# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11/15/2024** | **Rikka Eng** |  |

## 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

Rikka Eng

**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:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

**Intro:** Alright, so I work at Global Rain. An outsourced software company hired by Artemis Fowl to modernize their features and business. Artemis Fowl works in entrepreneurs, businesses, and government agencies around the world. A part of Artemis Fowl’s motive is their huge desire for security, particularly focusing on RESTful Web applications and APIs.

**Value of Secure Communication:** Secure communications are critical to maintain customer trust and ensuring the confidentiality and integrity of financial data. Considering the nature of Artemis Financial business operations, from other businesses to government agencies. Artemis deals with extremely sensitive information, and any compromise could result in regulatory penalties, loss of business reputation, or even lawsuits.

Rajputg0n66. (2024, July 8). Secure communication in distributed system. GeeksforGeeks. <https://www.geeksforgeeks.org/secure-communication-in-distributed-system/>

**International transactions:** Given the context, we are not explicitly sure whether the company is involved in international transactions. However, due to the nature of their specialization in RESTful Web APIs, it is plausible that they would, as such security measures are necessary for nationwide communications. They could be partnered with foreign banks, financial institutions, or even a foreign business for transactions like savings, retirement plans, or insurance. If they do engage in such affairs, they will have to take into consideration several factors depending on the foreign region:

* **GDPR:** The General Data Protection Regulation is a legal thingy pushed by the UK that affects any organization handling personal data of EU citizens, regardless of what planet, universe, or even country the other entity resides. This includes Artimis Financial, and they will have to comply with security related measure when dealing with sensitive information that may regard EU business, insurance, or retirement plans.

Wolford Ben. (n.d.) What is the GDPR? Europe’s new data privacy and security law includes hundreds of pages’ worth of new requirements for organizations around the world. This GDPR overview will help you understand the law and determine what parts of it apply to you.From <https://gdpr.eu/what-is-gdpr/>

* **Multi-Currency:** Due to currency conversions. The company may expose some vulnerabilities:
  + When converting currencies, APIs often interact with third**-**party services or financial systems. Without proper encryption, this communication could expose sensitive data to potential data thieves.
  + Multi-currency transactions complicate fraud detection because of varying regulator frameworks across countries.
  + Different regions have strict requirements for currency conversion, this is evident by the Payment Services Directive 2(PSD2) law in the UK.

Open Web Application Security Project (OWASP). (n.d.). REST Security Cheat Sheet. From <https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html>

IR Team. (n.d.). What is ISO 20022 and How is it Changing? From <https://www.ir.com/guides/what-is-iso-20022>

Bbva. (2019, Oct 16). Everything you need to know about PSD2. From <https://www.bbva.com/en/economy-and-finance/everything-need-know-psd2/>

* **API Protocols**: Artemis Financials’ specialization in RESTful APIS is pivotal for having secure connections to other nations in terms of transactions and communications over software systems over the internet. This is essential for integrating various financial services and conducting cross-border operations.
  + **RESTful APIS**: they include security features such as HTTPS, OAuth 2.0, and authentication / authorization methods to ensure that sensitive financial data is protected during international transactions. These measures help prevent unauthorized access and data breaches.

Epico Finanace. (2024). Integrating Financial Solutions With REST API: A Complete Guide. From <https://www.epicofinance.com/post/integrating-financial-solutions-with-rest-api-a-complete-guide>

Stripe. (2024. July 29) Financial APIs explained: What they are, how they work, and how they’re changing fintech. From <https://stripe.com/resources/more/financial-apis-explained-what-they-are-how-they-work-and-how-they-are-changing-fintech>

**Government Restrictions:** Artemis Financial must consider governmental restrictions and regulations that could impact secure communications. Financial institutions like Artemis Financial often handle sensitive customer information, making compliance with various laws and regulations critical. Based on the scenario and the context of financial planning.

* **Gramm-Leach-Bliley Act:** This is a federal law that says: financial institutions must protect the privacy of consumers’ financial information otherwise there’ll be consequences, and messing with the feds is something nobody should ever do. The GLBA’s safeguards rule requires the development and implementation of comprehensive information security program to secure customer data.
* **GDPR:** Operations involving European Union clients. Artemis must comply and enforce strict data protection and privacy standards. This includes ensuring secure communication channels when handling personal data of EU citizens.
* **SEC**: Sec requires financial firms to maintain records of business communication including electronic messages.
* **FCA**: In the UK, the FCA has placed heavy demands in the use of encrypted messaging apps by financial institutions to prevent market abuse and ensure compliance with record-keeping requirements.
* **FTC**: The FTC also affects Artemis Financial, emphasizing the need for encryption and access controls to protect consumer financial information.
* **NIST**: When adhering to NIST’s cybersecurity frameworks can help Artemis Financial align with best practices and regulatory expectations for secure communications.
* **Secure and Trusted Communications networks Act of 2019**: The US law prohibits the use of certain foreign telecommunications equipment and services deemed to pose national security risks. Artemis Financial must ensure that its communication infrastructure complies with these.
* **Bank Secrecy Act**: US regulation require financial institutions to implement strong AML and KTC programs, which include secure communication protocols to detect and prevent financial crimes, such as money laundering. Regulators have intensified their focus on financial crime compliance, emphasizing the importance of secure and monitored communications.

Federal Trade Commission. (n.d.). Gram-Leach-Bliley Act. From <https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act>

U.S. Securities and Exchange Commission. (n.d.). Cybersecurity. From <https://www.sec.gov/securities-topics/cybersecurity#:~:text=The%20SEC%20provides%20cybersecurity%20guidance,their%20customers%20from%20cyber%20threats>.

Cash Justin. (2024. November 13). FCA hits on ‘elevator pitch’ amid brand refresh. From <https://www.fnlondon.com/articles/fca-prepares-fresh-probe-into-bankers-encrypted-messaging-use-638b421c>

Federal Trade Commission. (n.d.). FTC Safeguards Rule: What your business needs to know. From <https://www.ftc.gov/business-guidance/resources/ftc-safeguards-rule-what-your-business-needs-know>

National Institute of Standards and Technology. (n.d.). Cybersecurity. From <https://www.nist.gov/cybersecurity>

Library of Congress. (n.d.). H.R.4998 – Secure and Trusted Communications Networks Act of 2019. From <https://www.congress.gov/bill/116th-congress/house-bill/4998>

Anand Nupur. (2024 November 13). US regulators warn bankers about intensified focus on financial crime. From <https://www.reuters.com/markets/us/us-regulators-warn-bankers-about-intensified-focus-financial-crime-2024-11-13/>

**Advanced Persistent Threat:** APTs are cyberattacks that can span over weeks if not months. It’s where a bad guy will gain hidden access to a network like an undercover agent / crocodile under the water, and attack when the time comes right. Financial institutions are prime targets due to the sensitive data. These attackers often use planned techniques to exfiltrate data or disrupt operations.

Yasar Kinza. (2023, December). Advanced persistent threat (APT). from <https://www.techtarget.com/searchsecurity/definition/advanced-persistent-threat-APT#:~:text=An%20advanced%20persistent%20threat%20(APT)%20is%20a%20prolonged%20and%20targeted,to%20the%20target%20organization's%20network>.

**Ransomware:** Is basically what it says. It’s the bad guys holding software for ransom, similar to a hostage situation. Bad guys will encrypt important data, usually they’ve done this by gaining access to the system and blocking the company out of it through encryption methods. They’ll often demand payment for the key and grant the company access back to their own data.

Check Point. (n.d.). What is raansomeware? <https://www.checkpoint.com/cyber-hub/threat-prevention/ransomware/>

**Phishing:** This is a very common tactic; you’ve probably seen one of these on your cell phone or email. It’s when a cyber bad guy breaks the law using trickery in the emails or phone calls and messages employees into revealing sensitive information to grant access to network / company framework.

America’s Cyber Defense Agency. (2021 Feb, 1). Avoiding Social Engineering and Phishing Attacks. From <https://www.cisa.gov/news-events/news/avoiding-social-engineering-and-phishing-attacks>

**Supply Chain Attacks:** These attacks are when the bad guys use weaknesses in third-party vendors to infiltrate larger organizations, such as Artemis Financial. In this type of attack cyber bad guys will exploits weaknesses in other software companies that have a weak cyber infrastructure and using trusted relationships between the victim and third-party partners as a pathway to gain unauthorized access. It would be best for Artemis to keep this in mind when dealing with international deals.

Cloudflare. (n.d.). What is a supply chain attack? From <https://www.cloudflare.com/learning/security/what-is-a-supply-chain-attack/>

**Cryptojacking:** This is when a cyber bad guy gains access to Artemis Financial’s computers / CPUs to mine cryptocurrencies. This can degrade system performance in house and cause disruptions in company infrastructure.

Zemlin Greg. (2024, April 29). Cryptojacking Explained. From <https://www.wiz.io/academy/what-is-cryptojacking>

**Injection Attacks:** Injection attacks occurs when cyber bad guys exploit a vulnerability in the system then puts malicious / cyber bad guy data/commands into the input fields. They can effectively gain access via launching unwanted commands into a system. These attacks often exploit inadequate input validation or poor coding practices, making them a common threat.

SentinelOne. (2024, October 17). Injection Attacks: Types, Techniques, and Prevention. From <https://www.sentinelone.com/cybersecurity-101/cybersecurity/injection-attacks/#:~:text=Injection%20attacks%20are%20a%20type%20of%20attack%20in%20which%20an,due%20to%20poor%20input%20validation>.

**Man-in-the-middle attacks:** A MITM attacks is when a cyber bad guy hacks into the communications without Artemis Financial knowing. For Artemis Financial, this attack could involve a situation where an attacker eavesdrops on sensitive communications between company messages. The attacker might use this position to steal confidential information, and then use said confidential information to do cyber bad guy things. Like manipulate data within the system, or even inject malicious commands. They usually do this through deciphering encryptions in the WiFI networks, or DNS spoofing. We can defend against this by using HTTPS for all communications.

Imperva. (2024) Man in the middle (MITM) attack. From <https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/>

**Denial of Service (DoS):** DoS attacks are when an intruder disrupts services by infecting the systems with cyber venom. A bunch of over the top and insanely unwanted code that the infrastructure cannot handle starts being executed within the system. This renders them inoperable/useless and disrupts user experiences. Artemis along with other financial business are prime targets for such attacks. These attacks can lead to significant operational disruptions and financial losses. To mitigate these, Artemis should deploy web application firewalls, implement rate limiting, and establish incident response plans to ensure swift recovery from potential disruptions.

**Open-Source Libraries:** Open-source libraries is a community driven practice that has the benefit of being a cost-effective solution to software development by providing pre-built components to Artemis Financial. It is a collection of pre-written code made freely available to developers, which can be used to build software applications. The source code is openly accessible, allowing anyone to use or modify them. Rapid prototyping support, and community support are its upsides, However, has the down side of providing high security risks.

**Evolving Web Applications:** refers to the latest tools, frameworks, protocols, and methodologies that enhance the development, performance, and functionality of web applications.

**Villaex Technologies.** (2023, December 5). Evolving Web Development: Developing Patterns, Technologies, and Innovations. From <https://www.linkedin.com/pulse/evolving-web-development-developing-patterns-technologies-a2xyf/>

**Cloud Computing Adoption:** This is one that’s basically become industry standard. A cloud-based infrastructure removes near all potential hardware limitations on the server side of things. In many cases it even adds other benefits, such as scalability, flexibility, and cost-efficiency. Cloud services enable rapid deployment of applications and even provide disaster recoveries in the event that it’s needed. However, it is also essential to implement security measures to protect sensitive financial data within said cloud.

**Open Banking Standards:** Open banking is a framework that enables the sharing of financial data between banks and Artemis Financial through APIs. While this has the benefit of making Artemis seem better for the economy as it promotes competition, it also helps customers use their prefer payment methods on a secure channel.

**Blockchain Technology:** Blockchain is a digital ledger type technology for record keeping in the system that has the benefit of being decentralized. This means that no single entity has control over the data. It records transactions or data in a series of blocks that are linked together to form a chain. Each block contains information such as a list of transactions, a timestamp, and a unique identifier called a hash. This makes block chaining ideal for Artemis Financial in areas such as cross-border payments and trade finance. As problems such as the risk of fraud and cyber attacks are less likely as the transactions would be encrypted and only valid entries are recorded and only verified through controlled mechanisms.

**DevOps:** DevOps is the term that bridges the gap between IT and Software Development. It’s a set of practices and tools that effectively make a certain type of culture and aims to streamline the entire software development lifecycle for most things coding, and this include cyber security. Similar to Agile and Scrum frameworks, it’s highly used in most STEM related programs in development.

**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:** Secure input validation is important to prevent injection attacks, such as SQL injection or command injection, these attacks can compromise sensitive data and lower Artemis Financials reputation. The company needs to validate and sanitize all user inputs to ensure data is not exposed, and all malicious practices will be stopped from entering the system. This is particularly critical for APIs used in financial transactions, which handle sensitive user data.

**Secure API:** APIs were mentioned heavily in Artemis Financial ‘s briefings. This is a practice enables secured communication between employees and external systems such as banks or other companies through secured channels. This can be done through practices such as using HTTPS and OAuth for authentication. These practices can prevent unauthorized access, data breaches, or man-in-the-middle attacks.

**Cryptography:** Cryptography is another essential component. It’s the practice of encrypting sensitive data, such as personal information, transactions details, and communications. Proper encryption standards such as TLS 1.3 ensure data security during transit and at rest. Artemis Financial must identify and mitigate vulnerabilities in its encryption methods to ensure secure data storage and communication channels. Particularly for compliance with regulation like GDPR, and maintaining their reputation and integrity.

**Client/Server:** Developing secure systems that safeguard client and server interactions is critical for protecting against unauthorized access, data breaches, and ensuring seamless communication between components. This involves implementing security measures for both frontend and backend systems to mitigate risks at every layer of interaction. Key threats such as session hijacking, cross-site scripting (XSS), and cross-site request forgery (CSRF) can compromise client-server communication if not addressed. Techniques like enforcing secure cookies, using HTTP headers (Content Security, Policy, X-Frame-Options), and implementing end-to-end encryption, play a vital role in maintain the confidentiality and integrity of data. Regularly updating server configurations, employing authentication and authorization framework (OAuth 2.0), and monitoring for unusual activity are essential steps in bolstering security. Additionally, strong logging and monitoring of client-server communications enable quick detection and responses to potential attacks to add insult to injury in the vein of all this.

**Code Quality:** Maintaining high code quality is more important than just making sure the software is in a working condition. It prevents errors that could introduce security vulnerabilities into the system. For example, unattended and forgotten possible error messages can actually disclose critical system information. Cyber bad guys may exploit these weaknesses in the system to gain unauthorized access or launch data breachers. This is why secure coding practices ensures that the software adheres to industry standards, strong memory management, and honestly removing bugs is also good for the customer. Regular code reviews, combined with adherence to secure design patterns significantly reduce risks such as buffer overflows, unhandled exceptions, and other security flaws, ultimately enhancing the reliability and security of the application.

**Encapsulation:** Encapsulation involves using secure data structures and access controls to limit the exposure of critical components within the application. This principle is vital for protectinginternal logic and sensitive data. By securely structuring data and limiting access, Artemis Financial can reduce the attack surface available to malicious actors. This is an industry standard practice that can provide clean modular code and allow for Artemis to expand its systems, if necessary, in a less problem some way, while also giving it a stricture that is secure in how it limits the access to its data.

**3. Manual Review**

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

**Input Validation & Sanitization:** Several parts of the codebase lack input validation and sanitization, making them susceptible to injection attacks or improper data handlings. CRUD.JAVA has inputs that are passed to the constructors that are not validated or sanitized. CRUDController.java, the name parameter is directly used without validation. GreetingCOntroll.java. The name parameter is directly incorporated into the response without checks. We can place validation and sanitization parameters on all inputs at the points of entry to prevent malicious payloads or incorrect data formats. Use frameworks like the Spring Validator to do this.

**Secure API interactions:** APIs are critical components of any modern software application. They facilitate communication between services and external systems. However, the API endpoints in CRUDController.java such as /read lack essential mechanism to ensure secure communication and prevent unauthorized access. A lack of authentication and authorization checks could lead to vulnerabilities in the system and be abused by attackers who could use them to gain access to sensitive data or perform problematic actions. We can fix this by added strong security measures such as integrating OAuth or JWT for secure API access. These methods provide a strong framework to verify if the identities of users are legit or not. Additionally using HTTPS across all API endpoints is critical to ensure data in transit is encrypted, protecting it from interception or tampered by MITM attacks.

**Hardcoded Credentials:** A hardcoded database credential in DocData.java represents a security vulnerability. It’s not advised to hardcode important things like usernames and passwords. If it’s in the source code, that’s like keeping your social security number in your wallet. Cyber bad guys could use these credentials to gain unauthorized access to the database. This not only risks compromising sensitive data but also makes it significantly harder to rotate credentials, which is an essential part of maintaining system security. To mitigate this, credentials should be stored securely using environment variables or encrypted configuration files. For example, environment variables can be accessed programmatically during runtime without embedding sensitive information into the codebase, while encrypted configuration files can add another layer of protection. If these methods are deemed impractical, the hardcoded credentials should at least be removed entirely and replaced with secure methods for managing access.

**Lack of Prepared Statements:** DocData.java has a database query that is not parameterized, leaving the system vulnerable to SQL injection attacks. Without prepared statements, user inputs could be used maliciously to alter the structure of SQL queries, allowing attackers to access, modify, or delete sensitive data. To address this vulnerability, the use of parameterized queries is critical, as they ensure user inputs are treated as data rather than executable code. Implementing prepared statements not only mitigates SQL injections risks but also strengthens the overall integrity of the system by providing a standardized and secure way to interact with databases.

**Encapsulation:** The concepts of encapsulation are improperly used in multiple classes cross the codebase. This could lead to potential risks of unauthorized access unintended modifications to sensitive data. For example customer.java has variables such as account\_number and account\_balance that are accissble directly. This bypases any controls of validations. Another one is myDateTime.java, the variable mySecond, myMinute, and myHour are publicly accessible, which can lead to unintended alterations. We can fix this by implementing security measures. These variables should be declared as privates with controlled access provided through getters and setters. Proper encapsulation ensures that sensitive data is accessed and modified only in a secure and intentional manner, minimizing risks and improving code clarity.

**Error Handling:** Error handling in DocData.java is incomplete for database connections. This poses a secure risk, as the lack of effective exception handling in database connection logic could lead to system crashes or the exposure of sensitive information through error messages. Comprehensive error handling should be implemented to manage database-related issues gracefully. This includes logging errors securely without revealing sensitive details and ensuring that exceptions are properly caught and managed to prevent disruptions. Additionally, using centralized logging mechanisms can help in identifying and addressing errors while maintaining the confidentiality and integrity of the systems.

**Logging and Monitoring:** The code lacks logging and monitoring mechanisms. These are critical for detecting and responding to unexpected behavior or security incidents. We can fix this by adding logging systems to track actions, such as API requests, errors, and suspicious activity. Libraries like Log4j or SLF4j are useful for structured and secure logging.

**Placeholder Methods:** Incomplete implementation of methods can lead to undefined behaviors and disrupt the functionality of an application, particularly when those methods are invoked inadvertently or misused. In myDatTime.java, the methods retrieveDateTime() and setMyDateTime() are currently placeholders without any functional code, leaving them prone to errors or unexpected results if called in their current state. We can address this by completing them effectively.

**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

|  |  |  |
| --- | --- | --- |
| bcprov-jdk15on-1.46.jar | CVE-2023-33202 | Bouncy Castle for Java before 1.73 contains a potential Denial of Service (DoS) issue within the Bouncy Castle org.bouncycastle.openssl.PEMParser class. This class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN.1 data through the PEMParser causes an OutOfMemoryError, which can enable a denial of service attack. (For users of the FIPS Java API: BC-FJA 1.0.2.3 and earlier are affected; BC-FJA 1.0.2.4 is fixed.) |
| bcprov-jdk15on-1.46.jar | 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. |
| bcprov-jdk15on-1.46.jar | 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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000345](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000344](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000343](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000342](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000341](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000339](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2016-1000338](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2018-5382](https://nvd.nist.gov/vuln/detail/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. |
| bcprov-jdk15on-1.46.jar | [CVE-2017-13098](https://nvd.nist.gov/vuln/detail/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." |
| bcprov-jdk15on-1.46.jar | [CVE-2013-1624](https://nvd.nist.gov/vuln/detail/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. |
|  |  |  |
| hibernate-validator-6.0.18.Final.jar | [CVE-2020-10693](https://nvd.nist.gov/vuln/detail/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 | [CVE-2023-35116](https://nvd.nist.gov/vuln/detail/CVE-2023-35116) | jackson-databind through 2.15.2 allows attackers to cause a denial of service or other unspecified impact via a crafted object that uses cyclic dependencies. NOTE: the vendor's perspective is that this is not a valid vulnerability report, because the steps of constructing a cyclic data structure and trying to serialize it cannot be achieved by an external attacker. |
| jackson-databind-2.10.2.jar | [CVE-2021-46877](https://nvd.nist.gov/vuln/detail/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 (2 GB transient heap usage per read) in uncommon situations involving JsonNode JDK serialization. |
| jackson-databind-2.10.2.jar | [CVE-2022-42004](https://nvd.nist.gov/vuln/detail/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. |
| jackson-databind-2.10.2.jar | [CVE-2022-42003](https://nvd.nist.gov/vuln/detail/CVE-2022-42003) | In FasterXML jackson-databind before versions 2.13.4.1 and 2.12.17.1, 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. |
| jackson-databind-2.10.2.jar | [CVE-2020-36518](https://nvd.nist.gov/vuln/detail/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. |
| jackson-databind-2.10.2.jar | [CVE-2020-25649](https://nvd.nist.gov/vuln/detail/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. |
|  |  |  |
| log4j-api-2.12.1.jar | [CVE-2021-44832](https://nvd.nist.gov/vuln/detail/CVE-2021-44832) | Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2. |
| log4j-api-2.12.1.jar | [CVE-2021-45105](https://nvd.nist.gov/vuln/detail/CVE-2021-45105) | Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3 and 2.3.1) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0, 2.12.3, and 2.3.1. |
| log4j-api-2.12.1.jar | [CVE-2021-45046](https://nvd.nist.gov/vuln/detail/CVE-2021-45046) | It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments. Log4j 2.16.0 (Java 8) and 2.12.2 (Java 7) fix this issue by removing support for message lookup patterns and disabling JNDI functionality by default. |
| log4j-api-2.12.1.jar | [CVE-2021-44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228) | Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects. |
| log4j-api-2.12.1.jar | [CVE-2020-9488](https://nvd.nist.gov/vuln/detail/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 | [CVE-2023-6378](https://nvd.nist.gov/vuln/detail/CVE-2023-6378) | A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. |
| logback-core-1.2.3.jar | [CVE-2021-42550](https://nvd.nist.gov/vuln/detail/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 | [CVE-2022-1471](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-41854](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-38752](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-38751](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-38750](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-38749](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2022-25857](https://nvd.nist.gov/vuln/detail/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. |
| snakeyaml-1.25.jar | [CVE-2017-18640](https://nvd.nist.gov/vuln/detail/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. |
|  |  |  |
| spring-boot-2.2.4.RELEASE.jar | [CVE-2023-20883](https://nvd.nist.gov/vuln/detail/CVE-2023-20883) |  |
| spring-boot-2.2.4.RELEASE.jar | [CVE-2023-20873](https://nvd.nist.gov/vuln/detail/CVE-2023-20873) |  |
| spring-boot-2.2.4.RELEASE.jar | [CVE-2022-27772](https://nvd.nist.gov/vuln/detail/CVE-2022-27772) |  |
|  |  |  |
| spring-boot-starter-web-2.2.4.RELEASE.jar |  |  |
|  |  |  |
| spring-core-5.2.3.RELEASE.jar | [CVE-2023-20863](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2023-20861](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2022-22971](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2022-22970](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2022-22968](https://nvd.nist.gov/vuln/detail/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-core-5.2.3.RELEASE.jar | [CVE-2022-22965](https://nvd.nist.gov/vuln/detail/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-core-5.2.3.RELEASE.jar | [CVE-2022-22950](https://nvd.nist.gov/vuln/detail/CVE-2022-22950) | n 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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2021-22060](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2021-22096](https://nvd.nist.gov/vuln/detail/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-core-5.2.3.RELEASE.jar | [CVE-2021-22118](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2020-5421](https://nvd.nist.gov/vuln/detail/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. |
| spring-core-5.2.3.RELEASE.jar | [CVE-2016-1000027](https://nvd.nist.gov/vuln/detail/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. |
|  |  |  |
| spring-core-5.2.3.RELEASE.jar |  |  |
|  |  |  |
| spring-web-5.2.3.RELEASE.jar |  |  |
|  |  |  |
| spring-webmvc-5.2.3.RELEASE.jar |  |  |
|  |  |  |
| tomcat-embed-core-9.0.30.jar | [CVE-2024-21733](https://nvd.nist.gov/vuln/detail/CVE-2024-21733) | Generation of Error Message Containing Sensitive Information vulnerability in Apache Tomcat.This issue affects Apache Tomcat: from 8.5.7 through 8.5.63, from 9.0.0-M11 through 9.0.43. Users are recommended to upgrade to version 8.5.64 onwards or 9.0.44 onwards, which contain a fix for the issue. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-46589](https://nvd.nist.gov/vuln/detail/CVE-2023-46589) | Improper Input Validation vulnerability in Apache Tomcat.Tomcat from 11.0.0-M1 through 11.0.0-M10, from 10.1.0-M1 through 10.1.15, from 9.0.0-M1 through 9.0.82 and from 8.5.0 through 8.5.95 did not correctly parse HTTP trailer headers. A trailer header that exceeded the header size limit could cause Tomcat to treat a single request as multiple requests leading to the possibility of request smuggling when behind a reverse proxy. Users are recommended to upgrade to version 11.0.0-M11 onwards, 10.1.16 onwards, 9.0.83 onwards or 8.5.96 onwards, which fix the issue. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-45648](https://nvd.nist.gov/vuln/detail/CVE-2023-45648) | Improper Input Validation vulnerability in Apache Tomcat.Tomcat from 11.0.0-M1 through 11.0.0-M11, from 10.1.0-M1 through 10.1.13, from 9.0.0-M1 through 9.0.81 and from 8.5.0 through 8.5.93 did not correctly parse HTTP trailer headers. A specially crafted, invalid trailer header could cause Tomcat to treat a single request as multiple requests leading to the possibility of request smuggling when behind a reverse proxy. Users are recommended to upgrade to version 11.0.0-M12 onwards, 10.1.14 onwards, 9.0.81 onwards or 8.5.94 onwards, which fix the issue. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-42795](https://nvd.nist.gov/vuln/detail/CVE-2023-42795) | Incomplete Cleanup vulnerability in Apache Tomcat.When recycling various internal objects in Apache Tomcat from 11.0.0-M1 through 11.0.0-M11, from 10.1.0-M1 through 10.1.13, from 9.0.0-M1 through 9.0.80 and from 8.5.0 through 8.5.93, an error could cause Tomcat to skip some parts of the recycling process leading to information leaking from the current request/response to the next. Users are recommended to upgrade to version 11.0.0-M12 onwards, 10.1.14 onwards, 9.0.81 onwards or 8.5.94 onwards, which fixes the issue. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-44487](https://nvd.nist.gov/vuln/detail/CVE-2023-44487) | The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-41080](https://nvd.nist.gov/vuln/detail/CVE-2023-41080) | URL Redirection to Untrusted Site ('Open Redirect') vulnerability in FORM authentication feature Apache Tomcat.This issue affects Apache Tomcat: from 11.0.0-M1 through 11.0.0-M10, from 10.1.0-M1 through 10.0.12, from 9.0.0-M1 through 9.0.79 and from 8.5.0 through 8.5.92. The vulnerability is limited to the ROOT (default) web application. |
| tomcat-embed-core-9.0.30.jar | [CVE-2023-28708](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2022-42252](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-43980](https://nvd.nist.gov/vuln/detail/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-core-9.0.30.jar | [CVE-2022-34305](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2022-29885](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-41079](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-33037](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-30640](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-25329](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-25122](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2021-24122](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2020-17527](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2020-13943](https://nvd.nist.gov/vuln/detail/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. |
| tomcat-embed-core-9.0.30.jar | [CVE-2020-13935](https://nvd.nist.gov/vuln/detail/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. |
|  |  |  |
| tomcat-embed-websocket-9.0.20.jar |  |  |

**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.

**Upgrade to Latest version:** This is the advice that is mentioned everywhere. Upgrading all of the thingys to their latest version will solve at least 90% of vulnerabilities within the system (I took that statistic from thin air). Software developers consistently release updates to patch security flaws. Obviously, this won’t solve everything.

(Bouncy Castle: CVE-2023-33202, CVE-2016-1000352, CVE-2016-1000346, CVE-2016-1000345, CVE-2016-1000344, CVE-2016-1000343, CVE-2016-1000342, CVE-2016-1000341, CVE-2016-1000339, CVE-2016-1000338, CVE-2018-5382.

Log4j: CVE-2021-44228, CVE-2021-44832, CVE-2021-45105, CVE-2021-45046, CVE-2020-9488

Tomcat: CVE-2024-21733, CVE-2023-46589, CVE-2023-45648, CVE-2023-42795, CVE-2023-44487

SnakeYAML: CVE-2022-1471, CVE-2022-41854, CVE-2022-38752, CVE-2022-38751, CVE-2022-38750, CVE-2022-38749, CVE-2022-25857

Spring Framework: CVE-2023-20863, CVE-2023-20861, CVE-2022-22971, CVE-2022-22970, CVE-2022-22965, CVE-2021-22060, CVE-2020-5421)

**Disabling JNDI:** JNDI lookup in the Log4j is one thing that can be disabled to stop weaknesses like CVE-2021-44228. Simply disabling JNDI ensures that external inputs cannot be used to dynamically fetch and execute malicious code form remote LDAP or RMI servers. This is particularly important in environments where log messages or configuration parameters might be influenced by untrusted sources, such as user input in web applications. (**Log4j:** CVE-2021-44228, CVE-2021-45046, CVE-2020-9488)

**Server Configuration:** HTTP smuggling, improper certificate validation, and data leaks caused by poorly configured HTTP headers are such vulnerabilities. Properly configured servers ensure that malicious inputs or misconfigurations cannot be exploited to compromise system security. We can mitigate this by correctly configuring missing HTTP headers to prevent such attacks, along with implementing certificate validation to ensure proper validation of SSL/TLS certificates. Another method includes configurations like rejectIllegalHeader=true in Tomcat to be set to enable. This ensures that any malformed or unexpected HTTP headers are rejected by the server. (**Tomcat:** CVE-2023-41080, CVE-2023-28708, CVE-2021-30640**)**

**YAML Parsing:** YAML is a popular due to its simplicity and readability. However, when processing untrusted YAML content, there are significant risks such as remote code execution (RCE) and denial of service (DoS) attacks. SnakeYAML, a widely used library for YAML parsing, allows for the instantiation of arbitrary Java objects during deserialization, which can be exploited by attackers to execute malicious code of exhaust system resources. We can mitigate this using SnakeYamal’s SafeConstructor. This feature can be used to restrict the types of objects that can be instantiated during deserialization. (**YAML Parsing:** CVE-2022-1471, CVE-2022-41854)

**Input Validations:** Implement strong input validation methods or sanitization inputs for all user-provided data. Properly checking all inputted data before they are processed by an application is almost essential to protecting against vulnerabilities. Failing to validate input effectively can lead to vulnerabilities such as injection attacks, deserialization exploits. And other forms of application misuse. (**Tomcat**: CVE-2023-46589, CVE-2023-45648. Hibernate Validator: CVE-2020-10693. Jackson Databind: CVE-2020-36518, CVE-2020-25649)

**Limit Resource Usage:** Introduce limits on recursion depth and resource usage during parsing or processing of data structures. This can effectively mitigate Denial of Service vulnerabilities and prevent resource exhaustion attacks. By introducing limits on rescurion depth, memory usage, and request handling, application can defend against malicious payloads deigned to overload their processing capabiltiies. This approach is particularly relevant for vulnerabiltiies related to deeply nested structures, excessive memory usage, or regulated input processing. (**Jackson Databind:** CVE-2023-35116, CVE-2020-36518, Logback: CVE-2023-6378, SnakeYAML: CVE-2022-38752, CVE-2022-38751)

**Authentication and Authorization**: apply strong authentication and role-based access control (RBAC) for all critical application components. With this we can ensure temporary directories and uploaded files are accessible only to authorized users, allowing for a privilege-based framework that provides protection to sensitive data and systems. **(Log4J:** CVE-2021-44832. **Spring Framework:** CVE-2021-22118)

**Secure Cryptographic Practices:** Replace weak cryptographic algorithms or implementations with strong ones. The strong ones prevent attackers from exploiting vulnerabilities in cryptographic algorithms or implementations. Weak ones lead to data breaches, unauthorized access, and compromised systems. We can mitigate this by upgrading to RSA key lengths, with a minimum of 2048-bit keys for RSA to meet current standards. AES encryption ensures that AES encryption uses secure modes like CBC with proper padding or strong GCM. (Bouncy Castle: CVE-2017-13098, CVE-2016-1000343)

**Testing and Monitoring:** Implementing automated vulnerability scans and penetration testing as part of a secure development lifecycle (SDLC). This includes deploying monitoring tolls to detect anomalous patterns such as repeated request cancellations or infinite loops caused by malicious payloads. (Tomcat: CVE-2023-44487, CVE-2020-17527. Spring Framework: CVE-2021-22096)

**Remove Legacy Features:** Deprecate outdated or insecure library features present exploitable vulnerabilities that attacker can target. Deprecating and removing legacy features, particularly those that are no longer supported. Removing these reduces the risk of exploitation and ensures applications align with current security standards.This is especially relevant for components like SnakeYAML and Log4j, where older configurations have been shown to introduce critical vulnerabilities. (Log4j: CVE-2021-45046. SnakeYAML: CVE-2017-18640)