****

# 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 5](#_Toc32574612)

[3. Manual Review 6](#_Toc32574613)

[4. Static Testing 7](#_Toc32574614)

[5. Mitigation Plan 9](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **01/20/2021** | **Jon Frodin** | **This is the first version of this document. It assesses threats to Artemis Financial and proposes a mitigation plan to prevent the exploitation of vulnerabilities. It includes, as well, a code review and dependency assessment.** |

## 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

Jon Frodin

## 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?

*Secure Communication*

Secure communication is vital when dealing with software that uses personal information. Even with software that doesn’t deal with personal information, secure communication is needed to prevent against attack. When personal information is involved, though, that need becomes far more pressing. Artemis Financial deals with financial plans and insurance for their clients, among other things, and those plans are specific to the individual. Information must be gathered about an individual in order to create a plan, so the communication and data must go back and forth. People will not be willing to provide sensitive information like salary and investment portfolios to a company that cannot keep that data secure and completely away from prying eyes.

*International Transactions*

Upon review of the documents provided by Artemis Financial, and the existing code base, I see nothing that indicates the presence of international transactions. However, there is also nothing that says or indicates that there will *not* be international transactions. The principles of security would dictate that we treat this as a possibility even if it is not outright stated, because it is also not outright excluded. With that being said, Artemis Financial will need a robust method of securing communications, both foreign and domestic.

*Governmental Restrictions*

In the United States, there are currently no regulations on the encrypting or securing of communications that need to be assessed. However, this is still something to plan ahead for in the United States because bills are being introduced with enough regularity to be cause for concern. The most recent proposal was put before the US Senate on June 23, 2020 called the Lawful Access to Encrypted Data Act.[[1]](#footnote-1) If a bill like this were to pass, our system would need to be designed in such a way that it would be easy to institute a compliance module if necessary. Since it is undetermined at this time whether international communications exist, it is important at this point to note that if Artemis Financial does do international business, then communications need to be accessible by law enforcement in countries where it is required. Many countries have this requirement, including Australia, Canada, the UK, France, and more.[[2]](#footnote-2)

*External Threats*

External threats abound in the modern world. Since Artemis Financial will be working with sensitive financial and personal information, the risks of attack are even greater. The number one threat I see for this company is an individual or small group attack. I do not think any government or other high-level actor would be interested in taking this company on. Threats can come from input in the form of invalid inputs, from SQL injection, from leaking data into the browser history (username and password, for example), and from dependency vulnerabilities. Bots can also be a threat. They can be programmed to try usernames and passwords until they are able to log in successfully. This could result, if attempted enough times and quickly enough, into a denial of service. Threats also exist in authorized users accessing data they are not authorized to access. This could be an end user, for example, accessing administrator files through privilege escalation. A robust API would also be needed to thwart potential attacks.

*Modernization Requirements*

Open-source libraries and even some actively maintained software can present unforeseen threats to the system. Modernization comes with many challenges. For example, this company deals in financial planning. They may want to use compiled stock/bond information put out by another company. Integrating this into the system could pose additional threats to the system. Other software out there that could be used to enhance this program could pose additional risks if their API and other structures are not built securely. Evolving web-application technologies are also something to consider when modernizing. Web applications can do more and more now, but that often exposes the application to security risks when other applications need permissions or have vulnerabilities. Also, attackers get more sophisticated as time goes on and find new ways to penetrate a system. It is important to default to fail in these instances.

## 2. Areas of Security

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

The primary areas of security include the following:

* *Input Validation –* Input validation is crucial any time you are dealing with input. This program does allow input, or at least leave itself open to input. For example, line 12 of CRUDController.java allows input to be passed in the form of an expected string.[[3]](#footnote-3) Validation of that string, to avoid failures or SQL injection, will make this program secure. Input validation is essential because we are asking the user to enter something into our system. If they have malicious intent, this could do serious damage or result in unauthorized access of private data.
* *APIs –* Because this application is designed to be run in an outside environment, such as a web browser, a good API will be essential. This defines how third-party software interacts with our program. The API will determine what methods are acceptable or unacceptable as well as what data is acceptable or unacceptable. Furthermore, this project may involve using third-party software which this software may depend on. To mitigate the risks posed by using someone else’s software, a secure API is vital.
* *Cryptography –* Cryptography is included in this list because of the uncertainty of international transfers. As discussed above, cryptography is regulated when it is an export item, so if secure, encrypted communications are used and Artemis Financial does do business overseas, then steps must be taken to secure the data but also do so in a manner that complies with the laws of both the United States and the destination country.
* *Code Error* – I included code error because the description of that is secure error handling. I see this as going hand-in-hand with input validation because errors caused by an input need to be handled properly. Any errors, especially those relating to input validation, need to be handled properly. Errors also need to be handled properly to avoid privilege escalation and mitigate other vulnerabilities.
* *Code Quality –* Code quality is included because it is an area of concern whenever an API and input validation are present. Quality code will ensure that no data is unintentionally exposed and it will also ensure that methods that are restricted from end users will not be made available to them. In short, only authorized users will get the data and methods that match their tier of authorization if quality code is used.

## 3. Manual Review

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

The following represents my findings upon manually reviewing the code base provided with this project for Artemis Financial.

* On reviewing the code, I first looked for input validation. The first place I looked for validation was the pom.xml file included in the code base. I did this because they could have written some Apache type validators in this file.[[4]](#footnote-4) I then looked to places where user input is coming in. One primary place was in the greeting controller. The input here is just assumed to be a string and not passed through any sort of validation.[[5]](#footnote-5) I was able to enter many things into the program, but was not able to verify if anything was taken in because there was no output (addressed under API). Another area that requests information also expects a string, but nothing works on it and it has no validators built-in.[[6]](#footnote-6)
* Next, I looked for an API and if it was present, would have evaluated how it worked. Instead, I found absolutely nothing. Even without a visual or working API, the program can accept data. This particular program accepts that data in a completely unsecure way. It accepts data when needed within the URL rather than in the POST method, which will leak into the browser history and be a potential exploit.[[7]](#footnote-7) In this program, even when text was entered into the URL, nothing would display and every page would yield an error, but it could still be vulnerable because it is taking raw user input and doing something with it. Also, this program has no discernable way to interact with it unless you are looking at the source code. If this were published, there would have to be a document or portal telling you how to actually interact with it. An API needs to have some sort of way to interact that is discernable to the user if it is to be considered a RESTful API.[[8]](#footnote-8)
* I followed this up by looking for any sort of cryptography in case there are international communications. This was a quick review because I did not find any sort of encryption at all. If international communications are needed by Artemis Financial, they will have to develop some sort of encryption system that complies with United States and international regulations and laws.
* Fourth, I looked for code error particularly in error handling. The one that stood out to me was in the DocData.java class. This class had a method for reading a document when given a key and a value. This does include a try/catch block but does not have any actual error handling.[[9]](#footnote-9) No other classes have any sort of error handling to speak of, so there was nothing else to evaluate there.
* Lastly, I looked at code quality. The code that exists is okay, though not entirely functional. It needs to be expanded a lot in order to get this program fully ready. The API, as mentioned, makes nothing available to the end user informing them what is expected. Input validation needs to be implemented. Input must be moved from the URL to the POST method to secure the data and prevent it from leaking into browser history. Code quality is a superset of the API and input validation. The issue with this aspect is to actually code, and code securely, the subsets.

## 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

All of the dependencies and vulnerability ID’s found below are from the same reference, footnoted here.[[10]](#footnote-10)

* bcprov-jdk15on-1.46.jar – Bouncy Castle Crypto package works on cryptographic algorithms for JDK 1.5 through JDK 1.7 (vulnerability ID: cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-cryptography-api:1.46.\*.\*.\*.\*.\*.\*.\*)
  + This potentially has several issues. One of the major ones, involving cryptography algorithms, can have a big impact on secure communications as well as exposing sensitive data. This version of Bouncy Castle (1.46) has a vulnerability where the second party DH public key is not fully validated and can lead to invalid keys being used to reveal details about the other party’s private key.[[11]](#footnote-11) If the private key of another user is found, it can make encryption useless. Even exposure of the algorithm used can lead to finding the key and render those “secure” communications essentially public.
* jackson-databind-2.10.2.jar – General data-binding functionality for Jackson that works on core streaming API (vulnerability ID: cpe:2.3:a:fasterxml:jackson:2.10.2.\*.\*.\*.\*.\*.\*.\* - AND – cpe:2.3:a:fasterxml:jackson-databind:2.10.2.\*.\*.\*.\*.\*.\*.\*)
  + This vulnerability can allow an attacker to submit an XML file that defines something outside of the intended scope of the application and have that data sent back as an error message.[[12]](#footnote-12) This can be used to allow an attacker to view sensitive data if they submit something that accesses files in the right way. The error message can be informative, but sometimes can be overly-informative in this instance.
* log4j-api-2.12.1.jar – Apache Log4j API (vulnerability ID: cpe:2.3:a:apache:log4j:2.12.1.\*.\*.\*.\*.\*.\*.\*)
  + This vulnerability is capable of impacting the SSL. The certificate checks performed during a session could either be not validated at all or validated incorrectly. This can allow software to connect to a malicious host while believing it is a trusted host.[[13]](#footnote-13) The lack of validation of an SSL certificate essentially renders that level of protection useless and gives users a false sense of security by seeing the little lock icon or the green address bar.
* snakeyaml-1.25.jar – YAML 1.1 parser and emitter for Java (vulnerability ID: cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25.\*.\*.\*.\*.\*.\*.\*)
  + This vulnerability can lead to denial of service (DoS). The reason is because the document type definition does not control the number of recursive definitions inside it. This can cause exponential growth when the data is parsed.[[14]](#footnote-14) DoS happens when a website or server has more coming in than it can handle and shuts down. An infinite, or even a very large but not infinite, recursion can be seriously taxing on any computer. It is critical to defend against this since DoS and DDoS attacks are among the most common.[[15]](#footnote-15)
* spring-core-5.2.3.RELEASE.jar – Spring Core (vulnerability ID: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release.\*.\*.\*.\*.\*.\*.\* - AND – cpe:2.3:a:springsource:spring\_framework:5.2.3:release.\*.\*.\*.\*.\*.\*.\* - AND – cpe:2.3:a:vmware:springsource\_spring\_framework:5.2.3:release.\*.\*.\*.\*.\*.\*.\*)
  + This vulnerability impacts versions of Spring all the way up to 5.2.8 and the vulnerability is that protections against RFD (reflected file download) attacks can be bypassed using the jssessionid path.[[16]](#footnote-16) RFD is quite an interesting and scary exploit. It can be used to completely control the targeted system.[[17]](#footnote-17) This exploit gets around the more traditional methods of blocking RFD.
* tomcat-embed-core-9.0.30.jar – Core Tomcat implementation (vulnerability ID: cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\* - AND - cpe:2.3:a:apache\_software\_foundation:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\* - AND - cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*)
  + This dependency involves many vulnerabilities that are known and documented. One major vulnerability is that it can be exploited to give unauthorized people access to sensitive information. This can work when the code inserts information directly, indirectly, or that the code manages resources that intentionally contain sensitive information and the resources become available to unauthorized users.[[18]](#footnote-18) If a user can access a method that involves sensitive information, they could accidentally stumble upon it. They could also deliberately mess with the program method to try and further exploit it to gain that data that is not protected.
* tomcat-embed-websocket-9.0.30.jar – Core Tomcat implementation (vulnerability ID: cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\* - AND - cpe:2.3:a:apache\_software\_foundation:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\* - AND - cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*)
  + See previous – details are the same and the CPEs are the same.

## 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 general mitigation plan involves transforming this DevOps pipeline into a DevSecOps pipeline. Since this project is still in the early stages, it would help to get everyone in the development team in a security mindset now. That way things will be built with security in mind and we can go forward with this project the right way. If we do this, then things like input validation, which has been cause for concern for the NIST many times,[[19]](#footnote-19) can be mitigated before this project goes into operation.

Specifically, the threats identified above have mitigation measures to be taken as well. Again, the early stage of development works to our advantage here. Those measures will be outlined in the list below.

* bcprov-jdk15on-1.46.jar – mitigating this threat will involve using a version numbered 1.56 or later because the later versions check the key parameters on agreement calculation.[[20]](#footnote-20) Keeping keys secure is what makes cryptography worth it in the first place. It is of critical importance to have those keys calculated properly and stored. Another part of this is making sure that they are in fact validated properly before any data is exchanged.
* jackson-databind-2.10.2.jar – mitigating this threat will involve configuring the XML parser and validator to disable eternal entity expansion.[[21]](#footnote-21) This means that attackers will no longer be able to submit altered DTD files within the XML documents because the parser will catch it and ignore the malicious code, if done properly.
* log4j-api-2.12.1.jar – mitigating this threat will involve not using certificate pinning or to make sure that all properties of the certificate are validated before pinning with a special focus on the hostname.[[22]](#footnote-22) Browsers trust the SSL certificate and so do the end users of those browsers. It would not do us much good to give people a false sense of security when using our platform to have an SSL certificate that is not enforced properly.
* snakeyaml-1.25.jar – mitigating this threat will involve prohibiting the use of DTDs or to use a parser that limits the expansion of those recursive definitions.[[23]](#footnote-23) Recursive calls can generate huge loops, either by mistake of by design. I use recursion often in code and it always needs to be checked. If a parser limits the expansion, that could stop this sort of attack in its tracks. Some recursion may take place, but it would be prevented from expanding to such an extent as to actually cause a DoS.
* spring-core-5.2.3.RELEASE.jar – There are no mentioned mitigation measures listed associated with this vulnerability, however, the current version is not listed in this set of vulnerable Spring frameworks. Therefore, a mitigating measure could be upgrading to the latest version, 5.3.1.[[24]](#footnote-24) Always using updated software helps to ensure the latest patches to known vulnerabilities are applied. That being said, it is also risky to use the newest thing because it doesn’t always have the real-world exposure to find, and therefore identify publicly, a vulnerability yet.
* tomcat-embed-core-9.0.30.jar & tomcat-embed-websocket-9.0.30.jar – mitigating these threats will involve using the principle of least privilege, along with safe zones in the system, will help to mitigate this threat. This allows sensitive information to remain within the safe zone.[[25]](#footnote-25) If sensitive data is places within the bounds of a secure structure, and that structure has authorization enforcement mechanisms in place, the leaking of that data becomes dramatically less likely.

1. https://www.judiciary.senate.gov/press/rep/releases/graham-cotton-blackburn-introduce-balanced-solution-to-bolster-national-security-end-use-of-warrant-proof-encryption-that-shields-criminal-activity [↑](#footnote-ref-1)
2. https://www.loc.gov/law/help/encrypted-communications/gov-access.pdf [↑](#footnote-ref-2)
3. CRUDController.java lines 12-13 [↑](#footnote-ref-3)
4. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch01.html#ch01> § Input Validation: Apache Struts [↑](#footnote-ref-4)
5. GreetingController.java lines 15-18 [↑](#footnote-ref-5)
6. CRUDController.java lines 12-13 [↑](#footnote-ref-6)
7. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch01.html#ch01> § HTTP/S POST Request [↑](#footnote-ref-7)
8. <https://spring.io/guides/tutorials/rest/> § What makes something RESTful? [↑](#footnote-ref-8)
9. DocData.java lines 21-35 [↑](#footnote-ref-9)
10. Dependency-Check-Report (included in submission documents as an HTML file) [↑](#footnote-ref-10)
11. <https://nvd.nist.gov/vuln/detail/CVE-2016-1000346#vulnCurrentDescriptionTitle> [↑](#footnote-ref-11)
12. <http://cwe.mitre.org/data/definitions/611.html> [↑](#footnote-ref-12)
13. <http://cwe.mitre.org/data/definitions/295.html> [↑](#footnote-ref-13)
14. <http://cwe.mitre.org/data/definitions/776.html> [↑](#footnote-ref-14)
15. <https://nordvpn.com/blog/hacking/> [↑](#footnote-ref-15)
16. <https://nvd.nist.gov/vuln/detail/CVE-2020-5421> [↑](#footnote-ref-16)
17. <https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf> [↑](#footnote-ref-17)
18. <http://cwe.mitre.org/data/definitions/200.html> [↑](#footnote-ref-18)
19. <https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=input&search_type=all> [↑](#footnote-ref-19)
20. <https://nvd.nist.gov/vuln/detail/CVE-2016-1000346#vulnCurrentDescriptionTitle> [↑](#footnote-ref-20)
21. <http://cwe.mitre.org/data/definitions/611.html> [↑](#footnote-ref-21)
22. <http://cwe.mitre.org/data/definitions/295.html> [↑](#footnote-ref-22)
23. <http://cwe.mitre.org/data/definitions/776.html> [↑](#footnote-ref-23)
24. <https://en.wikipedia.org/wiki/Spring_Framework> [↑](#footnote-ref-24)
25. <http://cwe.mitre.org/data/definitions/200.html> [↑](#footnote-ref-25)