****

# 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** | **2/22/2024** | **David Gerardi** |  |

## 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

David Gerardi

## Algorithm Cipher

I recommend using SHA-256, which stands for “Secure Hash Algorithm” for Artemis Financial. SHA-256 was created in 2001 and was designed by the United States National Security Agency. SHA-256 is commonly used for things like data transmission, password storage, and digital signatures. Using SHA-256 we can convert any amount of information into a secure 256bit unique hash. Since there are 256 bits in the hash, there are over 1.18 \* 10^77 different combinations. The chances of it ever being brute forced are astronomically slow. SHA is neither symmetric or asymmetric because it is a hash function. Atemis Financial asked for data verification in the form of a checksum, so a hash algorithm is required not full encryption.

## Certificate Generation

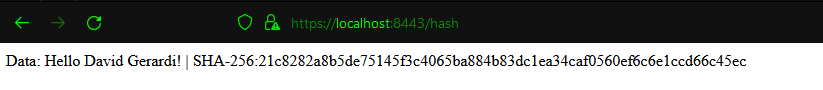
Insert a screenshot below of the CER file.

A screenshot of a computer program

Description automatically generated

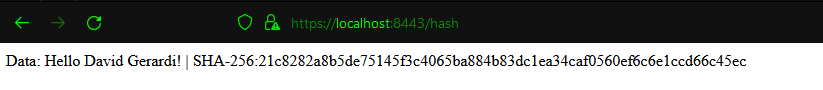
## Deploy Cipher

Insert a screenshot below of the checksum verification.



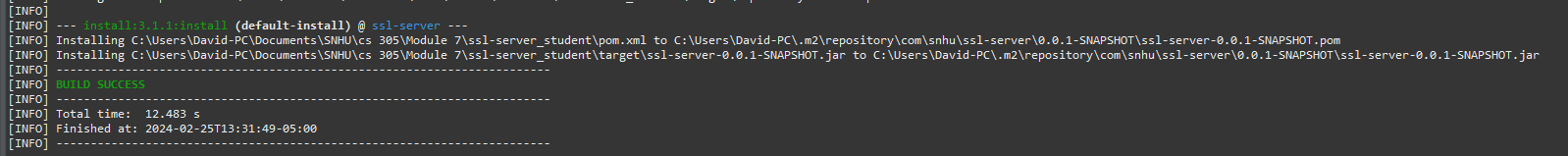
## Secure Communications

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



## Secondary Testing

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



A screenshot of a computer

Description automatically generated

## Functional Testing

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

A computer screen shot of a keyboard

Description automatically generated

## Summary

When refactoring this code, I had to pay extra attention to Cryptography, Code Quality, and Secure API interactions in accordance with the Vulnerability Assessment Flow Diagram. For cryptography, we had to generate a secure checksum where we used SHA-256. For Code Quality, we had to practice safe coding by adding the checksum, enabling HTTPS, and checking all dependencies with OWASP dependency checker. For APIs, we had to get everything working with HTTPS for a secure connection.

## Industry Standard Best Practices

We took three main measures to mitigate known security vulnerabilities. Firstly, Enabling the checksum introduced cryptography, which is extremely hard to break. Secondly, we enabled HTTPS to make the site safer and more secure. For our third measure against vulnerabilities, we manually reviewed the code for any errors, and used the OWASP dependency checker to go through all known dependency vulnerabilities and either updated them or disabled false positives.