GLOBALRAIN

Artemis Financial Vulnerability Assessment Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	1/23/24	Brayden Massa	

Client

Instructions

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

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

Developer
Brayden Massa

1. Interpreting Client Needs

Secure communications within this application are paramount to its success. Since Artemis Financial is a financial company, keeping their users' money safe and secure is essential to the success of the business. A breach in a customer's information could lead to a loss of trust from their clients as well as punitive damages from lawsuits. If Artemis Financial does business internationally, international compliance standards and regulations need to be considered. Financial organizations need to follow governmental compliance standards regarding data security and integrity, to protect their customers from hackers. Threats exist from attackers trying to gain customers' financial information, to steal their money/identities. Other threats exist to the organization specifically, where attackers can attempt to escalate privileges or flood information systems to try to shut down the web system. When using open-source libraries, staying up to date on the newest versions and being aware of their known vulnerabilities is crucial to maintaining a secure application, and this should be done regularly to avoid threats. Evolving web application technologies also must be studied, to keep up with their security standards and best practices, because not doing so could lead to new vulnerabilities and threats down the line.

2. Areas of Security

Based on the needs of the client and what they aim to achieve with their software, the areas of security that require the most focus are input validation, secure API interactions, cryptography, code quality, and encapsulation. Input validation and SQL parameterization are both essential to the application's security, since the application makes HTTP requests that receive user input, which could contain malicious input for diverse types of injection attacks. Secure API interactions are necessary to communicate with the application's API when making requests, so that no flaws in the system can be exploited by attackers. Cryptography is important for storing customers' personal information, financial information, and any other information that needs to be properly encrypted to ensure its security. Using secure coding patterns is essential for any secure coding, because failure to follow the best security patterns and protocols could lead to vulnerabilities within the code, allowing hackers to bypass other security measures and attack your application. Encapsulation is essential so the data structures within the application work as intended and cannot be mutated in a way to create new vulnerabilities for attacks.

3. Manual Review

Upon manual review of the code, there are multiple vulnerabilities that exist and should be addressed. In the myDateTime.java class, the fields are not encapsulated, meaning they can be accessed from external classes directly. This should be encapsulated, so that malicious manipulation of the class structure is not possible. In GreetingController.java, the greeting() method makes a request to the /greeting endpoint but does validate the input in any way. This input is concatenated to create a new Greeting object, however malicious or unexpected input could compromise its security, especially without proper error handling. The DocData.java class has database information listed directly in the code, which is not the best practice. Attackers can see this information and attack the database, leading to compromised data. These fields should be environment variables, not visible to the public. In customer.java, the account_balance variable is not properly encapsulated, which means it could be

manipulated from outside the bounds of the class. This compromises the data and should be safely stored and protected to prevent this. In the CRUDController.java class, a request is made to the /read endpoint, and takes an input parameter. This input is not validated and is used to create a new CRUD object. Malicious input could lead to unexpected functionality, which is always a security concern.

4. Static Testing

After running the tests, these are the found known vulnerabilities with third party software (ignoring low confidence vulnerabilities):

- Bouncy Castle Crypto version 1.46
    - Codes CVE-2016-1000338 and CVE-2016-1000342 say the DSA and ECDSA do not validate ASN 1 encoding of signature verification, so attackers can inject extra elements in the sequence which could lead to invisible data within a signed structure. It is recommended to upgrade beyond version 1.55
    - Code CVE-2016-1000343 states that the DSA key pair generator can generate a weak private key when used with default values. It is again recommended to upgrade beyond version 1.55
    - Codes CVE-2016-1000352 and CVE-2016-1000344 state that the DHIES and ECIES implementations allow the use of ECB mode which is regarded as unsafe. Support has since been removed for this after version 1.55 of the software
    - Code CVE-2016-1000341 says DSA signature generation is vulnerable to timing attacks, which could compromise the private value of the signatures. It is recommended to upgrade beyond version 1.55
    - Code CVE-2016-1000345 says the DHIES/ECIES CBC mode is vulnerable to padding oracle attack.
    - Code CVE-2017-13098 discusses a ROBOT vulnerability where an attacker can recover the private key when a TLS cipher suite is using RSA key exchange.
    - Code CVE-2020-15522 has a timing issue within the EC math library that exposes information about the private key.
    - Code CVE-2023-33202 contains a DoS issue where the org.bouncycastle.openssl.PEMParser class parses OpenSSL PEM encoded streams, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN 1 data causes an OutOfMemoryError.
    - Code CVE-2016-1000339 explains that the primary engine class used for AES can leak information on the AES key being used.
    - Code CVE-2015-7940 does not validate a point within the elliptic curve, making it easy for remote attackers to obtain private keys through a series of ECDH key exchanges.
    - Code CVE-2018-5382 says the default BKS keystore uses a 16-bit long HMAC, which can compromise the integrity of a BKS keystore.
    - Code CVE-2013-1624 says the TLS implementation does not carefully consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to make distinguishing attacks and plaintext-recovery attacks through statistical analysis of timing data for crafted packets.
    - Code CVE-2016-1000346 discusses that the other party's DH public key is not fully validated, which can lead to invalid keys revealing details about the other party's private key when static Diffie-Hellman is in use.

- Hibernate Validator 6.0.18
    - Code CVE-2020-10693 says a bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid, allowing attackers to bypass input sanitization controls.
- Jackson Databind version 2.10.2
    - Code CVE-2020-25649 discusses a flaw where entity expansion is not secured properly. This makes it vulnerable to XXE attacks, which could compromise data integrity
    - Code CVE-2020-36518 allows a StackOverflow exception and DoS due to a large depth of nested objects
    - Code CVE-2021-46877 discusses a vulnerability where attackers can cause a DoS involving JsonNode JDK serialization
    - Code CVE-2022-42003 says resource exhaustion can occur due to a lack of check in primitive value deserializers
    - Code CVE-2022-42004 says resource exhaustion can occur because a lack of a check in BeanDeserializer_deserializeFromArray
    - Code CVE-2023-35116 says attackers can cause a DoS via a crafted object that uses cyclic dependencies
- Apache Log4j API version 2.12.1
    - Code CVE-2020-9488 says improper certificate validation with host mismatch in the SMTP appender can allow SMTPS connection to be intercepted by a man-in-the-middle attack, leaking log messages sent through that appender.
- Logback Core version 1.2.3
    - Code CVE-2023-6378 says a serialization vulnerability allows for a DoS attack by sending poisoned data.
    - Code CVE-2021-42550 discusses how an attacker with the required privileges to edit configurations can craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers.
- Snake YAML version 1.25
    - Code CVE-2022-1471 discusses how the module's Constructor class has no type of restriction, which through deserialization can lead to remote code execution.
    - Code CVE-2017-18640 allows recursive entity expansion during a load operation, which could lead to DoS attacks.
    - Code CVE-2022-25857 allows DoS attacks because it is missing nested depth limits for collections.
    - Codes CVE-2022-38749, CVE-2022-38751, CVE-2022-38752, CVE-2022-41854, and CVE-2022-38750 are vulnerabilities where parsing untrusted YAML files could lead to DoS attacks with malicious YAML inputs.
- Spring Boot version 2.2.4
    - Code CVE-2023-20873. Any application deployed to Cloud Foundry could be at risk to a security bypass.
    - Code CVE-2022-27772 explains how it is vulnerable to temporary directory hijacking, which affected the "org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir" method.
    - Code CVE-2023-20883 is vulnerable to DoS attacks if Spring MVC is used with a reverse proxy cache.
- Spring Core version 5.2.3 and Spring Web MVC version 5.2.3

    - Code CVE-2022-22965 is a CISA exploited vulnerability where a Spring MVC or Spring WebFlux application could be vulnerable to remote code execution through data binding. It is exploitable if the application is run on Tomcat as a WAR.
    - Code CVE-2021-22118 discusses how a WebFlux application is vulnerable to privilege escalation where an attacker can maliciously write and read files uploaded to the WebFlux app.
    - Code CVE-2020-5421 discusses a reflected file download attack vulnerability based on the browser used.
    - Codes CVE-2022-22950, CVE-2023-20863 and CVE-2023-20861 discuss how a malicious SpEL expression could cause a DoS condition.
    - Code CVE-2022-22971 discusses how an application with a STOMP over WebSocket endpoint is vulnerable to a DoS by an authenticated user.
    - Code CVE-2022-22968 discusses the improper handling of casing compromises the security of the fields that are affected.
    - Code CVE-2022-22970 discusses how applications with file uploads are vulnerable to DoS attacks if they use data binding to set a MultiPartFile or Part to a field in a model.
    - Codes CVE-2021-22096 and CVE-2021-22060 discuss how malicious input can cause the insertion of additional log files.
- Core Tomcat version 9.0.30
    - Code CVE-2020-1938 is a CISA known exploited vulnerability. Extra care must be taken when trusting incoming connections to Apache Tomcat, since AJP connections are trusted more than HTTP connections. Attackers can exploit these connections
    - Code CVE-2020-11996 allows a specially crafted sequence of HTTP requests to trigger high CPU usage, which could lead to the server being unresponsive.
    - Code CVE-2020-13934 explains how an h2c direct connection to Tomcat did not release the HTTP/1 processor after upgrading to HTTP/2. IF enough requests are made, an OutOfMemoryException could lead to a DoS
    - Code CVE-2020-13935 says the payload length in a WebSocket frame was not validated, meaning invalid payload lengths could trigger an infinite loop, leading to a DoS
    - Code CVE-2020-17527 says Tomcat could re-use an HTTP request header value from the previous stream, leading to the possibility of leaked information between the requests.
    - Code CVE-2021-25122 says that when responding to new h2c connection requests, Tomcat could duplicate request headers and a limited amount of request body from one request to another, meaning both users could see the results from one of the users' requests.
    - Code CVE-2021-41079 says Tomcat did not properly validate incoming TLS packets, meaning a specially crafted packet could be used to trigger an infinite loop, leading to DoS
    - Code CVE-2022-29885 says the documentation for the EncryptInterceptor incorrectly stated it enabled Tomcat clustering to run over an untrusted network. It does not protect against all risks associated with running over any untrusted network, particularly DoS risks.
    - Code CVE-2022-42252 says if Tomcat was configured to ignore invalid HTTP headers by setting rejectIllegalHeader to false, it did not reject a request containing invalid Content-Length header making a request smuggling attack possible
    - Code CVE-2023-44487 is a CISA known exploited vulnerability where the HTTP/2 protocol allows a DoS due to request cancellation resetting many streams quickly.

- Code CVE-2023-46589 says that improper input validation did not correctly parse HTTP trailer headers, meaning a header that exceeded the header size limit could cause Tomcat to treat single requests as multiple requests, which exposes the risk to request smuggling when behind a reverse proxy.
- Code CVE-2023-46589 states that if an attacker can control the contents and name of a file on the server and the server is configured to use the PersistenceManager with a FileStore, the PersistenceManager is configured with sessionAttributeClassNameFilter="null" and the attackers knows the relative file path from the storage location used by FileStore to the file the attacker has control over, then using a specifically crafted request could lead to remote code execution via deserialization of the file under their control.
- Code CVE-2021-25329 says the fix for CVE-2023-46589 was incomplete, meaning the previous circumstances in addition to a configuration edge case that was highly unlikely to be used, could lead to the same issue of RCE
- Code CVE-2021-30640 describes how a vulnerability in the JNDI Realm of Tomcat allows an attacker to authenticate using variations of a valid username and/or to bypass some protection provided by the LockOutRealm.
- Code CVE-2022-34305 says the Form authentication example in the example web application exposed an XSS vulnerability
- Code CVE-2023-41080 says that there was a vulnerability with URL Redirection to untrusted sites in the FORM authentication feature.
- Code CVE-2021-24122 describes how when serving resources from a network location using the NTFS file system, Tomcat was susceptible to JSP source code disclosure in some configurations. This is because of unexpected behavior from the JRE API FIle.getCanonicalPath(), which was caused by inconsistent behavior of the Windows API
- Code CVE-2021-33037 discusses how Tomcat did not correctly parse the HTTP transfer-encoding request header sometimes, leading to request smuggling when used with a reverse proxy.
- Code CVE-2023-42795 discusses an incomplete cleanup vulnerability when recycling various internal objects in Tomcat, which could lead to skipping parts of the recycling process, and lead to information leaks.
- Code CVE-2023-45648 says that improper input validation did not correctly parse HTTP trailer headers. An invalid trailer header could cause Tomcat to treat a single request as multiple, which could lead to request smuggling.
- Code CVE-2019-17569 explains how refactoring introduced regression. This led to Transfer-Encoding headers being incorrectly processed which led to a possibility of HTTP Request Smuggling.
- Code CVE-2020-1935 says that the HTTP header parsing code used an approach to end-of-line parsing that allowed invalid HTTP headers to be parsed as valid, putting it at risk to HTTP Request Smuggling
- Code CVE-2020-13943 describes that if an HTTP/2 client connecting to Tomcat exceeded the agreed max number of concurrent streams for a connection, a subsequent request made on that connection could contain HTTP headers from a previous request
- Code CVE-2021-43980 describes how the simplified implementation of blocking reads and writes exposed a long-standing concurrency bug, which could cause client connections to share an Http11Processor instance, which could make responses be received by the wrong client.

5. Mitigation Plan

According to the manual code review, the code needs to be secured in the highlighted areas. Privileged information should be hidden in environment variables. Also, SQL requests should be parameterized, and the inputs should be validated, to protect the application from injection attacks. Proper error handling should also be practiced here, to prevent distinct types of DoS attacks, as well as leaked information. Variables should also be properly encapsulated, to prevent malicious adaptations from being made to the code structure. The code should also be configured to use https protocols to protect data when traveling. Based on the static testing, the following mitigation plan should be followed to protect the application.

- Bouncy Castle Crypto package can be secured by upgrading the package beyond version 1.73, which should update all the found vulnerabilities
- Hibernate validator should be upgraded to version 6.0.20 or later to fix vulnerabilities
- Jackson databind should be updated to version 2.16.0 or later to fix the investigated vulnerabilities
- Log4j API should be upgraded to 2.12.3 or beyond to fix the found vulnerabilities
- Logback Core should be upgraded beyond version 1.2.7 to fix the vulnerabilities in the report
- SnakeYAML should be upgraded to 2.0 or beyond to fix the vulnerabilities in the report
- Spring Boot should be upgraded beyond version 2.5.15 to fix the vulnerabilities in the report
- Spring Core should be upgraded beyond version 5.2.24 to fix the vulnerabilities
- Spring Web should be upgraded beyond version 6.0.0 to fix the vulnerabilities in the report
- Spring Web MVC should be upgraded beyond version 5.2.24 to remove the vulnerabilities in the report
- Core Tomcat implementation should be upgraded to version 9.0.83 to fix the vulnerabilities found in the report.