# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **May. 25, 2025** | **Harrison Labrecque** | **Evaluating the program for software security vulnerabilities and applying suitable security improvements.** |

## Client



## Developer

Harrison Labrecque

**1. Interpreting Client Needs**

Secure communications are imperative for Artemis Financial, as the company manages highly sensitive client information, encompassing personal data, financial records, investment particulars, and insurance policies. It is critical to maintain the confidentiality, integrity, and availability of this data to preserve client trust and ensure compliance with financial industry regulations. The implementation of secure protocols, such as HTTPS, TLS, and data encryption, across all communication channels is essential to prevent unauthorized access, data breaches, and information leaks. For instance, using encrypted email for client communications or ensuring that all file transfers are done through secure channels. Additionally, secure communication practices serve to underscore the company's commitment to cybersecurity, a vital component of its brand reputation.

As a financial consulting firm catering to a diverse clientele, Artemis Financial is likely to engage in international transactions that necessitate the transfer of financial data across national borders. This situation introduces the need for compliance with global regulations, such as the General Data Protection Regulation (GDPR) in Europe, as well as data residency laws applicable in countries such as Canada and Australia. Non-compliance with these regulations can lead to severe penalties and damage to the company's reputation. Furthermore, the organization must ensure that strong encryption and secure data handling processes are harmonized with governmental restrictions that may impose limitations on the import, export, or strength of cryptographic technologies in specific regions. Artemis Financial needs to ensure its systems comply with these regulations to avoid potential legal and regulatory repercussions.

The modernization of Artemis Financial's software necessitates a forward-thinking approach to technology and security. The company must meticulously evaluate the adoption of open-source libraries, which may enhance efficiency and reduce development time but could also introduce vulnerabilities if inadequately assessed and updated. With the increasing dependence on RESTful APIs, microservices, and cloud-based infrastructures, the new attack surfaces must be fortified using industry best practices, including input validation, rate limiting, and rigorous authentication mechanisms. As web application technologies evolve, Artemis Financial must embrace agile security practices that incorporate continuous monitoring, threat modeling, and automated vulnerability assessments to address emerging cyber threats proactively.

**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.

Based on the vulnerability assessment process flow, several critical security domains are directly relevant to Artemis Financial’s web-based application, given its functionality involving sensitive financial data and client interactions. The initial focus on Architecture Review is not just a step, but a crucial process that ensures a comprehensive analysis of the overall system design of the application. This thorough review enables the identification of structural weaknesses that may be susceptible to exploitation, providing a solid foundation for the rest of the assessment. Given that Artemis Financial’s software operates online and encompasses user data input, Input Validation is imperative to prevent injection attacks, cross-site scripting, and various input-based vulnerabilities.

Advancing through the process flow, Cryptography assumes a crucial role in safeguarding both stored and transmitted financial data. It applies robust encryption protocols, ensuring that data is protected from unauthorized access. Furthermore, the software’s integration with external services and potentially third-party platforms elevates API security as a significant concern. Secure API interactions are essential to avert unauthorized data access or manipulation through system endpoints.

The architecture comprises both client-side and server-side components, underscoring the necessity of securing client/server interactions. This is vital to mitigate risks of session hijacking, data interception, or man-in-the-middle attacks. Additionally, attention to Code Error management and Code Quality is vital for Artemis Financial. The secure handling of errors, alongside adherence to secure coding standards, aids in eliminating bugs that could potentially lead to vulnerabilities.

Finally, processes related to Encapsulation and Code Review are not just additional steps but crucial measures that ensure that data structures are adequately protected. In contrast, key application components, such as data access, services, controllers, and APIs, are thoroughly scrutinized for possible security deficiencies. Collectively, these focus areas establish a comprehensive mitigation strategy to safeguard Artemis Financial’s software against both current and future threats, instilling confidence in the thoroughness of the security measures.

**3. Manual Review**

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

A screen shot of a computer code

AI-generated content may be incorrect.

In the CRUDController.java file, there exists a potential information disclosure vulnerability due to the controller returning the result of doc.toString() directly to the user. If the toString() method of the DocData class reveals sensitive internal state or configuration details, this information could be inadvertently exposed through the /read endpoint. Furthermore, the business\_name parameter from the request is accepted without proper validation or sanitization, which may expose the application to serious risks of injection or data manipulation if this input is utilized elsewhere in the system. This could lead to severe data breaches or system malfunctions. Additionally, the usage of the generic @RequestMapping annotation without specifying an HTTP method can create unintended access, as it permits all HTTP methods by default.

A screen shot of a computer program

AI-generated content may be incorrect.

The primary security concern in the DocData.java file lies in the hardcoded database credentials ("root", "root") found in the read\_document method. This poses a significant risk, especially if the code is exposed or reused in different environments. Furthermore, the SQL connection string is set to use localhost without any encryption or connection pool management, a configuration that is clearly inadequate for production settings. This inadequacy underscores the need for immediate action. Additionally, the method fails to sanitize the key and value inputs, potentially opening the door to SQL injection vulnerabilities, should these values be used in future queries (which are currently unimplemented). Lastly, the absence of robust exception handling, beyond merely printing the stack trace, can result in poor error management.

A screen shot of a computer program

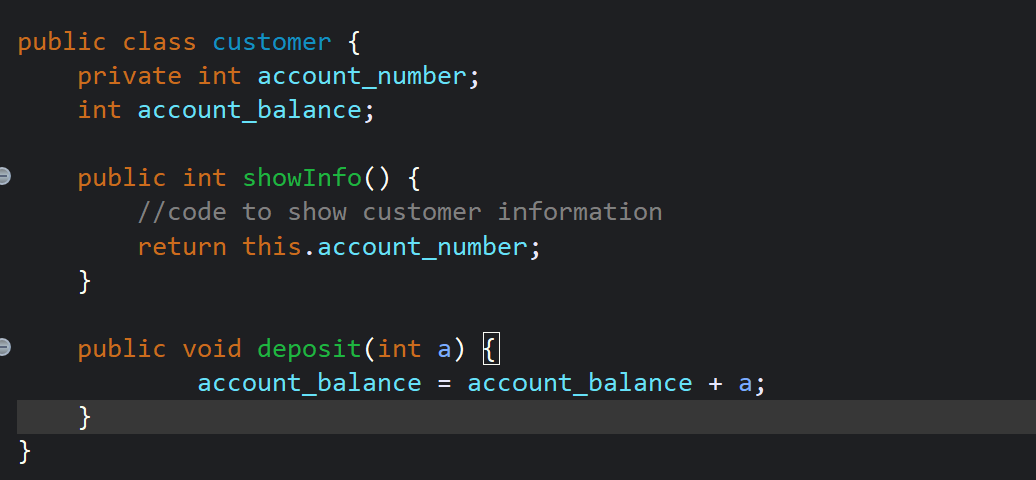
AI-generated content may be incorrect.

The Greeting.java class is designed to be straightforward and well-organized; however, it does present a potential security concern depending on its usage. Although it does not directly interact with user input or external systems, the content field is publicly accessible via the getContent() method without any form of output encoding or sanitization. If this content is set with unsanitized user input (for example, from a request) and subsequently rendered in a web page or API response, it could introduce cross-site scripting (XSS) vulnerabilities in downstream applications. While the class itself does not manage input or output directly, it is essential to ensure that content is sanitized adequately at the controller or service level before being displayed in any HTML context.

A screen shot of a computer code

AI-generated content may be incorrect.

In the GreetingController.java file, a vulnerability is found in the greeting method. The name parameter from the HTTP request is directly embedded into the response using String.format, without any input sanitization or output encoding. This lack of input sanitization or output encoding means that if a user provides malicious input, such as HTML or JavaScript code, it will be included in the content field of the Greeting object. When this content is rendered in a browser without appropriate escaping, it can lead to a reflected cross-site scripting (XSS) attack, potentially enabling attackers to execute arbitrary scripts in the user's browser.



In the `customer.java` class, there are several potential issues that may lead to vulnerabilities or improper behavior. Firstly, the class does not employ proper encapsulation; both `account\_number` and `account\_balance` should ideally be set as private to prevent direct access or modification from outside the class. This would uphold the principles of object-oriented design and help protect sensitive data. Additionally, the `deposit` method lacks validation to ensure that the deposit amount is positive. This oversight allows users to potentially deposit a negative amount or zero, which could result in erroneous behavior within the application. Furthermore, there is no mechanism in place to handle potential overflows when adding to the `account\_balance`, which could pose problems if the balance becomes excessively large. Lastly, while the `showInfo` method returns the `account\_number`, it has the potential to disclose sensitive customer information if not adequately secured.

A screen shot of a computer program

AI-generated content may be incorrect.

In the CRUD.java file, the absence of input validation and sanitization for the content and content2 fields presents potential security vulnerabilities that we need to address immediately. Mainly, if these values originate from user input, without appropriate handling, these fields could expose the application to risks such as Cross-Site Scripting (XSS) or SQL Injection, particularly if the data is incorporated into web responses or database queries. This highlights the earlier emphasis on Input Validation, which is essential for preventing injection attacks and maintaining the integrity of user-supplied data. Furthermore, the redundancy of having two fields (content and content2) that store the same value—without any clear necessity—could complicate future development or debugging efforts, thereby increasing the likelihood of errors or oversights. By addressing these issues through comprehensive input validation and careful data management, we can enhance the application’s security posture, aligning with the broader strategy outlined in the vulnerability assessment process.

A screen shot of a computer program

AI-generated content may be incorrect.

The issues identified in the `myDateTime.java` class underscore the critical importance of implementing robust input validation and maintaining data integrity, as emphasized in previous discussions. Failing to validate the input for seconds, minutes, and hours in the `setMyDateTime` method may allow invalid data to infiltrate the system, potentially leading to unexpected behavior or logical errors—issues that could jeopardize the application's functionality. This connects to the broader theme of input validation within the vulnerability assessment process, which is vital for preventing logical flaws and ensuring that user- or system-generated data remains consistent and trustworthy. The significance of maintaining data integrity cannot be overstated, as it is the cornerstone of a reliable and secure application. Furthermore, the lack of encapsulation for the time-related fields highlights the necessity of securing internal data structures to prevent unauthorized access and manipulation, echoing earlier points made about encapsulation and code review. Collectively, these concerns emphasize the need for a comprehensive approach to securing both the logical components and data structures of the application.

**4. Static Testing**

**bcprov-jdk15on-1.46.jar**

**Vulnerability IDs:  
CVE-2023-33202 – Potential Denial of Service (DoS) via PEMParser class due to crafted ASN.1 data causing OutOfMemoryError.  
CVE-2016-1000338 – Improper validation of ASN.1 signature encoding in DSA, allowing injection of extra elements into the signature sequence.  
  
Severity: High**

**Affected Versions: 1.46**

**Documentation: These vulnerabilities were reported in the National Vulnerability Database (NVD) and impact cryptographic data handling, specifically parsing and signature verification logic. For more information, please refer to the reference section.**

**Recommendations: Upgrade to version 1.47 or later. Ensure cryptographic operations comply with current standards, and avoid using outdated parsing and signature verification methods.**

**hibernate-validator-6.0.18.Final.jar**

**Vulnerability IDs:**

**CVE-2020-10693 – A flaw in the message interpolation processor enables invalid EL expressions to be evaluated as if valid. This allows attackers to bypass input sanitation controls, potentially leading to exploitation of user-controlled data.**

**Severity: Medium**

**Affected Versions: 6.0.18**

**Documentation: This vulnerability allows attackers to bypass input sanitization mechanisms in error messages, which may lead to the execution of unintended commands or scripts. For more details, refer to the reference section of this document.**

**Recommendations: Upgrade to version 6.0.19 or higher. Ensure that the validation mechanism in Hibernate is protected against deserialization vulnerabilities, and implement proper sanitation of user-controlled data to prevent remote code execution or other exploits.**

**jackson-databind-2.10.2.jar**

**Vulnerability IDs:**

**CVE-2023-35116 – Jackson-databind through 2.15.2 allows attackers to cause a denial of service (DoS) or other unspecified impacts via crafted objects with cyclic dependencies. Vendor’s perspective: The vulnerability cannot be triggered by external attackers due to the required steps for creating a cyclic structure.**

**CVE-2021-46877 – Jackson-databind 2.10.x through 2.12.x (before 2.12.6) and 2.13.x (before 2.13.1) allows attackers to cause a denial of service (DoS) with 2 GB heap usage per read, affecting JsonNode JDK serialization.**

**CVE-2022-42004 – Jackson-databind before 2.13.4 allows resource exhaustion due to deep nested array deserialization in customized deserialization scenarios.**

**CVE-2022-42003 – Jackson-databind before 2.13.4.1 and 2.12.17.1 is vulnerable to resource exhaustion from deep wrapper array nesting when the UNWRAP\_SINGLE\_VALUE\_ARRAYS feature is enabled.**

**CVE-2020-36518 – Jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service due to a large depth of nested objects.**

**CVE-2020-25649 – Jackson-databind lacked proper entity expansion security, making it vulnerable to XML External Entity (XXE) attacks, with the highest threat being data integrity.**

**Severity: High**

**Affected Versions: 2.10.2**

**Documentation: These vulnerabilities affect Jackson-databind and expose systems to potential denial-of-service attacks, resource exhaustion, and security risks. For further details on these vulnerabilities, refer to the reference section.**

**Recommendations: Upgrade to version 2.10.3 or a newer patched version to mitigate risks from remote code execution (RCE) or denial-of-service (DoS) attacks stemming from unsafe deserialization mechanisms.**

**log4j-api-2.12.1.jar**

**Vulnerability IDs:**

**CVE-2021-44832: Apache Log4j2 versions 2.0-beta7 through 2.17.0 are vulnerable to remote code execution (RCE) via a JDBC Appender with a JNDI LDAP data source URI when an attacker controls the target LDAP server.**

**CVE-2021-45105: Apache Log4j2 versions 2.0-alpha1 through 2.16.0 are affected by uncontrolled recursion from self-referential lookups, which can lead to a denial of service (DoS) attack.**

**CVE-2021-45046: A flaw in Apache Log4j2 where the fix for CVE-2021-44228 was incomplete, allowing remote code execution in non-default configurations with JNDI Lookups.**

**CVE-2021-44228: A critical vulnerability in Apache Log4j2 2.0-beta9 through 2.15.0 allows remote code execution (RCE) through JNDI features when an attacker can control log messages or parameters.**

**CVE-2020-9488: An improper certificate validation in Apache Log4j SMTP appender allows an attacker to intercept an SMTPS connection, resulting in potential information leakage of log messages.**

**Severity: Low**

**Affected Versions: 2.12.1**

**Documentation:  
 These vulnerabilities affect Log4j versions 2.12.1, exposing systems to potential risks such as remote code execution (RCE), denial of service (DoS), and other issues like information leaks. These vulnerabilities stem from JNDI-related attacks, uncontrolled recursion, and improper validation of certificates in the affected versions.  For further details on these vulnerabilities, refer to the reference section.**

**Recommendations:  
 While the severity is low, it is highly recommended to update to a later version of Log4j to avoid potential issues with future vulnerabilities and ensure adherence to the latest security practices. Consider upgrading to the most recent versions where critical security fixes have been implemented, particularly concerning JNDI vulnerabilities and improper certificate validation.**

**logback-classic-1.2.3.jar**

**Vulnerability IDs:**

**CVE-2023-6378: A serialization vulnerability in the Logback receiver component allows attackers to mount a Denial-Of-Service (DoS) attack by sending poisoned data.**

**CVE-2021-42550: In Logback version 1.2.7 and prior, an attacker with privileges to edit configuration files could execute arbitrary code loaded from LDAP servers by crafting a malicious configuration.**

**Severity: High**

**Affected Versions: 1.2.3**

**Documentation:  
 These vulnerabilities expose Logback 1.2.3 to significant risks, including potential denial-of-service (DoS) attacks via serialized data poisoning and the possibility of remote code execution through malicious LDAP configurations.  For further details on these vulnerabilities, refer to the reference section.**

**Recommendations:  
 Upgrade to version 1.2.4 or higher to mitigate the known vulnerabilities, preventing the risk of DoS attacks and unauthorized code execution through malicious configuration files. Upgrading to the latest patched version will help maintain the integrity and security of the logging system.**

**logback-core-1.2.3.jar  
 Vulnerability IDs:  
 CVE-2023-6378: A serialization vulnerability in the Logback receiver component allows attackers to mount a Denial-Of-Service (DoS) attack by sending poisoned data.**

**CVE-2021-42550: In Logback version 1.2.7 and prior, an attacker with privileges to edit configuration files could execute arbitrary code loaded from LDAP servers by crafting a malicious configuration.**

**Severity: High  
 Affected Versions: 1.2.3**

**Documentation:  
 These vulnerabilities expose Logback 1.2.3 to critical risks, including the potential for denial-of-service (DoS) attacks through malicious serialized input and the threat of remote code execution by exploiting insecure configuration via LDAP. These risks compromise both the availability and integrity of applications using the affected library. For further details on these vulnerabilities, refer to the reference section.**

**Recommendations:  
 Upgrade to version 1.2.4 or higher to mitigate the identified vulnerabilities. This prevents exploitation through serialized data and unsafe configuration loading. Upgrading to the latest available version is strongly recommended to ensure full protection and maintain a secure logging infrastructure.**

**snakeyaml-1.25.jar  
 Vulnerability IDs:**

**CVE-2022-1471: SnakeYaml’s Constructor() class does not restrict which types can be instantiated during deserialization. Deserializing attacker-controlled YAML content can lead to Remote Code Execution (RCE).**

**CVE-2022-41854: Parsing untrusted YAML with SnakeYaml can result in Denial-of-Service (DoS) due to stack overflow from crafted input.**

**CVE-2022-38752: DoS vulnerability via stack overflow when parsing user-supplied YAML.**

**CVE-2022-38751: DoS vulnerability via stack overflow due to recursive YAML structures.**

**CVE-2022-38750: DoS vulnerability from stack overflow via maliciously crafted YAML content.**

**CVE-2022-38749: Similar stack overflow-based DoS vulnerability via recursive YAML.**

**CVE-2022-25857: DoS vulnerability caused by lack of depth limitations for nested collections.**

**CVE-2017-18640: The alias feature before version 1.26 allows entity expansion during loading, potentially leading to billion laughs-style DoS attacks.**

**Severity: Critical  
 Affected Versions: 1.25**

**Documentation:  
 SnakeYaml 1.25 is affected by critical security flaws including remote code execution and several forms of denial-of-service (DoS) attacks. These issues arise from unrestricted deserialization, lack of input depth limitations, and improper handling of recursive or aliased structures. Applications parsing untrusted YAML using this version are at high risk. It is particularly vulnerable to exploits that crash the parser or execute arbitrary code. For further details, refer to the listed CVEs in the reference section.**

**Recommendations:  
 Upgrade to version 1.26 or later, though version 2.0+ is strongly recommended for enhanced security features, including safe constructors and improved deserialization safeguards. If parsing untrusted input, always use SafeConstructor to restrict the classes that can be deserialized and prevent RCE. Applying these mitigations ensures YAML parsing remains secure and reliable across applications.**

**spring-boot-2.2.4.RELEASE.jar  
 Vulnerability IDs:**

**CVE-2023-20883: Denial-of-Service (DoS) attack possible in Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older if Spring MVC is used with a reverse proxy cache.**

**CVE-2023-20873: Security bypass vulnerability affecting Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, especially when deployed on Cloud Foundry.**

**CVE-2022-27772: Temporary directory hijacking vulnerability in versions prior to 2.2.11.RELEASE, affecting the AbstractConfigurableWebServerFactory.createTempDir method.**

**Severity: Critical  
 Affected Versions: 2.2.4**

**Documentation:  
 Spring Boot version 2.2.4 is affected by multiple critical vulnerabilities that expose applications to denial-of-service attacks, security bypass, and temporary directory hijacking. These vulnerabilities could allow attackers to disrupt services or escalate privileges, impacting application security and availability. Since this version is no longer supported, it is essential to upgrade to maintain protection.**

**Recommendations:  
 Upgrade to Spring Boot version 2.2.5 or later, ideally the latest stable version, to remediate these critical security issues and ensure ongoing support and patching.**

**spring-boot-starter-web-2.2.4.RELEASE.jar  
 Vulnerability IDs:**

**CVE-2023-20883: Allows denial-of-service (DoS) attacks when Spring MVC is used with a reverse proxy cache.**

**CVE-2023-20873: Enables security bypass in applications deployed to Cloud Foundry.**

**CVE-2022-27772: Vulnerable to temporary directory hijacking that may lead to privilege escalation.**

**Severity: Critical  
 Affected Versions: 2.2.4**

**Documentation:  
 This version of Spring Boot Starter Web is vulnerable to several critical security issues. These vulnerabilities expose the application to denial-of-service conditions, security bypasses, and privilege escalation risks. Since 2.2.4 is no longer supported, timely upgrades are crucial to maintaining security and stability.**

**Recommendations:  
 Upgrade to version 2.2.5 or later to resolve these issues and protect your application against DoS attacks, security bypass, and temporary directory hijacking vulnerabilities.**

**spring-context-5.2.3.RELEASE.jar**

**Vulnerability IDs:**

**CVE-2023-20863: Allows denial-of-service (DoS) attacks via specially crafted Spring Expression Language (SpEL) expressions.**

**CVE-2023-20861: Enables denial-of-service (DoS) conditions through malicious SpEL expressions in multiple Spring Framework versions.**

**CVE-2022-22971: Authenticated users can cause denial-of-service (DoS) attacks on STOMP over WebSocket endpoints.**

**CVE-2022-22970: Denial-of-service (DoS) vulnerabilities occur during file uploads when relying on data binding for MultipartFile or javax.servlet.Part fields.**

**CVE-2022-22968: Case sensitivity issue in disallowedFields patterns that may fail to protect some fields from unauthorized binding.**

**CVE-2022-22965: Remote code execution (RCE) vulnerability in Spring MVC or WebFlux on Tomcat when deployed as WAR files via unsafe data binding.**

**CVE-2022-22950: Denial-of-service (DoS) attacks via malicious SpEL expressions in older Spring Framework versions.**

**CVE-2021-22060: Malicious input can cause insertion of additional log entries, leading to potential log injection attacks.**

**CVE-2021-22096: Similar to CVE-2021-22060, allows insertion of extra log entries by supplying crafted input.**

**CVE-2021-22118: Privilege escalation vulnerability by recreating temporary storage directories, allowing file overwrite or unauthorized read access.**

**CVE-2020-5421: Remote File Disclosure (RFD) attack via jsessionid path parameter bypassing protections in older Spring versions.**

**CVE-2016-1000027: Potential remote code execution (RCE) through Java deserialization of untrusted data in certain Spring Framework configurations.**

**Severity: Critical**

**Affected Versions: 5.2.3**

**Documentation:  
 Spring Context version 5.2.3 contains several critical and medium severity vulnerabilities that expose applications to denial-of-service (DoS) attacks via specially crafted SpEL expressions, privilege escalation risks, remote code execution (RCE) in certain deployment scenarios, and other security issues affecting session management, file upload handling, and logging. These vulnerabilities may allow attackers to disrupt services, bypass security controls, or expose sensitive data.**

**Recommendations:  
 Upgrade to version 5.2.4 or higher to fix these security issues and improve overall application security posture.**

**spring-core-5.2.3.RELEASE.jar**

**Vulnerability IDs:**

**CVE-2023-20861: Allows denial-of-service (DoS) attacks using crafted Spring Expression Language (SpEL) expressions.**

**CVE-2022-22971: Authenticated users can cause DoS attacks on STOMP over WebSocket endpoints.**

**CVE-2022-22970: DoS vulnerability via data binding on file uploads with MultipartFile or javax.servlet.Part.**

**CVE-2022-22968: Case sensitivity flaw in disallowedFields causes ineffective data binding protection.**

**CVE-2022-22965: Remote code execution (RCE) via unsafe data binding in Spring MVC/WebFlux on Tomcat WAR deployments.**

**CVE-2022-22950: DoS attacks via malicious SpEL expressions in older Spring Framework versions.**

**CVE-2021-22060: Malicious input can inject additional log entries, leading to log injection vulnerabilities.**

**CVE-2021-22096: Similar log injection issue allowing crafted input to add log entries.**

**CVE-2021-22118: Privilege escalation by recreating temporary storage directories, allowing unauthorized file access or overwrite.**

**CVE-2020-5421: Remote File Disclosure (RFD) via jsessionid path parameter bypassing older protections.**

**CVE-2016-1000027: Potential remote code execution from Java deserialization of untrusted data.**

**Severity: Critical**

**Affected Versions: 5.2.3**

**Documentation: Multiple vulnerabilities affecting Spring Core 5.2.3 have been documented and detailed in various CVE entries published by the National Vulnerability Database (NVD). These reports provide in-depth technical descriptions and impact assessments of issues such as denial-of-service attacks, remote code execution, privilege escalation, and log injection.**

**Recommendations: Update to the latest Spring Core version (5.2.4 or higher) to address multiple vulnerabilities involving denial-of-service attacks, remote code execution, privilege escalation, and log injection.**

**spring-expression-5.2.3.RELEASE.jar**

**Vulnerability IDs:**

**CVE-2023-20863: Allows denial-of-service (DoS) attacks via specially crafted Spring Expression Language (SpEL) expressions in vulnerable Spring versions.**

**CVE-2023-20861: Enables denial-of-service (DoS) attacks through malicious SpEL expressions in certain Spring Framework releases.**

**CVE-2022-22971: Authenticated users can trigger a denial-of-service attack on STOMP over WebSocket endpoints.**

**CVE-2022-22970: File upload handling is vulnerable to denial-of-service attacks when using data binding on MultipartFile or javax.servlet.Part.**

**CVE-2022-22968: Case sensitivity in disallowedFields patterns causes incomplete protection in data binding.**

**CVE-2022-22965: Remote code execution (RCE) vulnerability in Spring MVC/WebFlux applications deployed as WAR files on Tomcat via unsafe data binding.**

**CVE-2022-22950: Denial-of-service (DoS) attacks possible through crafted SpEL expressions in older Spring versions.**

**CVE-2021-22060: Malicious input may cause injection of extra log entries, leading to log injection vulnerabilities.**

**CVE-2021-22096: Similar log injection vulnerability caused by malicious input affecting multiple parts of the Spring Framework.**

**CVE-2021-22118: Privilege escalation via recreation of temporary storage directories allows unauthorized file access or overwriting.**

**CVE-2020-5421: Remote File Disclosure (RFD) vulnerability through jsessionid path parameter bypassing existing protections.**

**CVE-2016-1000027: Potential remote code execution due to Java deserialization of untrusted data under certain conditions.**

**Severity: Critical  
 Affected Versions: 5.2.3**

**Documentation:  
 This version of spring-expression contains multiple critical security vulnerabilities, including denial-of-service, remote code execution, privilege escalation, and log injection. These vulnerabilities stem primarily from unsafe Spring Expression Language (SpEL) parsing, improper data binding, and inadequate protection of internal application components, affecting various Spring Framework versions including 5.2.3.**

**Recommendations:  
 Upgrade to a newer version to mitigate potential vulnerabilities in the expression language parser, which could lead to arbitrary code execution and elevate privilege exploitation.**

**spring-web-5.2.3.RELEASE.jar  
 Vulnerability IDs:**

* **CVE-2023-20863 – Denial-of-service (DoS) via specially crafted SpEL expression.**
* **CVE-2023-20861 – DoS vulnerability through crafted SpEL expression.**
* **CVE-2022-22971 – DoS via STOMP over WebSocket by an authenticated user.**
* **CVE-2022-22970 – DoS through file upload handling with model binding.**
* **CVE-2022-22968 – Case sensitivity flaw in disallowedFields configuration can lead to insecure data binding.**
* **CVE-2022-22965 – Critical remote code execution (RCE) vulnerability known as Spring4Shell, exploitable via WAR deployment on Tomcat.**
* **CVE-2022-22950 – DoS via malicious SpEL expression input.**
* **CVE-2021-22060 – Log injection vulnerability via crafted user input.**
* **CVE-2021-22096 – Additional log injection vectors not covered by previous fixes.**
* **CVE-2021-22118 – Privilege escalation through overwriting files in WebFlux multipart upload.**
* **CVE-2020-5421 – Reflected File Download (RFD) bypass using the jsessionid path parameter.**
* **CVE-2016-1000027 – Potential remote code execution via Java deserialization of untrusted data.**

**Severity: Critical  
 Affected Versions: 5.2.3**

**Documentation:  
All the vulnerabilities mentioned are documented in the National Vulnerability Database (NVD). These include denial-of-service attacks, insecure data binding, remote code execution, log injection, privilege escalation, and the bypassing of security filters. The NVD offers technical details, severity ratings, and recommended mitigation steps for each issue.**

**Recommendations:  
 Update to Spring Web version 5.2.4 or higher to mitigate issues related to broken authentication, insecure direct object references, denial-of-service (DoS), remote code execution (RCE), and other security flaws that could lead to unauthorized access or resource exposure.**

**spring-webmvc-5.2.3.RELEASE.jar  
 Vulnerability IDs:**

**CVE-2023-20863 – Denial-of-service (DoS) via crafted SpEL expressions causing server overload.**

**CVE-2023-20861 – DoS through specially crafted SpEL expressions.**

**CVE-2022-22971 – DoS attack on STOMP over WebSocket endpoint by authenticated user.**

**CVE-2022-22970 – DoS vulnerability in file upload handling using data binding.**

**CVE-2022-22968 – Case sensitivity issue in disallowedFields causing ineffective field protection.**

**CVE-2022-22965 – Critical remote code execution (RCE) vulnerability via data binding in WAR deployments.**

**CVE-2022-22950 – DoS caused by crafted SpEL expressions.**

**CVE-2021-22060 – Log injection allowing malicious input to add extra log entries.**

**CVE-2021-22096 – Additional log injection vulnerabilities.**

**CVE-2021-22118 – Privilege escalation via multipart file upload temporary storage exploitation.**

**CVE-2020-5421 – Reflected File Download (RFD) bypass using jsessionid path parameter.**

**CVE-2016-1000027 – Potential RCE via Java deserialization of untrusted data.**

**Severity: Critical  
 Affected Versions: 5.2.3**

**Documentation:  
 These vulnerabilities expose the application to denial-of-service attacks, remote code execution, privilege escalation, log injection, and security bypass risks primarily due to improper input validation, unsafe SpEL expression handling, and insecure multipart file processing. Exploiting these issues can lead to server crashes, unauthorized data access, and system compromise. Full technical details, impact ratings, and mitigation steps are documented in the National Vulnerability Database (NVD).**

**Recommendations:  
 Upgrade to the latest patch release of Spring WebMVC to avoid vulnerabilities in request handling, improper input validation, and resource exposure that could facilitate unauthorized access.**

**tomcat-embed-core-9.0.30.jar**

**Vulnerability IDs:**

**CVE-2025-31651: Improper Neutralization of Escape, Meta, or Control Sequences. Rewrite rules could be bypassed, potentially circumventing security constraints.**

**CVE-2025-24813: Path Equivalence vulnerability leading to Remote Code Execution and/or Information Disclosure via Default Servlet with writes enabled.**

**CVE-2024-52316: Unchecked error condition in Jakarta Authentication could allow authentication bypass.**

**CVE-2024-38286: Allocation of Resources Without Limits, leading to OutOfMemoryError via TLS handshake abuse.**

**CVE-2024-24549: HTTP/2 request header limit mismanagement causing denial of service.**

**CVE-2024-23672: Incomplete cleanup on WebSocket connections causing resource exhaustion (DoS).**

**CVE-2024-21733: Error messages containing sensitive information (information disclosure).**

**CVE-2023-46589: Improper input validation leading to request smuggling via HTTP trailer headers.**

**CVE-2023-45648: Improper input validation causing request smuggling via invalid trailer headers.**

**CVE-2023-42795: Incomplete cleanup causing information leakage between requests.**

**CVE-2023-44487: HTTP/2 stream resets causing denial of service (resource consumption).**

**CVE-2023-41080: Open redirect vulnerability in FORM authentication’s default web app.**

**CVE-2023-28708: Missing Secure attribute on session cookies via RemoteIpFilter and reverse proxy.**

**CVE-2022-42252: Request smuggling possible due to invalid Content-Length headers.**

**CVE-2021-43980: Concurrency bug causing responses to be sent to wrong client.**

**CVE-2022-34305: Cross-site scripting (XSS) in Form authentication example app.**

**CVE-2022-29885: Incorrect documentation on EncryptInterceptor’s network security.**

**CVE-2021-41079: Denial of service via malformed TLS packets with OpenSSL.**

**CVE-2021-33037: HTTP transfer-encoding header parsing flaw enabling request smuggling.**

**CVE-2021-30640: Authentication bypass and LockOut Realm protection weakening in JNDI Realm.**

**Severity: Critical**

**Affected Versions: Apache Tomcat 9.0.30**

**Documentation:**

**The security issues in Tomcat Embed Core version 9.0.30 include vulnerabilities that could lead to information disclosure, session hijacking, remote code execution, and authentication bypass. Additional flaws make the system susceptible to denial of service attacks, request smuggling, and cross-site scripting, all of which pose risks to server stability and client security. Furthermore, there are concerns regarding open redirects, insecure cookie handling, and concurrency issues, which may result in data leaks and mixed-up responses. Additional technical information can be found in the NIST National Vulnerability Database (NVD).**

**Recommendations:  
 Upgrade to Apache Tomcat version 9.0.31 or higher. This update contains critical fixes addressing information disclosure, session hijacking, remote code execution, denial of service, and other serious security flaws that could compromise confidentiality, integrity, and availability of your applications.**

**tomcat-embed-websocket-9.0.30.jar  
 Vulnerability IDs:**

* **CVE-2025-31651: Bypass of rewrite rules leading to potential security constraint bypass.**
* **CVE-2025-24813: Path equivalence flaw causing remote code execution and information disclosure via file uploads.**
* **CVE-2024-52316: Authentication bypass via unchecked error conditions in custom Jakarta Authentication components.**
* **CVE-2024-38286: Resource allocation without limits, enabling OutOfMemoryError via TLS handshake abuse.**
* **CVE-2024-24549: HTTP/2 improper input validation causing denial of service.**
* **CVE-2024-23672: WebSocket connections not cleaned up correctly, leading to resource exhaustion DoS.**
* **CVE-2024-21733: Sensitive information leakage through error messages.**
* **CVE-2023-46589: Improper HTTP trailer header parsing, enabling request smuggling.**
* **CVE-2023-45648: Invalid trailer headers causing potential request smuggling attacks.**
* **CVE-2023-42795: Incomplete recycling of internal objects leaking information between requests.**
* **CVE-2023-44487: HTTP/2 request cancellation exploited for denial of service.**
* **CVE-2023-41080: Open redirect vulnerability in FORM authentication.**
* **CVE-2023-28708: Missing secure attribute on cookies behind reverse proxies.**
* **CVE-2022-42252: HTTP header validation bypass enabling request smuggling.**
* **CVE-2021-43980: Concurrency bug causing response mix-up between clients.**
* **CVE-2022-34305: Reflected XSS vulnerability in Form authentication example.**
* **CVE-2022-29885: Documentation error about EncryptInterceptor and network security.**
* **CVE-2021-41079: TLS packet validation flaw causing infinite loop DoS.**
* **CVE-2021-33037: HTTP transfer-encoding header parsing issue enabling request smuggling.**
* **CVE-2021-30640: JNDI Realm authentication bypass and LockOut Realm protection weakness.**

**Severity: Critical**

**Affected Versions: 9.0.30**

**Documentation:  
 Security vulnerabilities affecting the Tomcat WebSocket component include issues that may lead to denial of service (DoS), man-in-the-middle (MitM) attacks, and improper handling of protocol frames. These vulnerabilities could compromise communication confidentiality and availability. Detailed information and mitigation steps are available through Apache Tomcat Security Advisories and the NIST National Vulnerability Database (NVD).**

**Recommendations:  
 Ensure Tomcat is updated to the latest version to address vulnerabilities in WebSocket protocol handling, which could be exploited to cause DoS or man-in-the-middle (MitM) attacks.**

**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.

The manual code review and static testing conducted on Artemis Financial's software have uncovered several critical and high-risk vulnerabilities that demand immediate attention. The manual review identified issues such as unvalidated input parameters, hardcoded database credentials, and inadequate error handling, particularly within classes such as CRUDController, DocData, and GreetingController. These vulnerabilities significantly increase the application’s susceptibility to injection attacks, sensitive information disclosure, and improper input handling. For instance, the DocData class connects to the database without employing parameterized queries or input sanitization, making it highly vulnerable to SQL injection attacks. Furthermore, the exposure of business logic through unsecured REST endpoints indicates a significant lack of proper encapsulation and access control.

The static analysis, which utilized the OWASP Dependency-Check tool (version 12.1.0), uncovered a total of 175 vulnerabilities across 16 outdated and insecure dependencies. Noteworthy libraries, including bcprov-jdk15on, jackson-databind, logback, and various Spring components, were found to harbor multiple Common Vulnerabilities and Exposures (CVEs), some of which are classified as critical. These vulnerabilities encompass deserialization issues, risks of remote code execution, and compromised authentication mechanisms. Neglecting to address these outdated dependencies may lead to severe exploitation scenarios, including privilege escalation or complete system compromise. The root cause of these issues stems from the usage of obsolete third-party libraries without appropriate version control or security vetting, as evidenced in the Maven POM file.

To effectively address the identified vulnerabilities, it is crucial to implement a comprehensive mitigation plan. This plan should include the relocation of all hardcoded secrets to secure configuration files or environment variables using tools such as Spring Vault. It should also enforce rigorous input validation through robust validation libraries or regular expressions, ensuring that user-supplied data is sanitized before processing. Furthermore, the plan should prioritize the update of every vulnerable library to its latest secure version, accompanied by regression testing to verify compatibility. The adoption of secure coding practices—such as centralized error handling, cautious logging, and dependency scanning—should become standard within the development lifecycle. Establishing a DevSecOps pipeline will significantly enhance the assurance that future code modifications are automatically scanned for both dependency and logical vulnerabilities prior to deployment.

**6. References**

National Institute of Standards and Technology. (2023, November 23). *CVE-2023-33202.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-33202>

National Institute of Standards and Technology. (2018, June 1). *CVE-2016-1000338.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2016-1000338>

National Institute of Standards and Technology. (2020, May 6). *CVE-2020-10693.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2020-10693>

National Institute of Standards and Technology. (2023, June 14). *CVE-2023-35116.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-35116>

National Institute of Standards and Technology. (2023, March 18). *CVE-2021-46877.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-46877>

National Institute of Standards and Technology. (2022, October 2). *CVE-2022-42004.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-42004>

National Institute of Standards and Technology. (2022, October 2). *CVE-2022-42003.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-42003>

National Institute of Standards and Technology. (2022, March 11). *CVE-2020-36518.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2020-36518>

National Institute of Standards and Technology. (2020, December 3). *CVE-2020-25649.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2020-25649>

National Institute of Standards and Technology. (2021, December 28). *CVE-2021-44832.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-44832>

National Institute of Standards and Technology. (2021, December 18). *CVE-2021-45105.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-45105>

National Institute of Standards and Technology. (2021, December 14). *CVE-2021-45046.* National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-45046>

National Institute of Standards and Technology. (2021, December 10). *CVE-2021-44228.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>

National Institute of Standards and Technology. (2020, April 27). *CVE-2020-9488.*National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2020-9488>

National Institute of Standards and Technology. (2023, November 29). *CVE-2023-6378*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-6378>

National Institute of Standards and Technology. (2021, December 16). *CVE-2021-42550*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-42550>

National Institute of Standards and Technology. (2022, December 1). CVE-2022-1471. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-1471>

National Institute of Standards and Technology. (2022, November 11). CVE-2022-41854. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-41854>

National Institute of Standards and Technology. (2022, September 5). CVE-2022-38752. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-38752>

National Institute of Standards and Technology. (2022, September 5). CVE-2022-38751. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-38751>

National Institute of Standards and Technology. (2022, September 5). CVE-2022-38750. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-38750>

National Institute of Standards and Technology. (2022, September 5). CVE-2022-38749. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-38749>

National Institute of Standards and Technology. (2022, August 30). CVE-2022-25857. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-25857>

National Institute of Standards and Technology. (2019, December 11). CVE-2017-18640. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2017-18640>

National Institute of Standards and Technology. (2023, May 26). *CVE-2023-20883*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20883>

National Institute of Standards and Technology. (2023, April 20). *CVE-2023-20873*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20873>

National Institute of Standards and Technology. (2022, March 30). *CVE-2022-27772*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-27772>

National Institute of Standards and Technology. (2023, May 26). *CVE-2023-20883*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20883>

National Institute of Standards and Technology. (2023, April 20). *CVE-2023-20873*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20873>

National Institute of Standards and Technology. (2022, March 30). *CVE-2022-27772*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-27772>

National Institute of Standards and Technology. (2023, April 13). *CVE-2023-20863*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20863>

National Institute of Standards and Technology. (2023, March 23). *CVE-2023-20861*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-20861>

National Institute of Standards and Technology. (2022, May 12). *CVE-2022-22971*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-22971>

National Institute of Standards and Technology. (2022, May 12). *CVE-2022-22970*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-22970>

National Institute of Standards and Technology. (2022, April 14). *CVE-2022-22968*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-22968>

National Institute of Standards and Technology. (2022, April 1). *CVE-2022-22965*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-22965>

National Institute of Standards and Technology. (2022, April 1). *CVE-2022-22950*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-22950>

National Institute of Standards and Technology. (2022, January 10). *CVE-2021-22060*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-22060>

National Institute of Standards and Technology. (2021, October 28). *CVE-2021-22096*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-22096>

National Institute of Standards and Technology. (2021, May 27). *CVE-2021-22118*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-22118>

National Institute of Standards and Technology. (2020, September 19). *CVE-2020-5421*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2020-5421>

National Institute of Standards and Technology. (2020, January 2). *CVE-2016-1000027*. National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2016-1000027>

National Institute of Standards and Technology. (2025, April 28). *CVE-2025-31651 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2025-31651>

National Institute of Standards and Technology. (2025, March 10). *CVE-2025-24813 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2025-24813>

National Institute of Standards and Technology. (2024, November 18). *CVE-2024-52316 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2024-52316>

National Institute of Standards and Technology. (2024, November 7). *CVE-2024-38286 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2024-38286>

National Institute of Standards and Technology. (2024, March 13). *CVE-2024-24549 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2024-24549>

National Institute of Standards and Technology. (2024, March 13). *CVE-2024-23672 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2024-23672>

National Institute of Standards and Technology. (2024, January 19). *CVE-2024-21733 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2024-21733>

National Institute of Standards and Technology. (2023, November 28). *CVE-2023-46589 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-46589>

National Institute of Standards and Technology. (2023, October 10). *CVE-2023-45648 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-45648>

National Institute of Standards and Technology. (2023, October 10). *CVE-2023-42795 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-42795>

National Institute of Standards and Technology. (2023, October 10). *CVE-2023-44487 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-44487>

National Institute of Standards and Technology. (2023, August 25). *CVE-2023-41080 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-41080>

National Institute of Standards and Technology. (2023, March 22). *CVE-2023-28708 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2023-28708>

National Institute of Standards and Technology. (2022, November 1). *CVE-2022-42252 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-42252>

National Institute of Standards and Technology. (2022, September 28). *CVE-2021-43980 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-43980>

National Institute of Standards and Technology. (2022, June 23). *CVE-2022-34305 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-34305>

National Institute of Standards and Technology. (2022, May 12). *CVE-2022-29885 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2022-29885>

National Institute of Standards and Technology. (2021, September 16). *CVE-2021-41079 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-41079>

National Institute of Standards and Technology. (2021, July 12). *CVE-2021-33037 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-33037>

National Institute of Standards and Technology. (2021, July 12). *CVE-2021-30640 Detail*. NVD - National Vulnerability Database. <https://nvd.nist.gov/vuln/detail/CVE-2021-30640>