****

# 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** | **6/17/2023** | **Scotty Intondi** |  |

## 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

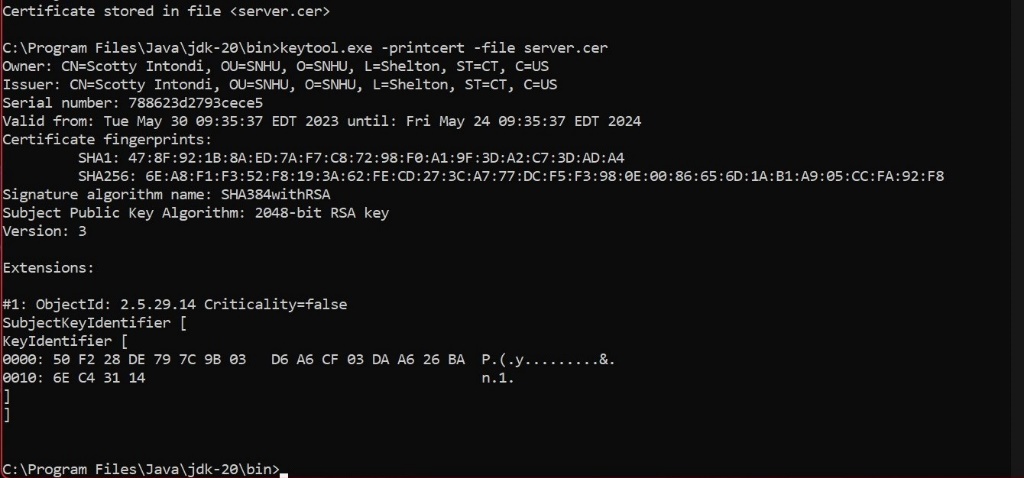
Scotty Intondi

## Algorithm Cipher

The SHA-256 algorithm will be used in Artemis Financial as it will provide the highest level of security when sending data externally. This type of algorithm uses 256-bit keys for encryption in which the numbers in the encryption are randomly generated and using 256 bits takes 1.1579209e+77 guesses which to put in perspective is more than the number of atoms in the known universe and computers are just not that powerful enough to crack a SHA-256-bit algorithm at this time. This will be paired with Asymmetric-key hashing where the encryption key is mathematically bound to the decryption key. The encryption key is public whereas the decryption key is private keeping from attackers being able to essentially gain access to the data.

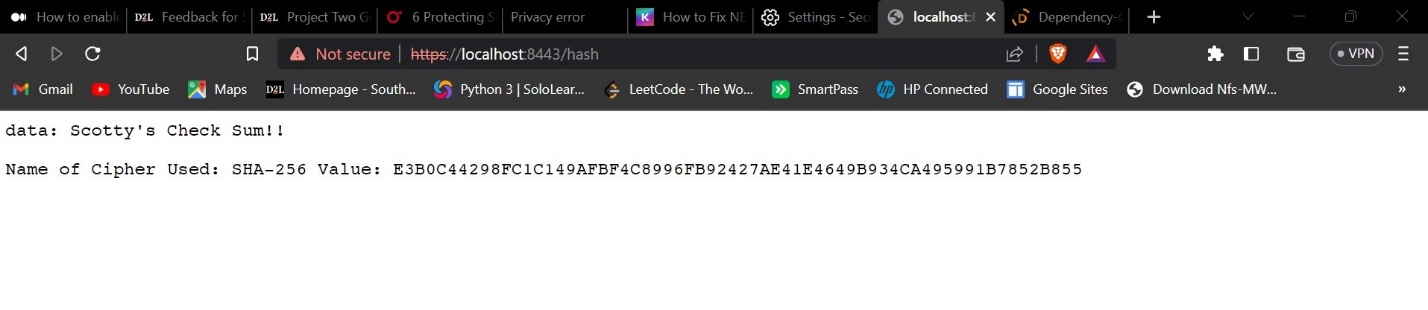
## Certificate Generation

Insert a screenshot below of the CER file.



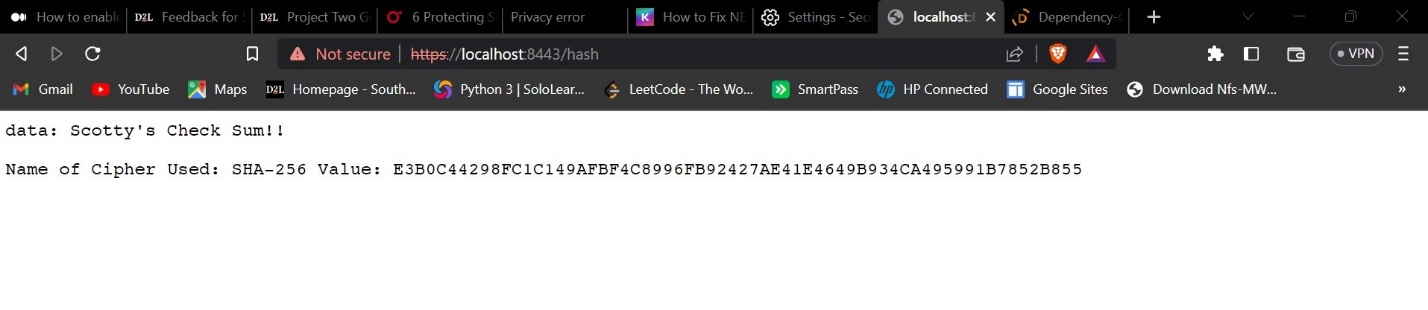
## Deploy Cipher

Insert a screenshot below of the checksum verification.

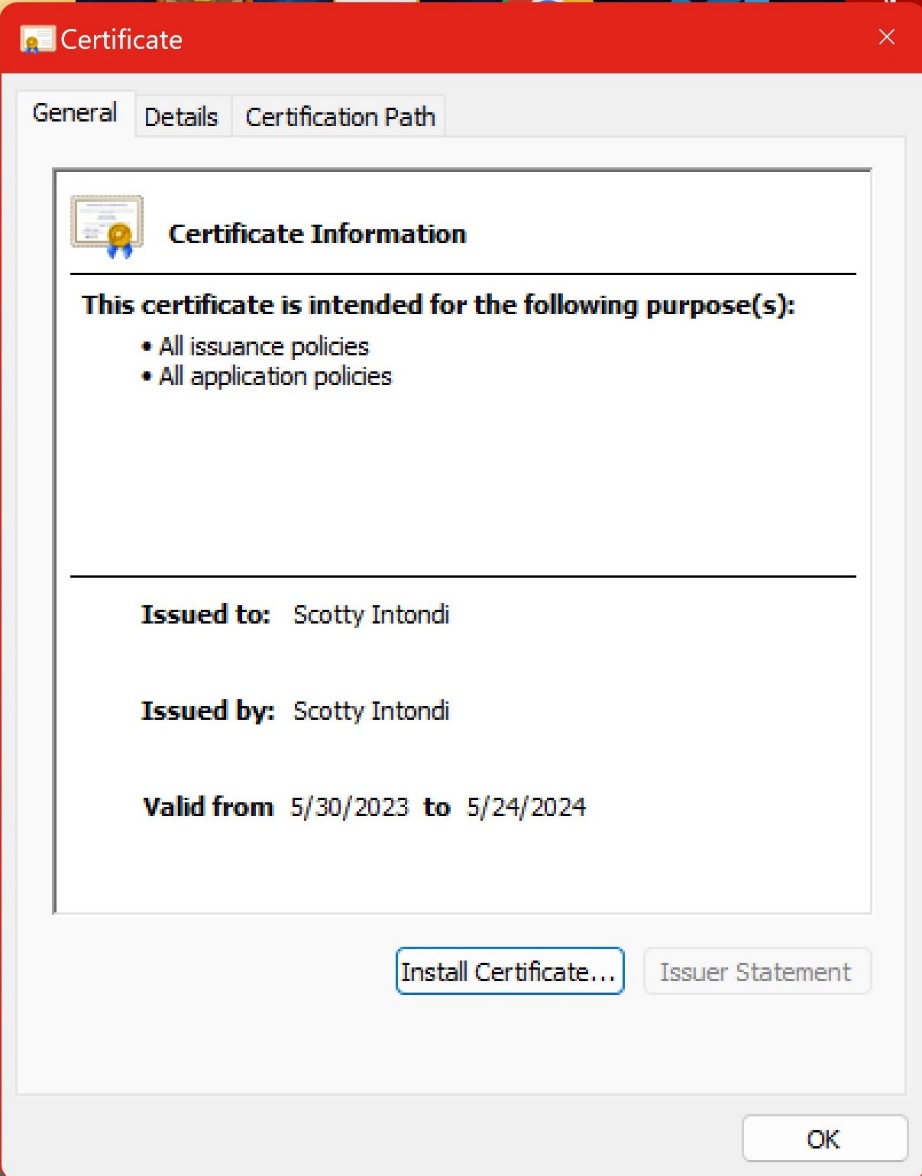


## Secure Communications

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

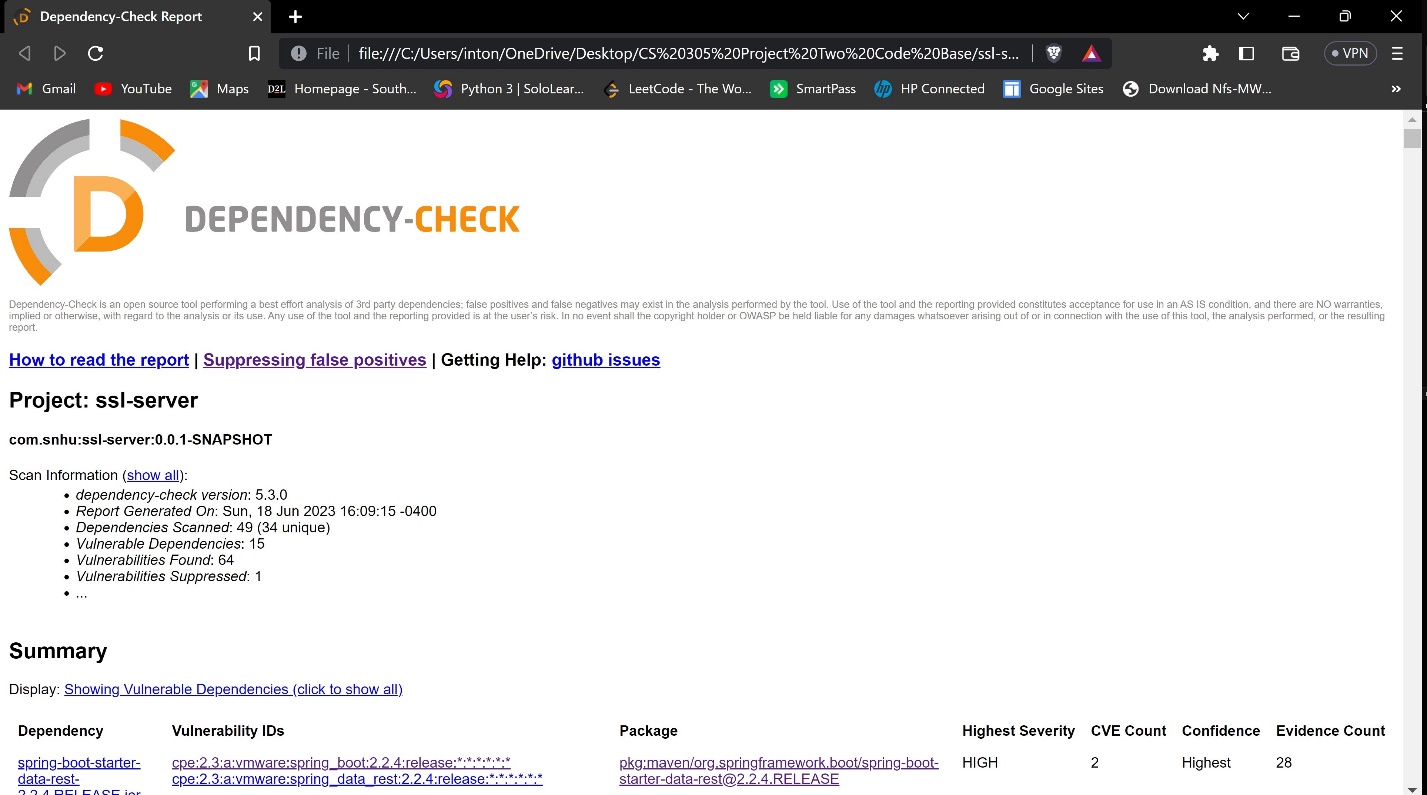


I wasn’t able to get self-signing secure with the truststore



## Secondary Testing

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



## Functional Testing

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

*@RestController*

class ServerController {

private static final char[] ***HEX\_ARRAY*** = "0123456789ABCDEF".toCharArray();

private String getHash(String input) {

try {

MessageDigest messageDigest = MessageDigest.*getInstance*("SHA-256");

byte[] messageDigestMD5 = messageDigest.digest();

return *bytesToHex*(messageDigestMD5);

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

return input;

}

public static String bytesToHex(byte[] bytes) {

char[] hexChars = new char[bytes.length \* 2];

for (int j = 0; j < bytes.length; j++) {

int v = bytes[j] & 0xFF;

hexChars[j \* 2] = ***HEX\_ARRAY***[v >>> 4];

hexChars[j \* 2 + 1] = ***HEX\_ARRAY***[v & 0x0F];

}

return new String(hexChars);

}

*@RequestMapping*("/hash")

public String myHash() {

String data = "Scotty's Check Sum!!";

String hash = getHash(data);

return "<p>data: " + data + "</p><p> Name of Cipher Used: SHA-256 Value: " + hash;

}

}

## Summary

To start off we create a bean component code @RestController which gives an entry point onto API calls. This is followed by nesting @RequestMapping which iterates the entry points which in this case is /hash. In the myHash method, the REST calls to the API which is the hashed data converted to string.

The code creates an array from hex 0x00 – 0xFF which comes to 256 bits. The pox.xml file houses the dependency versions and other various tools for the maven environment. The one we are particularly interested in is the dependency checker where it scans for vulnerabilities in dependencies in the program. Some vulnerabilities do not have a patch, yet, which would yield into warnings to the server and to subdue them you would have to suppress the dependency until there is an update. Updating periodically in the pom file will help ensure dependencies are patched and possible attacks are mitigated.

## Industry Standard Best Practices

Keeping dependencies updated help with mitigating possible attacks on the program.