****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **04/18/2021** | **Neil Doherty** | **Attached refactored program** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Neil Doherty

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

To guarantee the integrity of transferred data for Artemis Financial, a secure encryption algorithm must be utilized. The best choice, given the company’s needs, is the AES-256 encryption algorithm. AES-256 is an acronym for the Advanced Encryption Standard 256, where the integer refers to the number of bits associated with the given algorithm. It functions by taking a unique string and returning a unique hash value for that string. To decrypt the data, a key is needed. Any individual without the key cannot access the original data. Any change to the underlying data will result in a change to the hash value, showing a clear indication that the data is not as the author intended. This will provide a reliable way to verify the data and keep it secure. The string for the data contains numbers and characters that are random and only make sense when analyzed with the secret key. Randomness makes it extremely unlikely that an attacker would guess the correct combination of characters to decrypt the message.

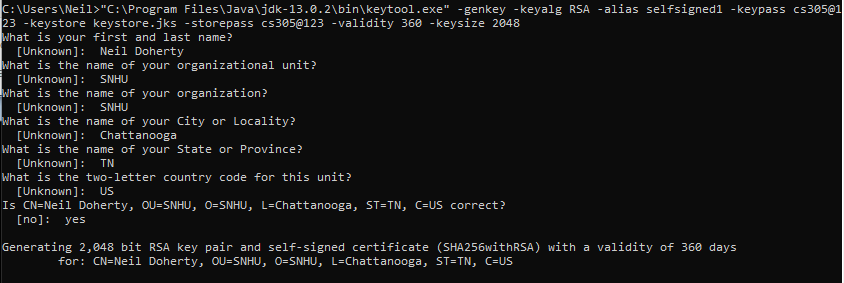
The AES algorithm is a symmetrical algorithm, meaning that both the sender and receiver need to have the key to both encrypt and decrypt the data. A non-symmetrical approach would have a public and private key, where the public key is available to anyone and the private is only made available to the intended recipient.

The AES algorithm is extremely secure, with no known vulnerabilities at this time. The main risks associated are if an unauthorized individual gains access to the key or if a brute force attack is successful. While the former is a possibility, the latter is extremely unlikely as the time needed to brute force the algorithm would be unrealistic. The AES algorithm has been around since the early 2000’s and is currently the most popular type of encryption available. Over the years, encryption algorithms have changed to become more secure, but the principle behind how data is encrypted has not. Keeping the key needed to decrypt a message secure is just as important as the security of the algorithm. An unbreakable algorithm is no use if it can be bypassed all together.

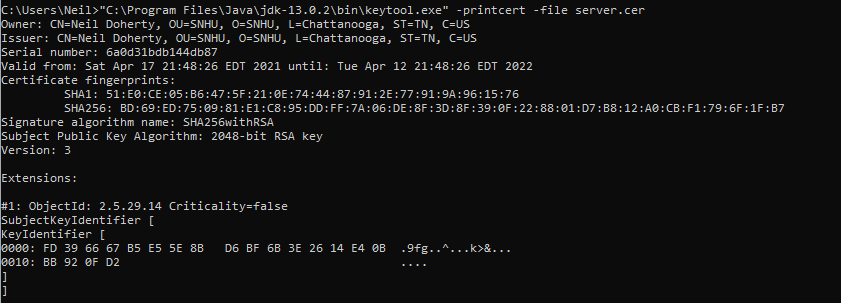
## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



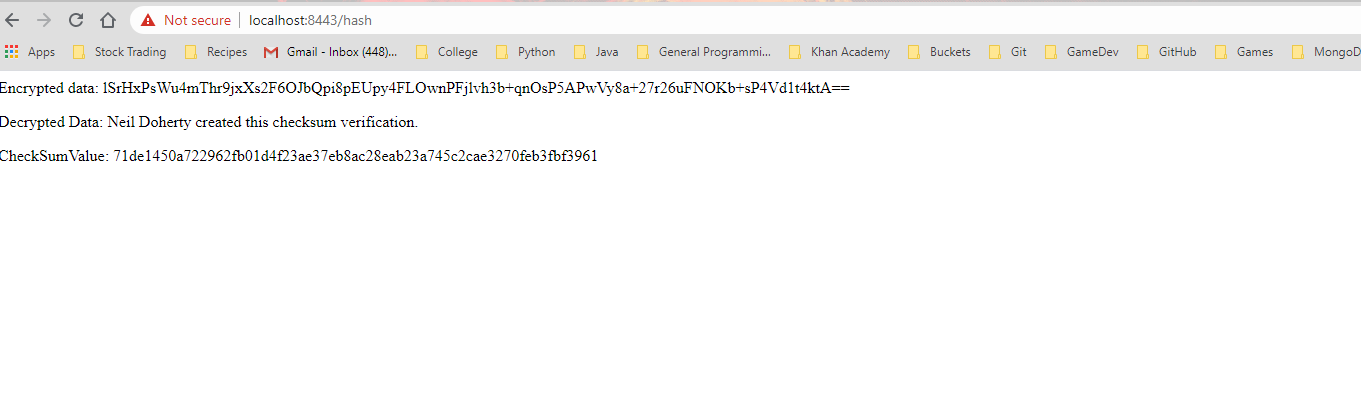




## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

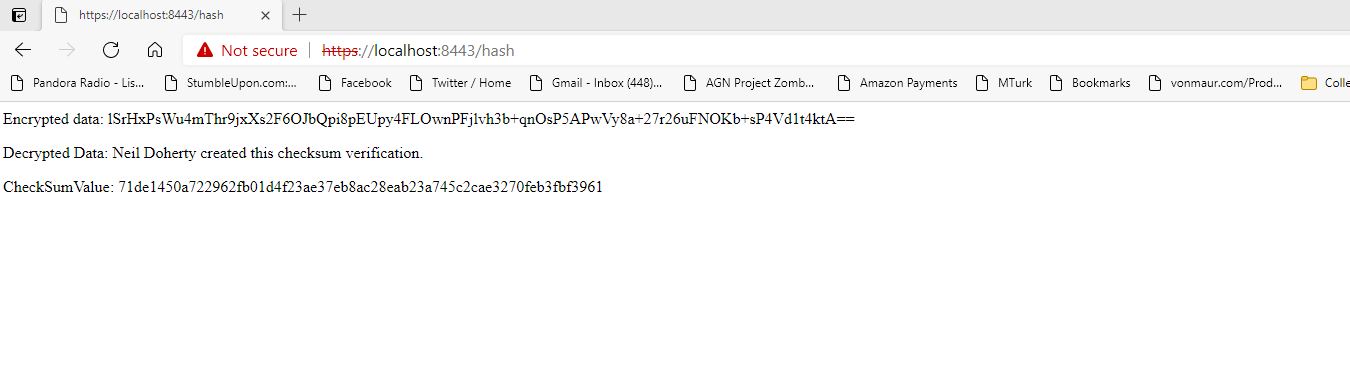
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

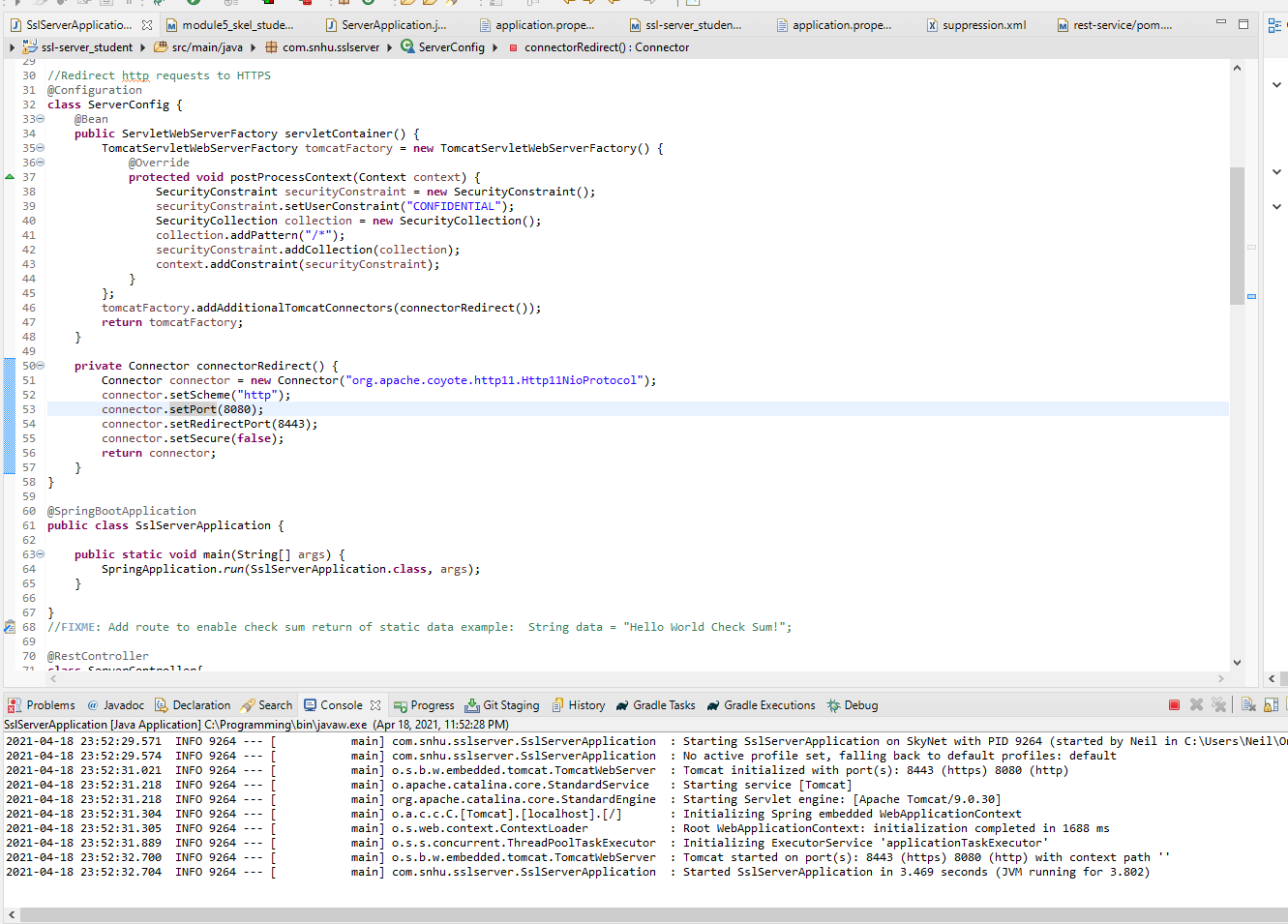
* Insert a screenshot below of the web browser that shows a secure webpage.



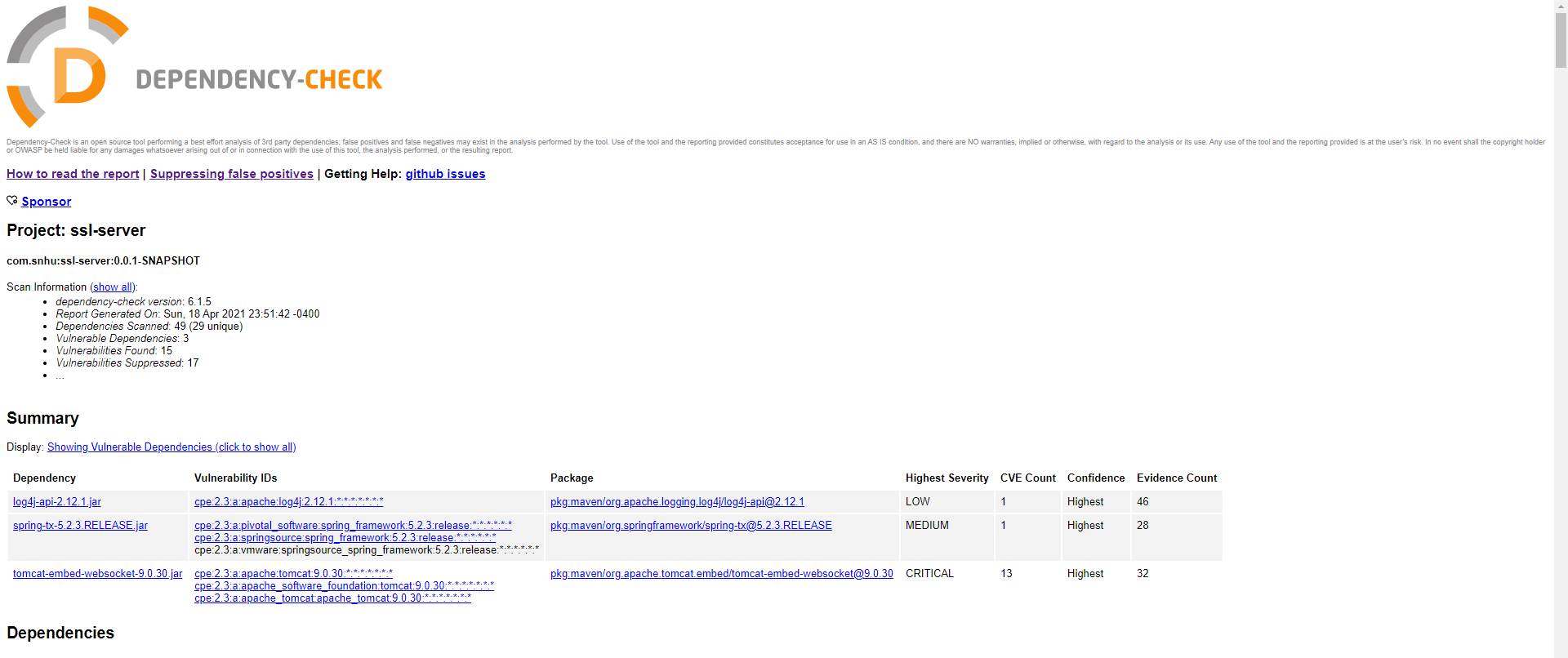
## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors



* + A screenshot of the dependency check report

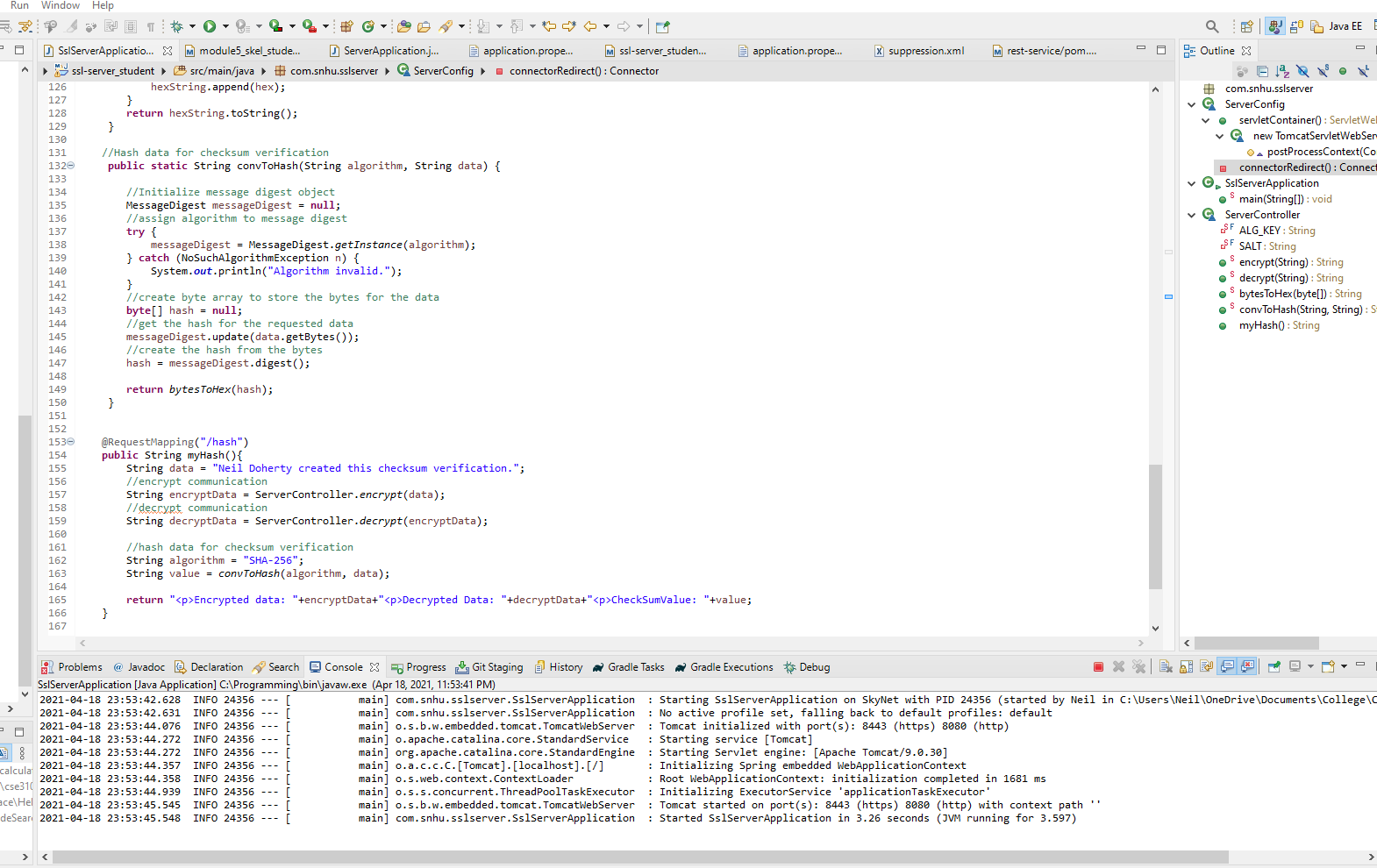


No additional vulnerabilities were identified.

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.



## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

The code has been refactored by adding a checksum verification using the AES-256 encryption algorithm to encrypt and decrypt the original data. Additionally, a server configuration was added to ensure that all HTTP requests are converted to HTTPS. The application.properties file was updated to include the correct keystore information and add an HTTP port for forwarding HTTP requests. Finally, the pom.xml file was updated to reflect the dependency report with some errors suppressed. The areas of security that were addressed are Secure API Interactions, Cryptography, and Secure Error Handling. The API interactions were validated with the dependency check. Cryptography was utilized by encrypting the data on the page. Secure Error Handling were addressed by ensuring that if the encryption algorithm was invalid, the page would not display.

The process for adding layers of security was based on the needs of the company and the data that would be transmitted. Utilizing a secure and practical encryption algorithm will guarantee that the data being displayed on the page is as the company intended and is not altered. It also prevents any individual without the appropriate key from being able to view the data. The decision to route all HTTP requests to HTTPS will help protect the company’s customers from a variety of attacks. HTTPS indicates the site is secure and eliminates potential theft of customer data. By approaching the security of the site with a focus on the customer, we can make decisions that will minimize their risk. By doing so, it helps the business maintain the integrity of our systems.

To maintain the current level of security, regular dependency checks should be performed to stay aware of potential security threats. If a vulnerability is identified in a way that may impact the site, research can be performed to identify and institute an appropriate fix. This process should also be applied to the encryption algorithm used. While there are no current vulnerabilities to the AES-256 algorithm, there is always the potential for that to change. Should the algorithm no longer be considered secure, the algorithm could be changed to another available option. Finally, all proposed changes should be handled in a testing branch prior to being implemented. Updates should not be made to the live branch of the code until they have been thoroughly reviewed for any potential vulnerabilities. If these processed are implemented, the code will remain secure.