****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Instructions 3](#_Toc32574609)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 4](#_Toc32574612)

[3. Manual Review 4](#_Toc32574613)

[4. Static Testing 4](#_Toc32574614)

[5. Mitigation Plan 4](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **20 September 2020** | **Cinnamon George** |  |

## Client



## Instructions

Deliver this completed vulnerability assessment report, identifying your findings of security vulnerabilities and articulating recommendations for next steps to remedy the issues you have found.

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

## Developer

Cinnamon George

## 1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Consider the following regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions about secure communications to consider?
* What external threats might be present now and in the immediate future?
* What are the “modernization” requirements that must be considered, such as the role of open source libraries and evolving web application technologies?

I have been tasked with examining Artemis Financials web-based software to identify any security vulnerabilities. The first thing that I am going to interrupt is client needs to understand what the security requirements are going to be. Artemis Financials deals with a ton of personal information from their clients. Some of the items that the work with their clients on is creating individualized plans for savings, retirement, investments, and insurance for their patrons. Dealing with all this personal information the value of secure communications, code, web application will be high. Even if someone was able to just get information on just one client, this would then could cause attacks on their clients. This could lead to other vulnerabilities. Based on the scenario I do not believe that there are any international transactions. This is where some further information would be needed. For example, does the company allow for the clients to be able to update their information or ask for money out of their savings plan as in most cases this can be transfer into a different account for them to be able to use. As of right now there is no government restrictions. The only thing that I would say is to be careful with any information that was government issued. An example of this would be a client’s social security.

Based on the scenario it seems that they are not using a very up to date system. External threats that are currently present or could be in the future could include hacks, breeches, infection with the malware, etc. Some of these external hacks could lead to the code source and be able to make changes that could affect the company, it severs for example which could lead down the road to even bigger problems. Some of the modernization requirements could include user access controls to be able to make changes, this would be important if their clients have access to be able to update or see their information. This was that it is controlled. Another thing would be to see what the current vulnerabilities are and be able to come up with a solution to be able to prevent those from happening. There could also be potential threats to the APIs which so far has been restful but could potentially have problems in the future. Also, any improvements made to the current security implementations will to provide protection to everyone’s information from the companies to their clients.

## 2. Areas of Security

Referring to the Vulnerability Assessment Process Flow Diagram, identify which areas of security are applicable to Artemis Financials software application. Justify your reasoning for why each area is relevant to the software application.

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. This is not something a believe because it is not dealing with solving codes, it could be if the code changed, but the code is not encrypted as far as I am aware just based off of the scenario it does not seem that they have anything encrypted. I will say that this is one that I would need further information because this could be something that comes into play with their signatures that validated the user. Based on the scenario I am unsure of exactly what it is asking.

## 3. Manual Review

Continue working through the Vulnerability Assessment Process Flow Diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

A screenshot of a computer screen

Description automatically generated

Figure 3.1

A screenshot of a computer screen

Description automatically generated

Figure 3.2

A screenshot of a computer screen

Description automatically generated

Figure 3.3

A screenshot of a computer screen

Description automatically generated

Figure 3.4

A screenshot of a computer screen

Description automatically generated

Figure 3.5

I am going to start out by stating the second level of the vulnerability assessment process flow chart that will be taken a look at through the manel review of Anthemis Finicals code. This includes views, models, controllers, data access, services, plugins, and APIs. The first thing that I am going to discuss is APIs. I have not been able to find them within the code. (This is something I have not seen but will come with more experience that I will be able to identify.) This is something that would be a vulnerability and will have to ensure that it is kept secure. This was identified by the scenario. The next three that I am going to discuss all work together in a model. This would be the views, the models, and the controllers. Refer to figure 3.1 and 3.5. These are a few examples of the controllers that are used in this company’s software. The controller accepts the input from the user and accepts and converts it to commands for the model. These three tend to work together. An example would be the Greeting controller. The expressive command that is implemented for input will be what the user will put the input into. This is looking for users input of the name or the name of the company that added within the application. Another thing that should be noted with this is the endpoint it evaluates the expression passed rather than just reading it as a string for display, and the number endpoint which is vulnerable to an index out of bounds condition. The next thing would be the model. The model is could be a number of things. Within the resources I did notice that I spring framework was being used. A model would also reference the classes, objects within them as this creates the data structures. Refer to figure 3.4 for an example of the classes within this code. This is a greeting class. This is an example of a model within the code. There are various levels of security. Classes are one that need to be evaluated for code quality as well as errors, will help to prevent any vulnerabilities. The view is the user who in this case is going to be customers or employees of Atemis Finical. Refer to figure 3.2 as this is an example of a user class. This is where validations would need to be made in order to be sure that only the people that are allowed to have access. For example the customer will not need to have access to certain aspects of information, they may just be able to update information. This also falls into the data access of who is allowed to access certain types of information. This part will deal will that type of validation. Next is a service this is something that runs in the background without anyone’s knowing. An example to this would be the Date and Time class. Refer to figure 3.3. The documents and time stamps used for their work will be time stamped with this program. A service is a program that will run in the background without the user knowing, this is a prime example of it. Based of the code I do not believe that there is a plug in. This is a group of external code that is used to help the code. From my review I was not able to find any of these. For this company there is going to need to be a lot of code quality and error check, before moving on to more complicated matters.

## 4. Static Testing

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from dependency check report. Include the following:

1. The names or vulnerability codes of the known vulnerabilities
2. A brief description and recommended solutions provided by the dependency check report
3. Attribution (if any) that documents how this vulnerability has been identified or documented previously

A screenshot of a computer

Description automatically generated

Figure 4.1

A screenshot of a computer

Description automatically generated

Figure 4.2

The first set that I am going to discuss from the HTML report is the ones that could be false positives, or what they need to be aware of that could be problematic in the future. Based on the results of the HTML file there are eighteen dependences but only five of them in a very high severity. I will discuss all eighteen of the dependences. I will save the ones that have the severities for the end. For this set please refer to figure 4.1 above. The first one is classmate-1.5.1.jar which deals with “library for introspecting types with full generic information including resolving of field and method types.” The next one is hibernate-validator-6.0.18.Final.jar which deals with “Hibernates Bean Validation (JSR-380) reference implementation”. The next one is Jackson-core-2.10.2.jar which deals with “Core Jackson processing abstractions (aka Streaming API), implementation for JSON”. The next one is Jackson-databind-2.10.2.jar which is “General data-binding functionality for Jackson: works on core streaming API”. The next one is Jakarta.annotation-api-1.3.5.jar which deals with the “Jakarta Annotations API”. The next one is Jakarta.validation-api-2.0.2.jar which deals with “Jakarta Bean Validation API”. The next one is jboss-logging-3.4.1.Final.jar which deals with “The JBoss Logging Framework”. Next is jul-to-slf4j-1.7.30.jar which deals with “JUL to SLF4J bridge”. The next one is logback-core-1.2.3.jar which deals with the “logback-core module”. Next is the slf4j-aoi-1.7.30.jar which deals with “The slf4j API”. The next one is spring-boot-2.2.4.RELEASE.jar which deals with “Spring Boot”. The next one is spring-core-5.2.3.RELEASE.jar which deals with “Spring Core”. The next one is tomcat-embed-el-9.0.30.jar which deals with “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.

The next set that I am going to discuss are the ones that are not false positives and are the ones that current vulnerabilities. For this set please refer to figure 4.2 above. The first one is bcprov-jdk15on-1.46.jar which deals with “The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 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. The next one is snakeyami-1.25.jar which deals with “YAML 1.1 parser and emitter for java”. This one is a high severity. The vulnerability ID is CVE-2017-18640. This vulnerability deals with the alias features in the snake YAML 1.18 that allows entity expansion during a load operation. The next one is spring-boot-starter-json-2.24.RELEASE.jar which deals with “Starter for reading and writing json”. This is a high severity. 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 next one is tomcat-embed-core-9.0.30.jar which deals with “Core Tomcat implementation. This is a critical severity. 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 severity. 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 etc. This allows for local hackers to escalate from group tomcat to the root. This has a high severity. 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 severity. The seventh vulnerability ID is CVE-2020-1935. This vulnerability deals with HTTP header parsing code used an approach to end-of-line parsing 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 severity.

## 5. Mitigation Plan

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial’s software application.

The first steps that I would recommend would be to start out with fixing the errors found within the code, and ensuring that the quality of the code is up to par. This is the easiest thing to be able to start with. This will mainly with troubleshooting, debugging etc. My next suggestion is another simple one, is making sure that the system is update with the latest versions. Based on the static testing a lot of these vulnerabilities were based on earlier versions. This would be as simple as making sure that the software and all the resources that this is using like for example the spring framework is up to date on the latest version this would handle a lot of the current vulnerabilities. One suggestion that I would make is to add identity and access control. This is not to the web application but to the actually code itself that nothing can become duplicated or changed without permission, or known user. This would create a layer of defense, limits what the users of the web application has access to. The next suggestion I would make would be to make a hard code policy. This can be checks throughout the application. This would make it harder for to make changes to. Another item that I would suggest would be fail open, as a attacker could use a null which could cause an exception. It never hurts to be on the safe side. My next suggestion would be to build centralized access control engine/decision maker. AS this will help to control was is process through the input. The next suggestion that I would make would be to create output encoding that escapes the users input before it makes a dynamic query. With the input and output, as well as the outside source. I believe that this would be the best to start with to help to protect the code. Like an program it will have to be monitored quickly for any changes that do occur bugs, hacks, defects, ect and make those changes and fixes quickly to prevent any further problems. As shown above in the static testing there was eighteen dependencies. So most of these did not have a vulnerability ID. There are five 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 persons 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 in preventing the vulnerabilities. Like stated above updating the versions will help to elimate a lot of these problems, especially if they are still using a older version.

References

George, Cinnamon (2020). Maven Dependency Check Report. Generated from: Long, Jeremy (2020). Maven Dependency Check Report. Generated from: <file:///Users/cinnamon/Desktop/dependency-check-report.html#l1_991c96a4e31e6c19e2b9136c8955bd423f2dc4c7>