****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Artemis Financial Vulnerability Assessment Report 1](#_Toc134963995)

[Document Revision History 3](#_Toc134963996)

[Client 3](#_Toc134963997)

[Instructions 3](#_Toc134963998)

[Developer 4](#_Toc134963999)

[1. Interpreting Client Needs 4](#_Toc134964000)

[2. Areas of Security 4](#_Toc134964001)

[3. Manual Review 6](#_Toc134964002)

[4. Static Testing 13](#_Toc134964003)

[5. Mitigation Plan 26](#_Toc134964004)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/18/2023** | **Christopher Trimmer** | **Document creation and initial submission** |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In the report, identify your findings of security vulnerabilities and provide recommendations for the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Christopher Trimmer

## Interpreting Client Needs

Artemis Financial has hired Global Rain to help modernize and secure one of their RESTful web API’s. Our company Global Rain specializes in custom software design and development. Our mission statement is, “Security is everyone’s responsibility”, and we support entrepreneur’s, businesses, and government agencies around the world. Artemis Financial has recognized the need for secure applications and is therefore motivated to modernize their applications and protect their organization from external threats.

This report is the initial software security assessment of Artemis Financials’ web application. The assessment will consist of interpreting Artemis Financials’ motivation for security, the areas of security we will focus on, a manual code inspection, a static code analysis, and a mitigation plan.

* *Value of secure communications*

Artemis Financial is a consulting company that develops individualized financial plans for their customers. Their focus includes savings, retirement, investments, and insurance plans. Secure communications are one of the most important areas that Artemis Financial values for several reasons. First, trust is a major consideration when customers seek financial help. If financial information was ever leaked or compromised, it would break the trust relationship and severely damage the reputation and credibility of the company. It would be extremely difficult for Artemis Financial to recover from a breach of trust, and they would most likely lose existing customers and potential future customers.

Second, financial information is widely considered to be one of the top security drivers to maintain compliance with industry standards and laws (Manico & Detlefsen, 2015). The Gramm-Leach-Bliley Act (GLBA) applies directly to Artemis Financial. The following are penalties associated with being found non-compliant with the GLBA (Wilson Consulting Group, n.d.):

* + A fine of up to $100,000 for each violation
  + A fine of up to $10,000 for officers and directors of the financial institution
  + Imprisonment for up to five years
  + Revocation of licenses
* *International transactions*

Our company supports companies around the world, which means have intellectual capital that can be leveraged by companies who wish to conduct business internationally. Considering that many businesses now operate globally, Artemis Financial is motivated to help individuals and companies who are operating within U.S. boundaries, as well as around the world. Therefore, it is important to consider that the web application has a security posture that ensures they are compliant according to international laws, as well as the GLBA. The Federal Trade Commission (FTC) provides specific guidance and policies for conducting business internationally (FTC, 2013).

* *Government restrictions*

As mentioned above, the GLBA, and applicable FTC and international laws will be a major driver for the way that information is handled by Artemis Financial. The web application must be developed in a way that ensures compliance with standards, codes, and laws that apply to any of their customers. The following are other specific government laws and regulations that will be applicable:

* + Consumer Privacy Protection Act
  + Homeland Security Act
  + Federal Information Security Management Act (FISMA)
  + Cybersecurity Information Sharing Act (CISA)
  + Multiple states in the U.S. also have laws that must be adhered to
  + International laws
* *External threats now and in future*

According to OWASP (2021), the “Top 10” threat categories are as follows: Broken Access Control, Cryptographic Failures, Injection, Insecure Design, Security Misconfiguration, Vulnerable and Outdate Components, Identification and Authentication Failures, Software and Data Integrity Failures, Security Logging and Monitoring Failures, and Server-Side Request Forgery. These are broad categories that group together related vulnerabilities. The fact is that there are so many attack vectors and vulnerabilities now, that every aspect of an application should be analyzed from a security perspective.

It is difficult to predict what the future security threats will be because software development is always evolving, companies are increasingly dependent on software, and technology is continuously evolving. A plug-in that has has no known vulnerabilities today, may have two vulnerabilities next week for example. However, we can at least use resources available today to help us prepare for the future.

* + Follow open-source resources such as OWASP, the NVD database by NIST, the CVE database by Mitre, and many other trusted security resources.
  + Incorporate security in the SDLC to form an SSDLC.
  + Incorporate Continuous Security (CS) into the CI/CD pipeline.
  + Transition from a DevOps to a DevSecOps process model.
  + Identify current trends:
    - More business is being done remotely (work from home, e-commerce, etc.).
    - Increased use of cloud computing (SaaS, PaaS, IaaS, etc.).
    - Increased use of artificial intelligence (AI) and machine learning (ML).
    - Increase use of API’s.
* *Modernization requirements*
  + Artemis Financial is developing a RESTful web-based API. Representational State Transfer (REST) is an architectural style that is based on a client-server relationship in which a server will respond to client requests for some sort of resource or information. Application Programming Interfaces (APIs) are ways that programs communicate and interact with each other. While using a RESTful web-based API is not the only way to communicate over the web, it has become the most recognized way that applications are used and developed today in support of e-Commerce and improves the ways businesses and individuals work.
* Artemis Financial is coding their application using Java and the Spring Framework. Java is a powerful object-oriented programming language that is used by millions of people throughout the world. Some of the main benefits of Java are that it is platform agnostic, and it has a lot of support for distributed, web-based applications. The Spring Framework is a framework that enables Java programmers to easily integrate capabilities from an eco-system of API’s that are designed to work together. It enables developers to focus on the development aspect of programming, rather than on the complexity of getting various API’s and technologies to work together. Spring Boot is an extension of the Spring Framework that contains the most core capabilities of an API, and helps people quickly get an application developed.

Companies like Artemis Financial are motivated to use a framework like Spring for several reasons: 1) it takes the complexity out of integrating multiple API’s from varying sources; 2) it drastically speeds up the development process and therefore the application can be pushed to market quickly; and, 3) the framework is continuously tested, and is used by many other customers, therefore vulnerabilities can be found and fixed more quickly.

* According to Contrast Security (2014), during the period of 2012 – 2014, “the use of open-source software has more than doubled from 6 billion to 13 billion component downloads per year”. Companies use open-source software for a variety of reasons: cost savings, improved time to market an application, added security and stability, added capabilities with less complexity, and the flexible and adaptable nature of open-source software making it highly preferable.

However, open-source software also comes with disadvantages. It can require expertise to integrate into an existing application, an application can end up dependent on the open-source product and therefore become limited if the open-source product is no longer supported becomes inferior, and the open-source product can add security vulnerabilities. According to OWASP (2021b), “Vulnerable and Outdated Components” ranked as the 6th most critical security risk to web applications, which is showing an increasing trend compared to being ranked 9th in 2017. As the production and use of open-source software increases, so too do the security risks and vulnerabilities.

* + Web application development will be a continually evolving practice, just as technology and business trends continue to evolve. Companies like Artemis Financial want to do what they do best – in this case, provide financial planning to customers. They are not interested in re-inventing the wheel in software development or coming up with new software technology. Rather, they want to use a software development process that is current and helps them get their product to market quickly and securely. Hence, frameworks like Spring are great for companies like Artemis Financial.

Artemis Financial also expects to be positioned for the future, ensuring that their web application we be adaptable to future web application trends and technologies. Probably the most profound and highly leveraged technology right now and in the foreseeable future is cloud computing. Cloud computing enables a company to scale their resources and deliver products quickly, easily, and economically. Artificial Intelligence and Machine Learning are dramatically changing how companies can obtain and use data to help their businesses. Jeyaraman (2023) has identified these and other innovations will enhance the future of the web, including: progressive web apps (PWAs), web VR/AR, and others. Using a framework like Spring so that new and existing API’s can be leveraged to keep up with the future is probably the best thing a company can do right now.

## Areas of Security

To organize our analysis, we will be using the Vulnerability Assessment Process Flow Document (VAPFD) as our guide for performing the overall security assessment. The VAPFD is a guide for conducting an architectural review of the application and for reviewing the code for possible security vulnerabilities. It consists of two layers, with seven areas of concern in each layer. Each area of concern is basically a way to categorize common security concerns into manageable groups to help make the review process standardized and consistent. Using this VAPFD in conjunction with Artemis Financials’ development lifecycle is a part of an overall strategy for implementing their Secure Software Development Lifecycle (SSDLC).

The first layer of the VAPFD consists of the following areas: input validation, API’s, Cryptography, client/server, code error, code quality, and encapsulation. After reviewing the current state of the web application, my approach to performing a security assessment will focus on all seven areas.

* *Cryptography* – As discussed in the introduction section, cryptography is an important part of the security posture for this application. We need to ensure that any web-based communication is performed using HTTPS, and that interactions such as handshakes, session establishment and control, and information exchange are performed using secure mechanisms such as SSL/TLS. Furthermore, since Artemis Financial is in the financial industry, it is critical to ensure that data is encrypted while in transit and at rest.
* *Client/Server* – As identified in the introduction, Artemis Financial will be performing communication over the web, reaching national and international clients. We naturally assume that there will be interaction with backend servers running Tomcat or some other server infrastructure. Furthermore, during the code review we note that we will be interfacing with a MySQL database server. Therefore, we will need to focus attention on securing connection strings, data, and transactions with external databases.
* *Code Error* – code errors are a security focus area because we must be vigilant to ensure things like error messages and error propagation are not the cause of vulnerabilities. For example, error messages could inadvertently expose information about the application that hackers could use to exploit a vulnerability. Furthermore, error propagation can be used to make the application vulnerable to DDoS attacks. Keeping this type of security in mind is always part of our development strategy.
* *Input validation* – Some of the most common types of attacks and vulnerabilities are caused by user input. This includes buffer overflow, cross-site scripting, various forms of injection attacks. It is also possible that meaningful attempts at performing input validation by the developer is the cause of a vulnerability. For example, a complicated regular expression could cause a ReDoS vulnerability (Manico & Detlefsen, 2015). Therefore, input validation will be addressed as a major area of concern in the assessment. Note that this could be addressed as part of the code quality category, however I feel it is crucial to address it separately to highlight how important it is.
* *API’s* – we are using the Spring Framework which is essentially a collection of APIs. Therefore, we must ensure that any vulnerabilities of the API’s we use are addressed. According to Contrast Security (2014), “eighty percent of the code in today’s applications come from libraries and frameworks, but the risk of vulnerabilities in these components is widely ignored and {underappreciated}”. During a yearlong study performed by Sonatype and Aspect Security, they documented that Spring MVC was the most downloaded library (more than 10-million downloads) (Contrast Security, 2014). During the same timeframe, the Spring framework was downloaded over 18 million times (Contrast Security, 2014). There have also been known vulnerabilities with various Spring API’s and components (Spring Expression Language Injection, for example). Since Artemis Financials’ web application will be based on the Spring framework, as well as other API’s, it is crucial that we focus attention on making sure the APIs are as secure as possible.
* *Code Quality* – code quality is a very broad category. It basically addresses any type of coding concerns. This includes ensuring we are using security best practices, and security-based design patterns such as Execute Around Method pattern for java (Oracle, 2019). Furthermore, we should note that our coding practices should ensure the application maintains compliance with the Gramm-Leach-Bliley Act (GLBA), federal and state laws, and international laws.
* *Encapsulation* – our application will have various classes used to represent objects. Therefore, we must ensure that information exchanged among interacting objects, and between internal and external systems is secure, and that our classes are highly cohesive. The classes in our application right now include the Application, Greeting, and Greeting Controller. As the project grows, we will be adding more classes, so ensuring our classes are built with encapsulation in mind will prepare us for future modifications to our code base.

## Manual Review

The following section will detail specific security areas of concern based on the current code base. This process involves the second layer of the VAPFD and is comprised of performing a code review of the following seven areas: views, models, controllers, data access, services, plug-ins, and APIs.

* Views – based on the current code, we are not given any of the view pages to review. In future versions of the application, we will need focus attention in this area. View pages can be implemented using .html, .jsp or other types, so our assessment in this area will be dependent on how they intend to implement the views.
* Models – there are several model classes in the application: CRUD.java, customer.java, DocData.java, Greeting.java, and myDateTime.java. I will discuss various concerns applicable to these classes.
  + Greeting.java

A screen shot of a computer program

Description automatically generated with medium confidence

* + - The two class attributes, id and content, have private access modifiers which means they are not directly exposed to external objects in the code. To access the data contained in these variables, objects will use the public getter methods. There are no setter methods, which means the data is “read only” to external objects. The only way to set the variables is via the class constructor which means that the data will be set during object creation. Finally, the attributes are also marked as final, which prevents them from being changed after an object is created. Therefore, this satisfies our concern for encapsulation of objects created from this class.
* However, note that setting the data directly in the constructor as it is currently coded means that there is no error checking, no data validation, and most importantly there is no security control. For the content attribute, any string value can be input to that field which can lead to open text input exploitation. Similarly, the id attribute is a long datatype. Since there is no input validation on this attribute, the code is vulnerable to overflow exploitation and possibly input exploitation.
  + CRUD.java

A screen shot of a computer program

Description automatically generated with medium confidence

* The two member attributes are marked as private and final, which is fine. However, it is not clear the intention of these variables. Based on Oracle’s naming convention standards, variable names should be meaningful and mnemonic (Oracle, 2019). As they are currently coded, it is very difficult to distinguish what information will be stored that makes them unique. The single parameter constructor sets both attributes to the same value, which further begs the question of what the difference is between these two attributes.
* Data is set via the constructor, which means that error checking and any type of text sanitizing would need to be performed when the object is created, or it will need to be added to the model class.
  + customer.java

A screen shot of a computer

Description automatically generated with medium confidence

* From a code quality perspective, we should be consistent with file naming conventions. Oracle suggests that class names should begin with capital letters (Oracle, 2019). Also note that account\_balance attribute is not explicitly marked as private which is inconsistent within this class, as well as when compared to other class attributes. Furthermore, account\_number and account\_balance are not consistent with Oracle’s recommended naming convention (Oracle, 2019).
* There is no constructor in this class. This means that Java will create a default constructor which serves to initialize the private variables to their default values. So, the accessor methods will work correctly in this scenario. However, the issue is that there is no way to set (or modify) the account\_number attribute. Therefore, it will be set to zero (0) for every customer object that is created because that is the default value set by the default constructor for int data types. As there is no way to modify the account number, every customer will have the same account number, which is a major concern. We should also seek clarity on using int for an account number, as String may be a more suitable datatype to use.
  + DocData.java

A screen shot of a computer program

Description automatically generated with low confidence

* From a code quality perspective, the read\_document method is not consistent with Oracle naming conventions, nor with other code in this app. According to Oracle, methods should camel-case, and there should not be an underscore between each word (Oracle, 2019).
* The DocData class is used to connect to the MySql backing database. It correctly uses a try/catch block for connecting to the database, however the connection is never closed. This could lead to a multitude of problems and undefined behaviors, such as exhaustion of resources leading to crashes and other DoS attacks (Manico & Detlefsen, 2015).
* A more critical security violation with this code is that the connection string is hardcoded with username and password of the root account. Obviously, anybody reading this file would then be able to access the backing database with highest level privileges which is a vulnerability. Furthermore, the connection string does not set security properties such as integrated security, trusted server certificate, and encrypt.

* + myDateTime.java

A screen shot of a computer

Description automatically generated with medium confidence

* From a code quality perspective, this class does not follow Oracle’s standard naming conventions (same reasoning as described for customer.java class). Also, the three attributes (mySecond, myMinute, and myHour) are not explicitly marked as private which is inconsistent with other classes in the application, and generally not good practice. The accessor method is named slightly different than the mutator (retrieveDateTime vs. setMyDateTime).
* Controllers – there are two controller classes implemented in the application: CRUDController.java, and GreetingController.java.
  + CRUDController.java

A screen shot of a computer program

Description automatically generated with low confidence

* + - It is not clear the intention of what will be returned when the controller returns a CRUD object. The code is calling the toString() function on a DocData object, which is a polymorphic function that all base objects in Java inherit. However, the DocData class does not override the toString() function to return meaningful data. Without it being overridden, the toString() function will return information about the object itself that is then set in both content fields in a CRUD object. Both of those fields have public accessor methods attached to them, which would therefore make information about the object publicly readable. This could be a critical vulnerability as undefined and exploitable information could be exposed.
  + GreetingController.java

A screen shot of a computer program

Description automatically generated with low confidence

* Open text input is allowed to be input into the Greeting constructor, which means that error checking and any type of text sanitizing would need to be performed when the object is created, or it will need to be added to the model class.
* RestServiceApplicationTests.java

A screen shot of a computer program

Description automatically generated with medium confidence

* + There are no unit tests implemented on current model and controller classes.
* Plug-ins and APIs – This section will be a review of the pom.xml file, which includes information about the dependencies, APIs, and plug-ins.

Based on the current pom.xml file in Artemis Financials’ codebase, they have hardcoded Spring Boot to version 2.2.4 (reference below screenshot), which also impacts the dependencies (second screenshot).

A screen shot of a computer code

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

The below screenshot shows that they are hardcoded to use Java version 8.

A picture containing text, screenshot, font, black

Description automatically generated

The below screenshot shows that the application is bound to version 1.46 of BouncyCastle. BouncyCastle is considered a leading cryptography dependency. As we will learn from the static analysis, version 1.46 is out of data and contains vulnerabilities.

A screen shot of a computer code

Description automatically generated with low confidence

Purely based on a manual code review, we can research these API’s and determine their vulnerabilities, what the current versions are, etc. Note that using Java 8 is not necessarily a vulnerability, as millions of applications are built in this version and Oracle is continuing to support the LTS versions. However, later versions of Java (Java 9+) are considered more secure and have performance enhancements.

## Static Testing

The following screenshots are the result of running the Maven Dependency-Check Plug-In on Artemis Financials’ codebase as it stands right now. I have included the screenshot of the overall results, as well as screenshots of the summary table that identifies the vulnerable dependencies, the CPE ids, and additional statistical data.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

The output indicates that there are 13 vulnerable dependencies and a total of 69 vulnerabilities found based on those dependencies. The following tables lists the CVE codes and descriptions for each dependency. Note that some dependencies have the same vulnerabilities as their core (for example, Spring and Tomcat). In cases where the vulnerabilities are listed multiple times in the NVD database, I only list them once here in the table.

|  |  |  |  |
| --- | --- | --- | --- |
| Dependency | Dependency Info | Published Vulnerability Codes | Description |
| bcprov-jdk15on-1.46.jar | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7 | CVE-2013-1624 | The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, a related issue to CVE-2013-0169. |
| CVE-2015-6644 | Bouncy Castle in Android before 5.1.1 LMY49F and 6.0 before 2016-01-01 allows attackers to obtain sensitive information via a crafted application, aka internal bug 24106146. |
| CVE-2015-7940 | The Bouncy Castle Java library before 1.51 does not validate a point is withing the elliptic curve, which makes it easier for remote attackers to obtain private keys via a series of crafted elliptic curve Diffie Hellman (ECDH) key exchanges, aka an "invalid curve attack." |
| CVE-2016-1000338 | In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure. |
| CVE-2016-1000339 | In the Bouncy Castle JCE Provider version 1.55 and earlier the primary engine class used for AES was AESFastEngine. Due to the highly table driven approach used in the algorithm it turns out that if the data channel on the CPU can be monitored the lookup table accesses are sufficient to leak information on the AES key being used. There was also a leak in AESEngine although it was substantially less. AESEngine has been modified to remove any signs of leakage (testing carried out on Intel X86-64) and is now the primary AES class for the BC JCE provider from 1.56. Use of AESFastEngine is now only recommended where otherwise deemed appropriate. |
| CVE-2016-1000341 | In the Bouncy Castle JCE Provider version 1.55 and earlier DSA signature generation is vulnerable to timing attack. Where timings can be closely observed for the generation of signatures, the lack of blinding in 1.55, or earlier, may allow an attacker to gain information about the signature's k value and ultimately the private value as well. |
| CVE-2016-1000342 | In the Bouncy Castle JCE Provider version 1.55 and earlier ECDSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure. |
| CVE-2016-1000343 | In the Bouncy Castle JCE Provider version 1.55 and earlier the DSA key pair generator generates a weak private key if used with default values. If the JCA key pair generator is not explicitly initialised with DSA parameters, 1.55 and earlier generates a private value assuming a 1024 bit key size. In earlier releases this can be dealt with by explicitly passing parameters to the key pair generator. |
| CVE-2016-1000344 | In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider. |
| CVE-2016-1000345 | In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES/ECIES CBC mode vulnerable to padding oracle attack. For BC 1.55 and older, in an environment where timings can be easily observed, it is possible with enough observations to identify when the decryption is failing due to padding. |
| CVE-2016-1000346 | In the Bouncy Castle JCE Provider version 1.55 and earlier the other party DH public key is not fully validated. This can cause issues as invalid keys can be used to reveal details about the other party's private key where static Diffie-Hellman is in use. As of release 1.56 the key parameters are checked on agreement calculation. |
| CVE-2016-1000352 | In the Bouncy Castle JCE Provider version 1.55 and earlier the ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider. |
| CVE-2017-13098 | BouncyCastle TLS prior to version 1.0.3, when configured to use the JCE (Java Cryptography Extension) for cryptographic functions, provides a weak Bleichenbacher oracle when any TLS cipher suite using RSA key exchange is negotiated. An attacker can recover the private key from a vulnerable application. This vulnerability is referred to as "ROBOT." |
| CVE-2018-5382 | The default BKS keystore use an HMAC that is only 16 bits long, which can allow an attacker to compromise the integrity of a BKS keystore. Bouncy Castle release 1.47 changes the BKS format to a format which uses a 160 bit HMAC instead. This applies to any BKS keystore generated prior to BC 1.47. For situations where people need to create the files for legacy reasons a specific keystore type "BKS-V1" was introduced in 1.49. It should be noted that the use of "BKS-V1" is discouraged by the library authors and should only be used where it is otherwise safe to do so, as in where the use of a 16 bit checksum for the file integrity check is not going to cause a security issue in itself. |
| CVE-2020-0187 | In engineSetMode of BaseBlockCipher.java, there is a possible incorrect cryptographic algorithm chosen due to an incomplete comparison. This could lead to local information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation.Product: AndroidVersions: Android-10Android ID: A-148517383 |
| CVE-2020-26939 | In Legion of the Bouncy Castle BC before 1.61 and BC-FJA before 1.0.1.2, attackers can obtain sensitive information about a private exponent because of Observable Differences in Behavior to Error Inputs. This occurs in org.bouncycastle.crypto.encodings.OAEPEncoding. Sending invalid ciphertext that decrypts to a short payload in the OAEP Decoder could result in the throwing of an early exception, potentially leaking some information about the private exponent of the RSA private key performing the encryption. |
| hibernate-validator-6.0.18.Final.jar | Hibernate's Bean Validation (JSR-380) reference implementation. | CVE-2020-10693 | A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages. |
| jackson-databind-2.10.2.jar | General data-binding functionality for Jackson: works on core streaming API | CVE-2020-25649 | A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity. |
| CVE-2020-36518 | jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects. |
| CVE-2022-42003 | In FasterXML jackson-databind before 2.14.0-rc1, resource exhaustion can occur because of a lack of a check in primitive value deserializers to avoid deep wrapper array nesting, when the UNWRAP\_SINGLE\_VALUE\_ARRAYS feature is enabled. Additional fix version in 2.13.4.1 and 2.12.17.1 |
| CVE-2022-42004 | In FasterXML jackson-databind before 2.13.4, resource exhaustion can occur because of a lack of a check in BeanDeserializer.\_deserializeFromArray to prevent use of deeply nested arrays. An application is vulnerable only with certain customized choices for deserialization. |
| log4j-api-2.12.1.jar | The Apache Log4j API | CVE-2020-9488 | Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1 |
| logback-core-1.2.3.jar | logback-core module | CVE-2021-42550 | In logback version 1.2.7 and prior versions, an attacker with the required privileges to edit configurations files could craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers. |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java | CVE-2022-1471 | SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond. |
| CVE-2017-18640 | The Alias feature in SnakeYAML before 1.26 allows entity expansion during a load operation, a related issue to CVE-2003-1564. |
| CVE-2022-25857 | The package org.yaml:snakeyaml from 0 and before 1.31 are vulnerable to Denial of Service (DoS) due missing to nested depth limitation for collections. |
| CVE-2022-38749 | Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stackoverflow. |
| CVE-2022-38751 | Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stackoverflow. |
| CVE-2022-38752 | Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stack-overflow. |
| CVE-2022-41854 | Those using Snakeyaml to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stack overflow. This effect may support a denial of service attack. |
| CVE-2022-38750 | Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stackoverflow. |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot | CVE-2022-27772 | \*\* UNSUPPORTED WHEN ASSIGNED \*\* spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method. NOTE: This vulnerability only affects products and/or versions that are no longer supported by the maintainer. |
| spring-boot-starter-web-2.2.4.RELEASE.jar | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container. | CVE-2022-27772 | \*\* UNSUPPORTED WHEN ASSIGNED \*\* spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method. NOTE: This vulnerability only affects products and/or versions that are no longer supported by the maintainer. |
| spring-core-5.2.3.RELEASE.jar | Spring Core | CVE-2022-22965 | A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it. |
| CVE-2021-22118 | In Spring Framework, versions 5.2.x prior to 5.2.15 and versions 5.3.x prior to 5.3.7, a WebFlux application is vulnerable to a privilege escalation: by (re)creating the temporary storage directory, a locally authenticated malicious user can read or modify files that have been uploaded to the WebFlux application, or overwrite arbitrary files with multipart request data. |
| CVE-2020-5421 | In Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versions, the protections against RFD attacks from CVE-2015-5211 may be bypassed depending on the browser used through the use of a jsessionid path parameter. |
| CVE-2022-22950 | In Spring Framework versions 5.3.0 - 5.3.16 and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial of service condition. |
| CVE-2022-22971 | In spring framework versions prior to 5.3.20+ , 5.2.22+ and old unsupported versions, application with a STOMP over WebSocket endpoint is vulnerable to a denial of service attack by an authenticated user. |
| CVE-2023-20861 | In Spring Framework versions 6.0.0 - 6.0.6, 5.3.0 - 5.3.25, 5.2.0.RELEASE - 5.2.22.RELEASE, and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition. |
| CVE-2023-20863 | In spring framework versions prior to 5.2.24 release+ ,5.3.27+ and 6.0.8+ , it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition. |
| CVE-2022-22968 | In Spring Framework versions 5.3.0 - 5.3.18, 5.2.0 - 5.2.20, and older unsupported versions, the patterns for disallowedFields on a DataBinder are case sensitive which means a field is not effectively protected unless it is listed with both upper and lower case for the first character of the field, including upper and lower case for the first character of all nested fields within the property path. |
| CVE-2022-22970 | In spring framework versions prior to 5.3.20+ , 5.2.22+ and old unsupported versions, applications that handle file uploads are vulnerable to DoS attack if they rely on data binding to set a MultipartFile or javax.servlet.Part to a field in a model object. |
| CVE-2021-22060 | In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. This is a follow-up to CVE-2021-22096 that protects against additional types of input and in more places of the Spring Framework codebase. |
| CVE-2021-22096 | In Spring Framework versions 5.3.0 - 5.3.10, 5.2.0 - 5.2.17, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. |
| spring-web-5.2.3.RELEASE.jar | Spring Web | CVE-2016-1000027 | Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data. |
| Plus all codes from core |  |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC | Same codes from spring-core |  |
| spring-beans-5.2.3.RELEASE.jar | Spring Beans | CVE-2022-22965 | A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it. |
| spring-context-5.2.3.RELEASE.jar | Spring Context | CVE-2022-22968 | In Spring Framework versions 5.3.0 - 5.3.18, 5.2.0 - 5.2.20, and older unsupported versions, the patterns for disallowedFields on a DataBinder are case sensitive which means a field is not effectively protected unless it is listed with both upper and lower case for the first character of the field, including upper and lower case for the first character of all nested fields within the property path. |
| spring-expression-5.2.3.RELEASE.jar | Spring Expression Language  (SpEL) | CVE-2022-22950 | In Spring Framework versions 5.3.0 - 5.3.16 and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial of service condition. |
| CVE-2023-20861 | In Spring Framework versions 6.0.0 - 6.0.6, 5.3.0 - 5.3.25, 5.2.0.RELEASE - 5.2.22.RELEASE, and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition. |
| CVE-2023-20863 | In spring framework versions prior to 5.2.24 release+ ,5.3.27+ and 6.0.8+ , it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition. |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation | CVE-2020-1938 | When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defence-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations. |
| CVE-2020-11996 | A specially crafted sequence of HTTP/2 requests sent to Apache Tomcat 10.0.0-M1 to 10.0.0-M5, 9.0.0.M1 to 9.0.35 and 8.5.0 to 8.5.55 could trigger high CPU usage for several seconds. If a sufficient number of such requests were made on concurrent HTTP/2 connections, the server could become unresponsive. |
| CVE-2020-13934 | An h2c direct connection to Apache Tomcat 10.0.0-M1 to 10.0.0-M6, 9.0.0.M5 to 9.0.36 and 8.5.1 to 8.5.56 did not release the HTTP/1.1 processor after the upgrade to HTTP/2. If a sufficient number of such requests were made, an OutOfMemoryException could occur leading to a denial of service. |
| CVE-2020-13935 | The payload length in a WebSocket frame was not correctly validated in Apache Tomcat 10.0.0-M1 to 10.0.0-M6, 9.0.0.M1 to 9.0.36, 8.5.0 to 8.5.56 and 7.0.27 to 7.0.104. Invalid payload lengths could trigger an infinite loop. Multiple requests with invalid payload lengths could lead to a denial of service. |
| CVE-2020-17527 | While investigating bug 64830 it was discovered that Apache Tomcat 10.0.0-M1 to 10.0.0-M9, 9.0.0-M1 to 9.0.39 and 8.5.0 to 8.5.59 could re-use an HTTP request header value from the previous stream received on an HTTP/2 connection for the request associated with the subsequent stream. While this would most likely lead to an error and the closure of the HTTP/2 connection, it is possible that information could leak between requests. |
| CVE-2021-25122 | When responding to new h2c connection requests, Apache Tomcat versions 10.0.0-M1 to 10.0.0, 9.0.0.M1 to 9.0.41 and 8.5.0 to 8.5.61 could duplicate request headers and a limited amount of request body from one request to another meaning user A and user B could both see the results of user A's request. |
| CVE-2021-41079 | Apache Tomcat 8.5.0 to 8.5.63, 9.0.0-M1 to 9.0.43 and 10.0.0-M1 to 10.0.2 did not properly validate incoming TLS packets. When Tomcat was configured to use NIO+OpenSSL or NIO2+OpenSSL for TLS, a specially crafted packet could be used to trigger an infinite loop resulting in a denial of service. |
| CVE-2022-29885 | The documentation of Apache Tomcat 10.1.0-M1 to 10.1.0-M14, 10.0.0-M1 to 10.0.20, 9.0.13 to 9.0.62 and 8.5.38 to 8.5.78 for the EncryptInterceptor incorrectly stated it enabled Tomcat clustering to run over an untrusted network. This was not correct. While the EncryptInterceptor does provide confidentiality and integrity protection, it does not protect against all risks associated with running over any untrusted network, particularly DoS risks. |
| CVE-2022-42252 | If Apache Tomcat 8.5.0 to 8.5.82, 9.0.0-M1 to 9.0.67, 10.0.0-M1 to 10.0.26 or 10.1.0-M1 to 10.1.0 was configured to ignore invalid HTTP headers via setting rejectIllegalHeader to false (the default for 8.5.x only), Tomcat did not reject a request containing an invalid Content-Length header making a request smuggling attack possible if Tomcat was located behind a reverse proxy that also failed to reject the request with the invalid header. |
| CVE-2020-9484 | When using Apache Tomcat versions 10.0.0-M1 to 10.0.0-M4, 9.0.0.M1 to 9.0.34, 8.5.0 to 8.5.54 and 7.0.0 to 7.0.103 if a) an attacker is able to control the contents and name of a file on the server; and b) the server is configured to use the PersistenceManager with a FileStore; and c) the PersistenceManager is configured with sessionAttributeValueClassNameFilter="null" (the default unless a SecurityManager is used) or a sufficiently lax filter to allow the attacker provided object to be deserialized; and d) the attacker knows the relative file path from the storage location used by FileStore to the file the attacker has control over; then, using a specifically crafted request, the attacker will be able to trigger remote code execution via deserialization of the file under their control. Note that all of conditions a) to d) must be true for the attack to succeed. |
| CVE-2021-25329 | The fix for CVE-2020-9484 was incomplete. When using Apache Tomcat 10.0.0-M1 to 10.0.0, 9.0.0.M1 to 9.0.41, 8.5.0 to 8.5.61 or 7.0.0. to 7.0.107 with a configuration edge case that was highly unlikely to be used, the Tomcat instance was still vulnerable to CVE-2020-9494. Note that both the previously published prerequisites for CVE-2020-9484 and the previously published mitigations for CVE-2020-9484 also apply to this issue. |
| CVE-2021-30640 | A vulnerability in the JNDI Realm of Apache Tomcat allows an attacker to authenticate using variations of a valid user name and/or to bypass some of the protection provided by the LockOut Realm. This issue affects Apache Tomcat 10.0.0-M1 to 10.0.5; 9.0.0.M1 to 9.0.45; 8.5.0 to 8.5.65. |
| CVE-2022-34305 | In Apache Tomcat 10.1.0-M1 to 10.1.0-M16, 10.0.0-M1 to 10.0.22, 9.0.30 to 9.0.64 and 8.5.50 to 8.5.81 the Form authentication example in the examples web application displayed user provided data without filtering, exposing a XSS vulnerability. |
| CVE-2021-24122 | When serving resources from a network location using the NTFS file system, Apache Tomcat versions 10.0.0-M1 to 10.0.0-M9, 9.0.0.M1 to 9.0.39, 8.5.0 to 8.5.59 and 7.0.0 to 7.0.106 were susceptible to JSP source code disclosure in some configurations. The root cause was the unexpected behaviour of the JRE API File.getCanonicalPath() which in turn was caused by the inconsistent behaviour of the Windows API (FindFirstFileW) in some circumstances. |
| CVE-2021-33037 | Apache Tomcat 10.0.0-M1 to 10.0.6, 9.0.0.M1 to 9.0.46 and 8.5.0 to 8.5.66 did not correctly parse the HTTP transfer-encoding request header in some circumstances leading to the possibility to request smuggling when used with a reverse proxy. Specifically: - Tomcat incorrectly ignored the transfer encoding header if the client declared it would only accept an HTTP/1.0 response; - Tomcat honoured the identify encoding; and - Tomcat did not ensure that, if present, the chunked encoding was the final encoding. |
| CVE-2019-17569 | The refactoring present in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50 and 7.0.98 to 7.0.99 introduced a regression. The result of the regression was that invalid Transfer-Encoding headers were incorrectly processed leading to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner. Such a reverse proxy is considered unlikely. |
| CVE-2020-1935 | In Apache Tomcat 9.0.0.M1 to 9.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99 the HTTP header parsing code used an approach to end-of-line parsing that allowed some invalid HTTP headers to be parsed as valid. This led to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner. Such a reverse proxy is considered unlikely. |
| CVE-2020-13943 | If an HTTP/2 client connecting to Apache Tomcat 10.0.0-M1 to 10.0.0-M7, 9.0.0.M1 to 9.0.37 or 8.5.0 to 8.5.57 exceeded the agreed maximum number of concurrent streams for a connection (in violation of the HTTP/2 protocol), it was possible that a subsequent request made on that connection could contain HTTP headers - including HTTP/2 pseudo headers - from a previous request rather than the intended headers. This could lead to users seeing responses for unexpected resources. |
| CVE-2023-28708 | When using the RemoteIpFilter with requests received from a reverse proxy via HTTP that include the X-Forwarded-Proto header set to https, session cookies created by Apache Tomcat 11.0.0-M1 to 11.0.0.-M2, 10.1.0-M1 to 10.1.5, 9.0.0-M1 to 9.0.71 and 8.5.0 to 8.5.85 did not include the secure attribute. This could result in the user agent transmitting the session cookie over an insecure channel. |
| CVE-2021-43980 | The simplified implementation of blocking reads and writes introduced in Tomcat 10 and back-ported to Tomcat 9.0.47 onwards exposed a long standing (but extremely hard to trigger) concurrency bug in Apache Tomcat 10.1.0 to 10.1.0-M12, 10.0.0-M1 to 10.0.18, 9.0.0-M1 to 9.0.60 and 8.5.0 to 8.5.77 that could cause client connections to share an Http11Processor instance resulting in responses, or part responses, to be received by the wrong client. |
| tomcat-embed-websocket-9.0.30.jar |  | CVE-2020-8022 | A Incorrect Default Permissions vulnerability in the packaging of tomcat on SUSE Enterprise Storage 5, SUSE Linux Enterprise Server 12-SP2-BCL, SUSE Linux Enterprise Server 12-SP2-LTSS, SUSE Linux Enterprise Server 12-SP3-BCL, SUSE Linux Enterprise Server 12-SP3-LTSS, SUSE Linux Enterprise Server 12-SP4, SUSE Linux Enterprise Server 12-SP5, SUSE Linux Enterprise Server 15-LTSS, SUSE Linux Enterprise Server for SAP 12-SP2, SUSE Linux Enterprise Server for SAP 12-SP3, SUSE Linux Enterprise Server for SAP 15, SUSE OpenStack Cloud 7, SUSE OpenStack Cloud 8, SUSE OpenStack Cloud Crowbar 8 allows local attackers to escalate from group tomcat to root. This issue affects: SUSE Enterprise Storage 5 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP2-BCL tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP2-LTSS tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP3-BCL tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP3-LTSS tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP4 tomcat versions prior to 9.0.35-3.39.1. SUSE Linux Enterprise Server 12-SP5 tomcat versions prior to 9.0.35-3.39.1. SUSE Linux Enterprise Server 15-LTSS tomcat versions prior to 9.0.35-3.57.3. SUSE Linux Enterprise Server for SAP 12-SP2 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server for SAP 12-SP3 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server for SAP 15 tomcat versions prior to 9.0.35-3.57.3. SUSE OpenStack Cloud 7 tomcat versions prior to 8.0.53-29.32.1. SUSE OpenStack Cloud 8 tomcat versions prior to 8.0.53-29.32.1. SUSE OpenStack Cloud Crowbar 8 tomcat versions prior to 8.0.53-29.32.1. |
| Plus all codes from tomcat embedded core |  |

## Mitigation Plan

This section will detail the mitigation plan for vulnerabilities discussed throughout the assessment. I will discuss the mitigation plan for manual review and static analysis separately.

*Manual review vulnerability mitigation plan*

The mitigation plan for the manual review will be to provide recommendations for each of the areas discussed in the manual review of each class, if necessary, as well as the pom.xml file.

* Greeting.java
  + Implement validation and sanitization of the open text input String attribute.
* CRUD.java
  + Rename the class variables content and content2 to meaningful, mnemonic names so it is clearer what information is being stored in them and what the difference is between them.
  + Implement validation and sanitization of the content attributes as they are open text input.
* customer.java
  + Rename the class to Customer.java. Set account\_balance and account\_number attributes to be explicitly private. Set both variables to final. Rename both variables to camel case and remove the underscore.
  + Make recommendation that account\_number should be a String datatype instead of int.
  + Add a constructor.
  + Add a way to create and set a unique account number for each customer. This can be accomplished in the constructor or a private mutator function per each attribute.
* DocData.java
* It is considered better practice to use the “try-with-resources block” (Manico & Detlefsen, 2015), instead of standard Try-Catch block when using resources. Try-with-resources blocks will automatically close connections regardless of whether the connection was successful or not. They were introduced in Java 7 and should be used in any application built from that version and forward. If we are using a version of Java prior to this, then we should have a finally clause and manually close the database connection using the .close() method.
* Rename the read\_document method to readDocument.
* Store the database password that is currently hardcoded in the connection string in a separate file that has restricted access. Then obtain the password from the file and store it in a variable and use that variable in the connection string. Another option is to avoid using the DriverManager.getConnection() method entirely, and instead use the MySQL Connector/J driver to interface with JDBC. Connector/J is certificate-based and uses SSL to encrypt data that is transferred between JDBC and the MySQL server. The connection string should be something like this:

String connectionUrl =

“jdbc:sqlserver://localhost:<port number>;” +

“databaseName=twkAppData;integratedSecurity=true;” +

“encrypt=true; trustServerCertificate=false;” +

“trustStore=storeName;trustStorePassword=storePassword;” +

“hostNameInCertificate=hostname”;

* myDateTime.java
  + Attributes should be marked as private, so they are consistent with other classes in the application. Furthermore, rename the accessor method to getMyDateTime so it is consistent with standard naming conventions of mutators and with the name of the class.
* CRUDController.java
  + Override the toString() function in DocData class to provide sanitized/safe information that is appropriate for the DocData class to consume or create a parameterized constructor in DocData that handles requests appropriately and use this constructor instead of unparameterized constructor.
* GreetingController.java
  + Implement validation and sanitization of the greeting since it is open text input.
* RestServiceApplicationTests.java
  + Need to implement unit testing of all model and controller classes.
* Spring Framework (including Boot and any others to be integrated later)
  + Upgrade to latest version
* BouncyCastle
  + Upgrade to latest version
* Java version 8
  + Consider using later version of Java

*Static vulnerability mitigation plan:*

The Dependency-Check Plugin provides a list of vulnerabilities with the dependencies identified in the code base. The results identify 13 vulnerable dependencies, with a total of 69 vulnerabilities. In our previous assessment, Spring Framework was showing more dependencies with vulnerabilities even though the same version is coded in the pom.xml file. So, I incorporated those previous Spring vulnerabilities into this report because there are Spring vulnerabilities that are part of the framework. It is also important to note that many of the vulnerabilities are duplicated on multiple dependencies. For example, Spring Boot has multiple dependencies (including the Core, Web, and WebMVC components), and there are several vulnerabilities that apply to each component. The Tomcat dependency is another example where there are duplicate vulnerabilities. I want to also point out that the project is forced to use Java version 8 as it is explicitly programmed into the pom.xml file (version 1.8). It might be worth recommending upgrading to a later version of Java (9 or greater).

I have built a chart of the 13 vulnerable dependencies and a high-level strategy for mitigating the vulnerabilities. The mitigation strategy will be based on the vulnerability assessment process flow diagram (VAPFD) by identifying the areas of security that I thought are most relevant. Based on researching the vulnerability on the CVE, NVD, and other web resources, the manual mitigation for all these CPE’s is to upgrade our dependencies to later versions where the vulnerability has been addressed. We should also take steps to ensure we are following secure coding practices in any upgraded version as well. Even though we upgrade a dependency to a later version, there could still be unknown vulnerabilities, so it is important to test the dependency on our own.

I want to focus on the Spring Boot vulnerabilities as that dependency is in our pom.xml file. It is currently set to use version 2.2.4, which is out of date. So, our focus for mitigation of this is to upgrade it to the latest version (3.0.x). This should be done for the Core, Web, and WebMVC components as well. Also note that the Tomcat vulnerabilities can also be mitigated by blocking or disabling the Apache JServ Protocol (AJP). This is a special case considering some organizations may not have access to, or the ability to quickly upgrade their servers. Therefore, they can still mitigate the vulnerability using the work around until they can upgrade.



References

*Code Conventions for the Java Programming Language: 9. Naming Conventions*. (n.d.). Www.oracle.com. <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>

Contrast Security. (2014). *The Unfortunate Reality of Insecure Libraries Contrast Security*. <https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf>

*CS305 Vulnerability Assessment Process Flow Diagram*. (n.d.). Southern New Hampshire University.

*CVE - Common Vulnerabilities and Exposures (CVE)*. (2016). Mitre.org. <https://cve.mitre.org/index.html>

FTC. (2013, March). *International*. Federal Trade Commission. <https://www.ftc.gov/policy/international>

*Hacked... Because of an Insecure Library*. (n.d.). Www.contrastsecurity.com. Retrieved May 9, 2023, from <https://www.contrastsecurity.com/security-influencers/hacked-because-of-an-insecure-library>

Jeganathan, S. (2019). DevSecOps: A Systemic Approach for Secure Software Development. *ISSA Journal*, *17*(11), 20–27. International Security & Counter Terrorism Reference Center.

Jeyaraman, M. (2023, February 23). *8 Critical Innovations in the Evolution of Web Technology*. TurboFuture. <https://turbofuture.com/internet/The-Evolution-of-Web-Technology-Understanding-the-Past-and-Present-of-the-Web>

*Libraries & Application Security, Part 1: Known Vulnerabilities*. (n.d.). Www.contrastsecurity.com. Retrieved May 9, 2023, from <https://www.contrastsecurity.com/security-influencers/libraries-and-application-security-part-one-known-vulnerabilities>

Manico, J., & Detlefsen, A. (2015). *Iron-Clad Java : Building Secure Web Applications /*. New York, N.Y. Mcgraw-Hill Education.

NIST. (2019). *NVD - Home*. Nist.gov. <https://nvd.nist.gov/>

Oracle. (2019). *Secure Coding Guidelines for Java SE*. Oracle.com. <https://www.oracle.com/java/technologies/javase/seccodeguide.html>

OWASP. (2019). *Input Validation · OWASP Cheat Sheet Series*. Owasp.org. <https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html>

OWASP. (2021a). *A06 Vulnerable and Outdated Components - OWASP Top 10:2021*. Owasp.org. <https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/>

OWASP. (2021b). *OWASP Top Ten*. Owasp.org; OWASP. <https://owasp.org/www-project-top-ten/>

*Security Injection: Buffer Overflow - CS0 Java*. (2023). Towson.edu. <https://cisserv1.towson.edu/~cyber4all/modules/nanomodules/Buffer_Overflow-CS0_Java.html>

Wilson Consulting Group. (n.d.). *GLBA Compliance Consulting | GLBA Assessment Services*. Wilsoncgrp.com. Retrieved May 14, 2023, from <https://wilsoncgrp.com/gramm-leach-bliley-act-compliance>