****

# 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** | **July 15, 2022** | **Valerie Smith** | **Create initial security plan and review** |

## Client



## Developer

Valerie Smith

## 1. Interpreting Client Needs

There are many values in secure communications to a company. The top value would be to protect the data of the customers to ensure that the customer’s information is not exploited by a hacker. Another value of secure communications to a company would be to protect the company’s own assets, such as the data the company is responsible for, as well as the companies web domain and employee information.

Having secure communications is of value to a company to protect the things that most hackers are looking for, such as money, data theft, espionage, user harassment and identity theft, personal vendetta’s such as an employee grudge, and hackers that hack for infamous reasons, such as the group Anonymous. A data breach or a hacking event can cost the company financially as well as the trust of their customers is broken. (Manico & Detlefson, 2015)

For the information given, it not obvious that the company is doing any international transactions. However, if the company will be doing business internationally with the rest service that is provided, there are serious concerns with the implementation, such as a lack of role-based access control, which is a standard according to INCITS (Manico & Detlefsen, 2015).

According to Manico & Detlefsen (2015), access control mechanisms are required for enterprise and government applications. Various other compliance standards are PCI-DSS for credit card management, HIPAA for health care data, Gramm-Leach-Bliley standards for financial information, and other international privacy laws that are particularly required in Europe (Manico & Detlfsen, 2015).

Current and future security threats exist and will continue to exist anytime an application works with untrusted data, which is data that is not static to the application (Manico & Detlfson, 2015). Untrusted data can come into the application from forms that accept user input, file uploads, databases, internal applications, and other various outside sources. (Manico & Detlfson, 2015).

Intercepting web proxies, the lack of input validation, SQL Injection, XML and JSON based injection, command injection, DOS and DDoS attacks, Man in the Middle attacks, phishing, ransomware, and DNS spoofing are all current cyber security attack examples (*Top 20 most common types of cyber attacks*).

Modernization efforts should have critical application priorities addressed, such as scalability, capacity, availability, reliability, maintainability, service and manageability, environmental and interoperability, and security and regulatory concerns. The customer of the application should be kept top of mind to ensure that the customer gets an application that will suit their needs and enable them to get their work done, rather than something that is too much for them to manage, such as overengineering (Harding, 2022).

Functional business requirements should be used to support all requirements and security standards and elements should be updated. Third party software, such as integrations with open-source software, infrastructure and hardware should be examined for risk and evaluated to ensure that changes to them won’t introduce unnecessary risk (Harding, 2022).

## 2. Areas of Security

1. Input Validation – The Artemis Financial application contains objects that will be entered by the user of the application, such as company name, username, password, and the ability to read the documents that are returned from the database.
2. API’s – Secure API Interactions will be necessary for this application as it will be used to upload documents to a database.
3. Cryptography – Encryption may need to be examined in depth for this application as it is accepting passwords and contacting a database via a jdbc string. Cryptography may also be a part of the financial requirements that included in regulatory standards such as Gramm-Leach-Bliley.
4. Client Server – The Artemis application is a financial application that will accept user input as well as return the customer’s information, so it is assumedly going to work with a client UI of some unlisted type or be integrated with another application that will be utilizing the rest service. Client server security protocols should be followed, such as incorporating access control and role-based access control. As the code presented is a spring boot application, the application architecture should be created so it is not susceptible to Spring4Shell security vulnerabilities, which affects versions of applications that use Spring MVC. Clients that are calling the API’s generated by a Spring Boot application should be authorized and authenticated to use the API’s. Content Security Policies and using HTTPS can guard against Cross Site Scripting attacks. Parameterized Queries should be used in Spring Boot to recognize unsafe values that are being passed in. (Spring Boot - securing web applications)
5. Code Error:

Custom error handling and the appropriate error exception handling should be provided in order to eliminate the default Whitelabel error pages. Eliminating the Whitelabel pages is necessary to provide effective feedback to legitimate users of the application insights. (*Spring Boot Error Handling Guide*)

1. Code Quality:

Developer experience with Secure Coding techniques is a must when creating a rest API. Writing code that is modular, reusable, maintainable, immutable, and secure will help to guard against the accidental introduction of security vulnerabilities (*Code quality: Skills directory*).

1. Encapsulation:

Encapsulation threats exist due to an application failing to separate critical data and functionality between components, which can create leaks from an application. These problems can lead to cross domain attacks that allow unauthorized users to gain access to data. Error codes and left-over debugging statements can give clues to internal details about the application. (*Encapsulation vulnerabilities*)

## 3. Manual Review

There are no views created for this application, the template folder is empty. In the Greeting Controller, it seems a default greeting mapping exists with a string template. This default greeting needs to be removed as it exposes application inner workings and does not have authentication and authorization security controls around the endpoint.

The customer.java class model exposes the customer’s account balance as a public integer. There are no accessors for any of the fields in the class and no input validation on the variables, leaving the application exposed to any value being passed into the application. The showInfo method will expose any customer account number, providing access to the ‘this’ variable. The deposit() method accepts an identified integer ‘a’, which would allow anyone to add a random amount to the customer’s account balance. This function should be refactored so that it is not capable of passing untrusted data.

In the CRUD.java class, there is no input validation around the values that will be used to read the data coming back from the database. The public constructors are allowing access to the ‘this’ keyword, which allows access to the variables, as well as allowing any String value to be passed to the variables, exposing the application to risk. The first public constructor is taking a String content parameter and setting the value of it to two different variables, this constructor should be examined for relevancy,

refactored, or removed.

The CRUDController.java class has no security built around the endpoint, which exposes the application to numerous security vulnerabilities. There is no authorization/authentication in the application that determines the identity of who is allowed to use this controller. There is no input validation for the value business\_name, which can be any String value, exposing the endpoint. The controller creates a new CRUD document instance, yet it is unclear what this instance will do as it appears the method is not finished. This exposes the internal workings of the application to anyone who can use the endpoint.

The DocData.java implementation needs to be refactored in entirety. The class exposes a private id as a String, which could be any string that is passed into the application. The public constructor is exposing the entire class without any parameters or input validation. The read\_document method will accept any two strings with no validation and call the jdbc url directly via the browser. This url should be a constant that is not a part of the class as the entire class is exposed. The exception handling is not complete and will also expose inner workings of the application’s details if allowed to work in this fashion. The application should implement a secure call to the database along a more trusted url, such as with the use of https.

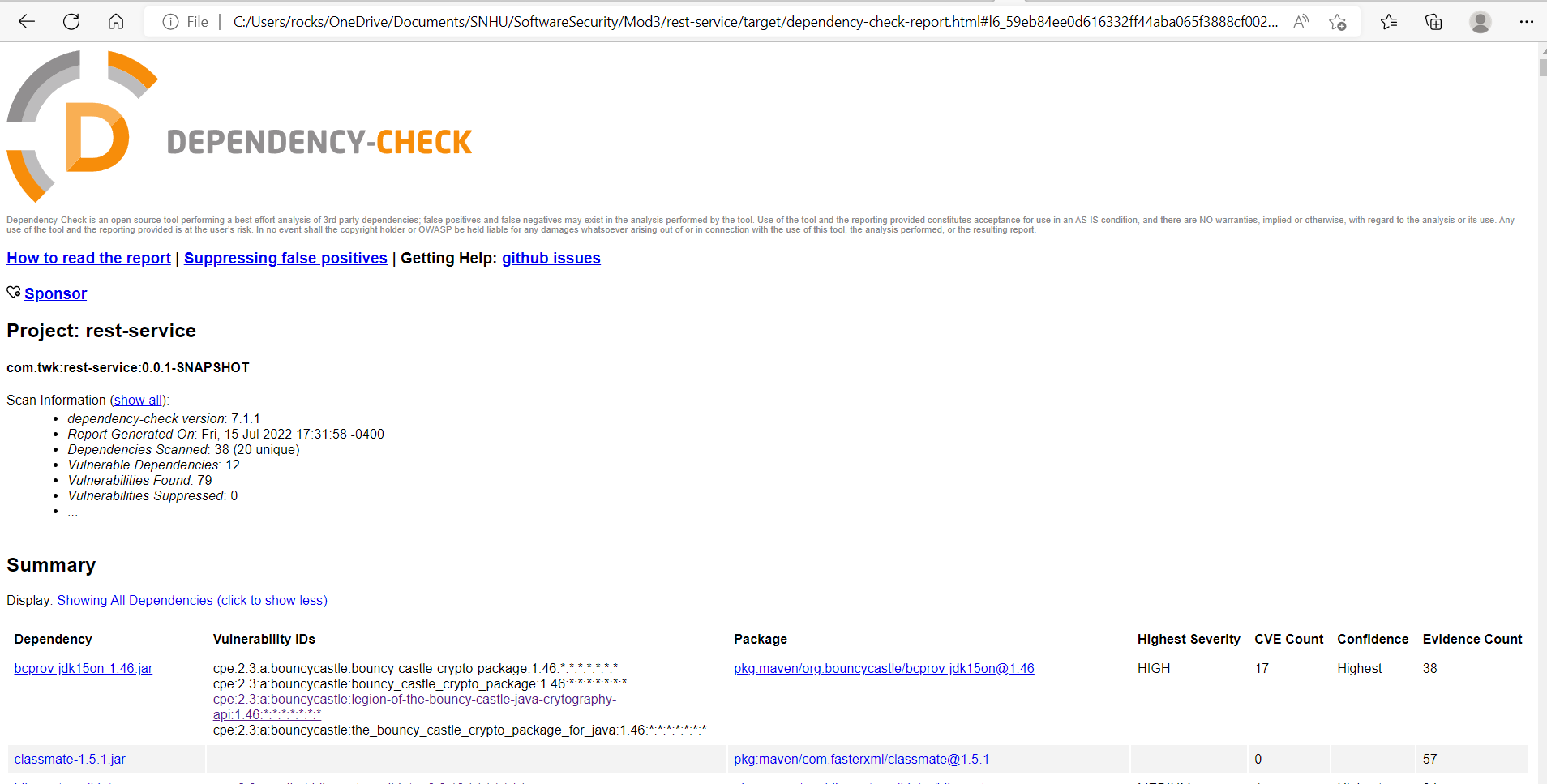
The myDateTime.java class exposes three integer variables to untrusted data and allows them to be used to create an array of three objects that can be any integer of any size or length. This class should be refactored properly and be created with a Date class that is native to the application to avoid the impending security vulnerabilities that will arise from it’s use. Proper input validation should be implemented in this class as well as the rest of the application. There is no exception or error handling in this class, which will then result in a Whitelabel error, exposing the internal workings of the application.

The application.properties file exposes the location path of the data on the server, which exposes the server to direct access by unauthorized actors. The server port is also listed on this file. These properties should be abstracted to a vault type of implementation, or at a minimum, encrypted and decrypted to avoid the values being shown at application run time, or exposed in a public repository that is subject to viewing by unauthorized parties. The properties can also be moved to a WebSecurityConfig via Spring Security.

The application is using an outdated version of Spring-boot-starter parent as well as Spring-boot-starter web, which are pulling in outdated tomcat versions as well as introducing other security vulnerabilities into the application. The bouncy castle jar has been pulled in on the pom.xml and is also outdated and subject to security vulnerabilities. The bouncy castle jar is not utilized in the application for its intended purpose and should be updated or removed from the application.

Unit testing has not been completed for this application. Proper unit testing will assist the developers in creating a more secure application and to ensure that the application works according to user specifications. Unit testing will guard against future changes to the application that may introduce errors and potential security vulnerabilities.

## 4. Static Testing



1. bcprov-jdk15on-1.46.jar –

Bouncy Castle is a java plug in that is used in java cryptography API’s. There are 12 connected CVE vulnerabilities, and it is listed as a Highest Severity on the vulnerability report. The Vuln IDs from the report are CVE-2016-100352, CVE- 2016-100346, CVE- 2016-1000345, CVE-2016-100344, CVE-2016-1000343, CVE-2016-1000342, CVE-2016-1000341, CVE-2016-1000339, CVE-2016-1000338, CVE-2018-5382, CVE-2017-13098, and CVE-2013-1624.

Some attributions complain of plaintext attacks when using noncompliant MACs, weak Bleichenbacher oracle when using TLS with RSA key change, attackers being able to infiltrate the default BKS keystore, unvalidated ASN.1 encoding, look up table leaks when using AES keys, DSA signatures being vulnerable to timing attacks, DSA key pair generators generate weak private keys when used with default values, ECB mode regarded as unsafe and is removed from support, decryption failing due to padding, and DH public keys not fully validated.

1. Hibernate-validator-6.0.18.Final.jar –

This dependency has the Vuln ID of code CVE-2020-10693, which is a flaw even if it is Hibernate Validator. This enables invalid EL expressions to be evaluated and marked as valid. This would affect the data in error messages not. This is considered a Medium severity issue.

1. Jackson-databind-2.10.2.jar –

This dependency introduces two Vuln ID codes of CVE-2020-36518 and CVE-2020-25649. Flaws in the FasterXML Jackson Databind exposes XML to XXE attacks with highest threats to data integrity. The High Severity CVSS is on versions 3.1, Medium on versions 2.0.

1. Jakarta-annotation-api-1.3.5.jar –

This dependency has a published vulnerability code of CVE-2022-31569, which is explained as an improper limitation of a pathname to a restricted directory, otherwise known as a Path Traversal.

1. Log4j-api-2.12.1.jar –

This dependency has a High Severity rating and introduces many Vuln ID codes: CVE-2021-44832, CVE-2021-45105, CVE-2021-45046, CVE-2021-44228, and CVE-2020-9488. The Apache log4j versions from 2.0—beta7 through 2.17.0 are vulnerable to remote code execution, or RCE. Other vulnerabilities are Thread Context Map Control that causes DOS attacks, malicious JNDI lookup patterns, the ability to execute arbitrary code loaded from LDAP servers due to unprotected LDAP endpoints, and SMTPS connections intercepted by man in the middle attacks.

1. Logback-core-1.2.3.jar –

In the logback version 1.2.7 and prior, attackers that have required edit privileges can create malicious configurations that will execute code from LDAP servers. This is listed as a Medium severity with a CVE count of 1.

1. Snake-yaml-1.25.jar –

This dependency has a Vuln ID of CVE-2017-18640. This concerns alias features in SnakeYML formatting which introduce issues from CVE-2003-1564.

1. Spring-boot-2.2.4.RELEASE.jar –

This dependency is listed as Highest criticality and has a CVE count of 1. It exposes Vuln ID of CVE-2022-27772, which affects spring boot versions prior to 2.2.11.RELEASE. This exposes the application to temporary directory hijacking within the AbstractConfigurableWebServerFactory.

1. Spring-core-5.2.3.RELEASE.jar –

This dependency is listed as a Critical vulnerability and has a CVE count of 9. The VULN IDs introduced by this dependency, however, list 10 CVE numbers:CVE-2022-22971, CVE-2022-22970, CVE-2022-22968, CVE- 2022-22965, CVE-2022-22950, CVE-2021-22060, CVE-2021-22096, CVE-2021-22118, CVE-2020-5421, and CVE-2016-1000027. This dependency has been shown to introduce vulnerabilities from DOS attacks pertaining to data binding scenarios as well as remote code execution RCE attacks via data binding. SpEL expressions can also be used in the older unsupported versions to cause DOS attacks. Log entries on unsupported versions may be vulnerable to attacks by allowing malicious input to be inserted into logging entries. In certain WebFlux applications, vulnerabilities are present that are introduced by authenticated users.

1. Spring-web-5.2.3.RELEASE.jar –

This dependency is listed as a Critical vulnerability and has a CVE count of 10. The VULN ID’s are the same ID’s that are listed for the spring-core-5.2.3.RELEASE.jar.

1. Tomcat-embed-core-9.0.30.jar –

This dependency is listed as a Critical vulnerability and has a CVE count of 18 that are known to be causes of security issues: CVE-2022-34305, CVE-2022-29885, CVE-2021-33037, CVE-2021-41079, CVE-2021-25329, CVE-2021-25122, CVE-2021-30640, CVE-2021-24122, CVE-2020-17527, CVE-2020-13943, CVE-2020-13935, CVE-2020-13934, CVE-2020-8022, CVE-2020-11996, CVE-2020-9484, CVE-2020-1938, CVE-2020-1935, and CVE-2019-17569. These issues surround the ability of various versions of Apache Tomcat to protect user data and expose various problems while parsing HTTP request headers. Vulnerabilities also exist in the JNDI Realm of Apache Tomcat while authenticating valid names as well as duplicate request headers. Some of the vulnerabilities are pointing to previous vulnerabilities that have not been addressed by Apache for the affected versions. The main vulnerabilities point to DOS attacks based on the use of the Apache Tomcat versions listed.

1. Tomcat-embed-websocket-9.0.30.jar

This vulnerability is listed as a Critical vulnerability and has a CVE count of 18, corresponding to the same CVE VULN Id’s for the tomcat-embed-core-9.0.30.jar dependency that is also listed here.

## 5. Mitigation Plan

1. Secure coding techniques in the Java language that cover encapsulation and Spring security techniques are to be reviewed by the developers and implemented in order to refactor the class models, the class controllers, and the application properties files. This will ensure that the issues with unprotected endpoints and input validation are corrected in the application as well as the information contained in the application.properties file is secure against outside attacks as well as protecting the application from untrusted data. Secure coding techniques will also assist the developers in the proper implementation to secure the JDBC http call. Secure coding techniques will assist the developers in the implementation of the unit tests that are necessary for this application.
2. The application will be updated to the most recent stable versions for the Spring-boot-starter parent and the Spring-boot-starter web in order to remove the existing security vulnerabilities that these framework jars introduce and to update the tomcat jars to a stable version. This will also remove the security vulnerabilities found with the snake yml, jarkarta, logback, Jackson, and log4j jars, however, these jars should be checked for the latest stable versions to overcome the security vulnerabilities. Developers will utilize a security vulnerability scan to routinely check that the existing security vulnerabilities have been removed and/or have not been reintroduced to the application.
3. Accessed based control and authentication will be incorporated into the application to ensure secure client server communications and to limit the use of the application to the correct validated users.
4. Security techniques such as Spring Security will be used in order to secure the endpoints of the application.
5. The default greeting endpoint will be removed from the application.
6. The date class model will be updated and refactored to include an imported Java Date library in order to remove the threat of the passing of untrusted data to the application.
7. Proper error and exception handling will be introduced in the application to ensure that Whitelabel pages are not shown, eliminating the exposure of the internal workings of the application. Valid logging techniques will be added to ensure that exceptions and errors are logged and can be reviewed.
8. The bouncy castle jar will be examined for proper usage and if found necessary to keep for encryption efforts, updated to the most stable version that does not introduce security vulnerabilities from an outside source and that it is properly working with cryptography efforts such as encryption and digital signatures.

References:

*Code quality: Skills directory*. HackerRank. (n.d.). Retrieved July 9, 2022, from https://www.hackerrank.com/skills-directory/code\_quality

*Encapsulation vulnerabilities*. Veracode. (n.d.). Retrieved July 16, 2022, from https://www.veracode.com/security/encapsulation-vulnerabilities

Harding, R. (2022, January 3). *5 application modernization considerations*. EQengineered. Retrieved July 16, 2022, from https://www.eqengineered.com/insights/5-application-modernization-considerations

Manico, J., & Detlefsen, A. (2015). *Iron-clad java building secure web applications*. McGraw-Hill Education.

Spring Boot Error Handling Guide. (n.d.). Retrieved July 16, 2022, from https://www.codejava.net/frameworks/spring-boot/spring-boot-error-handling-guide

*Spring Boot - securing web applications*. Tutorials Point. (n.d.). Retrieved July 9, 2022, from https://www.tutorialspoint.com/spring\_boot/spring\_boot\_securing\_web\_applications.htm

*Top 20 most common types of cyber attacks*. Fortinet. (n.d.). Retrieved July 16, 2022, from

<https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks>