



## CS 305 Project One Template

### Document Revision History

Version	Date	Author	Comments
1.0	[May 23,2024]	[Kevin Alexander]	

### Client



### Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend 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 include images or supporting materials. If you include them, make certain to insert them in 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

[Insert your name here.]

### 1. Interpreting Client Needs

Determine your client's needs and potential threats and attacks associated with the company's application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

.....

(Question 2)

Secure communications are of utmost importance to Artemis Financial as they deal with

sensitive financial information of their customers. By ensuring secure communications, the

company can protect the confidentiality, integrity, and authenticity of the data being transmitted

over their RESTful web API. This helps in preventing unauthorized access, data breaches, and

eavesdropping by malicious actors. Secure communications also build trust with

customers, enhance the company's reputation, and ensure compliance with data protection

regulations. Overall, the value of secure communications is crucial in safeguarding the company's

assets and maintaining a secure environment for conducting financial transactions.

- Yes, the company Artemis Financial makes international transactions as it provides financial services to customers around the world. In the context of protecting against external threats, it is crucial for Artemis Financial to ensure that its web-based software application is secure and resilient to potential security vulnerabilities that could be exploited by malicious actors. By conducting a thorough vulnerability assessment and implementing appropriate security measures, Artemis Financial can enhance the security of its systems and protect sensitive financial information during international transactions.
- Are there governmental restrictions on secure communications to consider? Yes, there are government restrictions on secure communications that companies need to consider when protecting against external threats. Depending on the country where the company



operates, there may be regulations and laws that dictate how data should be implemented, and how encryption should be implemented, and how secure communications should be maintained. In the United States, companies may need to comply with regulations such as HIPAA for healthcare data, GDPR for European Union citizens data, or PCI DSS for payment card information. It is essential for companies like Artemis Financial to be aware of and comply with these regulations to ensure secure communications and protect against external threats effectively.

- -DDos Attacks distributed denial of service attacks can overwhelm the web application with a volume of traffic, causing it to become slow or unavailable.

SQL Injection attackers might attempt to inject malicious SQL queries into the application's input fields to gain unauthorized access to the database.

-Cross-Site Scripting(XSS): This attack involves injecting malicious scripts into web pages viewed by other users. It can be used to steal sensitive information or session cookies.

Phishing Attacks then attackers might send deceptive emails or messages to users to trick them into revealing sensitive information such as login credentials.

Brute Force Attacks hackers might try to gain unauthorized access by systematically trying different combinations of usernames and passwords until they find the correct ones.

- (Question 2)

[-Use of Secure Coding Practices: Ensure that the development team follows secure coding practices to prevent common vulnerabilities such as injection attacks, cross-site scripting(XSS), and cross-site request forgery(CSRF).

Regular Security Updates: Stay up-to-date with security patches and updates for all software components, including operating systems, web servers, frameworks, and libraries, to address known vulnerabilities.

-Secure Communication:Implement secure HTTPS to encrypt data transmitted between clients and servers, protecting sensitive information from eavesdropping and man-in-the-middle attacks .

Authentication and Authorization:Implement strong authentication mechanisms,such as multi-factor authentication,and ensure that users have the appropriate permissions to access resources.

-Data Encryption”Utilize encryption techniques to protect sensitive data at rest and in transit,safeguarding customer information and financial data from unauthorized access.

-Role of Open-Source Libraries:When using open-source libraries,ensure that they are actively maintained,have a good security track record,and are regularly updated to address security vulnerabilities. Conduct regular security assessments of third-party libraries to identify and mitigate any potential risks.

-Evolving Web Application Techniques:Stay informed about evolving web application security techniques and best practices,such as implementing Content Security Policy(CSP),secure cookie attributes.]

## **2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

[-Input Validation:This is important to prevent malicious input from being processed by the application,which could lead to security vulnerabilities such as injection attacks.

-Code Review:Regular code reviews help identify security vulnerabilities and ensure that secure coding practices are followed throughout the development process.

-Vulnerability Assessment:Conducting vulnerability assessments helps identify weaknesses in the application that could be exploited by attackers.

-Cryptography:Using encryption and secure cryptographic algorithms is essential to protect sensitive



data stored and transmitted by the application.

- Secure API Interactions:Ensuring that APIs are accessed securely helps prevent unauthorized access and data breaches.

- Secure Data Structures:Implementing secure data structures helps protect sensitive information from unauthorized access or tampering.

- Secure Error Handling:Proper error handling mechanisms are crucial to prevent sensitive information leakage and potential security vulnerabilities.

- Secure Coding Practices/Patterns:Following secure coding practices and design patterns help reduce the likelihood of introducing security vulnerabilities in the application.

- Secure Distributed Architecture:Securely designing the application's architecture for distributed environments helps protect against various security threats in a distributed system.

- Secure Data Access: Implementing secure data access controls helps prevent unauthorized access to sensitive data within the application.

Each of these areas is relevant to Artemis Financial's Software application because they collectively contribute to building a secure and robust application that protects sensitive financial data,prevents unauthorized access,and mitigates potential.]

### **3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

[(CRUD.java)There are several security vulnerabilities in the provided code snippet: Missing closing braces for the constructors.Typo in the constructor parameter names(content vs content1). Lack of access modifiers for the class and methods(e.g,public,private,protected).No input validation or sanitization for the content parameters. No visibility modifiers for the instance

variables(content,content2).No validation for the content2 parameter in the second constructor.No encapsulation for the instance variables(direct access via getters).No authentication or authorization mechanisms. No secure coding practices implemented (e.g,input validation,output encoding,etc.)These vulnerabilities can lead to potential security risks such as injection attacks,data leakage,and unauthorized access. It is crucial to address these issues to ensure the code's security

(CRUDController.java)There are several security vulnerabilities in the code snippet:Import Statement

Issue:There is a typo in the import statement.Method Declaration Issue:The method 'CRUD' is declared

with a return type CUD',which seems to be a typo. It should be 'CRUD' to match the class name.Object

Creation Issue:The code tries to create an instance of 'DocData' and 'CRUD' classes,but the 'CRUD' class

is not defined in the provided code snippet.If 'CRUD' is supposed to represent the same class as

'CRUDController',then the return statement should be corrected.Missing Class Definition:The class

'DocData' is referenced but not defined in the provided code snippet.You need to define the

'DocData'class or import it if it's from an external library. RequestMapping

Annotation:The'RequestMapping'annotation is placed outside the class definition,which is incorrect. It

should be placed above the method that needs to be mapped to a specific URL. In this case,it should be

placed above the 'CRUD' method.

(customer.java)There are several issues and vulnerabilities in the provided code snippet:There is a syntax

error in the class definition.The variable"account number"should not have a space,it should be

"accountNumber".The"showInfo"method is missing a closing curly brace'.The "showinfo"method is

missing a closing curly brace.The "showInfo"method should return a String or void instead of int as it is

intended to display customer information.The "deposit"method is missing a closing curly brace. The

"deposit"method does not have proper validation for the deposit amount 'a'.It should check if the



amount is positive before adding it to account balance. The “account balance” variable should be private to encapsulate it to prevent direct access from outside the class. The class name should start with an uppercase letter according to Java naming conventions. These issues need to be addressed to improve the quality and security of the application.

(DocData.java) There are several security vulnerabilities in the provided code snippet:

- SQL Injection: The code is vulnerable to SQL injection attacks as it directly concatenates user input (key and value) into the SQL query without proper validation or sanitization. This can allow attackers to manipulate the query and potentially access or modify the database in unintended ways.

- Plain Text Password: Strong database password in plain text within the code is a security risk. It is recommended to use secure methods like environment variables or configuration files with restricted access to store sensitive information.

- Exception Handling: The exception handling in the code is not robust. Printing the stack trace using ‘e.printStackTrace()’ can expose sensitive information to attackers. It is advisable to handle exceptions more gracefully and log errors without revealing too much information.

- No Input Validation: The code lacks input validation for key and value parameters passed to the read document method. Proper input validation should be implemented to prevent unexpected behavior or attacks.

- Class.forName Security: Using ‘Class.forName(“com.mysql.jdbc.Driver”)’ to load the MySQL driver is not recommended for modern JDBC versions. It is better to use the ‘DriverManager.registerDriver’ method or rely on the JDBC driver auto-loading mechanism.

- Connection String: The connection string contains sensitive information like the database username and password. It is advisable to store such information securely and not hardcode it in the code.

hardcode it in the source code.

(Greeting.java) There are several security vulnerabilities in the provided code snippet:

- Syntax Error: There is a syntax error in the constructor where's `this, content = content;` should be `'this.content = content;'`.

- Insecure Object Initialization: The 'id' and 'content' fields are declared as 'final', which means they cannot be changed after initialization. However, there is no validation or sanitization of the input values, which could lead to potential security vulnerabilities like injection attacks.

- Lack of Input Validation: The constructor does not perform any input validation on the 'id' and 'content' parameters, which could lead to unexpected behavior or security issues.

- Lack of Encapsulation: The fields 'id' and 'content' are declared as 'public', which breaks encapsulation principles and can lead to uncontrolled access and modification of the class state.

These vulnerabilities should be addressed to ensure the code is secure and follows best practices.

(GreetingController.java) There are several security vulnerabilities in the provided code snippet:

- Lack of Input Validation: The code does not validate the input parameter "name" received from the client. This can lead to various security issues such as injection attacks (e.g., injection, XSS).

- Lack of output Encoding: The code does not encode the output before returning it to the client. This can lead to Cross-Site Scripting (XSS) vulnerabilities.

- Lack of Authentication and Authorization: The code does not include any authentication or authorization mechanisms to control access to the "/greeting" endpoint. This can lead to unauthorized access to sensitive information.

- Lack of HTTPS: If this code is part of a web service, it is important to ensure that HTTPS is used to encrypt the communication between the client and the server to prevent eavesdropping and man-in-the-middle attacks.



(myDateTime.java)The code provided has several issues related to security vulnerabilities:

-Access Modifiers:The variables 'mySecond','myMinute',and 'myHour'are declared as public,which

exposes them to direct access and modification from outside the class.It is recommended to use private access modifiers and provide public methods to access and modify these variables.

-Array Initialization:In the 'retrieveDateTime'method,the code tries to return a new integer array with size 'B',which is not defined in the code snippet. This could lead to an `ArrayIndexOutOfBoundsException` or other unexpected behavior. It should return an array with the correct size based on the number of elements to be retrieved.

-Incomplete Methods:The code contains incomplete method implementations with comments indicating "implementing accessor method".This could lead to compilation errors or unexpected behavior if not properly implemented.

-Input Validation:The `setMyDateTime`'method does not perform any input validation on the 'seconds','minutes',and 'hour' parameters.It is important to validate input data to prevent unexpected behavior or security vulnerabilities,

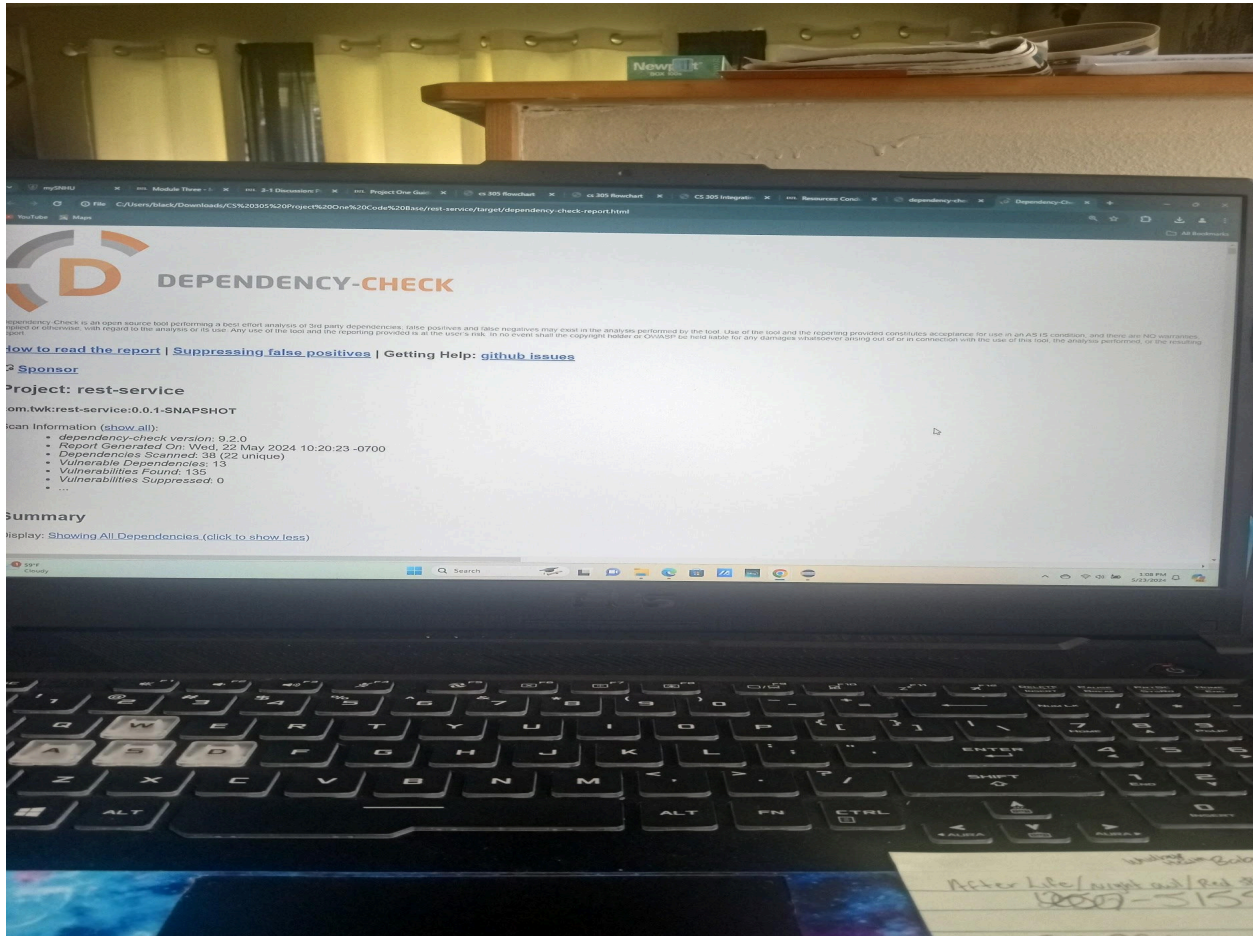
(RestServiceApplication.java)Based on the provided code snippet,there are no apparent security vulnerabilities present. The code seems to be a basic Spring Boot application that starts a REST service using `SpringApplication`. However,it's important to note that security vulnerabilities can arise from various aspects such as input validation,authentication,authorization,data handling,and more,which are not visible in the provided code snippet. It is recommended to conduct a thorough security review of the entire application codebase and configuration to identify and address any potential security issues.]

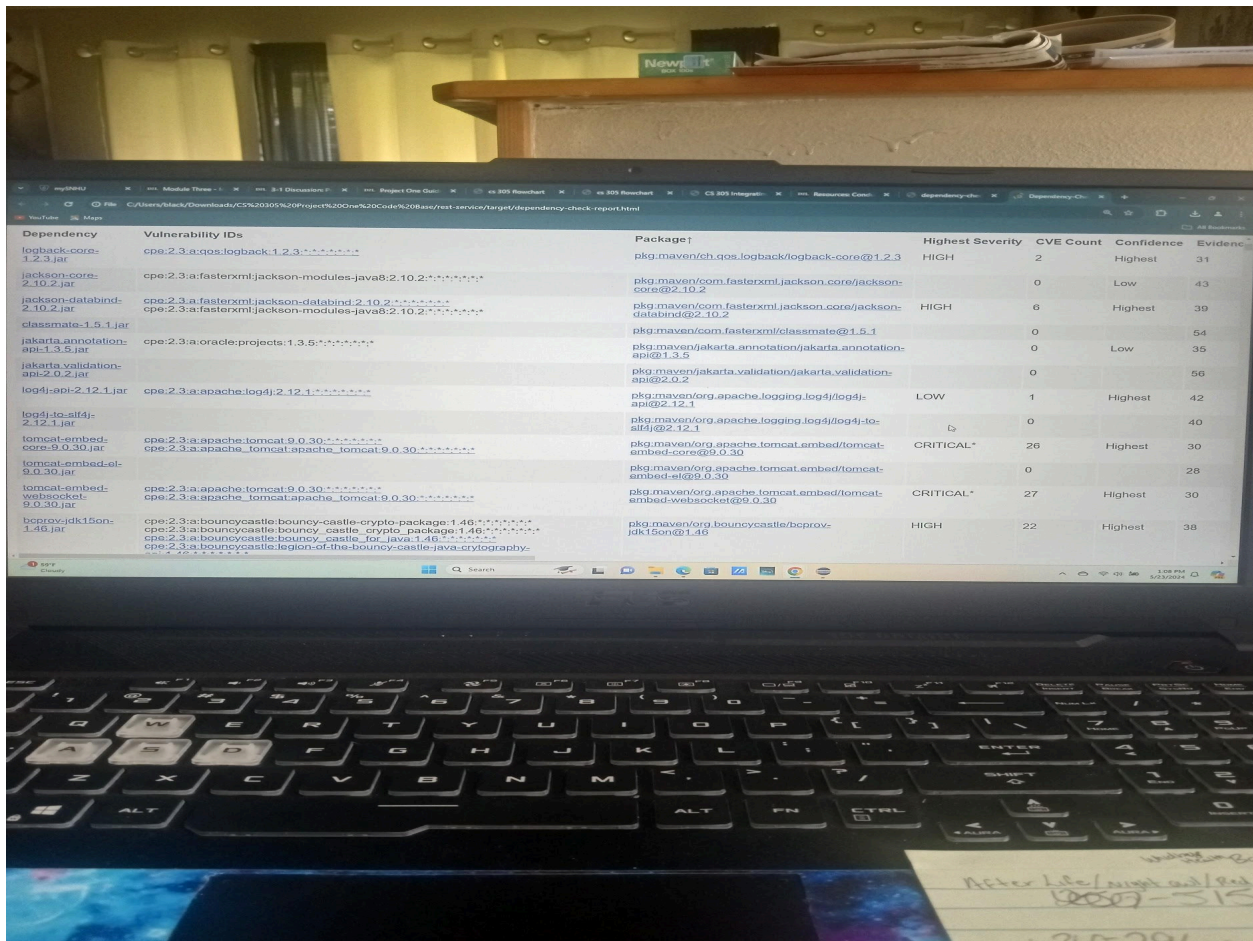
#### **4. Static Testing**

Run a dependency check on Artemis Financial's software application to identify all security vulnerabilities in the code. Record the output from the dependency-check report. Include the following items:

- The names or vulnerability codes of the known vulnerabilities

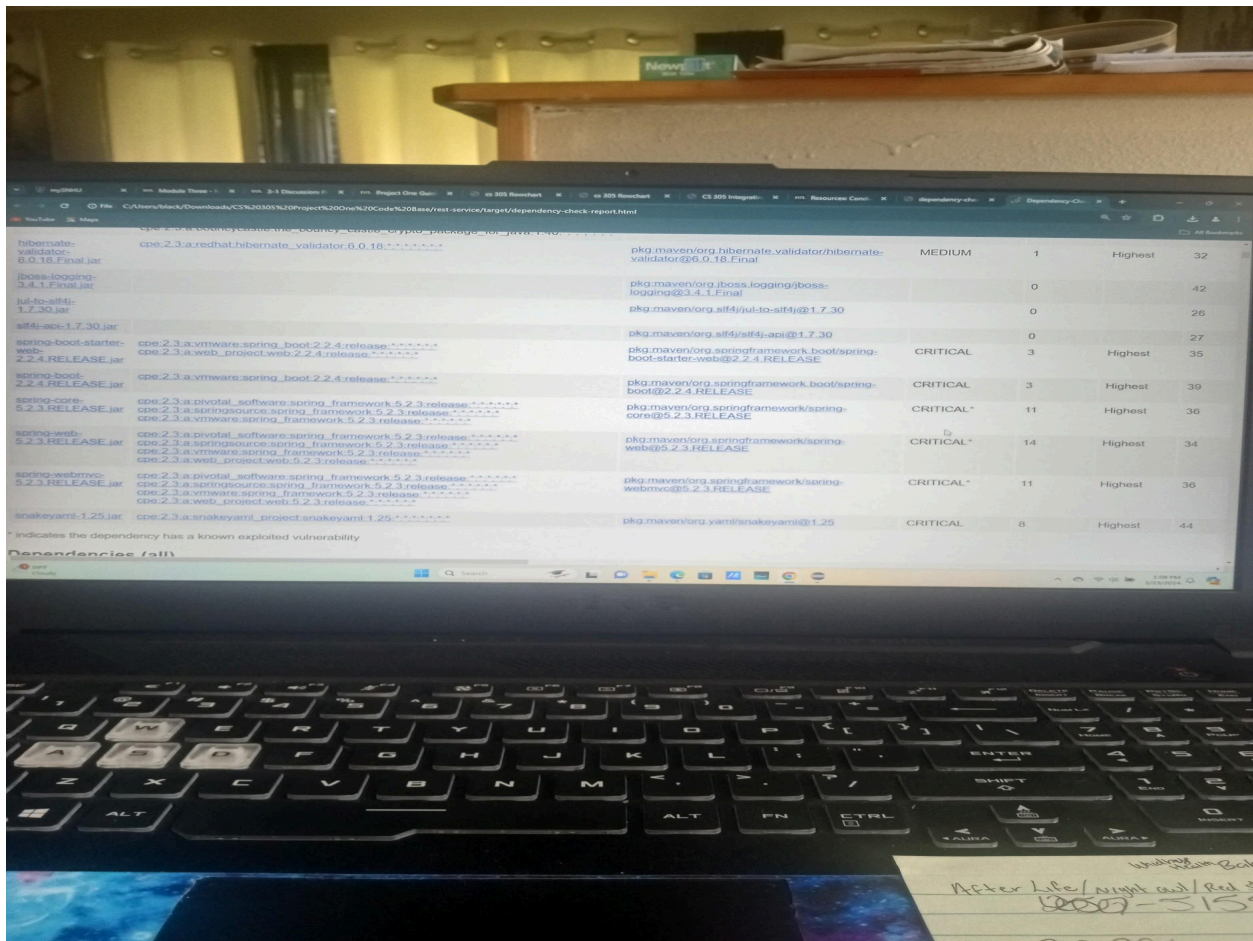
- A brief description and recommended solutions provided by the dependency-check report
- Any attribution that documents how this vulnerability has been identified or documented previously

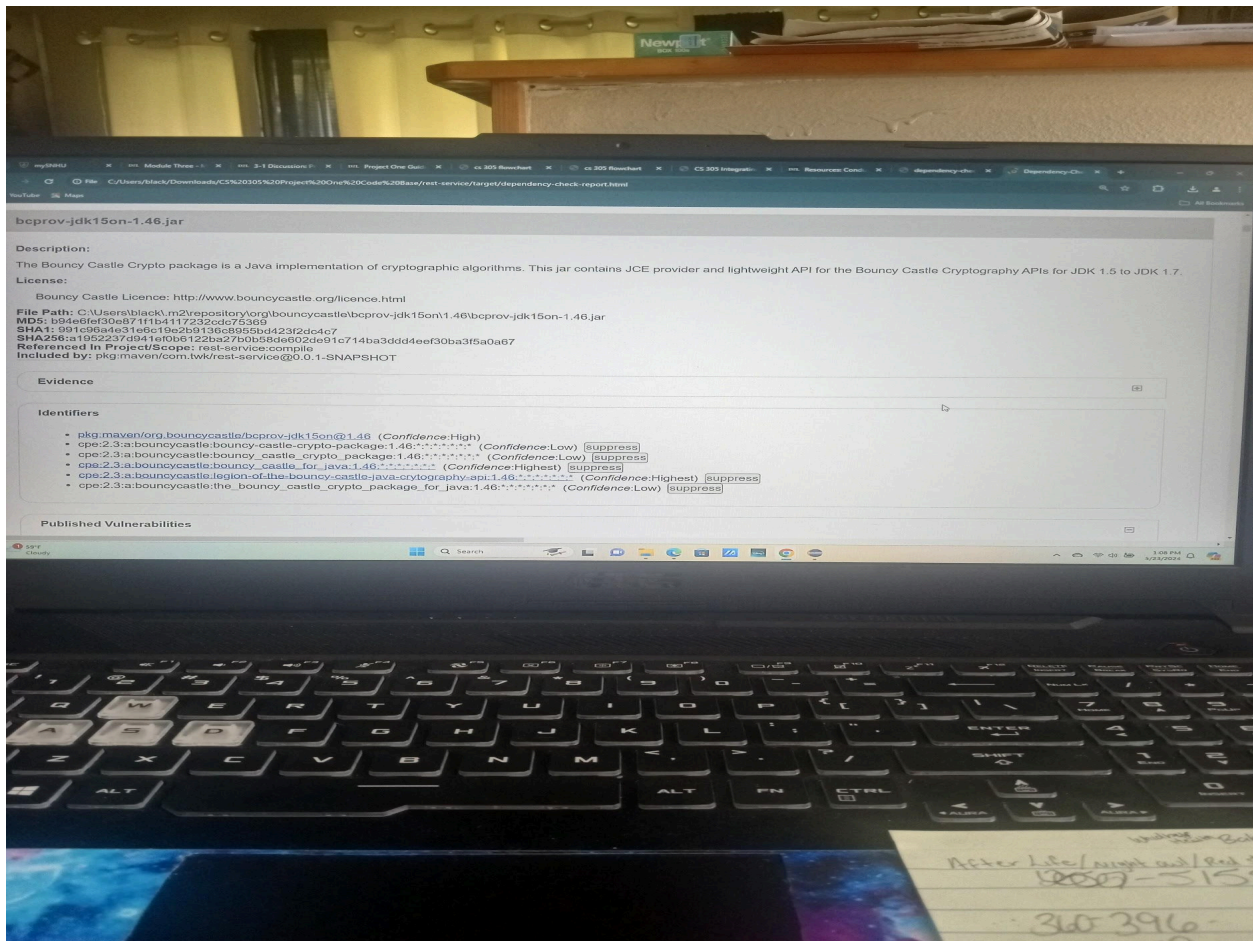




Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence
logback-core-1.2.3.jar	cpe:2.3:a:qos:logback:1.2.3:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.2.3	HIGH	2	Highest	31
jackson-core-2.10.2.jar	cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-core@2.10.2		0	Low	43
jackson-databind-2.10.2.jar	cpe:2.3:a:fasterxml:jackson-databind:2.10.2:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.10.2	HIGH	6	Highest	39
classmate-1.5.1.jar		pkg:maven/com.fasterxml/classmate@1.5.1		0		54
jakarta.annotation-api-1.3.5.jar	cpe:2.3:a:oracle:projects:1.3.5:*:*:*:*	pkg:maven/jakarta.annotation/jakarta.annotation-api@1.3.5		0	Low	35
jakarta.validation-api-2.0.2.jar		pkg:maven/jakarta.validation/jakarta.validation-api@2.0.2		0		56
log4j-api-2.12.1.jar	cpe:2.3:a:apache:log4j:2.12.1:*:*:*:*	pkg:maven/org.apache.logging.log4j/log4j-api@2.12.1	LOW	1	Highest	42
log4j-to-slf4j-2.12.1.jar		pkg:maven/org.apache.logging.log4j/log4j-to-slf4j@2.12.1		0		40
tomcat-embed-core-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@9.0.30	CRITICAL*	26	Highest	30
tomcat-embed-el-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-el@9.0.30		0		28
tomcat-embed-websocket-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-websocket@9.0.30	CRITICAL*	27	Highest	30
bcpkix-jdk15on-1.46.jar	cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46:*:*:*:*	pkg:maven/org.bouncycastle/bcpkix-jdk15on@1.46	HIGH	22	Highest	38







.]

## 5. Mitigation Plan

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial's software application.

[1.For CVE-2023-6378:

- Upgrade logback to a version that is not affected by the serialization vulnerability. In this case, versions from 1.20.0 up to(excluding) 1.2.13 are vulnerable. Therefore,upgrading to version 1.2.13 or higher would be recommended.
- Apply patches or fixes provided by the logback project to address the vulnerability.
- Implement proper input validation and sanitization to prevent malicious data from being deserialized.



- Monitor security advisors and updates from logback and other relevant sources for any new information or patches related to this vulnerability.

## 2.CVE-2021-42550:

- Upgrade logback to a version that is not affected by the arbitrary code execution vulnerability. In this case,versions prior to 1.2.7 are vulnerable. Therefore,upgrading to version 1.2.7 or higher would be recommended.

- Restrict privileges for editing configuration files to only trusted users to prevent unauthorized changes that could lead to arbitrary code execution.

- Implement secure coding practices and configurations to prevent the loading of arbitrary code from LDAP servers.

- Regularly review and audit configuration files for any unauthorized or suspicious changes.

Based on the provided information, the vulnerability described is related to CVE-2020-25649 in FasterXML Jackson Databind,which allows for XML external entity(XXE) attacks due to improper restriction of XML external entity reference. The highest threat from this vulnerability is data integrity.

The CVSS scores are as follows:

- CVSSv2 Base Score: MEDIUM(5.0)

- CVSSv3 Base Score: High(7.5)

To address these vulnerabilities,it is recommended to apply the appropriate patch provided by the vendor. In this case,the vulnerable software version is FasterXML Jackson Databind up to version 2.10.0 (excluding 2.10.5). The patch for this vulnerability is available in version 2.13.0.

To mitigate the risk associated with this vulnerability, it is crucial to update the FasterXML Jackson Databind library to a version that includes the fix for CVE-2020-25649,such as version 2.13.0 or later.

The vulnerability described is related to improper privilege management in Apache Tomcat when



handling Apache JServ Protocol(AJP) connections. These vulnerabilities can be exploited by attackers to gain higher trust levels than intended,potentially leading to unexpected security risks.

To mitigate this vulnerability, users are advised to take the following actions:

- Apply updates per vendor instructions: Users should update their Apache Tomcat installations to version 9.0.31 or later, as recommended by the vendor.

- Disable the AJP Connector if not required: If AJP connections are not necessary for the application, it is recommended to disable the AJP Connector to prevent potential exploitation.

- Review and adjust configurations: Users upgrading to the patched versions of Apache Tomcat may need to make small changes to their configurations to ensure proper security settings.

- Implement defense-in-depth strategies:Users are encouraged to adopt a defense-in-depth approach to security,which involves implementing multiple layers of security controls to protect against various attack vectors.

- Limit access to AJP ports:Ensure that AJP ports are not accessible to untrusted networks or users to reduce the risk of exploitation.

By following these recommendations and best practices,users can effectively mitigate the identified vulnerability in Apache Tomcat and enhance the security of their systems.

The security vulnerability identified as CVE-2020-1938 in Apache Tomcat involves improper privilege management related to the Apache JServ Protocol (AJP) connections. Apache Tomcat treats AJP connections as having higher trust than other types of connections, such as HTTP connections. This can lead to potential exploitation by attackers,allowing them to perform unexpected actions within the web application.

Mitigation for this vulnerability includes applying updates as per the vendor's instructions. Additionally, when using the Apache JServ Protocol(AJP),caution should be exercised in trusting incoming connections



to Apache Tomcat. It is recommended to disable the AJP Connector if not required, especially in versions 7.0.100 or later. Users are advised to follow a defense-in-depth approach and block vectors that allow the processing of arbitrary files.

In versions 9.0.31, 8.5.51 and 7.0.100 or later, changes were made to the default AJP Connector configuration to enhance security. Users upgrading to these versions may need to make minor adjustments to their configurations to ensure a more secure setup.

The CVE-2023-20873:

Vulnerability: Security bypass vulnerability affecting Spring Boot versions 3.0.0-3.0.5, 2.7.0-2.7.10 and older unsupported versions when deployed to Cloud Foundry.

-CVSSv3 Base Score: Critical(9.8)

-Mitigation:

-Users of 3.0.x versions should upgrade to 3.0.6+

-Users of 2.7.x versions should upgrade to 2.7.11+

-Users of older, unsupported versions should upgrade to 3.0.6+2.7.11+.

The CVE-2022-27772:

-Vulnerability: Temporary directory hijacking vulnerability affecting Spring Boot versions prior to 2.5.15.

-CVSSv2 Score: Medium(4.6).

-Mitigation: Upgrade to a non-vulnerable version.

For CVE-2023-20873, users should follow the provided mitigation steps to address the security bypass issue. For CVE-2022-27772, users should upgrade to a version beyond 2.5.25 to mitigate the temporary directory hijacking vulnerability.

The CVE-2022-22965 vulnerability affects VMware Spring Framework and can lead to remote code execution(RCE) via data binding in Spring MVC or Spring WebFlux applications running on JDK 9+.





The vulnerability has a Base Score of High(75) in CVSSV2 and critical(9.8) in CVSSV3. The required action to mitigate these vulnerabilities is to apply updates provided by the vendor. The due date for applying the updates is 2022-04-25. It is important to note that if the application is deployed as a Spring Boot executable jar, it is not vulnerable to this specific exploit.

Security vulnerabilities related to deserialization of untrusted data in the Pivotal Spring Framework through version 5.3.16. The vulnerabilities are classified as follows:

- CVE-2016-1000027:

- CVSSv2 Base Score: HIGH(75)

- Description: Potential security issue within the product, may or may not occur. Authentication mechanisms may be impacted due to reliance on deserialization of trusted data.

- CWE:502-Deserialization of Untrusted Data.

Deserialization of Untrusted Data:

- CVSSv3 Base Score: Critical(98)

- Description: Vulnerability related to Java deserialization of untrusted data. The impact depends on how the library is implemented. The vendor states that untrusted data is not an intended use case, and the product's behavior may not be as expected.

Mitigation for these vulnerabilities would typically involve updating the Pivotal Spring Framework to a version where the identified security issues have been addressed, implementing secure coding practices, and avoiding deserialization of untrusted data whenever possible to prevent potential exploits.

]