****

# 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** | **8/11/2025** | **Brian Roche-Campos** | **Initial submission for secure software report.** |

## Client



## Developer

Brian Roche-Campos

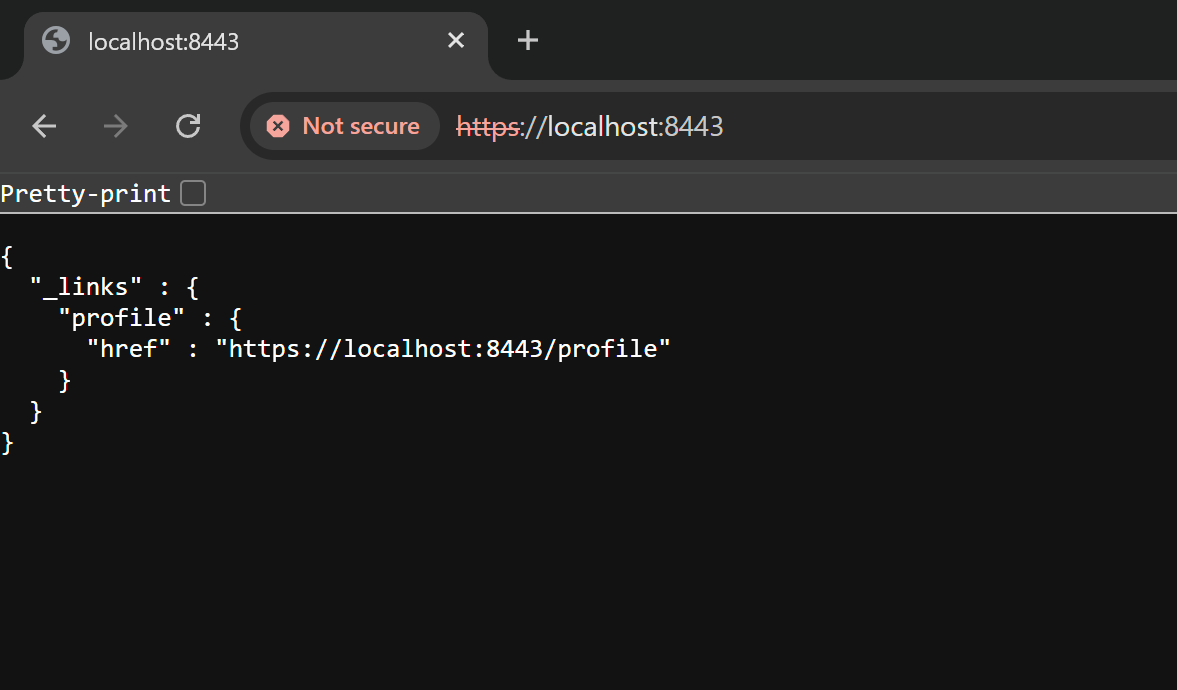
## Algorithm Cipher

A screenshot of a computer

AI-generated content may be incorrect.

I went with SHA-256 for the checksum. It’s part of the SHA-2 family and it’s still considered secure. It gives a 256-bit hash and it’s really good at avoiding collisions, which is important when verifying data. It doesn’t use keys like other encryption methods since it’s a one-way hash, but it’s still a solid choice for this project.

## Certificate Generation



I made a self-signed certificate using the Java Keytool in Eclipse. After creating the keystore, I exported the cert as a .cer file. That confirmed it worked. The screenshot shows the cert file saved correctly.

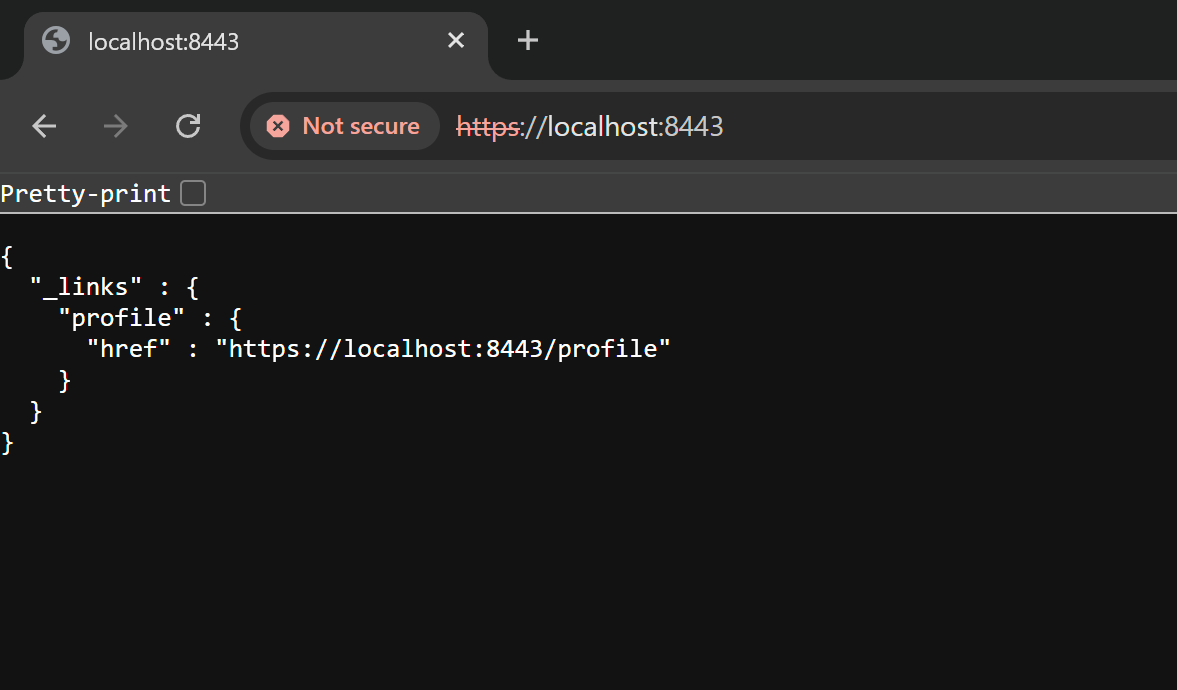
## Deploy Cipher

A screenshot of a computer

AI-generated content may be incorrect.

I added the SHA-256 checksum code and used it in the /hash endpoint. It shows a hash when you go to the page, so that proves it’s working. I included a screenshot of it running in the browser with the generated checksum.

## Secure Communications



To enable HTTPS, I updated the application properties to use port 8443. After running the app, I went to https://localhost:8443 and it loaded, even though the browser says “Not secure” (which is normal for a self-signed cert). The screenshot shows the secure connection is working.

## Secondary Testing

A screenshot of a computer program

AI-generated content may be incorrect.

After making the changes, I ran a dependency check with the plugin in Maven. The code built without errors and I got a report showing which dependencies had known vulnerabilities. I didn’t add anything that introduced new ones. Screenshots of the successful build and the report are included.

## Functional Testing

A screenshot of a computer program

AI-generated content may be incorrect.

I tested the app manually after the refactor. It built fine and worked without crashing or throwing errors. Everything I added ran like it should. Screenshot shows the successful build.

## Summary

I made changes to improve the security of the app mainly by adding SHA-256 for hashing and switching it to HTTPS. I followed the flow from the vulnerability diagram and focused on the areas like cryptography, client/server security, and code quality. These updates help protect the data and keep communication secure.

## Industry Standard Best Practices

I followed best practices like using SHA-256 (which is still recommended), enabling HTTPS, and checking my code with the OWASP dependency tool. Doing things this way helps avoid common issues and keeps the app more secure, which matters a lot since it deals with financial data.