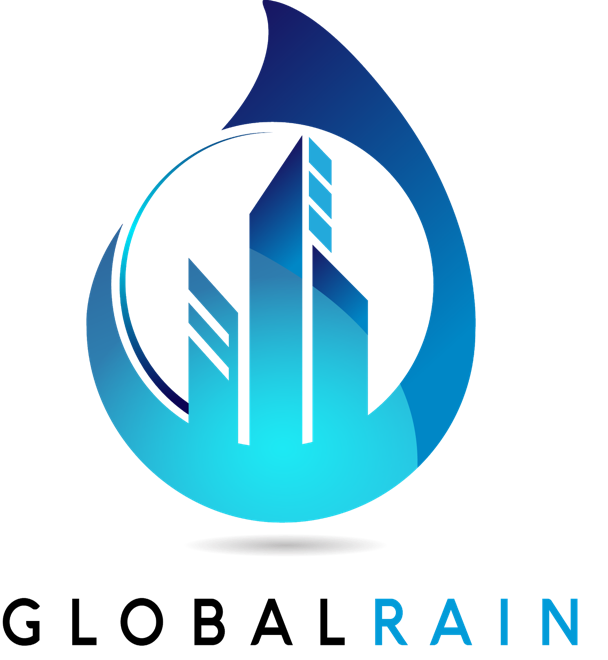
****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

**Table of Contents**

[Document Revision History 3](#_heading=h.gjdgxs)

[Client 3](#_heading=h.30j0zll)

[Instructions 3](#_heading=h.3znysh7)

[Developer 4](#_heading=h.2et92p0)

[1. Interpreting Client Needs 4](#_heading=h.tyjcwt)

[2. Areas of Security 4](#_heading=h.3dy6vkm)

[3. Manual Review 4](#_heading=h.1t3h5sf)

[4. Static Testing 4](#_heading=h.4d34og8)

[5. Mitigation Plan 4](#_heading=h.2s8eyo1)

## Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **5/23** | **Ryan Mitchell** | **Finalizing Report** |

## 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

Ryan Mitchell

## 1. Interpreting Client Needs

The company Artemis Financial is a financial consulting company that develops individualized financial plans for savings, retirement, investments, and insurance for their patrons. Given this, secure communications within the company are important, as well as to the client. However, the use of cryptography is overkill in my opinion. While sensitive data may exist within the spreadsheet files or financial plans, there should not be any directly threatening information to the client base such as credit card numbers, or social security present within the files. As far as transactions go, there may be some minor charges across clientele, but there is not any large scale currency transfer as this is not a banking application. Given the universal encouragement for secure data transfer, certain practices should be adhered to, but are not necessarily mandated by government agencies. Various threats can always be present, but given the nature of this company’s application, I would presume potential SQL injections, access control attacks such as privilege escalation, http request manipulation, cross site request forgery, etc. As for modernization, it is important to always test and use new dependencies as attacks will improve alongside defensive techniques. Open source libraries must always be tested for vulnerabilities, and web applications must always be on the lookout for new attack methods.

## 2. Areas of Security

Despite little information about the business dealings of this company and assuming it has a competent REST API, the pertinent areas of security I have identified as the most crucial to this application are input validation, client/server, code error, and encapsulation. With the application giving direct access to the consumers, I can assert that secure distribution is a must in the client/server structure of the application. With potential hijacking requests in mind, input validation is also important to prevent any attacks that are attempting to retrieve personal information from the client base. Cryptography could be important depending on the structure of the application. In order to prevent any other leaks, the application must also securely handle its code and data structures to help diminish any sort of attacks that depend on request manipulation and injections.

## 3. Manual Review

The first problem I recognize in my code review is a distinct lack of authentication of the current user. While there is connection testing in retrieving personal documents, there is no protection against impersonation or account data retrieval. Second, there is no RBAC or ABAC in place, let alone roles to be assigned. The inclusion of proper access control will allow for greater restrictions depending on the credentials an attacker may retrieve. Next, there is no sort of input validation to ensure that any requests being passed are not manipulated, such as character limits. The code itself is also compromised with such a small number of static fields allowing for source manipulation. Additionally, a request or action log may be used to help identify any strange or suspicious behavior as it is happening.

## 4. Static Testing

After running a dependency check using the maven plugin, all known vulnerabilities were identified. The first vulnerability was with the Bouncy Castle Crypto package that is a Java implementation that simply includes cryptographic algorithms. The dependency has CVE count of 16, meaning that there is a huge issue with it’s protection. I would recommend checking for updates, otherwise perhaps removing or replacing this package entirely. Next is a vulnerability with the Hibernate Validator vers. 6.1.2, [**CVE-2020-10693**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-10693), and involves a bug in the message interpolation that enables invalid EL expressions to be evaluated as if they were valid. This is an issue of input validation. Then there’s an issue with general data-binding functionality for the Jackson dependency, [**CVE-2020-25649**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649), where it does not have entity expansion properly secured and is a high threat of XXE attacks over data integrity. A vulnerability was also detected in the Apache Log4j SMTP appender, [**CVE-2020-9488**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9488), in which improper validation of certificate with host mismatch could allow an SMTPS connection to be intercepted by a man-in-the-middle attack. Next is an issue with the Alias feature in SnakeYAML 1.18, [**CVE-2017-18640**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-18640), which allows entity expansion during a load operation. Then we have a vulnerability with the Spring Framework, [**CVE-2020-5421**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-5421), where the current version is not properly protected against RFD attacks and may be bypassed depending on the browser used. This could perhaps be corrected with updated systems. Finally, the tomcat core and websocket both maintain a CVE count of 12 and 13 respectively, implying that both are critically vulnerable to NUMEROUS attacks, requiring major correction.

## 5. Mitigation Plan

The first step to mitigating threats is to under the types of attacks that are expected based on the vulnerabilities. With the issue of input validation, there should be measures to both detect and validate proper length of input and potentially blacklisting characters. The problem of entity expansion being take advantage of can be helped by also addressing input validation, and block input using the DOCTYPE tag. There must also be measures to validate the current client server connection to ensure that no proxies or means of interception are present in the client’s position. Overall, the first step should be to make sure all systems are properly updated. Using out of date software can be one of the biggest threats, as these companies are actively working towards patching the various holes that attackers can utilize. Once everything is up to date, the vulnerability test should be rerun. Within the application itself, there must also be some sort of role based access control to minimize the impact of leaked credentials being taken advantage of. If all these steps are taken, the amount of risks should be reduced, but a follow up static test would be beneficial once packages and dependencies are all up to date.