****

# 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** | **12-20-24** | **Brandon Ellis** |  |

## 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 Ellis

## Algorithm Cipher

SHA-256 is a strong and reliable cipher algorithm that helps avoid data collisions. It is part of the SHA-2 family of hash functions and works by using a 256-bit key to change a piece of data into a fixed-length string of random-looking characters and numbers. This output, called a hash value, is always 256 bits long. SHA-256 is secure for three main reasons. First, it’s nearly impossible to figure out the original data just by looking at the hash value; a brute-force attack would require trying 2²⁵⁶ different combinations. Second, it’s extremely unlikely for two different pieces of data to create the same hash value, which is known as a collision.

Today, the Advanced Encryption Standard (AES) is widely trusted and used by the U.S. government and many organizations. AES is very efficient in its 128-bit version but also supports 192- and 256-bit keys for situations that need stronger encryption. Interestingly, the concept of encryption has been around for thousands of years as a way to protect information.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## 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.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screen shot of a computer program

Description automatically generated

## Summary

The refactored code improves security by fixing key vulnerabilities, following the steps in the vulnerability assessment process. Changes include switching plaintext responses to JSON format, which prevents data injection and makes responses safer. Using Base64 encoding for hashes protects data from exposure, and initializing the hashing algorithm with @PostConstruct ensures it’s ready securely at runtime. These updates align with the mitigation phase of security testing, making the code more secure and reliable.

To add layers of security, the process started by identifying potential risks and applying best coding practices. Replacing generic error messages with specific ones avoids exposing sensitive details. The use of dynamic input (@RequestParam) improves flexibility, while recommending HTTPS protects data during transfer. Structured JSON responses prevent misuse and enhance compatibility. Overall, these updates make the software stronger against common security threats and follow secure development practices.

## Industry Standard Best Practices

To keep the software secure, I followed best practices by reducing the risk of exposing sensitive data and ensuring safe communication. Instead of sending hashes in plaintext, I switched to JSON responses, which helps prevent data injection and works better with modern APIs. I also used Base64 encoding for the hashes to protect them from being exposed. By initializing the hashing algorithm at runtime with @PostConstruct, I made sure the application runs securely from the start. Additionally, I recommended using HTTPS to protect data while it's being transferred, which follows common security standards.

Using these security practices helps the company by reducing the chances of data breaches and protecting sensitive information. It also builds trust with customers and shows the company cares about security. Following these best practices makes the software stronger against common threats, which helps prevent costly security issues and protects the company's reputation. By staying ahead of potential risks, the company shows its commitment to safety, which builds trust and ensures it meets important security regulations.