****

# 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** | **11 December 2022** | **Ethan Klukkert** |  |

## Client



## Developer

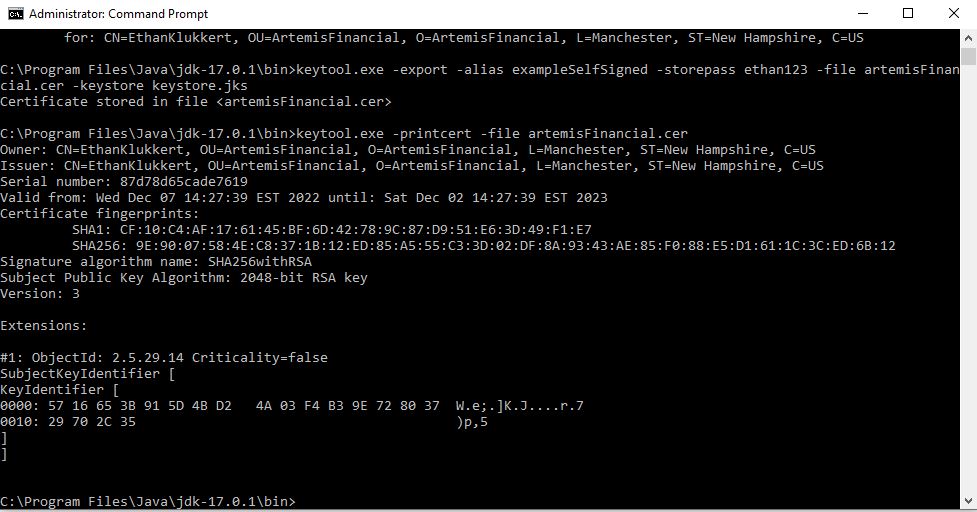
Ethan Klukkert

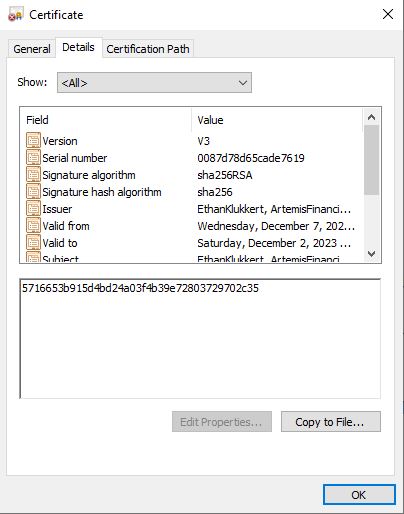
## Algorithm Cipher

Artemis Financial will use the Advanced Encryption Standard, or AES, algorithm cipher to encrypt files and communications over their application services. AES is a symmetric algorithm cipher, so two communication ports need to have a pre-arranged encryption key (Technopedia 2022). AES will also be accompanied by a hash function called SHA-256, which is a 256-bit hash function that returns a unique hash value for a document. AES will be used for a communication channel, where encrypted data can be later decrypted by a receiver. The application will use the 256-bit level for maximum security, especially since the application handles sensitive personal financial information. The cipher also has no known exploits or vulnerabilities that may make it exposed to attacks (Dutta 2021). Once data is received, it must be verified to ensure that no data has been altered. This is where SHA-256 will be used for the application to ensure that data is not altered while in transit, even if it has been encrypted by AES. SHA-256 uses a 256-bit level hash function to return 64 hexadecimal characters. Since the algorithm returns a constant number of 64 characters, there is theoretically another document that can produce the same value. However, this means that there are 2 to the power of 256 different combinations to go through in order to find the original message or another message that produces the same result (Technopedia 2022). Even though it is technically possible, it is practically impossible, therefore making it the most secure hash algorithm in use today. So, a message can generate a unique string of characters that can be used to compare to the hexadecimal value received versus the value it should be. This “value that it should be” is obtained by a certificate authority (Vitale 2017). A self-signed certificate and cipher is implemented in this document to demonstrate how a unique hash value is generated and data is encrypted to ensure data is secure and untampered.

## Certificate Generation

Here, this is an output of a self-signed certificate demonstrating how the SHA-256 algorithm is used to generate a self-signed certificate.





The image above displays some of the details of the certificate generated. This certificate file is attached to this report in a zip file.

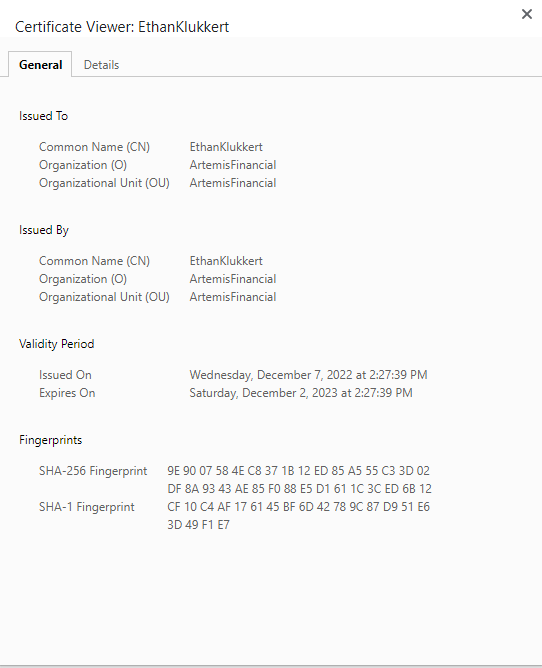
## Deploy Cipher

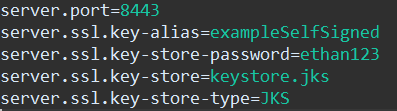
## This screenshot shows the cipher being deployed with a checksum verification.

## 

## Secure Communications

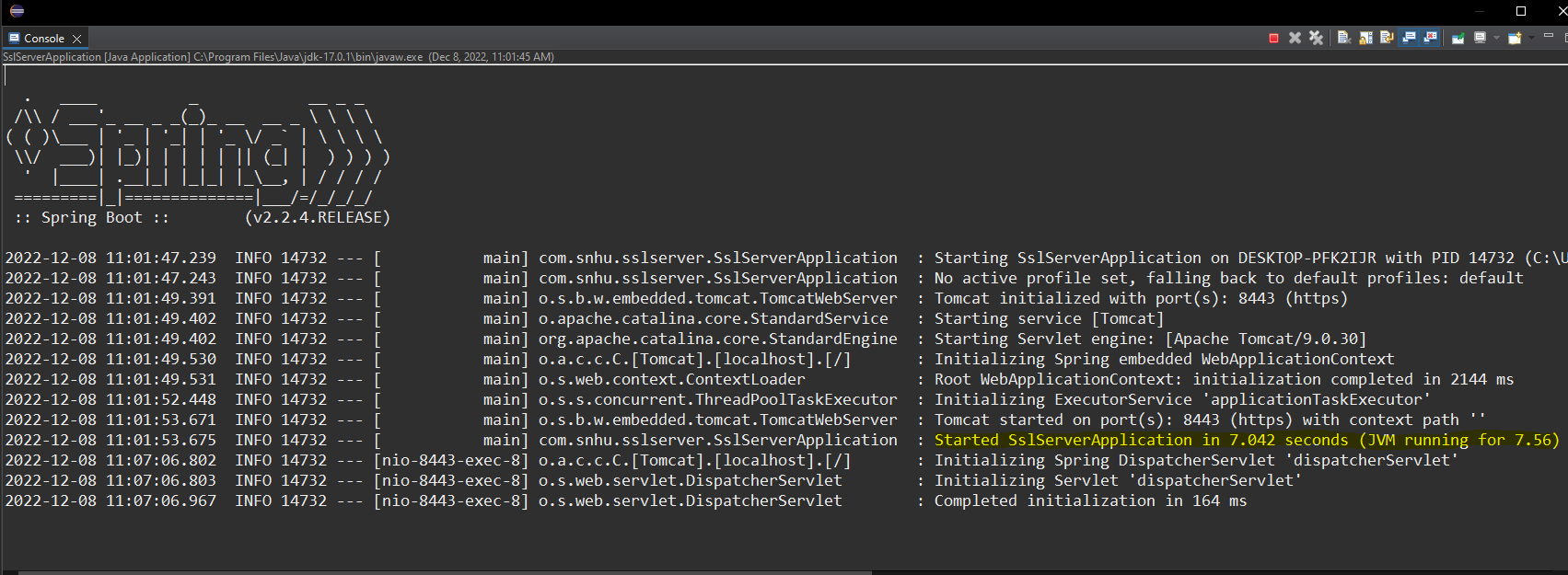
The connection itself must be secured using a valid certificate. The example above uses a self-signed certificate, which should not be used in production for a secure connection. A secure certificate can be obtained for free from Certbot (<https://certbot.eff.org/instructions?ws=apache&os=windows>) by installing it on the server’s machine and running the installer. This needs to be done on a live server other than localhost servers (Vitale 2017).





This image above shows the settings for enabling HTTPS on a server, where the keystore.jks file is linked within the project files (see attached code).

## Secondary Testing



The original server running the operations above can be seen running without errors in this console window. Also, a preliminary dependency check was conducted to note the pre-existing vulnerabilities in the application (see attached for full file).



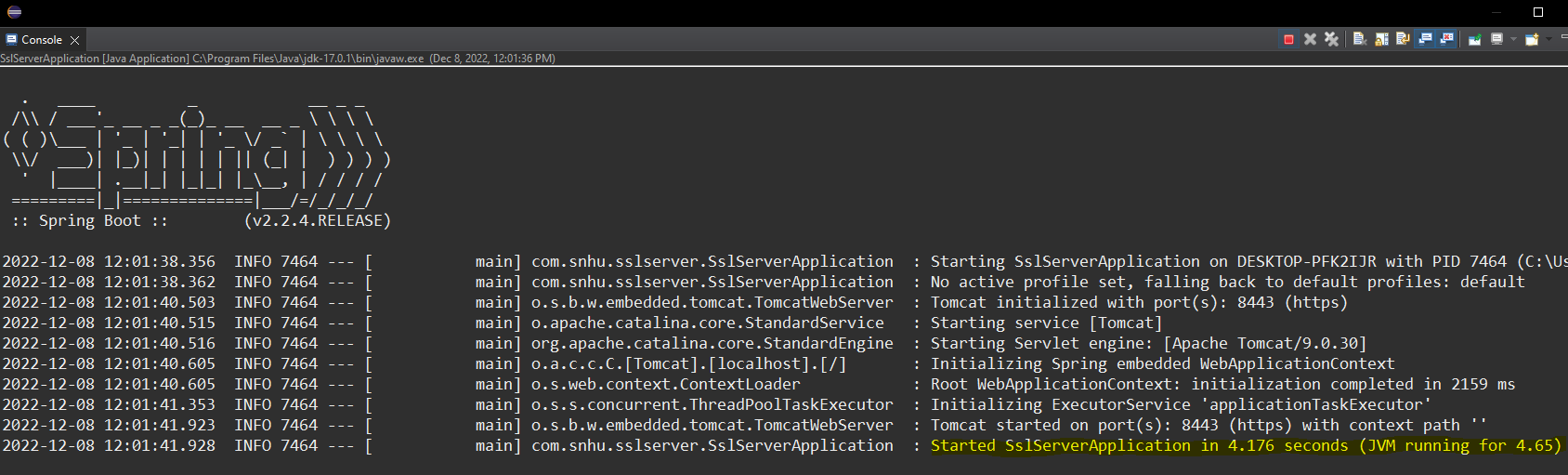
Then, the dependency check was rerun after the code was refactored.



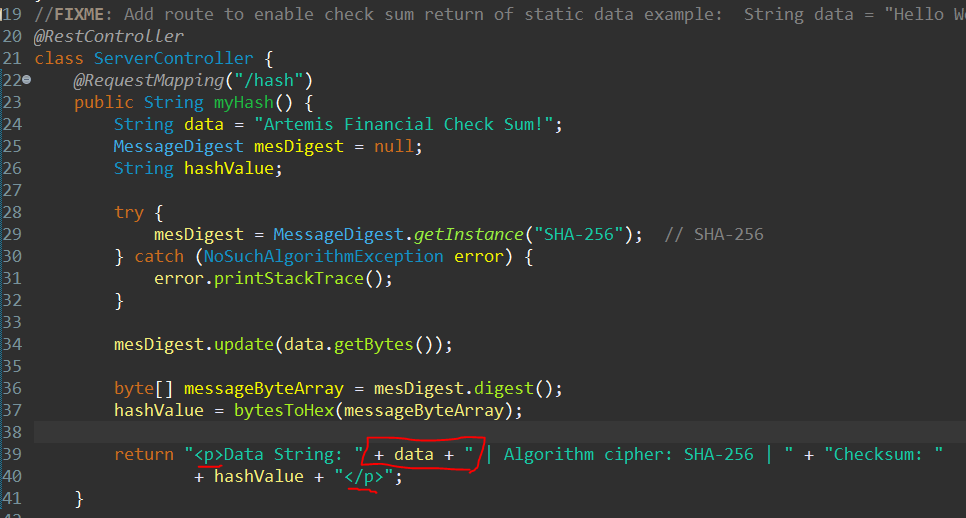
As the report states, no “new” vulnerabilities have been introduced to the application. Of course, there are still vulnerabilities to be mitigated or suppressed, but the number of vulnerabilities has not been increased due to the code being refactored (see attached html reports).

## Functional Testing

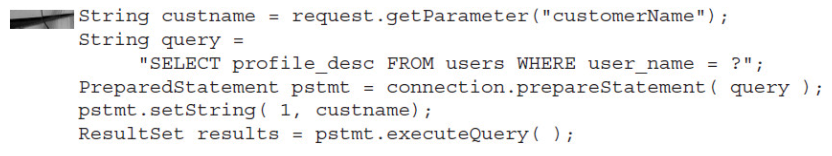
For functional testing, first the code is run without error as seen in the image below:



Then the code is examined for any logical and security vulnerabilities. Since the application is small at this stage, it is not evident that there are many potential issues. However, there is one issue that stands out, and that is at line 39 of the SslServerApplication.java file, where data is directly appended to the html! This allows attackers to input raw html into the browser, and so this means the attacker can put any harmful JavaScript code into html.



Also, the attacker can even reroute the user to any untrusted website through JavaScript. Even SQL injection can be implemented here with server access. Clearly this can be a serious issue for the user and application, and so the solution to this problem is parameterization and input validation (Manico 2014). A prepared statement in Java can prevent any untrusted data from escaping the original intentions of the input handler code:



Credit: *Iron-Clad Java, Chapter 7,* Jim Manico et al., September 2014

## Summary

In this project report, I refactored code to meet proper security protocols by securing connections, securing data reliability, and preventing code from becoming compromised in the application server. These particularly emphasized the importance of Input Validation, Cryptography, Client / Server security, and Code Quality. In the certificate generation section, we implemented a way for users to ensure that the connection to the application was legitimate and can be trusted. This makes sure that the user can send data to the right service. In the first section, we discussed how data can be transferred reliably across an unsecure network. This enables the user and application to send data to each other without fear of the data being intercepted and tampered with. Lastly, Input Validation and Code Quality were demonstrated using parameterization to ensure that untrusted data sent to the server cannot harm the server or users connected to the server. The data is sequestered in secure coding practices.

## Industry Standard Best Practices

We achieved several industry standards for web security by implementing several key actions. First, a certificate is established from a third-party CA to ensure that users and the application are actually communicating with each other, and no one else. Second, cryptographic ciphers are used to make data unreadable from eavesdroppers. This is also paired with a checksum verification to ensure that the data is not changed even in the slightest by comparing the hexadecimal output of the document to another instance of the hexadecimal output. Lastly, server code is ensured to have parameterization for input data. This means that even when a user is verified through authentication and authorization, their data cannot be used to cause harm such as SQL injection and Cross-Site Scripting (Manico 2014). In summation, the server can connect to users securely, send data securely, and even if the user is malicious the server nor users should be damaged.

References

Technopedia (August 2022). *What is 256-Bit Encryption?*  <https://www.techopedia.com/definition/29703/256-bit-encryption#:~:text=256%2Dbit%20encryption%20is%20refers,by%20even%20the%20fastest%20computers>.

Dutta, S. (Jan 2021). *This Unsolvable Problem is Worth Billions of Dollars* <https://medium.com/the-wisest-friends/this-unsolvable-problem-is-worth-billions-of-dollars-4fc28b876e03#:~:text=SHA%2D256%2C%20which%20is%20a,how%20is%20it%20so%20secure%3F>

Manico, J. et al. (September 2014) *Iron-Clad Java: Building Secure Web Applications.* McGraw Hill Computing.

Vitale, Thomas (July 2017). *How to enable HTTPS in a Spring Boot Java application* <https://www.thomasvitale.com/https-spring-boot-ssl-certificate/>