****

# 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** |
| --- | --- | --- | --- |
| **2.0** | **December 10, 2023.** | **Tinisha Cain-Beckford** | **Secure Software Report** |

## Client



## 

## Developer: Tinisha Cain-Beckford

## Algorithm Cipher

Artemis Financial is considering encrypting its archive files. Encrypting long-term archive files is a crucial step in safeguarding sensitive financial data. Employing best practices is essential to ensure robust security and adopting a strong encryption algorithm is paramount.

The Advanced Encryption Standard (AES) algorithm cipher is a symmetric-key encryption standard widely recognized for its security and efficiency. AES employs various cryptographic components, including hash functions and bit-level operations, to ensure robust encryption. The cipher uses different key sizes—128, 192, or 256 bits—to encrypt data, providing varying levels of security. These bit levels determine the complexity of the encryption and the resistance against brute-force attacks, with larger key sizes offering higher security due to increased possible key combinations.

AES is currently the industry standard in secure encryption, making it the best cipher choice for a financial company such as Artemis Financial. Factors like computational efficiency, ease of implementation, and hardware support also make it an excellent choice for long-term archiving. The symmetric 256-bit key ensures that the plain text data is logically encrypted using a combination of 2256 possible key values. This makes it essentially impenetrable even by a supercomputer using brute force attacks.

SHA-256 is a cryptographic hash function that produces a fixed-size output of 256 bits. It's extremely hard to find two different pieces of data that result in the same “fingerprint”, which makes accidental matches (collisions) very unlikely. This tool is used to check if data has been changed, to sign digital documents securely, and to maintain the authenticity of information in secure systems.

## Certificate Generation

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Terminal-** Server.cer file

A computer screen with white text

Description automatically generated

## Deploy Cipher

**SslServerApplication.java**

A computer screen shot of a program code

Description automatically generated

Connection to the server was made.

A screenshot of a computer

Description automatically generated

## Secure Communications

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

**Application.properties**

A screenshot of a computer

Description automatically generated

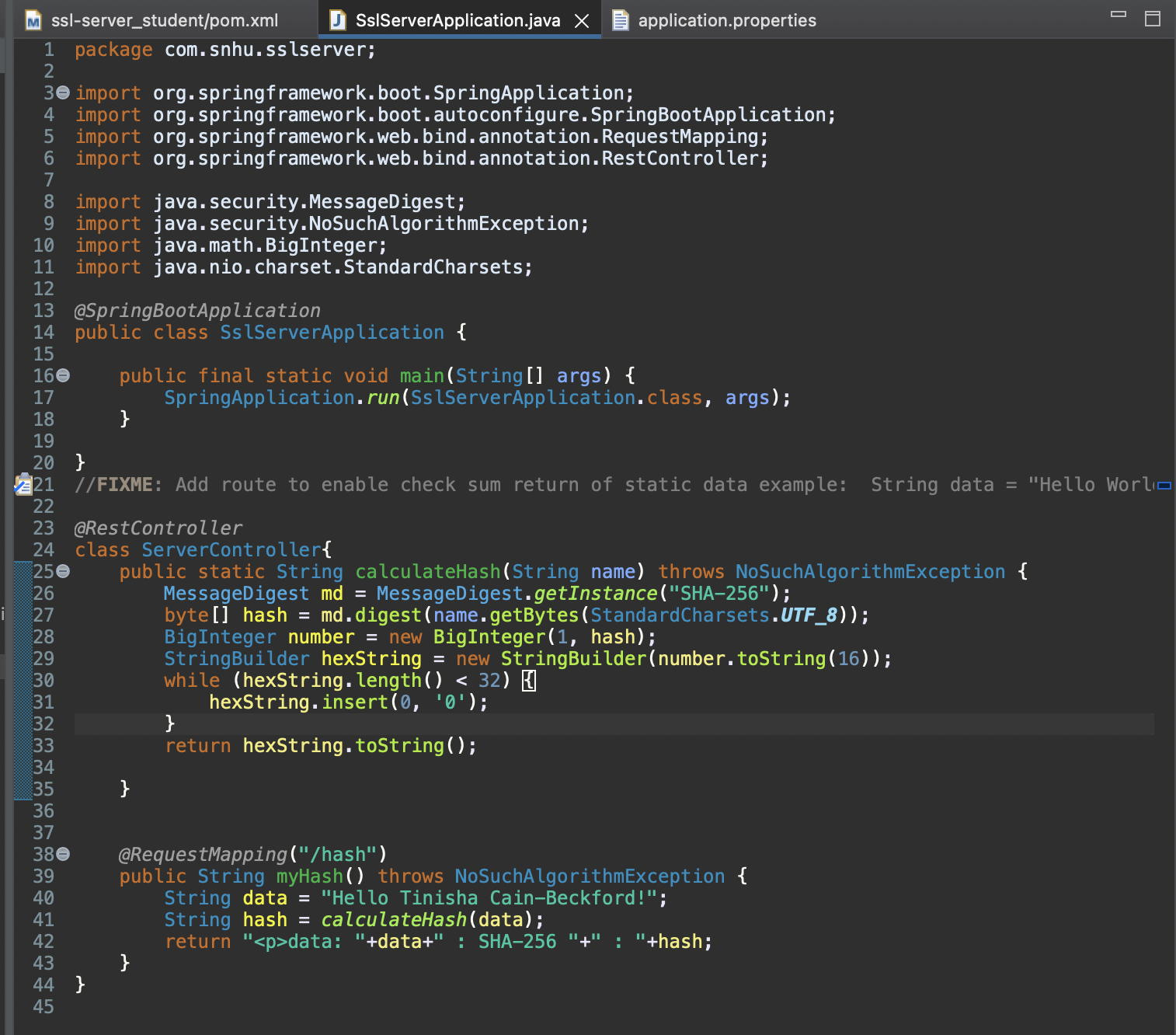
[**https://localhost:8443/hash-**](https://localhost:8443/hash-)Due to my OS, I was unable to connect to the server.

A screenshot of a black screen

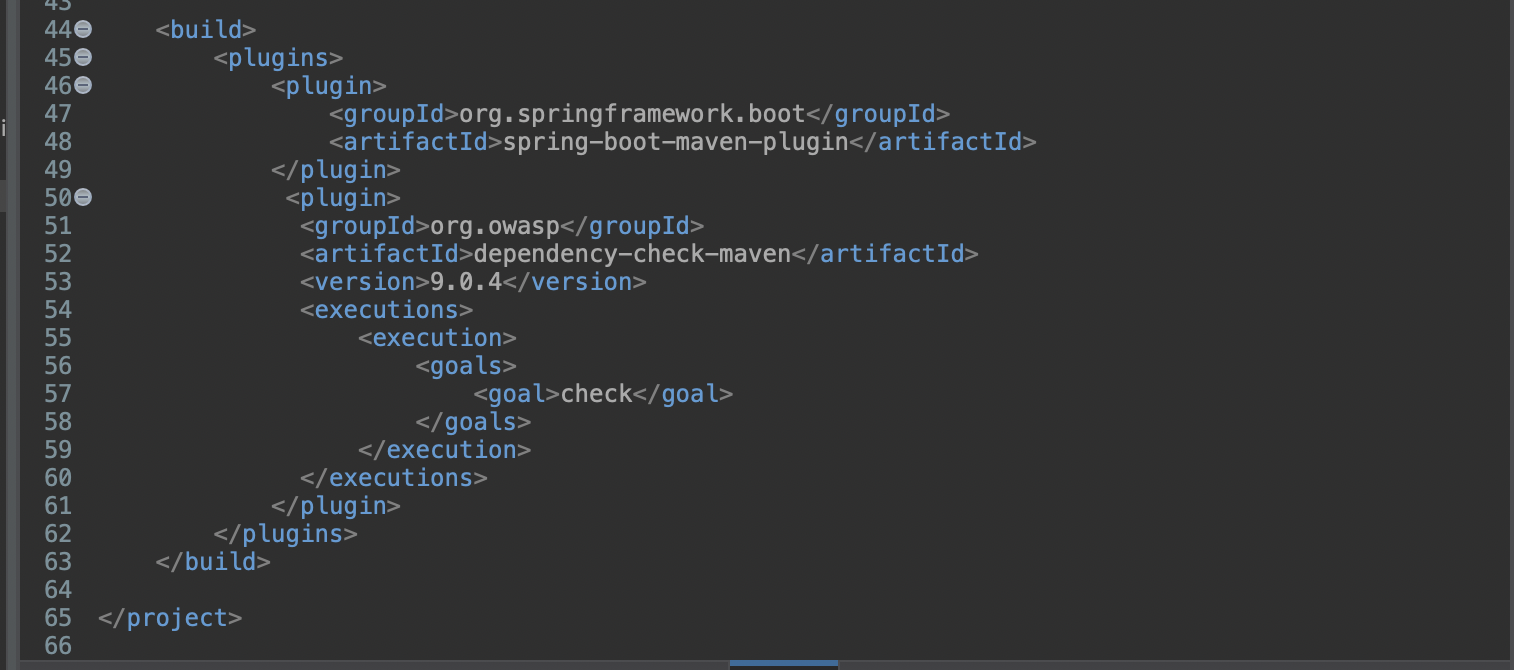
Description automatically generated

## Secondary Testing

**SslServerApplication.java-** A checksum was added to verify data integrity. It ensures that data is not tampered with.



**Pom.xml-** The current version of the Owasp Maven Dependency Check was added.



**Application.properties**

**A screenshot of a computer

Description automatically generated**

**Dependency-Check 1:**

<file:///Users/nisha/Desktop/Git-repos/Java/ssl-server_student/target/dependency-check-report.html>

A screenshot of a website

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

**Pom.xml-** I refactored the pom.xml file by adding a suppression file configuration.

A computer screen shot of text

Description automatically generated

**Suppression.xml –** To filter the false positives in the initial software dependency check, I suppressed dependencies that had incorrect CPE values and placed them in a suppression file. I determined it was false because the vendor, product, and version of the software being used was not the one listed as vulnerable.

A screen shot of a computer

Description automatically generated

**Dependency-Check 2:** <file:///Users/nisha/Desktop/Git-repos/Java/ssl-server_student/target/dependency-check-report.html#header-suppressed>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Summary

My approach to fortifying the client's software application involves a meticulous process encompassing Cryptography, Code Quality, and APIs. Cryptographic hash functions serve as invaluable tools for secure information sharing, forming the backbone of various cryptographic operations. These functions facilitate digital fingerprinting, commitment schemes, message authentication, digital signatures, stream ciphers, and more. Employing checksum verification guarantees the integrity of transmitted data, bolstering security measures. Furthermore, I reinforced security by implementing a self-signed certificate authority and establishing a secure server connection via HTTPS, enhancing overall protection against potential threats.

Code quality stands as the bedrock for developing resilient software, particularly seamless API integration. Well-structured, clean code streamlines the implementation of robust input validation mechanisms, effectively thwarting security vulnerabilities and data integrity compromises resulting from incorrect or malicious inputs. Moreover, high-quality code significantly influences the efficiency and manageability of API integration, reducing the likelihood of bugs, errors, and compatibility issues. To ensure code quality, I conducted a static test using a Maven dependency check, created a suppression file to mask false positive vulnerabilities, and meticulously reviewed application files to align with best practices.

Guaranteeing the integrity of data access orbits on secure API interactions. Rigorous implementation of stringent measures ensures that data remains accessible solely to authorized entities, minimizing the risks associated with unauthorized access. This comprehensive approach across Cryptography, Code Quality, and API security serves to fortify the software application, providing robust protection and reliability against potential security threats.

## Industry Standard Best Practices

To ensure the security of the software application, I employed industry-standard best practices across various key areas. Utilizing checksums allowed for data integrity verification, reducing the risk of malicious files compromising the system. Implementing robust server connections and hash functions prevented data leaks, preserving the confidentiality of sensitive information. Additionally, access control measures were rigorously enforced, restricting unauthorized access to critical system components, and maintaining a secure user interaction from the outset.

Adhering to industry-standard best practices in secure coding isn't merely a technical necessity; it's fundamental to Artemis Financial’s overall well-being. By prioritizing these practices, we establish a robust shield against potential security breaches, safeguarding invaluable assets like user data and system integrity. This approach bolsters the company's reputation and instills trust among users as well as underlining that we take their security and privacy seriously, cultivating a strong relationship built on trust. Furthermore, in today's digital landscape, where reports of security vulnerabilities spread rapidly, the perception of a secure system is crucial in preserving the company's credibility and market standing. Embracing these best practices thus becomes a cornerstone of the company's long-term success and resilience in an ever-evolving technological landscape.

**References:**

1. ClickSSL. (2023, July 18). 256-Bit *Encryption – Is AES 256-Bit Encryption Safe in Modern Times?* ClickSSL.net. https://www.clickssl.net/blog/256-bit-encryption
2. *Data encryption in finance*. (n.d.). Apricorn.com. https://apricorn.com/data-encryption-in-finance
3. Franklin, R. (2020, March 13). *AES vs. RSA Encryption: What Are the Differences?* Precisely.com. <https://www.precisely.com/blog/data-security/aes-vs-rsa-encryption-differences>
4. *How Do Banks Encrypt Data? - Newsoftwares.net Blog*. (2023, September 3). Www.newsoftwares.net. https://www.newsoftwares.net/blog/how-do-banks-encrypt-data/
5. *Java Security Standard Algorithm Names*. (n.d.). Docs.oracle.com. <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>
6. Probasco, L. (2017, April 25). *Encryption Requirements for Banks & Financial Services*. Townsendsecurity.com. https://info.townsendsecurity.com/encryption-requirements-for-banks-financial-services
7. *6 Protecting Sensitive Data - Iron-Clad Java [Book]*. (n.d.). Www.oreilly.com. https://learning.oreilly.com/library/view/iron-clad-java
8. *Secure Hash Algorithm - an overview | ScienceDirect Topics*. (n.d.). Www.sciencedirect.com. https://www.sciencedirect.com/topics/computer-science/secure-hash-algorithm
9. *Secure Hash Algorithms | Brilliant Math & Science Wiki*. (2010). Brilliant.org. <https://brilliant.org/wiki/secure-hashing-algorithms/>
10. *What is SHA? | SHA-1 | SHA-2 | Encryption Consulting*. (2020, September 23). https://www.encryptionconsulting.com/education-center/what-is-sha/#:~:text=SHA%20stands%20for%20secure%20hashing

‌

1. *keytool-Key and Certificate Management Tool*. (n.d.). Docs.oracle.com. https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

**‌**

1. Poettering, B., & Stebila, D. (2017). Double-authentication-preventing signatures. *International Journal of Information Security*, *16*(1), 1–22. https://doi-org.ezproxy.snhu.edu/10.1007/s10207-015-0307-8