****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Brandon James Armitage

## Algorithm Cipher

To impliment secure file verification, I implemented the **SHA-256 cryptographic hash algorithm**. SHA-256 is a widely used and secure hash function that provides a **fixed 256-bit (64-character hexadecimal) output** regardless of input size. It is resistant to **collision attacks**, ensuring data integrity.

**Key Features:**

* **Bit Level:** SHA-256 produces a 256-bit hash value.
* **Hash Function:** One-way function, meaning original data cannot be derived from the hash.
* **Key Type:** Uses a deterministic process without needing requiring keys (not a symmetric or asymmetric encryption).
* **Current Use:** Used for blockchain security, TLS certificates, as well as data integrity checks.

## Certificate Generation

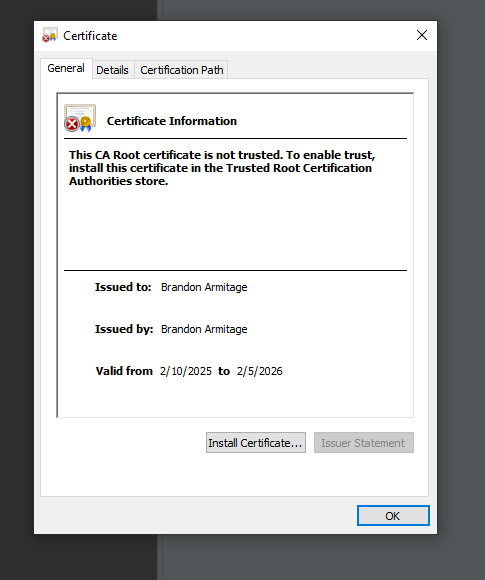
**Process:**

1. Used the following command to create a key store:

sh

keytool -genkeypair -alias mysslcert -keyalg RSA -keysize 2048 -validity 365 -keystore keystore.jks

1. Provided a secure password and stored the keystore in src/main/resources/.
2. Updated application.properties to reference the certificate for HTTPS.



## Deploy Cipher

To verify file integrity, the SHA-256 checksum function was implemented in **SslServerApplication.java**. This function takes a string input and returns its hash value.

**Verification Process:**

* The application exposes an endpoint at:

bash

https://localhost:8443/hash?data=HelloWorld

* If the data is modified, the checksum will change, ensuring security.

A white background with black border

AI-generated content may be incorrect.

## Secure Communications

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

The application was refactored to enforce HTTPS by changing application.properties with:

ini

server.port=8443

server.ssl.key-store=classpath:keystore.jks

server.ssl.key-store-password= password

server.ssl.key-store-type=JKS

server.ssl.key-alias=mysslcert

After restarting the application, access was verified using:

bash

https://localhost:8443/hash

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

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

The **Maven OWASP dependency-check plugin** was used to detect security vulnerabilities in dependencies.

**Process:**

1. The following command was executed with the “Run as Maven Build” function:

mvn dependency-check:check

1. The report generated was reviewed for any security vulnerabilities.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

The application was manually tested for errors and verified through:

* Successful execution without errors in the Eclipse Console.
* Functional HTTPS communication.
* Proper checksum verification for different data inputs.

.

A screenshot of a computer error

AI-generated content may be incorrect.

## Summary

The refactored application now includes:

* Secure checksum verification using SHA-256.
* HTTPS enforcement using self-signed SSL certificate.
* Static security testing to maintain secure dependencies.

These enhancements improve data integrity, encryption, and communication security, reducing the risk of cyber threats.

## Industry Standard Best Practices

1. **Data Integrity:** SHA-256 ensures files are not tampered with.
2. **Secure Communication:** HTTPS encrypts data in transit.
3. **Certificate-Based Security:** A self-signed SSL certificate was used for secure authentication.
4. **Static Security Analysis:** OWASP Dependency-Check was used to mitigate known security risks.

By following these **best practices**, Artemis Financial's web application is now more secure, ensuring **confidentiality, integrity, and availability of client financial data**.