphers use what are known as hash functions. Basically, the hash function takes regular data and makes it unreadable. To have a hash function convert the data, a key is required. AES allows for 128-bit, 192-bit, and 256-bit keys, which refers to the number of characters in the key. For comparison, the Data Encryption Standard was only 56 bits. Today, it can be brute-forced in only 362 seconds. AES’s 128-bit, however, would need upwards of 36 quadrillion years to brute-force. In this case, I would recommend Artemis Financial uses the 256-bit algorithm.

While these large keys are a big advantage for AES, there’s also a rather prominent disadvantage. AES encrypts every block of data the same way. If someone were able to access one part of your data, they would be able to access all of them. As such, Artemis Financial should take other precautions when using AES.

One precaution they should take is creating a random key. While a pattern would be easy for a person to remember, it would also make that key easier for someone else to find. A random key ensures better security. Another thing that affects security is the difference between symmetric and non-symmetric keys. Symmetric uses the same key for both encryption and decryption, whereas non-symmetric uses different keys.

## Certificate Generation

Insert a screenshot below of the CER file.

Graphical user interface, text, application

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

Graphical user interface, text, application, email

Description automatically generated

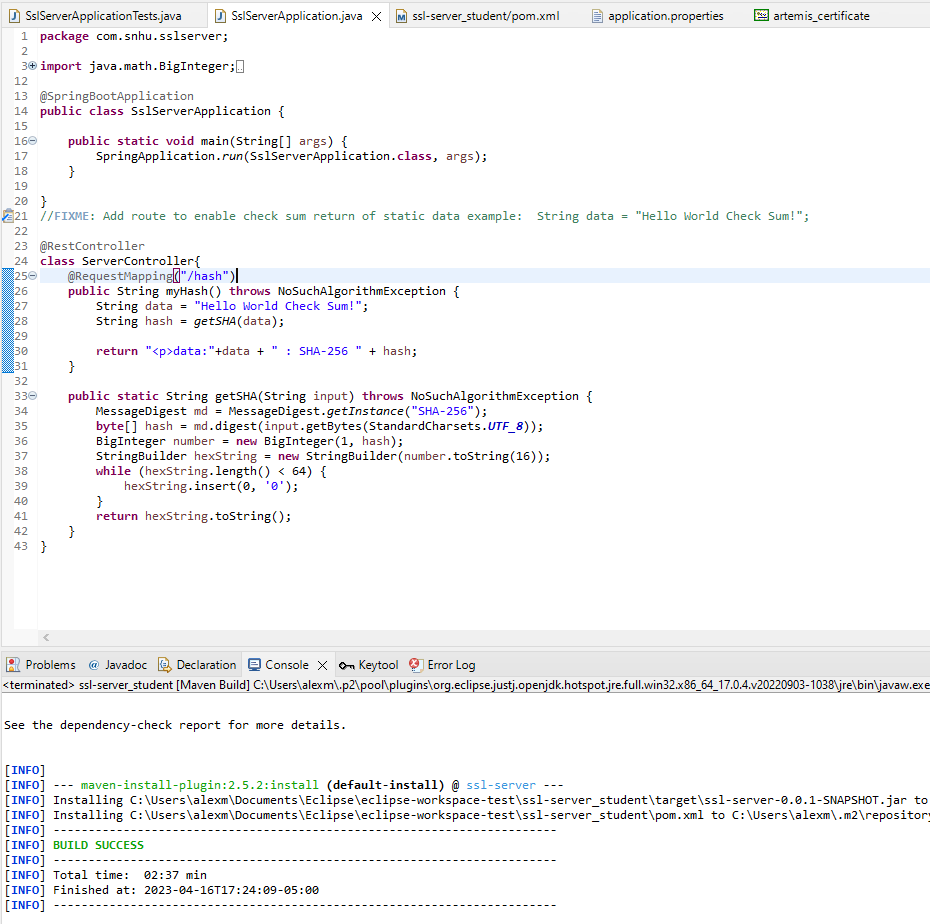
## Secure Communications

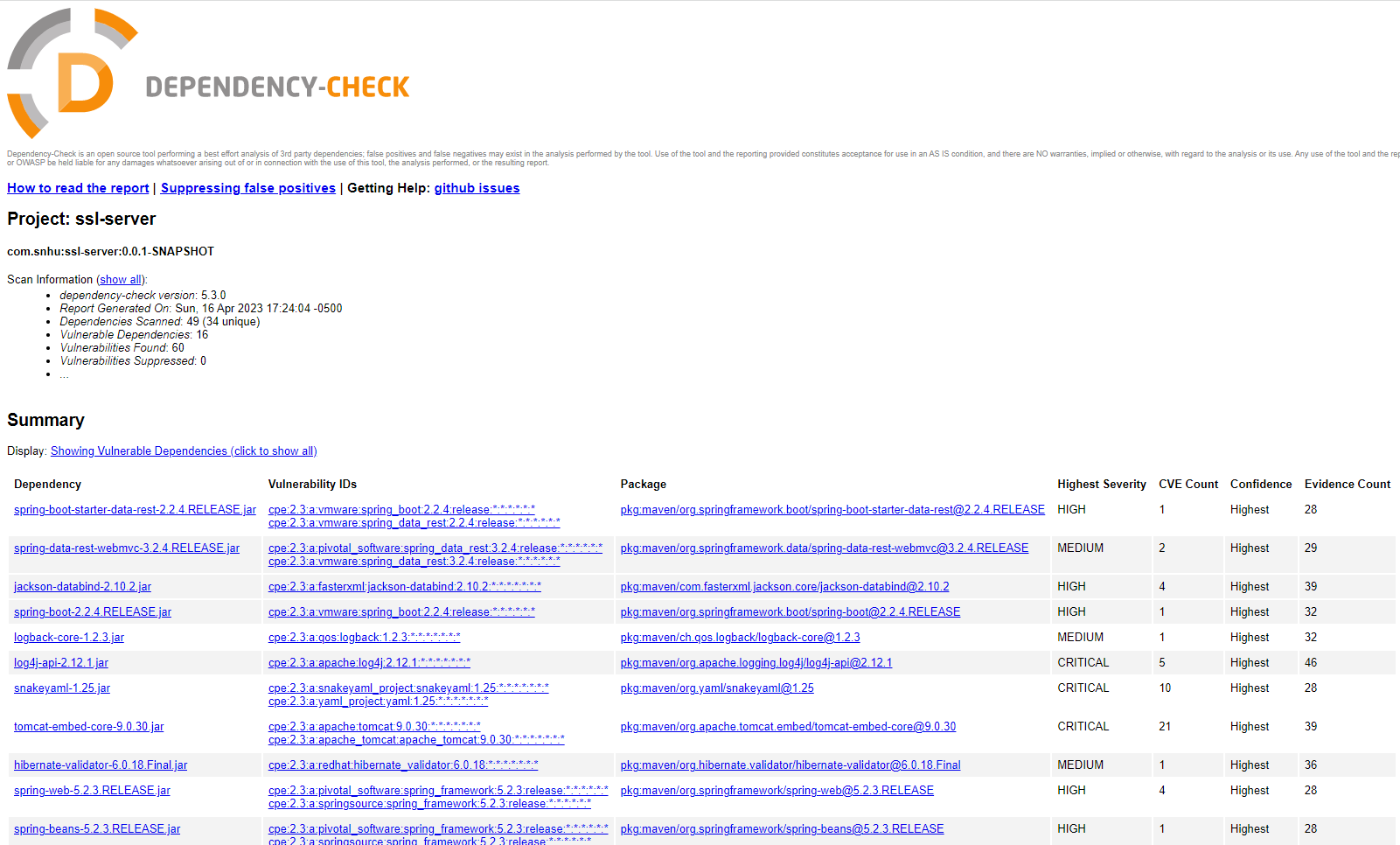
Insert a screenshot below of the web browser that shows a secure webpage.

I cannot. From what I can tell, my browser does not trust my certificate. I tried adding my certificate to Chrome’s list of trusted certificates, but that didn’t work, either.

## Secondary Testing

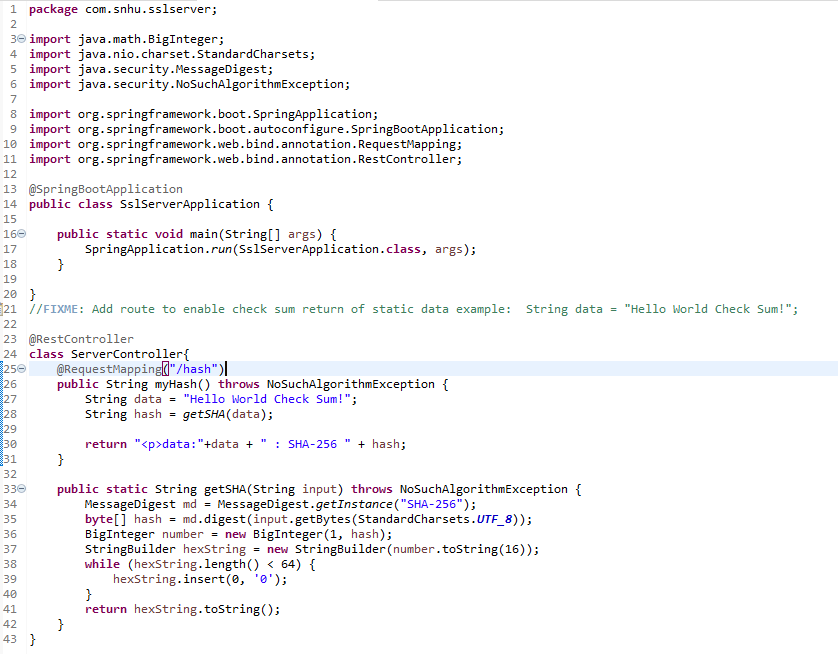
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



The major vulnerabilities seem to be the hardcoded data string in SslServerApplication.java and the password in application.properties. All someone would have to do is look at the source code and they could completely bypass the hash function and get the certificate’s password.

## Summary

The main security addition was the self-signed certificate, allowing for HTTPS to be used. This was done by first installing the Keytool plugin for eclipse, allowing me to generate the certificate from my IDE. With the keytool providing a “Create a new Certificate” button, it was just a matter of following prompts to create it. Once the certificate was created, I could add the relevant information (alias, password, keystore, and keystore type) to application.properties.

Another thing done was adding a hash function to encode the data we want to transmit. We then use the checksum to verify that it’s working properly.

The last thing is patching remaining vulnerabilities. By running a dependency check report, we can find the vulnerable dependencies. From there, they can be fixed depending on the issue, such as by performing an update. Unfortunately, I don’t actually know how to fix them. For instance, log4j-api-2.12.1.jar has an issue that is fixed in v2.12.3 and v2.13.1, but I don’t know what I would do with the downloaded .jar file afterwards.

## Industry Standard Best Practices

One of the best practices is ensuring everything is up to date. As my above example shows, many vulnerabilities get fixed in later updates, so previous exploits cannot be used by attackers. Another method of security that doesn’t apply to this application, but would apply to Artemis Financial’s final product, would be to set user privileges. A client does not need access to the same privileges that an admin does.