****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

**Gregory Dionisio**

**Department of Science, Southern New Hampshire University**

**CS 305: Software Security**

**Dr. Vivian Lyon**

**July 16, 2022**

Table of Contents

[Document Revision History 4](#_Toc32574607)

[Client 4](#_Toc32574608)

[Developer 4](#_Toc32574610)

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

[2. Areas of Security 6](#_Toc32574612)

[3. Manual Review 7](#_Toc32574613)

[4. Static Testing 9](#_Toc32574614)

[5. Mitigation Plan 10](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/11/2022** | **Gregory Dionisio** | **Developer** |
| **1.1** | **7/16/22** | **Gregory Dionisio** | **Developer** |

## Client



## Developer

Gregory Dionisio

## 1. Interpreting Client Needs

Artemis Financial creates custom financial plans for clients in the areas of retirement, investments, savings, and insurance. This means the company handles sensitive financial information for all their clients that if acquired by the wrong people could be detrimental for their clients and for the company. Artemis Financial web application is a high threat target for hackers and the transfer of information from the client and vice versa must be secure.

Due to the nature of their business Artemis Financial will need to deal with domestic and foreign transactions in their operations whether it be held savings accounts, potential investments, a client travelling abroad, or an international customer it must be assumed that data must travel internationally and be accessible in multiple countries. Another important consideration with a financial institution such as Artemis Financial is that there are more restrictions than other websites. The Graham-Leach-Bailey Act put into law that financial institutions using web based financial transactions must disclose to the public how they keep data secure during transactions and a failure to properly keep data safe can result in serious fines on top of the moral obligations to their clients (Manico, & Detlefsen, 2015).

The main external threat is hackers who are constantly evolving new and deceiving techniques to penetrate website data especially for a financial institution which can make them money. This is best prevented by following a vulnerability assessment process flow diagram. Another potential external threat could be anything that can physically destroy the server, explosions, earthquake, floods, etc. All data should be stored in multiple locations and preferably separate geographic areas to prevent such catastrophes.

Just as hackers are constantly creating new ways to compromise a website so must developers be constantly thinking of ways to build more secure code. It is helpful to be kept up to date on the latest breaches and potential loopholes that a hacker may use so that they can be closed. Keeping software up to date will often prevent known breaches from older software versions. Open-source libraries save a lot of time and can be well updated however this should not be assumed and is often not the case. Open-source software cannot be assumed to be secure, 26% of open source libraries have vulnerabilities and most vulnerabilities haven’t even been discovered yet (Manico, & Detlefsen, 2015). There are multiple ways to both manually and electronically search through libraries to look for insecurities. People are accustomed to being able to access data often and from anywhere and this creates more opportunities for hackers. To keep up with the demand new web technologies are constantly being developed however keeping up to date with the technologies and the potential security flaws that they may have can be a full-time job.

## 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.

The vulnerability assessment diagram is used to secure a website. The first important area for Artemis Financial application to consider is their Application Programming Interface (API). One area of API’s to be cautious of is during user input. This task of an API can lead to injection attacks that can cause damage to the site. Another consideration for an API is access control. Every user should be given as little control as possible to complete their tasks and this can result in a complicated control grouping for individuals. Access control needs to be considered at the start of the project to help prevent hacker access. The site may need to integrate multiple API to get the best functionality and for each API needs to be considered at the start of the project. Another area that hackers will attempt to penetrate is during a password reset form. This is such an abused way that it is recommended that web applications “do not support the forgot password workflow as part of your website” (Manico, & Detlefsen, 2015) for banking institutions.

With a financial application such as Artemis Financial built cryptography is used to transmit information through the internet safely. As the data is traveling it can be intercepted but cryptography will ensure that the messages cannot be read. Cryptography will also be used on the server to store passwords and data so that if accessed it will be difficult to decode.

The client/server communication is another area for concern. In a web application this will go through Hypertext Transfer Protocol (HTTP) or preferably Hypertext Transfer Protocol Secure (HTTPS). HTTPS secures the transaction by making the data unseeable, unchangeable, and the proper location during a transfer. All data from and to the client creates a critical control point that can be accessed by a hacker and needs to be monitored and handled properly to ensure there is no leak in the data, or unwanted access to the system.

The best defense against potential hackers is to build a secure application before a breach occurs by using secure coding. To write secure code against vulnerabilities takes effort. One should know common hacking tricks and use input validation, query parametrization, variable binding, and other secure coding techniques to mitigate these attacks. Once code is complete it needs to undergo multiple security analysis such as manual testing, dynamic testing, and static testing. These programs can sift through every file in the application and look for known vulnerabilities written into the code. These programs can even look through large open-source libraries quickly and efficiently. As security deficiencies are discovered in this process the code is rewritten more securely. While an application will never be written completely secure using best coding practices severely reduce the chance of ay hack and as well prevent the depth of the hack if one does occur.

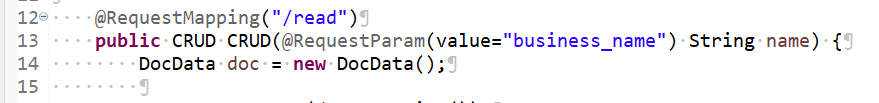
## 3. Manual Review

The first security vulnerability in the code is contained in the DocData.java file. This file is making a connection to the sql server on line 27. The issue is that it is testing with the root user profile which has unlimited access and as well the password is the same as the username and an easily guessed password.

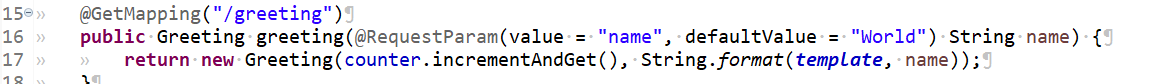
Text

Description automatically generated

The second security vulnerability is in the CRUDController.java on line 13. A parameter of business name is being passed into the DocData file which writes to sql. Hackers can use this to perform an SQL injection.



The third security vulnerability is in the GreetingController.java page on line 16. The page is grabbing a parameter and writing it directly onto the html page and this can lead to a html injection where the user can add their own code in this area to potentially cause a lot of damage.



A fourth potential security vulnerability (I do not know spring that well so maybe this login experience can be found somewhere else) is in the customer.java page on line 9. This page is showing account numbers and balance however there is no specifically set log in system to prevent unauthorized access to this sensitive information.

Text

Description automatically generated

## 4. Static Testing

The maven dependency check found 78 vulnerabilities. The first dependency was Bouncy Castle Dependency for cryptographic algorithms which had 17 vulnerabilities. The first five were CVE-201601000338 which can allow extra characters in a cryptographic signature now has third party patches to ensure the signatures are the proper length (Mitre 2020). CVE-2016-1000352 allowed the use of ECB mode, and ECB should be avoided (Mitre 2020). CVE-2016-1000343 is a vulnerability that a weak cryptographic key is made with the default settings, it is recommended to initialize JCA key with DSA parameters (Mitre 2020) . CVE-2016-1000344 is a vulnerability that allowed ECB which is unsafe to use thus the recommendation is to not use ECB mode (Mitre 2020). [CVE-2016-130](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-13098)41 which can cause a buffer in print function and is best fixed by using the most updated version (Mitre 2020).

The second dependency was the Hibernates Bean Validation containing one vulnerability of CVE-2020-10693 which passes unvalidated user input into the server and it is recommended to upgrade to version 6.0 or higher to prevent this (Red Hat 2022).

The third dependency is Jackson General Data Binding containing 2 vulnerabilities. The CVE-2020-25649 is a flaw allowing a hacker to infiltrate the data during an entity expansion and there does not seem to be a good remedy for this yet. CVE-2020-36518 is susceptible to denial-of-service attacks caused by overflow upgrading to version 2.13 or higher is recommended (Mitre 2022).

The fourth dependency is Apache Log4j contained one vulnerability CVE-2020-9488 that allowed a man in the middle attack by giving access to log messages which is fixed in versions 2.12 or higher (Apache Software Foundation 2022).

The fifth dependency was Logback Core with CVE-2021-42550 has a vulnerability that allows an entity to run malicious code which is mitigated using version 1.3 or later (SGCVP 2022).

The sixth dependency is YAML 1.1 which has one vulnerability CVE-2017-18640 which leaves the system vulnerable during load operations this is fixed by upgrading to a newer version without this weakness (Mitre 2022).

The seventh dependency is Spring Boot which contains one vulnerability of CVE-2022-27772 that allowed the directory to be hacked and this is removed when an updated version is used (Mitre 2022).

The eigth dependency is Spring Core containing nine vulnerabilities the first five are CVE-2020-22965 which allowed unauthorized access to the system and cause DOS attacks, this is resolved by using the latest version of the application (Hacker One 2021). CVE-2021-22118 allows users to gain larger access than granted and it is recommended updating past 5.3.7 (VMWare 2022) CVE-2020-5421allows an RFD attack to be possible through hacking which is relieved when update to the latest version (Pivotal Software Inc. 2022). CVE-2022-22950 which allows a hacker to cause a denial-of-service attack is fixed by updating to the latest version (VMware 2022).

The ninth dependency is Spring Web which has 10 vulnerabilities. CVE-2016-1000027 allows a hacker to input code to the remote server, which unfortunately cannot be fixed because de serialization is a wanted function(MITRE 2022). CVE-2022-22965 allows a hacker to use data binding to execute code on the sever and this is negated if deployed as an executable jar (VMware 2022). CVE-2021-22118 allows users greater access than originally granted and is recommended to update to the latest version (VMware 2022).

The tenth dependency is Core Tomcat which has 17 vulnerabilities. The vulnerabilities have mainly to do with incoming connection with Apache tomcat that allow hackers to affect the system. The first five vulnerabilities all recommend upgrading to the latest version of tomcat and they are, CVE-2020-1938, CVE-2020-11996, CVE-2020-13934, CVE-2020-13935, CVE-2020-17527 (Apache Software Foundation 2022).

The eleventh dependency is Tomcat Websocket containing 18 vulnerabilities. CVE-2020-1938, CVE-2020-8022 CVE-2020-11996 all recommend updating the software to prevent the vulnerability of hackers connecting with greater access than originally granted (Apache Software Foundation 2022).

## 5. Mitigation Plan

The access control vulnerabilities need to be addressed. The connection that is made in the DocData.Java file is dangerous. Making a connection using root with the root password also being root. Making a connection with root user is dangerous because it gives the user full access. The password being the same as username is a very poor choice that will be easily guessed. Even if they don’t guess this easy password a brute force attack will not take long to figure out a four-letter password. The user used to log in to the SQL should have as minimal control as needed and they passwords should be strong for both this user and the root user.

To fix the input validation vulnerabilities there should be no parameters that can be abused used. Building SQL applications without query parameterization leaves many vulnerabilities (Manico, & Detlefsen, 2015). The business name should not be passed as easily in the URL. As well all inputs should have limited number of input characters and common coding symbols blocked (blacklisting). However, the step before blacklisting is to “begin by defining only those characters and patterns that are known to be good” (Manico, & Detlefsen, 2015). For example, if it’s a phone number field there should be no letters, if name no numbers, and if email have the proper format.

This financial app is showing customer account numbers and balances however does not have a proper login function. This can result in people having access to these number who should not have it. A proper secure authentication functionality with access control can prevent this sort of attack. Also, if a virtual forgot password system is required it is recommended that 2 or more user specific information is requested, ask a security question, and use multifactor authentication before allowing a user to reset their password. (Manico, & Detlefsen, 2015)

Using the static tester, we discovered many vulnerabilities. The most important thing is that we keep all added plugins up to date and that we use plugins that are actively being updated regularly. New security vulnerabilities will come up, but the developers will typically prevent them in the next release following the discovery. The maven check should be run regularly as well to determine if any new vulnerabilities have come up that can be prevented as quick as possible through work arounds until developers send updates.

To safely transfer the account numbers to the users HTTPS and Secure Socket Layers or Transport Layer Security (Manico, & Detlefsen, 2015) should be used to ensure that hackers cannot intercept this data. On the client/Server side DDOS attacks need to be prevented by adding layers to prevent malicious logging in of usernames.

If a breach occurs or is in the process of occurring an efficient way to decrypt the events is by reviewing error logs. Java plugins that log errors or malicious looking activity need to be added but ensure that no revealing information is contained in the logs that a hackers could use if they gained access.

By determining the site load one can determine how much memory is needed and resources available to handle normal use plus a potential attack. The buffer needs to be as large as specified and enough allocated space is there to perform the actions.

**Works Cited**

Apache Software Foundation. (2022, May 10). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/cve-2020-9488>

Apache Software Foundation. (2022, July 12). NVD. Retrieved July 17, 2022, from

https://nvd.nist.gov/vuln/detail/CVE-2020-1938

Hacker One. (2021, November 23). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2021-22965>

Manico, J., & Detlefsen, A. (2015). *Iron-clad Java: Building secure web applications*. New York: Mc Graw Hill Education.

MITRE. (2020, October 20). NVD. Retrieved July 17, 2022, from https://nvd.nist.gov/vuln/detail/CVE-2016-1000338

MITRE. (2020, October 20). NVD. Retrieved July 17, 2022, from [https://nvd.nist.gov/vuln/detail/CVE-2016-1000352\](https://nvd.nist.gov/vuln/detail/CVE-2016-1000352\\)

MITRE. (2020, October 20). NVD. Retrieved July 17, 2022, from

https://nvd.nist.gov/vuln/detail/cve-2016-1000343

MITRE. (2020, October 20). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2016-1000344>

MITRE. (2020, October 20). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2017-13041>

MITRE. (2022, May 6). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2020-36518>

MITRE. (2022, April 18). NVD. Retrieved July 17, 2022, from

https://nvd.nist.gov/vuln/detail/CVE-2017-18640

MITRE. (2022, April 7). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2022-27772>

MITRE. (2022, June 10). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/cve-2016-1000027>

MITRE. (2022, May 19). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/cve-2022-22965>

RED HAT. (2022, May 10). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2020-10693>

RED HAT. (2022, May 10). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2020-25649>

Switzerland Government Common Vulnerability Program. (2022, March 29). NVD. Retrieved July 17, 2022, from <https://nvd.nist.gov/vuln/detail/CVE-2021-42550>

Pivotal Software Inc. (2022, June 23). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/cve-2020-5421>

Pivotal Software Inc. (2022, June 22). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2022-22950>

VMWare. (2022, May 12). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2021-22118>

VMware. (2022, May 12). NVD. Retrieved July 17, 2022, from

<https://nvd.nist.gov/vuln/detail/CVE-2021-22118>