****

# 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** | **6/20/25** | **Jordan Landry** | **Initial steps** |

## 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

Jordan Landry

## Algorithm Cipher

Artemis Financial’s primary goal is to deliver secure financial programs to clients around the world. To support this objective, I recommend implementing the **SHA-256** encryption algorithm. SHA-256 is widely recognized for its strength and reliability—it is virtually impenetrable through brute-force methods, making it an ideal choice for protecting sensitive financial data from unauthorized access.

**Why SHA-256?**  
 SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that processes input data into a fixed-length hash value. This function compresses the input, producing what is known as a hash value. The strength of SHA-256 lies in its 256-bit structure, which enables an enormous number of possible combinations, making it extremely difficult for attackers to guess or crack. The algorithm’s built-in randomness ensures high levels of unpredictability, further enhancing its security.

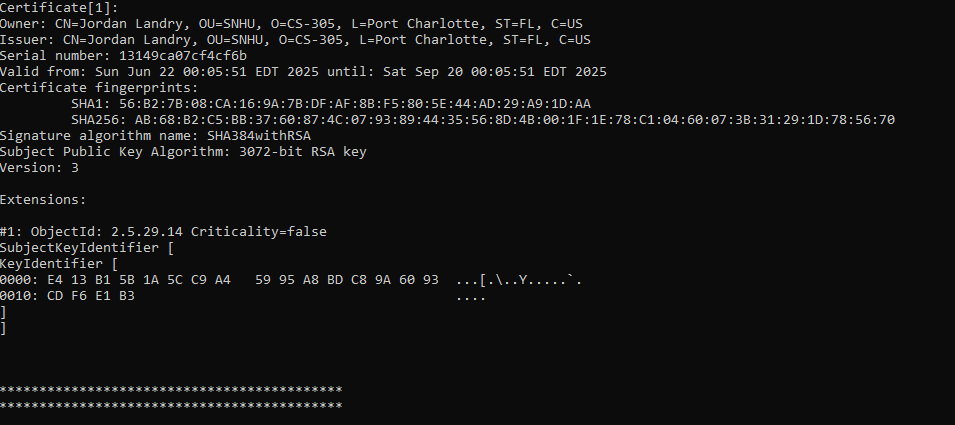
**Symmetric vs. Asymmetric Encryption**  
 Encryption generally falls into two categories: **symmetric** and **asymmetric**.

* **Symmetric encryption**, such as **AES-256**, uses a single key for both encryption and decryption. Its key advantage is efficiency—it requires less processing time and is simpler to implement (Yedakula, K., 2019).
* **Asymmetric encryption**, on the other hand, involves two keys: a public key for encryption and a private key for decryption. While more complex, it is often considered more secure and is commonly used for secure online communication.

**Historical Context**  
 Encryption techniques date back to around 600 BC, and the importance of protecting information has only grown with time (A Brief History of Encryption, Thales Group, 2016). As cybersecurity threats evolve, strong encryption like SHA-256 ensures that Artemis Financial can maintain the confidentiality and integrity of client data in a digitally connected world.

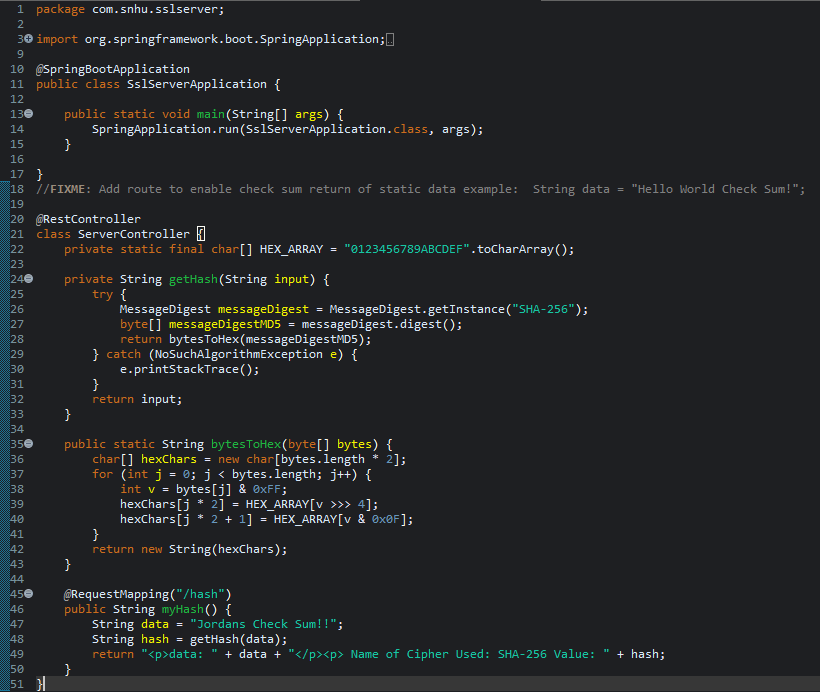
## Certificate Generation

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.

[Insert screenshots here.]

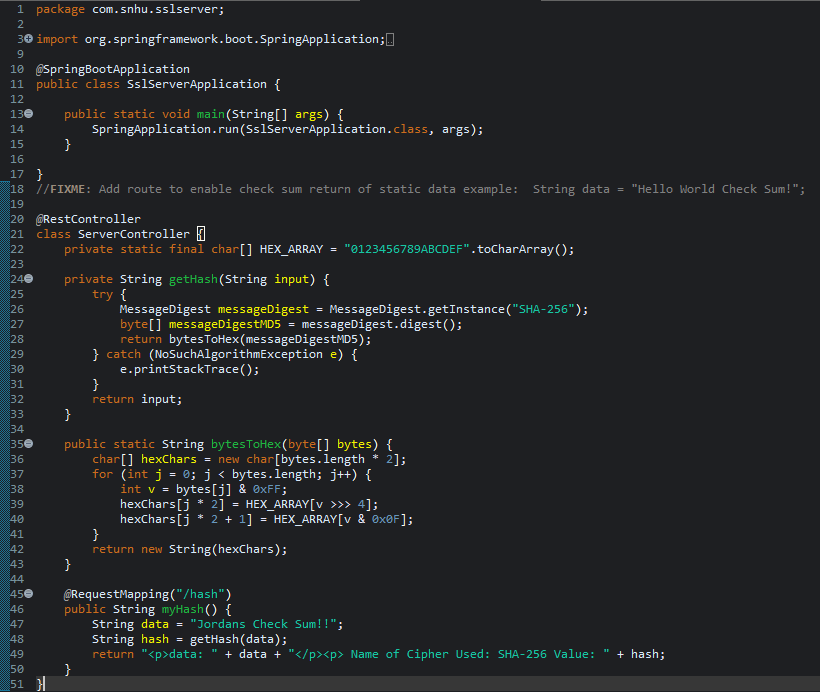
## Secondary Testing

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

[Insert screenshots here.]

## Functional Testing

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



## Summary

To improve our application's security and meet testing protocols, we made several important changes based on the **Vulnerability Assessment Process Flow Diagram**. These changes focused on secure communication, fixing known vulnerabilities, and protecting user data.

The first major update was adding **self-signed certificates** so the app could use **HTTPS**. This ensures secure communication between users and our server, helping users know they’re on our real site—not a fake one. This builds trust and protects the company’s reputation.

Next, we **refactored the pom.xml file** to fix all issues found during the **dependency check**. This removed outdated or risky libraries that could be exploited, keeping the app safer.

We also **verified the hashing function** with a checksum to make sure it scrambled user data correctly. This makes it much harder for anyone to recover or misuse that data.

Finally, we **patched all known vulnerabilities**, making sure the app and its parts are secure and up to date.

To keep the app secure going forward, we recommend these best practices:

* **Apply regular updates and patches** to fix new issues quickly.
* **Use the principle of least privilege**, giving users only the access they need.
* **Keep scanning for vulnerabilities** and include security checks in future development.

These security layers protect user data, build trust, reduce risk, and support the long-term health of the business.

## Industry Standard Best Practices

To maintain the security of the software application and address known vulnerabilities, I applied industry-standard secure coding practices and implemented key measures to ensure long-term protection. These practices are essential not only for protecting sensitive data but also for meeting regulatory requirements, reducing operational risk, and preserving the organization’s reputation. The following security measures should be maintained as part of the application lifecycle:

* **Input validation and sanitization** to prevent attacks such as SQL injection, cross-site scripting (XSS), and command injection.
* **Secure authentication and password management**, including strong password policies, hashing algorithms, and multi-factor authentication (MFA).
* **Principle of least privilege**, ensuring that users and applications only have the access necessary to perform their roles.
* **Secure data storage and transmission**, using encryption for data at rest and secure communication protocols like HTTPS.
* **Regular updates and patching** of the application and its dependencies to address known vulnerabilities promptly.
* **Proper error handling and logging** to avoid leaking sensitive information and to monitor for security threats.

By following these best practices, the company can safeguard its systems, build customer trust, reduce costs associated with data breaches, and strengthen its overall brand reputation.