****

# 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** | **7/12/2022** | **Chance Roy** |  |

## Client



## Developer

Chance Roy

## Interpreting Client Needs

Secure communication allows Artemis Financial the ability to assure their clients that their confidential data will be secure with them. It appears that Artemis Financial works within their own country, there is no information stating otherwise. Since the information about what country Artemis Financial is based out if isn’t provided, there is no way for us to conclude what governmental restrictions may or may not have an impact on the project. There will always be the constant external threat of attackers utilizing things such as proxies to intercept and modify user input (Manico & Detlefsen, 2014). There are ways to protect or mitigate against such attackers, but they will always be there, looking for new ways steal information and cause harm. When building the application and using open-source libraries, we need to be sure that these libraries will be around and in use in the future, or at the very least easily replaceable if they become obsolete. This same principle can be applied to external sources or pieces of code that aren’t manufactured by us.

## Areas of Security

First, we need to handle input validation. “Input validation should seek to define what constitutes good data and reject everything else (whitelisting).” (Manico & Detlefsen, 2014, ch. 1). This needs to be done to prevent abnormal data from interfering with the system. Next, we need to secure the restful API by making sure we only accept data sent through a secure TLS. Using a TLS has become the standard for protecting against network-based threats using cryptography (Manico & Detlefsen, 2014). This will protect user access and traveling data through encryption. For data that is at rest, we could outsource to a vendor such as Keyczar which “allows you to easily encrypt and sign data, and—perhaps even more importantly—Keyczar provides basic key management functionality.” (Manico & Detlefsen, 2014, ch.6)

Just like our API connections need to be secure; we also need to ensure that our client/server interactions our secured from impersonators and attackers that are seeking to steal data. As mentioned, before we could utilize TLS to help protect data that’s in transit, but we should also use a cloud-based infrastructure like Amazon Web Services. According to Amazon’s website, AWS allows you to automate security task to avoid human interference that could lead to errors and security breaches. This is one of the many benefits to using AWS for our client’s web application. Lastly, we have code quality and code error. Code errors can result in the web application malfunctioning and displaying sensitive information through stacks traces. This can be avoided by handling “uncaught exceptions via configuration in web.xml.” (Manico & Detlefsen, 2014, ch.9) Code quality should be maintained throughout development by following the Secure Coding Guidelines for Java SE.

## 3. Manual Review

We need to restrict data that can be seen or used throughout the program. For instance, the account balance variable within the customer class on line 5 isn’t specified as private. This could be potentially harmful because other areas of the program could accidentally alter this information. We also need to check for proper input validation within the myDateTime class when the methods are implemented. They seem to be missing wrappers and native methods, which could result problems such as overflow of integers and directory traversal attacks. (“Oracle,” n.d.) In the DocData class and customer classes, I do not see data being released after it’s been called. This could result in a denial-of-service.

## 4. Static Testing

* [cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api&cpe_version=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api%3A1.46) In versions 1.55 and earlier for Bouncy Castle JCE the DSA does not fully validate ASN. 1 encoding of signature on verification as well as a plethora of other issues related to the older versions of the software. Red Hat recommends updating the software. Reference: <https://access.redhat.com/errata/RHSA-2018:2669>
* [cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aredhat&cpe_product=cpe%3A%2F%3Aredhat%3Ahibernate_validator&cpe_version=cpe%3A%2F%3Aredhat%3Ahibernate_validator%3A6.0.18) This is a flaw in Hibernate Validator version 6.1.2 Final that enables invalid EL expression to be evaluated as if they were valid. A user on the Red Hat forums suggests passing user input as an expression variable by unwrapping the context to HibernateConstraintValidatorContext, but the optimal solution would be to update the Red Hat Jboss Enterprise Application Platform which has rectified the issue. Reference: <https://access.redhat.com/errata/RHSA-2020:3464>
* [cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Afasterxml&cpe_product=cpe%3A%2F%3Afasterxml%3Ajackson-databind&cpe_version=cpe%3A%2F%3Afasterxml%3Ajackson-databind%3A2.10.2) Versions of Jackson-databind before 2.13.0 allow a Java StackOverflow exception and denial of service through nested objects. This has since been fixed recent versions of Jackson-databind and thus should be updated to maintain system integrity.
* The Jakarta Annotations API allows for absolute oath traversal because the Flask send\_file function is used unsafely. This has been documented here: <https://github.com/github/securitylab/issues/669#issuecomment-1117265726> Currently, I see no fix to the issue.
* [cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Alog4j&cpe_version=cpe%3A%2F%3Aapache%3Alog4j%3A2.12.1) There are multiple vulnerabilities related toApache Log4j2 all of which are related to specific versions of the software. Here is a reference: <https://lists.apache.org/thread/s1o5vlo78ypqxnzn6p8zf6t9shtq5143> It is recommended to update to the latest version.
* [cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aqos&cpe_product=cpe%3A%2F%3Aqos%3Alogback&cpe_version=cpe%3A%2F%3Aqos%3Alogback%3A1.2.3) In logback versions 1.2.7 and older, attackers with the necessary credentials to edit configurations files could create a malicious configuration allowing to execute arbitrary code loaded from LDAP servers. Here is a detailed reference: <https://github.com/cn-panda/logbackRceDemo> According to this link here: <https://jira.qos.ch/browse/LOGBACK-1591> The problem has been resolved in recent versions of the software.
* [cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Asnakeyaml_project&cpe_product=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml&cpe_version=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml%3A1.25) Snakeyaml version 1.18 allows entity expansion during a load operation. It is recommended to upgrade snakeyaml to a version without CVE-1017-18650, noted here: <https://lists.apache.org/thread/obr7cqy8lk0dk4gp018pplkqnqsxpkff>
* [cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_boot&cpe_version=cpe%3A%2F%3Avmware%3Aspring_boot%3A2.2.4) Spring-boot versions prior to v.2.2.11 were vulnerable to temporary director hijacking. This only affects versions or products that are no longer supported by the manufacturer. The workaround is to set the java.io.tmpdir system environment variable to a directory that is exclusively owned by the executing user will fix this vulnerability for all operating systems. Fix is referenced here: <https://github.com/JLLeitschuh/security-research/security/advisories/GHSA-cm59-pr5q-cw85>
* [cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A5.2.3) A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. Siemens has released updates for multiple products and recommends updating to the latest versions. Reference here: <https://cert-portal.siemens.com/productcert/pdf/ssa-254054.pdf>
* [cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A5.2.3) Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. It is recommended that one should not expose HTTP invoker endpoints to untrusted clients, and it’s recommended to use any other messaging format like JSON. Reference: <https://github.com/spring-projects/spring-framework/issues/24434#issuecomment-582313417>
* [cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Atomcat&cpe_version=cpe%3A%2F%3Aapache%3Atomcat%3A9.0.30) Apache Tomcat treats AJP connects as having higher trust than a similar HTTP connection. The fact that the attacker must have permissions to upload documents helps to mitigate the problem. Firewalls can also help limit access to AJP ports. Referenced here: <https://support.blackberry.com/kb/articleDetail?articleNumber=000062739>
* [cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Atomcat&cpe_version=cpe%3A%2F%3Aapache%3Atomcat%3A9.0.30) This vulnerability appears to related to the same Apache Tomcat vulnerability above, so the same prevented measures apply.

## 5. Mitigation Plan

The majority of the vulnerabilities found during the static testing already have a recommended course of action associated with them, most of which suggest updating the reported software. Each of the vulnerabilities should be gone through and fixed by following the suggested repairs. Since this website will involve a client/server relationship we need to protect against denial-of-service. This can be done by ensuring that we release resources as soon as possible no matter the situation. (“Oracle,” n.d.) To ensure the input validation runs smoothly we need to define wrappers around native methods which, “can safely perform any necessary input validation prior to the invocation of the native method.” (“Oracle,” n.d.)

**References**

Manico, J., & Detlefsen, A. (2014). *Iron-clad java building secure web applications*. McGraw-Hill Computing.

Zedner, L. (2009). *Security*. Amazon. Retrieved July 16, 2022, from https://aws.amazon.com/security/

Secure coding guidelines for java SE. (n.d.). Retrieved July 17, 2022, from https://www.oracle.com/java/technologies/javase/seccodeguide.html