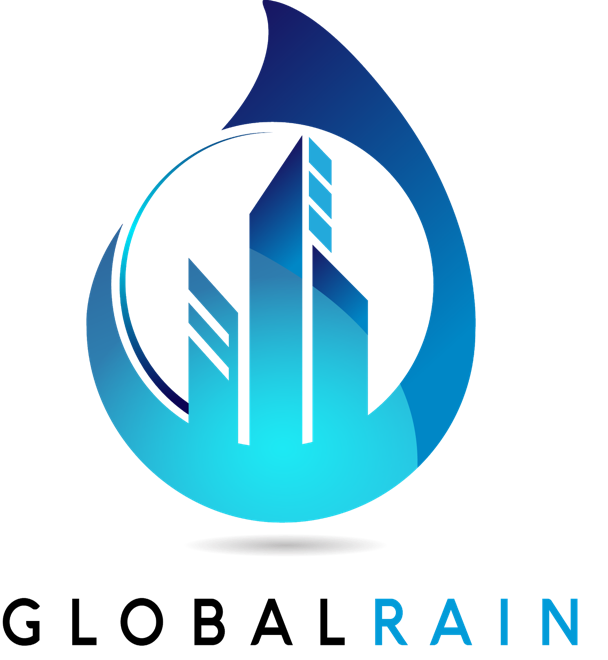
****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_1fob9te)

[Client 3](#_3znysh7)

[Instructions 3](#_2et92p0)

[Developer 4](#_tyjcwt)

[1. Algorithm Cipher 4](#_3dy6vkm)

[2. Certificate Generation](#_1t3h5sf) 5

[3. Deploy Cipher](#_4d34og8) 6

[4. Secure Communications](#_2s8eyo1) 7

[5. Secondary Testing](#_17dp8vu) 7

[6. Functional Testing](#_3rdcrjn) 9

[7. Summary](#_26in1rg) 11

## Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **10/17/2020** | **Nathaniel Holcombe** | **Project zip file is attached.** |

## 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

Nathaniel Holcombe

## 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.

My choice for secure communication between Artemis Financial’s web application and the server is TLS 1.3, which employs a suite of various algorithms for certificate verification, generation of public and private keys, and encryption of data. It uses a 2048-bit RSA encryption to encrypt the certificate data, the SHA-256 algorithm to create a certificate fingerprint by hashing the RSA encryption, an elliptic curve algorithm such as Curve25519 (Cryptography.io, 2020) to generate asymmetric public and private keys for secure socket establishment of data between the client and server, and then AES-128 GCM for symmetric encryption and exchange of bulk data after trust is established (Sullivan, 2018).

The SHA-256 algorithm is effective for verifying the authenticity of an RSA-encrypted server certificate because of its ability to create a unique, one-way representation of data using its hashing functions. The SHA-256 encryption is first padded with filler data to make its size a multiple of 512 bits, which is then processed by padding, parsing, and then repetitively mixing and hashing in a particular sequence to compute the final hash value of 64 hexadecimal characters (Bellet, 2018). There are 2^256 possible unique hash values from the SHA-256 algorithm, which allows for nearly any amount or variation of data to be hashed into a unique “fingerprint”, or checksum. This explains the usefulness of SHA-256 to verify the authenticity of a certificate, as it is highly unlikely that another SHA-256 encryption will result in a duplicated hash value, thereby offering high collision resistance.

The use of 2048-bit RSA encryption is effective in creating public and private keys of certificate data because it uses a trap-door function of multiplication of two very large prime numbers to create a key which is 617 digits long (Lake, 2018). RSA pads the certificate data using random numbers in order to obfuscate the structure of the data before it’s encrypted. The quality of these random numbers will result in a weaker or stronger defense against factoring of the key. High-quality random numbers can be obtained using a pseudo-random number generator, computer memory random data, or other means (Manico, 2015).

The asymmetry of the RSA means that the signing of certificate data into an RSA signature uses a private key, and a public key is used to verify the signature. Both keys were created within one mathematical process, and their usefulness depends on one key remaining a secret while the other is transmitted publicly. These public and private key pairs encrypt and decrypt the RSA signature of the certificate data, serving as variables in calculation.

After client and server trust has been established, TLS 1.3 uses an elliptic curve algorithm to establish a secure socket connection using a shared key, which is exchanged along with each message to be validated by the receiver. The key used is only valid for a single session, so that a new key derived with the elliptic curve algorithm is required at the next connection session. This provides a safety mechanism which prevents an attacker from reading future messages if the current encryption is cracked (Crawley, 2019).

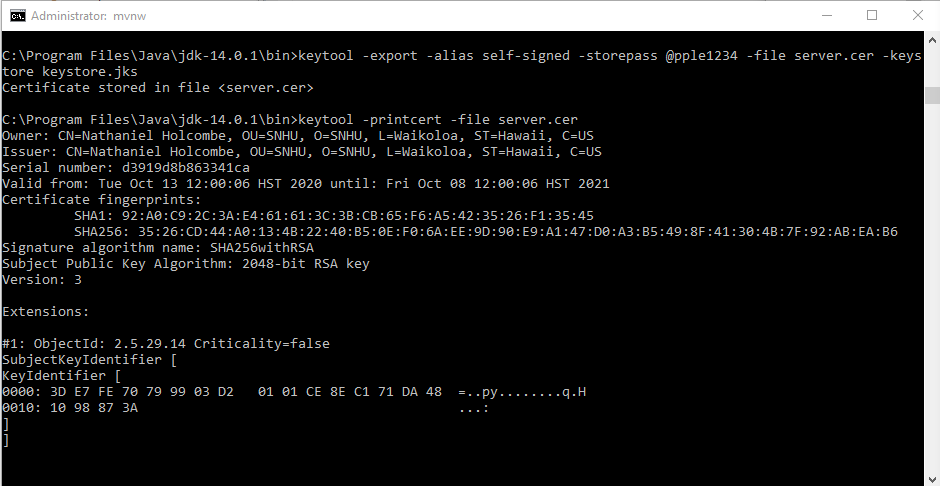
Within a secure socket connection, the message data itself are encrypted using AES-128, a secure symmetric encryption algorithm which is used to both encrypt and decrypt the message data. Symmetrical encryption is far more efficient than asymmetric, and AES-128 is much faster than RSA encryption (Wahid, 2018). The use of different algorithms for certificate signature and verification, securing the session, and encrypting message data within TLS 1.3 serve to increase the overall security of the process because cracking any one of these encryptions will not negatively impact the other information.

Historically, encryption algorithms have been developed and later compromised due to both the progression of technology according to Murphy’s law, and to the discovery of vulnerabilities in the encryption process. Those algorithms which used to provide security with only a 56-bit key (DES) are no longer adequate to withstand a brute-force attack because the computing power available has increased to a sufficient extent to cause such a feat to be possible in only 22 ¼ hours in the case of DES (Distributed.net, 2010). Newer algorithms have been developed in response to this challenge, partly through the promotion of cryptography contests to encourage innovation. An example of such a competition was the National Institute of Standards and Technology (NIST) AES competition, announced in 1997 and completed in October 2000 (Bernstein, 2014). AES therefore uses an algorithm created by Vincnet Rijmen and Joan Daemen of Belgium, called the Rijndael algorithm, and is the standard used to encrypt classified information by the US Government (Rouse, 2020). Undoubtedly, as time progresses and technology exponentially grows in its computing power, better encryption algorithms will continue to be created and put into use. Quantum computing is expected to provide both terrific challenges and terrific advancements to the world of cryptography because it uses quantum superposition to remove the need to use iteration in computations (Mosca, 2016).

## 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.



*The above screenshot displays the certificate information located in the server.cer file. It is valid for one year, after which a new one will be required. The fingerprints represent the unique hashed value of the certificate data, including the signature.*

## 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.

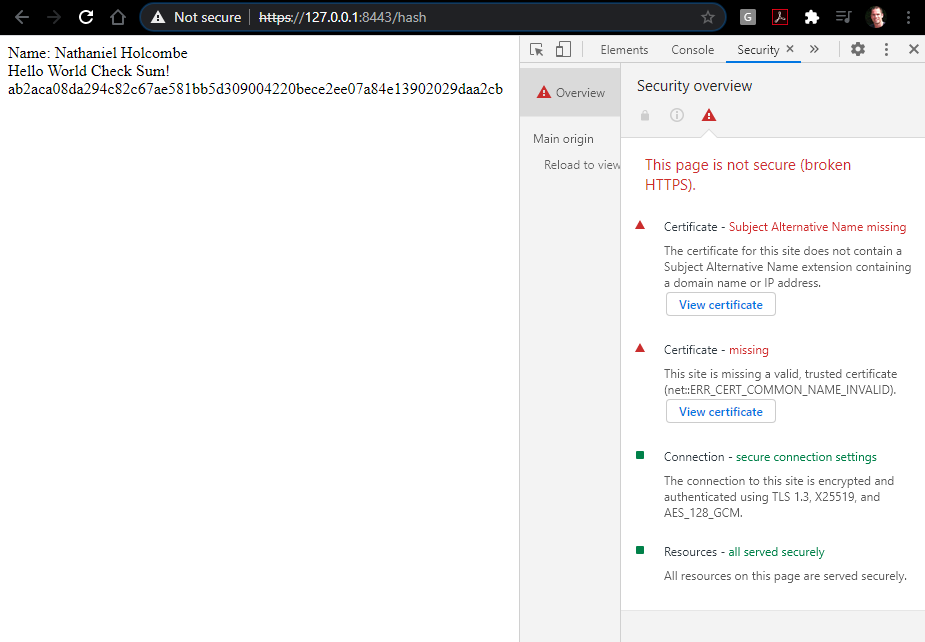


*The above screenshot displays a Chrome browser window, with my name, a data string to be encrypted, and the SHA-256 hash value (checksum) of the data string on the third line. Such a checksum would be used for verification of the integrity of a transmitted file to ensure it was not tampered with in transit.*

## 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.

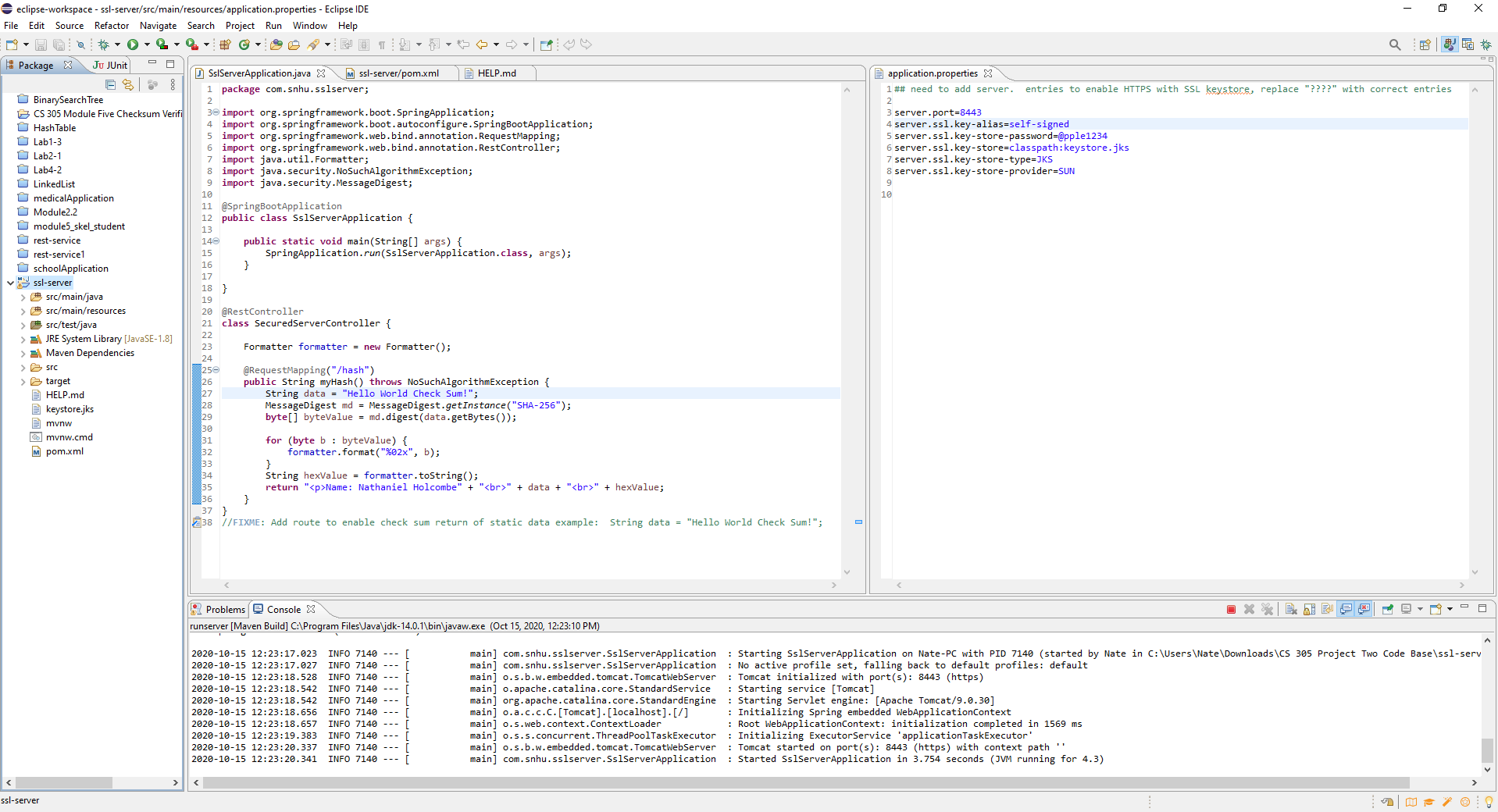


*The above screenshot shows the Chrome browser window, with the address bar displaying “https”, which is crossed out because of the alternative name and common name issues displayed in the Developer Tools > Security tab. HTTPS is secured hypertext transfer protocol which results from Transport Layer Security encryption of the client/server connection. This prevents malicious access to sensitive information sent between the Web Application and Server.*

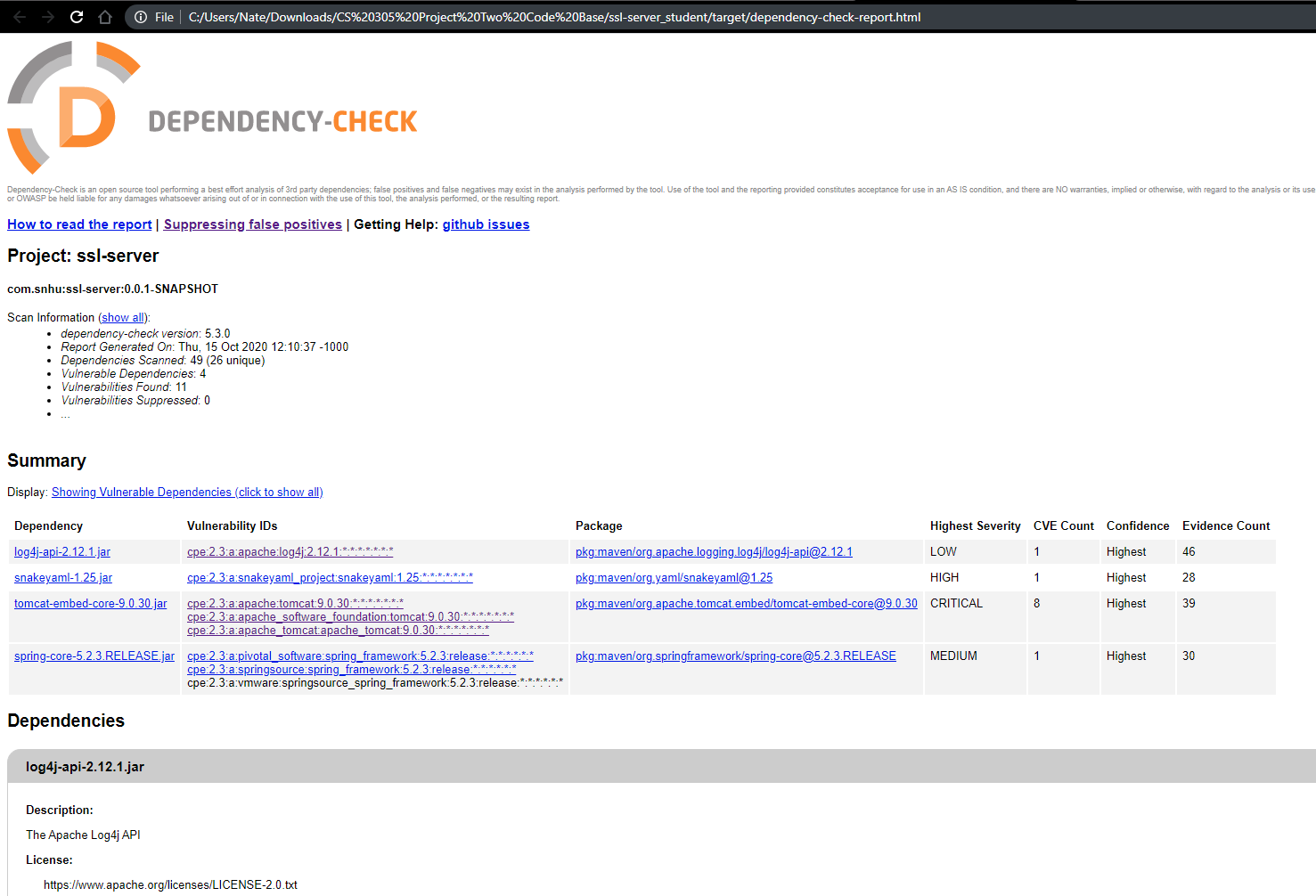
## 5. Secondary Testing

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



*The above screenshot displays the running application, with the refactored code of the SslServerApplication.java file on the left and the refactored code of the application.properties file on the right. There are no errors found with execution. Code has been added which configures the security of the connection (right side), and runs the application and maps the file verification (checksum) functionality to the “/hash” URL address.*

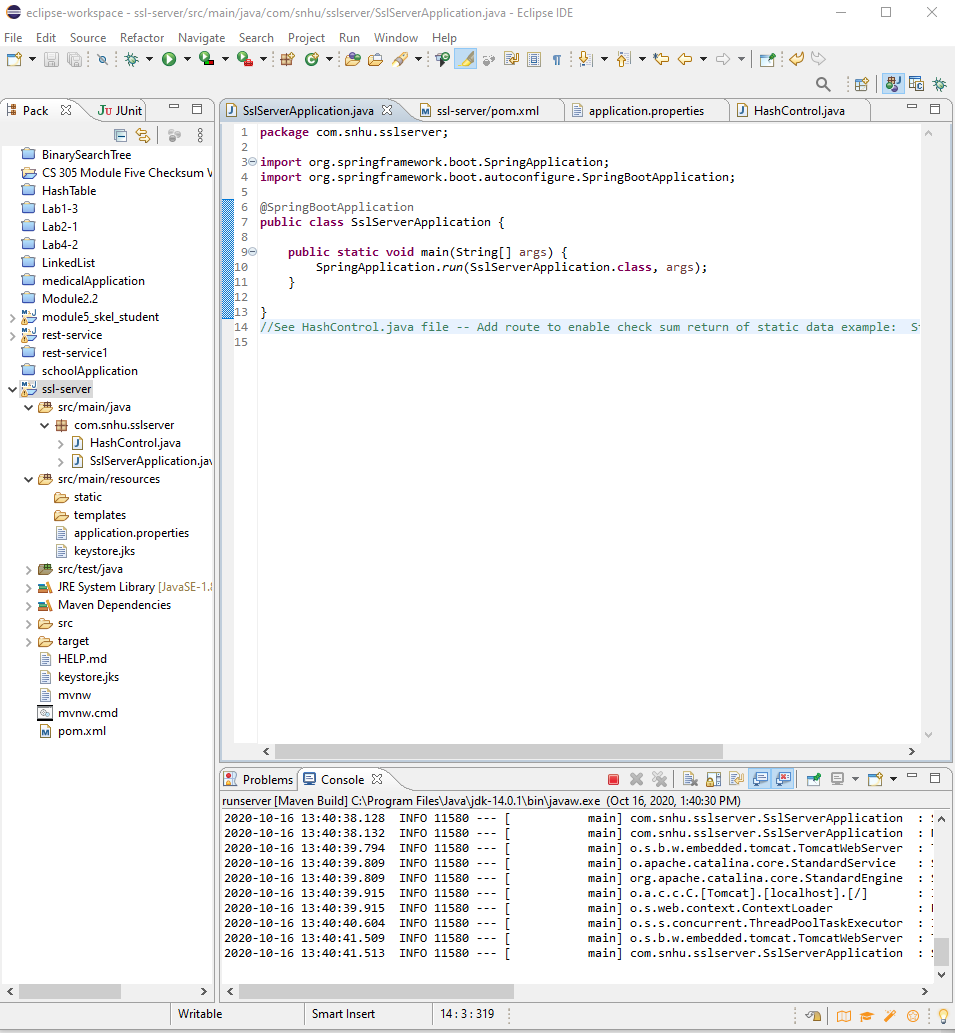


*The above screenshot of the dependency check page shows four dependencies but no additional vulnerabilities from the refactoring of the program to meet requirements. Unaddressed vulnerabilities introduce attack vectors to an application which may be addressed by updating to the latest version that includes bug fixes, seeking alternative tools/plugins, or removal.*

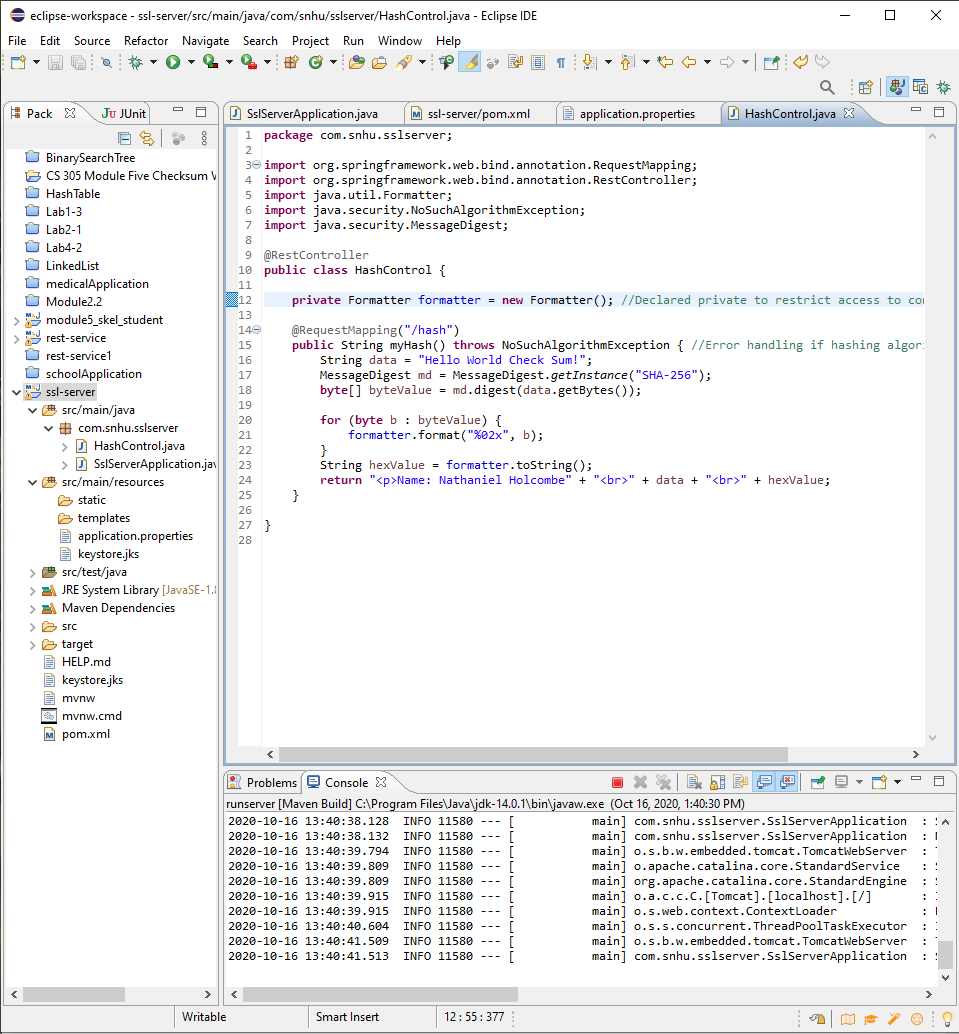
## 6. Functional Testing

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.



*Above screenshot: This demonstrates improved modularity of the application by separating the main application from the hash controller via a new class file. Maintenance and troubleshooting are both easier when functional components are separated within an application.*



*Above screenshot: This displays the HashControl.java file and HashControl class. Encapsulation of the Formatter object initialization via private access prevents unauthorized use outside of the HashControl class. Error handling of incorrect/unsupported hashing algorithms with NoSuchAlgorithmException thrown by the myHash() method prevents unexpected program behavior which may introduce a new attack vector.*

## 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.

*Input validation* - This is not applicable in the current version of the server code, but the client side web application must use input validation for all user input. There are no method parameters to address, but any which may exist in the future must be declared as private and wrapped in a public, Java-based wrapper method which performs input validation and error handling (Oracle, 2018).

*APIs* - This server application uses Spring framework with Spring Boot, JUnit testing, and Apache Maven provides the encrypted transmission of resources via TLS 1.3 -- the most recent version -- to enable HTTPS communication between the client and server. The application also uses public REST classes according to Oracle guidelines (Oracle, 2018). The application would benefit from the use of a Content Security Policy (CSP) to prevent cross-site scripting of user input (Raible, 2018).

*Cryptography* - The algorithms used here within TLS 1.3 ensure that the transmission of information between the client and server is made secure by first establishing trust between the web application and the server with the use of public and private encrypted keys created by the Java keytool and stored in the keystore. This keystore should be safely protected within a physically separate location from the server or web application installations, or can be secured with Spring Vault and HashiCorp Vault (Spring, 2020). The data to be transmitted is then rendered human-unreadable by symmetrically encrypting the information before it is sent using the AES-128 algorithm, after trust between the server and client has been established and a session key has been created using an elliptic curve algorithm. This data is then decrypted after receipt, along with verification of the EC-derived session key. These forms of security via cryptography are aligned with the SEC requirement that financial institutions encrypt all data in transit and at rest (SEC, 2019).

*Client/Server* - Financial and PI Information are being exchanged, so the use of HTTPS requests and responses is required. As described previously, a certificate is used to establish trust between the server and the web application, along with limited certificate legitimacy (360 days). Man-in-the-middle attacks are avoided through these as well as the use of user logins, passwords, and access permission/authorization with OpenID Connect (OIDC). The SEC requires user action traceability, so authentication is needed to be sure who is performing actions in the system (SEC, 2017).

*Code error* - The opportunity for errors to occur with either normal or malicious activity requires the use of error handling techniques to anticipate all possible reactions of the application to these scenarios. The “NoSuchAlgorithmException” in the HashController class handles an invalid algorithm choice and prevent unpredictable program behavior in response to errors in the hashing of files, or even of maliciously-intended files on the client end (Oracle, 2018).

*Code Quality* - The Formatter constructor is securely initialized as private, limiting its accessibility to only within the HashControl class. The addition of the Java SecurityManager would be useful here and in any other part of the application to check the security context before an operation is performed (Oracle, 2020). The use of security policies is an important part of SEC compliance (SEC, 2019).

*Encapsulation* - Java Guidelines advise to “declare any class or interface public if it is specified as part of a published API, otherwise, declare it package-private.” There are no sensitive classes in this server application.

*Views* - There are no views.

*Models* - There are no models.

*Controllers* - Note: The Formatter object is private to prevent access outside of the HashControl class.

*Data Access* - User access control should be included, along with login failure logging and a limit on the number of transactions in a given unit of time that a user can perform. These would protect the system from access by unauthorized entities and protect against a brute force attack attempting to guess the credentials (OWASP, 2010).

*Services* - There are no services in this application.

*Plug-ins* - The Apache Log4j API in the application is version 2.12.1. The mitigation listed for the vulnerability states to update to version 2.13.2 to address the issue of the current version not verifying the host name against the SSL certificate, allowing the SMTPS log files to be accessed via a man-in-the-middle attack (Apache, 2020). For the SnakeYAML dependency vulnerability, the mitigation is to disable XXE (Xml eXternal Entity) (CheatSheets, 2020).

*APIs* - API classes are declared public, in accordance with Oracle guidelines (Oracle, 2018).

My process for adding layers of security to the application was in response to the stated need of securely transmitting files between the web application and the server. These files consist of RESTful message resources, sensitive financial information, certificate signatures, public RSA keys and a session key. The encryption of these could be considered one layer of security which protects the integrity of the file transmission functionality of an internet-connected system, as transmitted data may be easily intercepted, yet not understood. Artemis Financial is thereby protected against malicious impersonators who may attempt to pose as the server or client and send a false key, and against brute force cracking of encrypted data because of the strength of the algorithms used. Secondly, the use of forward secrecy in the choice of the elliptic curve algorithm for the session traffic adds another layer of security to prevent reading of past sessions if another session key is decrypted, because each session key is unique. This means that a single occurrence of a compromised session key preserves the integrity of Artemis Financial’s other data transactions and minimizes the number of clients affected. The third layer of security is in the lack of accessibility of functions to unauthorized server users. While the authorization system has not yet been built, the basic program code is compatible with such a layer because classes, methods and variables are protected by encapsulation.

The mindset of a developer with a concern for security will be successful if this question is constantly asked: “how can this be used in a way other than intended?” In order to preserve the current level of security, new functionality must be added in as clear and modular a manner as possible, so as to not obscure its qualities of security behind a spaghetti of complexity. Where possible, new code must be built securely by minimizing duplication and maximizing encapsulation of components, classes, or other parts of the application. This will facilitate a uniformly-secured system because there is less of a likelihood that malicious actors can find alternate, under-secured routes to get around more secure areas which have received extra reinforcement.

All input must be validated and never assumed to be trustworthy. Data length, range, source, and context inside or outside a trust boundary are all considerations in order to prevent injection of code to produce aberrant behavior. Access to the system should also be restricted to authorized users, whose identity is verifiable with passwords and session ids. These approved users should not be allowed to log in from multiple devices and be limited to a safe number of actions in a given amount of time in order to reduce the possibility of denial of service attacks.

Another best practice is to continue to build in the security of new code from the beginning instead of retrofitting it afterward. Such a poor development approach would undoubtedly take more time in total and is unlikely to be as effective without significant refactoring of code. Such a retroactive concern for security also applies to adding third-party software which may bring with it certain vulnerabilities for which the best solution is to either avoid them or make sure they are updated soon after new releases. Use of open-source software may or may not imply a shorter release schedule, but greater popularity can provide more safety because of the number of eyes on the project.

The documentation of security measures taken, particularly for a client in the financial industry like Artemis Financial, will ease the annual SEC audit experience because security procedures are accessible in a centralized place. Policies regarding user privileges and trust boundaries which are documented also add to the level of transparency which is required for successful audits.

As mentioned previously, all data in transit and at rest should be encrypted, in accordance with the SEC’s requirements for financial institutions. The fact that financial and personally identifiable information is a common target of cybersecurity attacks highlights the importance of maintaining the secrecy of this data (Verizon, 2020).

*References:*

Individual Contributors. (2020). *X25519 Key Exchange.* Cryptography.io. <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/x25519/>

Sullivan, N. (2018, August 10). *A Detailed Look at RFC 8446 (a.k.a. TLS 1.3).* CloudFlare.com. <https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/>

NIST. (2012, August). *Recommendation for Applications Using Approved Hash Algorithms.* U.S. Department of Commerce. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf>

Bellet, C. (2018, January 3). *Part 5: Hashing with SHA-256.* Medium.com. <https://medium.com/biffures/part-5-hashing-with-sha-256-4c2afc191c40>

Lake, J. (2018, December 10). *What is RSA Encryption and How Does it Work?* Comparitech.com. <https://www.comparitech.com/blog/information-security/rsa-encryption/>

Crawley, K. (2019, May 7). *The Importance of Forward Secrecy in TLS 1.3.* Venafi.com. <https://www.venafi.com/blog/importance-forward-secrecy-tls-13>

Wahid, M., Ali, A., Esparham, B., Marwan, M. (2018). A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. *Journal of Computer Science Applications and Information Technology. 3(2): 1-7. DOI: 10.15226/2474-9257/3/2/00132.* <https://symbiosisonlinepublishing.com/computer-science-technology/computerscience-information-technology32.php>

Distributed.net. (2010, June 26). *DES-III: Success!* Distributed.net. <https://www.distributed.net/DES>

Bernstein, D. J. (2014, January 27). *AES: The Advanced Encryption Standard.* Competitions.cr.yp.to. <https://competitions.cr.yp.to/aes.html>

Rouse, M. (2020, April). *Advanced Encryption Standard (AES).* TechTarget.com <https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard#:~:text=The%20Advanced%20Encryption%20Standard%20>(AES,cybersecurity%20and%20electronic%20data%20protection.

Mosca, M. (2016, May 20). *Update on teh Quantum Threat, Mitigation, and Relevant Timelines.* ICMC 2016. <https://icmconference.org/wp-content/uploads/A21b-Mosca.ppt.pdf>

Raible, M. (2018, July 30). *10 Excellent Ways to Secure Your Spring Boot Appliciation.* Okta.com. <https://developer.okta.com/blog/2018/07/30/10-ways-to-secure-spring-boot>

U.S. Securities and Exchange Commission. (2017, n.d.). *Observations from Investment Adviser Examinations Relating to Electronic Messaging.* SEC. <https://www.sec.gov/files/OCIE%20Risk%20Alert%20-%20Electronic%20Messaging.pdf>

VMWare. (2020). *Spring Vault.* Spring.io. <https://spring.io/projects/spring-vault>

U.S. Securities and Exchange Commission. (2019, May 23). *Safeguarding Customer Records and Information in Network Storage - Use of Third Party Security Features.* SEC. <https://www.sec.gov/files/OCIE%20Risk%20Alert%20-%20Network%20Storage.pdf>

Oracle. (2018, September 27). *Secure Coding Guidelines for Java SE.* Oracle.com. <https://www.oracle.com/java/technologies/javase/seccodeguide.html>

Oracle. (2020). *Class SecurityManager.* Docs.Oracle.com. <https://docs.oracle.com/javase/7/docs/api/java/lang/SecurityManager.html>

U.S. Securities and Exchange Commission. (2019, April 16). *Investment Adviser and Broker-Dealer Compliance Issues Related to Regulation S-P - Privacy Notes and Safeguard Policies.* SEC. <https://www.sec.gov/files/OCIE%20Risk%20Alert%20-%20Regulation%20S-P.pdf>

Apache. (2020). <https://issues.apache.org/jira/browse/LOG4J2-2819>

CheatSheets. (2020). *XML External Entity Prevention Cheat Sheet.* OWASP.org. <https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html#JAXP_DocumentBuilderFactory>

Verizon. (2020, n.d.). *Data Breach Investigations Report.* Verizon Enterprise. <https://enterprise.verizon.com/resources/reports/dbir/2020/results-and-analysis/>