****

# 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 5](#_Toc102040760)

[4. Secure Communications 5](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 8](#_Toc102040763)

[7. Summary 8](#_Toc102040764)

[8. Industry Standard Best Practices 9](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **February 25, 2024** | **Buddy Marcey** |  |

## 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

Buddy Marcey

## Algorithm Cipher

I will use SHA 256 for this assignment. It is a 256-bit cipher, meaning there are 2256 possible variations of hashed result. This result is found by taking the input data and altering it over multiple rounds, making it extremely difficult to reverse-engineer the original text from the hashed message. SHA 256 generates the same result from the same input prompt every time, making it an excellent candidate for checksum operations, and has an extremely low collision chance. Symmetric keys involve a secret key that can be used to encrypt data and then also decrypt it by the receiving party. Asymmetric keys involve a private key to encrypt data, but a public key to decrypt. Neither are the case for this assignment though; we simply want to generate a string for the purposes of verifying data. This string will not need to be decrypted at all; the checksum will simply need to be compared by the receiving party to make sure the communication is legitimate. SHA 256 is sufficiently strong for today’s needs, but with the growth in the realm of quantum computing, it will eventually need to be replaced with a newer, more complex cipher. The DES encryption method used to be the standard in the 1970s, but as computers became more powerful it was eventually retired as a standard in 2005.

## Certificate Generation

Insert a screenshot below of the CER file.

A computer screen with green text

Description automatically generated

Screenshot of the .cer file from the command line, with the project open in Eclipse to show that the keystore and certificate are in the project folder.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

When I refactored my code, I followed the same instructions I had for module 5, but I changed this string by adding Project 2 so that it would generate a different Checksum Value.

## Secure Communications

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

A screenshot of a computer

Description automatically generated

I ran into an issue using both Firefox and Chrome. Chrome shows the screen shot in prompt 3, with the “Not Secure” tag, but then says that it isn’t secure because it is using a self-signed certificate. Firefox allows me to make an exception, denoted by the lock with a triangle on it next to the URL. I will add an additional screenshot showing that the site is pulling my certificate and using HTTPS:

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

This first screenshot shows a dependency check from the original codebase without any alterations at all:

A screenshot of a computer error

Description automatically generated

And then this second shows the check after refactor and proper updating:

A screenshot of a computer

Description automatically generated

And finally, after suppression:

A screenshot of a computer error

Description automatically generated

## Functional Testing

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

Screenshot of the refactored code and the program running with no errors:

A screenshot of a computer program

Description automatically generated

## Summary

Several additional security functions were added to the program to make it more secure. The processes from the VPD that were directly affected were APIs, Cryptography, Client/Server, Code Error, and Code Quality. API vulnerabilities were addressed by updating the versions of Spring, Maven, and Java in the pom.xml file. Cryptography was added by implementing the checksum. Client/Server relationships were developed by use of a specific port on the host and by certificate signing. Code Error and Quality were implemented through correctly and accurately developing the required code for the application to run properly.

## Industry Standard Best Practices

Several industry standard best practices were implemented in this project. RESTful endpoints were put in place, SHA 256 encryption was used for generating a checksum, and a certificate was generated for secure communication. Dependency checks were run, and issues addressed. These elements help keep the application secure, which is vitally important for any company doing business over the internet. Data breaches can deal a crushing blow in today’s competitive business world; consequences potentially include loss of property, high mitigation costs, regulatory fines, and perhaps most importantly, loss of trust from the customer base. It is very difficult to regain that lost trust after a breach, especially if it could have been prevented through more diligence when developing the software.

**References**

Callaghan, P. (n.d.). *Why you should use SHA-256 in evidence authentication*. https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication#:~:text=Collisions%20are%20incredibly%20unlikely%3A%20There,the%20exact%20same%20hash%20value.

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java: Building secure web Applications (Oracle Press)*. http://dl.acm.org/citation.cfm?id=2826076

Vadapalli, P. (2023, August 21). What is SHA 256 Algorithm? Functions & Applications. *upGrad blog*. https://www.upgrad.com/blog/sha-256-algorithm/