****

# 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** | **18 October 2020** | **Cinnamon George** |  |

## 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

Cinnamon George

## 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.

Based on the requirements Artemis Finical is looking for a way to have secure communications. Based on what I have learned having a algorithm that avoids collisions will help to ensure the security for this company. I think that a good fit this company will be using a message digest algorithm. Message digests are designed to protect the integrity of the data. A message digest also detects any changes or alterations to the message/data. This is something that will help the company as they work on creating insurance policies etc. This will be important for them to be able to track once the data has been altered. Also having the integrity of the data for their clients will help them in the long run to be able to keep returning clients. If information were to be hacked then it could be both harmful to the clients as well as the company. For the message digest there are four to be able to choose from MD2, MD5, SHA-1, SHA-2, SHA-3. I choose the SHA-256 for this. Starting with the secure hash algorithm. Starting with why I did not choose MD5 this was designed in 1991. “In 2004 a team of Chinese cryptanalysts presented a method for finding collisions”. With this the methods have improved and using this hash algorithm collisions can be found in under a minute. This a part of the requirements is to choose an algorithm that avoids collisions. Right off the bat this is not a algorithm that avoids the collisions and would be to prone to attack. Moving on to SHA-1 “So far, no explicit collisions have been found in SHA-1. However, theoretical analysis indicates that they can be found using significantly fewer than 280 hash-function evaluations required by the birthday attack.” Looking ahead I do not believe that this program is going to need to as secure as it needs to be. That is one thing that I have learned from this class is that you need to look at the possibilities. The ones with little to no known weakness. This helps to ensure that it is able to be secure as absolutely possible. At this time the SHA-2\* does not have any known weakness. All changes can be made over time I believe for this assignment that it was the best one to use. Then taking a look at the SHA-3 this is for fixed output. For this program it is not a fixed output so this one would not be able to work for what we need it to be able to. The main thing is to be able to avoid collisions as much as possible. Collisions occur when two distinct pieces of data have the same hash value, checksum etc. If the output were to change or there was more than one thing being added we would not want it to be encrypted the same. This will cause lots of problems when trying to decrypt the hash. As this is not a fixed output collisions could occur, so it is best to be able to avoid them.

When working towards securing communications it is important to use random numbers because they are a cryptographic key. It unlocks the content or locks the content of the encrypted message. The nice thing about this is that every message has a unique key. Which is good for Artemis Financial as the messages/data etc would have its one key. This can go over individual files in a folder. This would make it hard to be able to access all information if someone were able to access that specific key they would have to go through the long process again to be able to figure out how to be able to open up the next file. The algorithm cipher that I choose does use random numbers. This is essentially what a hash function does to help to generate these random numbers on top of the algorithm. Next I am going to talk symmetric vs non symmetric keys. Symmetric keys is a use of only one key to both encrypt and decrypt information. While having one key for both is much faster to be able to access information. This also has its disadvantages because the more that users the more keys that there are. This would mean that you would also need to be able to have key access management. For this company due to the lack of people that would be dealing with the information key management would be extremely simple and more secure that an non symmetric key and here is why. A non-symmetric key uses pairs of public keys and then uses a private key to decrypt the message. When it comes to using public keys it can be easier to get around, it makes it more of a vulnerability. Also the message digest uses a symmetric key.

Encryptions have been used for ages well before technology was ever created. This was most often used in military settings. As they would code messages to get them across enemy lines to various allies. Today encryptions are dealing with the same concept just on a different level than it was previously was. Today information is put out there. At the same time information needs to be secured for the benefit of well everyone. In order to do this the algorithms are constantly changing. They are being used in different ways or creating new ways to be able to improve the algorithms. This is due to technology changing. This is also due to the fact that people can be persistent when they are wanting information and can find ways around an encryption. New algorithms may not have known vulnerabilities. Which as the vulnerabilities appear is why you see then need for software security. A part of that is monitoring the code and then figuring out why it is the way it is. While I picked a algorithm that does not have know vulnerabilities it does not mean that this not need to be monitored. This is something to start with that is simple that will be known to be secure and wont have to worry about issue right of the bat with the data. This helps to be able to stay of top of it and be able to keep the data safer if something does appear.

## 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.

Graphical user interface, text, application, email

Description automatically generated

## 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.

[Insert screenshot(s) here.]

Graphical user interface, text, application

Description automatically generated

## 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.

Graphical user interface, application, Word

Description automatically generated

## 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

Graphical user interface, text, application, Teams

Description automatically generated

Figure 1

Graphical user interface, text, application, email, Teams

Description automatically generated

Figure 2

Graphical user interface, text, application

Description automatically generated

Figure 3

Figure 3 is the refactored code.

Figure 1 is all of the vulnerabilities

Figure 2 is the highest vulnerabilities.

Based on the results of the HTML file there are seventeen dependences but only five of them in a very high severity. I will discuss all seventeen of the dependences. I will save the ones that have the severities for the end. Classmate-1.5.1.jar which is the library for introspecting types with full generic information including resolving of field and method types. This one does not have a vulnerability ID. Next is hibernate-validator-6.0.18. Final jar which is hibernate’s bean validation(JSR-380) reference implementation. Next is Jackson-core-2.10.2.jar which is core Jackson processing abstractions (aka streaming API) Implemation for JSON. Next is jackson-databind-2.10.2.jar which is general databinding functionally for Jackson; works on core streaming API. Next is Jakarta.annotation-api-1.3.5.jar which is Jakarta annotations API. Next is Jakarta.validation-api-2.0.2.jar which is jakarta bean validation API. Next is jboss-logging-3.4.1.Final.jar which is the JBoss logging framework. Next is jul-to-slf4j-1.7.30.jar which is JUL to SLF4J bridge. Next is the log4j-api-2.12.1.jar which is the apache Log4j API. Next is logback-core-1.2.3.jar which is logback-core module. Next is slf4j-api-1.7.30.jar which is slf4j API. Next is spring-boot-2.2.4 Release Jar is the spring boot. Next is spring-core-5.2.3.RELEASE.jar which is the spring core. Next is tomcat-embeded-el-9.030.jar which is core tomcat implementation. While some of these have vulnerability IDs, I believe this could mean that they could have problems in the future as they are dependencies. Adding them to this report to be aware of them and what they are will be important to be aware of. Keep in mind that this is prior to the suppression and it is important to understand why false positives need to be ruled out. This will be discussed with the second report.

* Moving on to the ones that have the severity ratings. These five currently have vulnerabilities within the code. The first one to be discussed is a unknown severity this is the bcprov-jdk15on-1.46.jar which is the bouncy castle crypto package is a java implementation of cryptographic algorithms from APIS for JDK 1.5 – 1.7. The severity for this is unknown. There are thirteen different discrepancies for this one vulnerability. The first ones vulnerability ID CVE-2018-1000613 which deals with the Legion of Bouncy castle Legion Cryptography APIs 1.58 up to but not including 1.6. The use of externally controlled input to select classes or code can be accessed in a sense through handcrafted private keys. This seems to have been fixed at 1.6 and above. This ones severity is critical. The second Vulnerability ID is CVE-2016-1000352 deals with the JCE Provider version 1.55 and earlier. The ECIES implementation allowed the use of ECB mode which has been regarded as unsafe there for should not be supported by the provider. This is considered as high. The third vulnerability ID is CVE-2016-1000346 which deals with the version of 1.55 and earlier. This allowed for other party DH public keys to be fully validated which allowed certain users to be able to gain information on private keys to which they should not have uptained. This is considered low. The fourth vulnerability ID is CVE-2016-1000345 which deals with the version 1.55 and earlier. This dealt with the decryption failing which lead to padding oracle attack. This is considered as medium. The fifth vulnerability ID is CVE-2016-1000344 which deals with the JCE Provider version 1.55 and earlier. The DHIES implementation allowed the use of ECB mode which has been regarded as unsafe there for should not be supported by the provider. This is considered as high. The sixth vulnerability ID is CVE-2016-1000343 which deals with the version 1.55 and earlier. The DSA key generates a weak private key if used with default values. The seventh vulnerability ID is CVE-2016-100342 which deals with version 1.55 and earlier. The ECDSA does not fully validate ASN.1 encoding of signature on verification. This would inject extra elements into the signature. The eighth vulnerability ID is CVE-2016-1000339 which deals with primary engine class used for AES. This had a high table driven approach used in the algorithm it turns out that if the data channel on the CPU can be monitor the lookup table can cause a leak in information. The nineth vulnerability ID VE-2016-1000341 which deals the lack binding allowing for access to information on the signatures. The tenth vulnerability ID is CVE-2016-1000339 which deals with the encoding of signature on verification. The eleventh vulnerability ID is CVE-2018-5382 deals with file use in HMAC that is only 16 bits long which allowed the attacker to compromise the integrity. The twelfth vulnerability ID is CVE-2017-13098 is prior to the 1.03 version the API configured a weak Bleichenbacher oracle when any RSA key was exchanged which left the application vulnerable. The final vulnerability ID is CVE-2013-1624 which attacks on a noncompliant MAC check operation during the processing of malformed CBC padding. The next one is lig4j-api-2.12.1.jar which deals with “The Apache Log4j API”. This one is a low severity. The vulnerability ID is CVE-2020-94-88. This vulnerability deals with improper validation of the certificate with host mismatch. This could allow for a man in the middle attack which would leak any log messages through that apprehender. There is two in the highest severity. At the lowest severity level is log4j-api-2.12.1.jar which is the apache log4j API. The vulnerability ID is CVE-2020-94-88. This vulnerability deals with improper validation of the certificate with host mismatch. This could allow for a man in the middle attack which would leak any log messages through that apprehender. There is two in the highest severity. The first one is snakeyaml-1.25.jar which is YAML 1.1 parser and emitter for java. The vulnerability ID is CVE-2017-18640. This vulnerability deals with the alias features in the snakeYAML 1.18 that allows entity expansion during a load operation. The second one is spring-boot-starter-json-2.2.4.RELASE.jar which is the starter for reading and writing json. The vulnerability ID is CVE-2020-7712. This vulnerability deals with the package json before 10.0.0, it makes it possible to inject arbitrary commands using the parseLookup function. The dependency at the critical severity is tomcat-embed-core-9.0.30.jar which is the core tomcat implementation. This decency has eight vulnerabilities. The first vulnerability ID is CVE-2020-13935. This vulnerability deals with the payload length in a WebSocket frame was not to correctly validated in tomcat 10.0 and lower. Multiple requests with invalid payload lengths could lead to the denial of service. This has a high severity. The second vulnerability ID is CVE-2020-13934. This vulnerability deals with a h2c direct connection to Apache tomcat 10.0.0 and lower. If a sufficient number of such requests were made, an out of memory exception could occur leading to denial of service. This has a high servity. The third vulnerability ID is CVE-2020-8022. This vulnerability deals with a incorrect default permissions vulnerability in the packaging of the tomcat on SUSE Enterprise storage 5, Linux enterprise sever 12 ect.. This allows for local hackers to escalate from group tomcat to the root. This has a high servitiy. The fourth vulnerability ID is CVE-202-11996. This vulnerability deals with a specially crafted sequence of HTTP/2 requests sent to Apache tomcat 10.0.0 and lower. This could trigger high CPU usage after a few seconds. If to many requests were made then the server could become unresponsive. This is a high severity. The fifth vulnerability ID is CVE-2020-9484. This vulnerability deals with using apache tomcat 10.0.0 and lower an attacker is able to control the contents and name of a file on the sever, and the sever is configured to use the persistence manger with a file store, which is configured by attuites class. The attacker mush know the path in order to succeed. This is a high severity. The sixth vulnerability ID is CVE-2020-1938. This vulnerability deals with the Apace JServe protocol to which care must be taken with incoming connection. If this is exposed it could exploit configured IP addresses. Which would allow them for remote configurations. This is a critical seveity. The seventh vulnerability ID is CVE-2020-1935. This vulnerability deals with HTTP header parsing code used an approach to end-of-line parasing that allowed some invalid headers to be parsed as valid. This lead to the possibility of request smuggling. This is a medium severity. The eighth vulnerability ID is CVE-2019-17569. This vulnerability deals with refactoring present in apache tomcat 9.03 and lower. This lead to a regression that was invalid transfer-encoding headers were incorrectly processed leading to the possibility of HTTP request smuggling if it was located behind a reverse proxy incorrectly. This is a medium servity.

### Interpret Results

As shown above there was seventeen dependencies. So most of these did not have a vulnerability ID. There are four of them that were marked as potential threats based on the version that was being used. The others could have been likely to be false positives. The problem with false positives are that it can take time and resources away from the problems that are most present and most crucial. The ones that are most crucial are the ones that are most open to attacks. A lot of the above vulnerabilities that were talked about can be controlled by access control a suggestion that I would suggest would be creating a centralized in enforcement layer. This way the code as certain checks as it goes through the code to prevent people later. As well as having a server-side trusted data should help to drive access control. Also discussed as having a proxy issue which being able to change a person’s entailment on there server in real time could help to prevent attacks before or while they are happening. Also having defense in depth, meaning that having defense tactics throughout the code, will help to prevent attackers to get to the root, or gaining information that they should not have that would link to others information. With these strategies it may be able to help.

## 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.

[Insert screenshot(s) here.]

Graphical user interface, text, application

Description automatically generated

Figure 4

Graphical user interface, text, application

Description automatically generated

Figure 5

Looking at the dependcey report and doing a manual review I first took a look to see what decencies are active within this code. Based on this I then created a suppression file. Dependency Report 2: Without False PositivesReport: I am not going to restate it from above but the twelve that did not have an severity ratings. These are a part of the dependencies but do not have any known errors had this time. So these are not false positives the report is allowing us to know what these dependencies are, and what they are used for. This helps to know that if there is an error at some point shows up on one of these that are in the Library of our code. This helps to keep these in mind for the future. I ended up suprresioning the bouncy castle because this was not shown within the code.

Inteptert the Results: The parts that I took out are what is false positives. The first thing that I noticed when taking out the results is how they did not apply to the versions that were currently being used. For example a couple of these that were suppressed earlier or later versions that did not include the in the range of the versions that this issue impacts of the current version. When showing other team members or clients the report of where the issues are occurring this helps to make it easier to read, and know what needs to be worked on to ensure that the application is more secure. This report gives us things that do not have an issue, but should be kept an eye on, as well as what we need to watch for any security issues and what needs to be the most protected to due issues currently. Some of these fixes could just be an update to the version that is currently being used as later versions may not have any know errors, failures, hacks etc.

Next I am going to jump to the second level of architecture as it deals with doing a manuel review of the code. Based on the second level of the VAPFD which is the code review aspect which does static testing and takes a look at views, models, controllers, data access, services, plug-ins, APIs. The first thing that I am going to highlight is the views, controller, model. Due to this being a web application there is a controller which accepts input from the user and converts it to commands for the model or view. These three tend to work together. Figure 5 is an example of this is the local host as the user can access this and this shows the communication of model and view. In this image you can also see there is a rest controller this is a controller that is used for the spring web application. The entire application is an example of an API. This is due to this being a web application. An example of a plug in would be the imports that it is using from the spring work application. Based on the refactored code and the data now being encryted the data access will be limited to the people that have access to those keys only.

## 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.

First thing that I am going to discuss all the areas of security. The seven areas of architecture are Input Validation, APIs, Cryptography, Client/Sever, Code Error, Code Quality, and Encapsulation. These deal with the first level of Architecture. The areas that are affective for this company will be APIs, Encapsulation, Code Quality, Code Error, Input Validation, Client/Server. I do not believe that Cryptography is going to be an area of security that will pertain to this company. Next I am going to break down each aspect of the seven areas of code and explain why they could be a area of security threats or not. Starting with APIs, this was started in the company’s scenario that they use a RESTful web application programming interface (API). Already knowing that this is something that the company uses this is area that will need to be secured, and ensure that it monitored for an potential problems with the API. Often an API is a tool that software developers use to be able to push solutions faster, if a external threat was to access this it could harm various different aspects of the web application. Artemis financial uses its own custom software, this being said this is where we need to discuss the quality of the code, any code errors, and encapsulation. Encapsulation deals with secure data structures are an important part to any code. Secure data structures are what helps to give the program/application the proper layout, this will deal with items such as classes, object etc. Next thing that I am going to talk about is code quality which deals with secure coding practices and patterns. For any code this is important. The better quality the code the more secure it will be. If the code is poorly written the easier it will be open vulnerabilities. So this would be dealing with the code itself. Next on the list is code error which deals with secure error handling. This could be dealing with anything from errors created while writing the code, defects, bugs, etc. This will happen throughout the entire lifecycle of the code. The next vulnerabilities that I believe that is relevant to this client/sever. Due to this being a web application this is going to be dealing with client and servers the main vulnerability that I am aware at the moment for this one would it would open up to hackers if not handled properly. Next is input validation which deals with secure input and representations. I believe that input validation will be important to this company as there is a lot input with all the information that they will put in for their clients. The one that I am positive is not a vulnerability for this code is Cryptography which deals with encryption use and vulnerabilities based on the dependcy report we are able to see that there are vulnerabilities. Next is to come up with a plan to help stop those vulnerabilities and ensure that we are not using that version of that dependency. Also the data is now encrypted which helps when it comes to protecting the data. This company deals with some very sensitive information and it can be harmful if not encrypted. Which refactoring the code helped to be able to do that and only allow sures that have that key to be able to access that information.

Starting with the most important part is adding a layer of protection is the data. This is what the encrypting messages/data are used for. Then I added a hash function to ensure that the it was random. This helps each piece of data to be unique and harder to break into. The next layer is to the web application and only allowing certain users to be able to access the information. Any information put in will be encrypted as well. This is where their clients will have certain login to be able to identify them. This helps with access control and not allowing other users to have control. This helps the company from the code to the application. Layers of protection help to secure and makes it harder to get information as there are multiple layers to go through.

The first thing I would do to start out with some best practices is by laying out a security policy. This could be for how regularly that the system is monitored to look for vulnerabilities. For some web applications it would need to be constantly monitored. The next best practice is to encode the data prior to using it. This way as the data is changed it is protected from start to finish. Which for Atermis Financial which will help to keep the companies information secure as well as your clients will help your business. The encryption keys will only allow certain people. I would want to only let one or two people be in charge of handling the keys as this will help to protect them from other users. The less people that know the better secure that the information is.

References

(n.d.). Retrieved October 04, 2020, from <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html>

(n.d.). Retrieved from file:///Users/cinnamon/Desktop/dependency-check-report.html#l10\_a55e6d987f50a515c9260b0451b4fa217dc539cb