

**CS 305 Project One**

**Artemis Financial Vulnerability Assessment Report**

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **5/21/2020** | Stephen Cardone | Begin interpreting client needs |
| **1.1** | **5/23/2020** | Stephen Cardone | Analyze client needs and areas of security |
| **1.2** | **5/24/2020** | Stephen Cardone | Perform manual code review and static testing, and record findings. |

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

Stephen Cardone

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?

**[Include your findings here.]**

Artemis Financial is a company that works in the personal finance space. Their clients are individuals, and these individuals are trusting Artemis with significant amounts of personal and financial information. Artemis' clients will expect absolute security of their information. Any breach of security will be devistating to both Artemis' clients, and Artemis themsleves, as a breach of security could cause many clients to leave due to lack of trust.

Artemis may have clients with international money and will require the ability to transact and exchange foreign currency securely. There may be additional steps required to ensure security of information and transfer of information at an international scope.

The Government has many restrictions on data and communication over the internet. In order to ensure the security of Artemis' current system, I will need to familiarize myself with governmental restrictions. A good resource to begin looking in to these issues is <https://www.fcc.gov/general/broadband-and-internet-guides> the fcc website. Restrictions may also expand further when we consider international communication.

There are external threats to Artemis extend to both data stored on their systems, as well as any communication or data being transfered over the internet. For data stored on Artemis' systems, we need to be concerned about injection attacks where a bad actor could inject SQL or other code into our user input boxes to extract data. We will also need to ensure that we properly restrict data behind credentials and have protocols to prevent credentials from being inapropriatly accessed.

As far as modernizing Artemis' systems, we should make sure that any third party libraries used in Artemis' systems are updated. As third party libraries find security issues in their systems, they patch them. We should update to newest versions of libraries in order to get as many security patches as possible. We can also read about the specific versions to be aware of and fix whatever known issues we can.

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.

**[Include your findings here.]**

**Input Validation:** As I review Artemis' codebase for security, Input validation is one of the most important items to look into. Input validation is one of the more simple issues to identify, and also resolve. I will make sure to look for parameterized SQL.

**APIs:** Since Artemis' codebase contains s RESTful api, we will need to check to insure that the api's are properly restricted. I will do this in two ways. Firstly, I basically validate api input similar to input validation. for API's I will make sure that passing parameters or options with malicious intent does not result in unintended data being sent. Also, I will ensure that access to the APIs are properly restricted by user levels. If a user should not have permission to access some data, they should not be able to access it from the APIs.

**Client/Server:** Since Artemis supplies APIs, I will need to ensure that Artemis is properly protecting data as it is being communicated from client to server. Also I will need to ensure that permissions are properly enforced between client and server.

**Code Error:** I will check to make sure that restrcited data is not logged to the user in any error cases. A bad actor could intentionally trigger errors and if the error logs contain sensitive data, this could be a security breach.

3. Manual Review

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

**[Include your findings here.]**

**Update to latest Spring-boot-starter-parent**: The parent version in pom.xml:8 should be updated from 2.2.4 to 2.3.0. This ensures that we have the most recent depedency checkingin infastructure.

**Dependency Versions**: Most dependency versions in the application are not up to date. It is likely that they were up to date at the time of writing the application. It may be worth updating to latest dependency versions. This may improve the static testing results below.

**CRUDController.java:13**: On this line, you accept a parameter of "business\_name" in as a string. There is no input validation on this input. We do not trust the source of this parameter, so you need to validate input in some way.

**GreetingController.java:16**: On this line, you accept parameter input from the user as a string. There is no input validation on this string, so a user could potentially pass anything, including an attempted injection attack.

**DocData.java:26-27**: On these lines, you create a database connection. In this instance there are 2 issues. Firstly, you hard code a username and password. This is not reccomended. Also, there are no privilege checks. Any user will have access to this database connection regardless of who they are. You should ensure the user has the correct permissions to access the data before creating a DB connection.

**Customer.java:4-5**: These class level variables are a customers account data and account balance. These are both extremly sensitive pieces of information and need to be protected. At the very least, they will need to be private variables to prevent unintended access. Also, consider abstracting the data and storing the values outside of the program rather than storing them in the cstomer class.

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:

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

**[Include your findings here.]**

My testing has identified 1 vulnerable dependency with 15 vulnerabilities.

bcprov-jdk15on-1.46.jar (Bouncy Castle Crypto package)

* CVE-2013-1624: Java and C# libraries do not properly consider timing side-channel attacks on noncompliant MAC check operations.
* CVE-2015-6644: Info disclosure vulnerability could enable local malicious applications to gain access to user private info.
* CVE-2015-7940: Java library does not validate a point is within the elliptic curve.
* CVE-2016-1000338: DSA does not fully validate ASN.1 encoding of signiture on verification.
* CVE-2016-1000339: CPU can be monitored sufficently to leak information on the AES key being used.
* CVE-2016-1000341: DSA signature generation is vulnerable to timing attack
* CVE-2016-1000342: ECDSA does not fully validate ASN.1 encoding of signature on verification.
* CVE-2016-1000343: DSA key pair generator generates a weak private key when default values are used.
* CVE-2016-1000344: ECB mode is allowed which is unsafe.
* CVE-2016-1000345: Vulnerable to padding oracle attack
* CVE-2016-1000346: invalid keys can be used to reveal details about other keys
* CVE-2016-1000352: ECB mode is allowed which is unsafe.
* CVE-2017-13098: Use of JCE can allow an attacker to recover private keys from the application
* CVE-2018-1000613: private key deserialization that can result in Deserializing an XMSS/XMSS^MT private key can result in the execution of unexpected code.
* CVE-2018-5382 : keystore files use HMAC that is only 16 bits. Not long enough to be secure.

Resolution: The first step to resolving these issues is to update the version of Bouncy Castle being used. The majority of these issues are caused by the out of date version. Static testing would need to be re-assessed once the version is updated.

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.

**[Include your findings here.]**

Reccomendataions to Artemis:

Firstly, you should update all library dependencies to latest versions. Specifically target Bouncy Castel as this is the source of most of your security vulnerablities at this time. Once libraries are all up to date, focus on input validation for your API's. Make sure input validation is applied to all the areas I identified earlier in this document, as well as any future apis. Also, make sure that all account info (such as account balances and account numbers) are properly abstracted and restricted by permission checks. Finally, ensure that all database connections require proper authentication and are only available to those with the proper permissions, and ensure that any database username/passwords are abstracted and not coded in as plain text.