# Dependability Analysis - Apache commons-imaging library

Julian Alexis Orcinoli*
Università degli Studi di Salerno
Fisciano, Salerno, Italy
j.orcinoli@studenti.unisa.it

Santiago Morales Henao†
Università degli Studi di Salerno
Fisciano, Salerno, Italy
s.moraleshenao@studenti.unisa.it

Franco Nicolás Merenda‡
Università degli Studi di Salerno
Fisciano, Salerno, Italy
f.merenda2@studenti.unisa.it

## ABSTRACT

This paper presents a comprehensive analysis of software dependability in the context of the Apache Commons Imaging project, a vital component of the Apache Commons IO ecosystem. The study employs a multifaceted approach, leveraging software analytics, testing tools, and vulnerability assessment techniques to evaluate and enhance the dependability and code quality of the project.

The analysis begins with an exploration of SonarCloud, delving into the identification and resolution of bugs. Subsequently, various software testing tools, including JaCoCo, CodeCov, PiTest, and EvoSuite, are employed to assess code coverage, mutation testing, and automated test case generation. The study extends to vulnerability checking using OWASP Dependability Checker and FindSecBugs.

Results showcase improvements in code quality, increased code coverage, and the identification and remediation of software vulnerabilities. The paper contributes valuable insights into the efficacy of diverse tools and methodologies in ensuring software dependability. Through this investigation, we aim to provide a nuanced understanding of the strengths and limitations of each approach, offering guidance for future endeavors in the pursuit of reliable and robust software systems.

## CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; **Software maintenance tools**; *Open source model*.

## KEYWORDS

Apache Commons Imaging, Automated Test Case Generation, Bug Fixing, Code Coverage, Code Quality, CodeCov, EvoSuite, FindSecBugs, JaCoCo, Java Microbenchmark Harness, JMH, Mutation Testing, OWASP Dependability Checker, PiTest, Software Analytics, Software Dependability, Software Engineering, Software Testing Tools, Software Vulnerabilities, SonarCloud

## 1 INTRODUCTION

The evolution of software dependability, intrinsic to modern computing, spans from mainframes to interconnected networks, broadening its focus on reliability, fault tolerance, security, and safety. Pioneering works in the 1980s emphasized redundancy techniques like Triple-Modular Redundancy. Shifting to the 1990s, the spotlight turned to security and safety, addressing cyber threats in banking and hazards prevention in healthcare software. Software dependability encompasses reliability, availability, safety, security, maintainability, and fault tolerance. Recent years witnessed increased complexity in cloud computing and IoT, necessitating innovative approaches. Avizienis, Laprie, Randell, and Landwehr emphasized the role of fault tolerance. Ongoing contributions underscore the evolution of these principles. As technology advances, software dependability will continue to evolve, balancing functionality, performance, and societal impact in interconnected systems.

In the contemporary technological landscape, software systems play a pivotal role across diverse domains, shaping both personal and societal interactions. The escalating reliance on software underscores the critical need for dependability in these systems. This paper embarks on a comprehensive exploration of software dependability, covering diverse aspects such as project analysis, testing tools, and vulnerability checks.

Commencing with an overview of Apache Commons Imaging [5], a project within the Apache Commons IO ecosystem, this paper sets the foundation for subsequent analyses. Insights into the project's structure are provided, offering context to the broader discussions on software dependability.

Moving forward, Software Analytics takes center stage, delving into bug identification and fixes. This meticulous analysis provides a nuanced understanding of software quality and dependability through advanced analytical tools.

The exploration of Software Testing Tools follows, encompassing crucial tools like JaCoCo and CodeCov. The analysis spans code coverage evaluations before and after bug fixes, augmented by insights from Mutation Testing and Java Microbenchmark Harness.

The next section focuses on the Automated Generation of Test Cases, using a tool named EvoSuite. This section probes the efficacy of these tools in generating meaningful test cases, aligning with the overarching theme of software dependability.

The critical dimension of Software Vulnerabilities is dissected, providing a brief overview of tools like OWASP Dependability Checker and FindSecBugs. Practical insights into running these tools, fixing identified bugs, and engaging in insightful discussions enrich this section.

The ensuing discussion section serves as a platform to scrutinize findings, highlight encountered challenges, and address pivotal issues in the realm of software dependability.

In conclusion, this paper synthesizes a multifaceted exploration of software dependability, transcending traditional boundaries. Covering project analysis, analytics, testing tools, and vulnerability checks, it encapsulates the essence of reliability, security, and robustness in software systems.

### 1.1 Apache Commons Imaging

Apache Commons Imaging is a powerful open-source Java library that resides within the Apache Commons project. This library focuses on simplifying the complexities of image processing for Java developers. Boasting support for a diverse range of image formats,

*All authors contributed equally to this research.
†All authors contributed equally to this research.
‡All authors contributed equally to this research.

including popular ones such as JPEG, PNG, GIF, BMP, and TIFF, provides a versatile toolkit for developers working with images.

One notable aspect of Apache Commons Imaging is its commitment to cross-platform compatibility, ensuring that developers can seamlessly integrate the library into various operating systems and environments. The project maintains an active and collaborative community, welcoming contributions and fostering an environment conducive to continual improvement.

In conclusion, Apache Commons Imaging stands out as an essential tool for Java developers engaged in image-related tasks. Its extensive format support, versatile manipulation capabilities, and commitment to collaboration make it a reliable choice for projects requiring robust image processing functionality.

## 2 SONARCLOUD

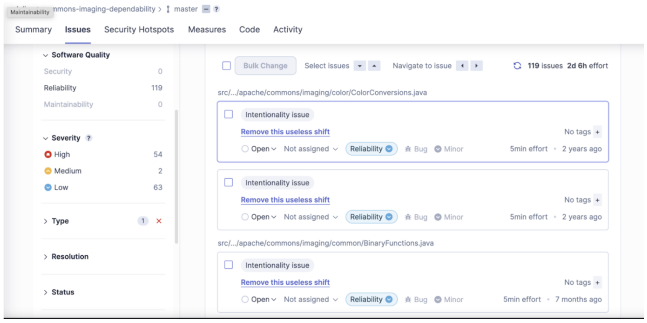



**Figure 1: Report before fix**

### 2.1 Analyzing SonarCloud

After analyzing the commons-imaging-dependability project on SonarCloud, we found the following issues in the code:

**High → 54**

In the high category, several errors are related to divisions where the divisor is not checked for the possibility of being 0. Therefore, if the divisor is 0, the code would throw an exception. There are also errors of the type where referencing a static member of a subclass from its parent during class initialization makes the code more fragile and prone to future errors. The program's execution will largely depend on the order of class and static member initialization.

**Medium → 2**

In the medium category, there are only 2 errors where an attempt is made to get the length of a palette that could be null.

**Low → 63**

Out of these 63 errors, most are related to the suggestion to remove a left shift, explaining that an int is a 32-bit variable and specifying that shifting an int by 32 is the same as shifting by 0, and shifting by 33 is the same as shifting by 1. Additionally, there are some errors related to number casting.

### 2.2 Bugs Fixed

To organize our tasks based on the errors found in **SonarCloud**[11], we created tasks in **JIRA**[1], and each of them was assigned to the three members of the group. We attempted to address the majority of the 119 issues related to reliability, of which we were able to fix 29.

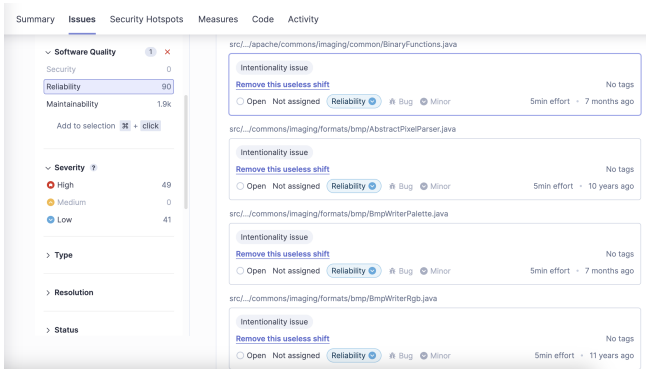



**Figure 2: Bugs Fixed**

## 3 SOFTWARE TESTING TOOLS

To measure and improve the robustness of the Apache Commons Imaging project's dependability, this section focuses on three key testing tools. First, we delve into the realm of code coverage using JaCoCo and CodeCov, unraveling their importance and impact. Following that, we explore mutation testing with PiTest, evaluating the resilience of test cases. The journey concludes with Java Microbenchmark Harness, providing insights into performance characteristics. Together, these tools form a robust arsenal to fortify and enhance the project's reliability.

### 3.1 Code Coverage - JaCoCo

JaCoCo [6] (Java Code Coverage) serves as a vital tool in assessing code coverage for Java applications, offering valuable insights into the extent of code execution during testing. It generates comprehensive reports that pinpoint the specific portions of the code exercised by tests.

Our investigation utilizing JaCoCo revealed a baseline of 77% Instructions Coverage and 63% Branches Coverage. This includes specific metrics such as 2448/6213 missed cyclomatic complexities, 3931/17299 missed lines, 511/2539 missed methods, and 16/431 missed classes.

A notable feature of JaCoCo lies in its ability to set a minimum expected coverage. This proactive measure ensures that code changes maintain a predefined level of coverage [7], contributing significantly to the overall robustness of the project.

### 3.2 Code Coverage - CodeCov

CodeCov [2] emerges as an instrumental tool, orchestrating a meticulous examination of code coverage within the Apache Commons Imaging project. This sophisticated platform specializes in unraveling the intricacies of test suite effectiveness.

**Key Metrics:**

(1) **Overall Code Coverage:** Our CodeCov analysis reveals a comprehensive code coverage of 71.17% across the entirety of the project. This quantitative metric serves as a substantive

indicator of the extent to which our test suites engage with the codebase.

(2) **Files with Lower Coverage Rates (In use with Below 0%):**
- ColorTools.java
- PixelDensity.java
- DataParserBitmap.java
- TagInfoSLongs.java
- ColorTools.java
- PbmWriter.java
- PgmWriter.java
- UncompressedDataReader.java
- ScanExpediterInterlaced.java
- PhotometricInterpreterCieLab.java
- PhotometricInterpreterYCbCr.java
- FieldTypeFloat.java

(3) **Line Coverage:** A granular examination exposes that among the 17,299 lines constituting the project, the project covers 12,311 lines.

Following the addition of targeted tests aimed at addressing files with 0% coverage, the overall code coverage has substantially increased to 71.82%. Out of a total of 17,303 lines, 12,427 are now covered. Notably, certain sections of the code remain only partially covered, emphasizing the ongoing commitment to fortify the software's robustness and reliability.

## 3.3 Mutation Testing - PiTest

Pitest [3] is an open-source mutation testing tool designed for Java projects. Its primary purpose is to evaluate the efficacy of a software test suite by introducing controlled and artificial changes (mutations) into the source code. The tool then analyzes how well the existing test suite detects and responds to these mutations. Pitest helps identify areas of weakness in test coverage, guiding developers in enhancing the robustness of their test suites and, consequently, improving overall code quality and reliability. After performing this testing tool these were the results:

*3.3.1* ***Mutant Coverage:*** The mutant coverage is decent, out of 15393 mutants introduced, 10217 were killed, The overall mutation coverage for the entire project is 66%, indicating that approximately 66% of introduced mutations are successfully detected by the test suite. Most packages scored a mutant coverage of around 52-73% for the interquartile range and a median of 0.61.

*3.3.2* ***Line Coverage:*** The overall line coverage for the project is relatively good, with most packages achieving a line coverage of around 74-86% for the interquartile range and a median of 0.79. The line coverage is at 77%, suggesting that tests cover a substantial portion of the codebase. Ranging from 38% (for just one datapoint) to 100 line coverage rate across all different packages indicates that a substantial portion of the code is covered by tests.

*3.3.3* ***Test Strength:*** The test strength metric is at 83%, which is a custom metric that seems to indicate a strong test suite.

For some packages, such as **.formats.jpeg.decoder**, **.formats.tiff.datareaders** and **.mylzw** the mutation coverage is notably high, suggesting a strong ability to detect mutations. Some packages, like **.png.transparencyfilters** and **.icc** have lower mutation coverage. It might be worth investigating whether there are opportunities to improve test coverage in these areas. The **.internal** package has relatively low line coverage, and it could benefit from additional tests.

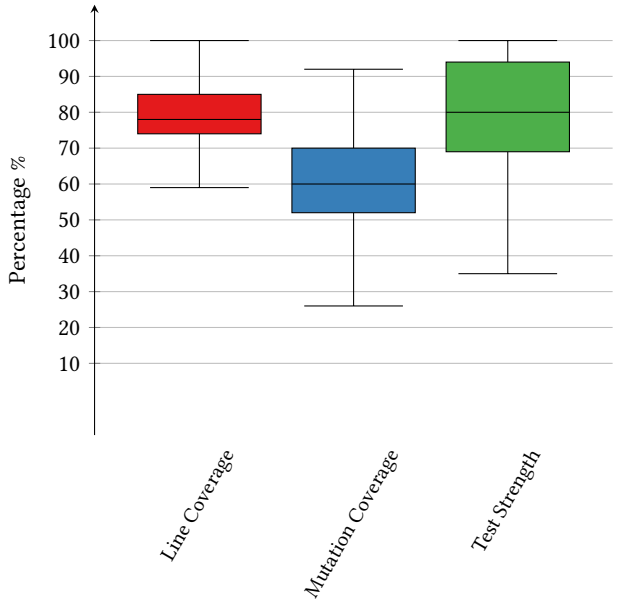



**Figure 3: Distribution of percentages across the 3 different metrics**

In the future it should be considered focusing on packages with lower mutation coverage and line coverage to improve the overall robustness of the test suite and explore specific mutations that were not covered to understand if they represent edge cases that should be addressed in the tests.

## 3.4 Java Microbenchmark Harness

JMH[9] is a powerful tool specifically designed for benchmarking Java code, allowing us to meticulously analyze the performance of our compression algorithms.

In the context of our image management project, an in-depth analysis of CPU-intensive image formats has revealed TIFF and WebP as the most demanding formats in terms of system resources. Following this, a detailed examination of test durations highlighted a specific, time-consuming scenario (`org.apache.commons.imaging.formats.tiff.TiffRoundtripTest.test`): the compression of TIFF images utilizing various algorithms (`TIFF_COMPRESSION_LZW`, `TIFF_COMPRESSION_PACKBITS`, `TIFF_COMPRESSION_DEFLATE_ADOBE`, and `Uncompressed`).

*3.4.1 Environment and Program Execution Specifications.* **Hardware Specifications:**
- CPU: Apple M1 Pro (2021)
- OS: MacOS Sonoma 14.2.1
- RAM: 16GB unified memory

**Program Execution Specifications (for report):**
- `OpenJDK version "1.8.0_392"`

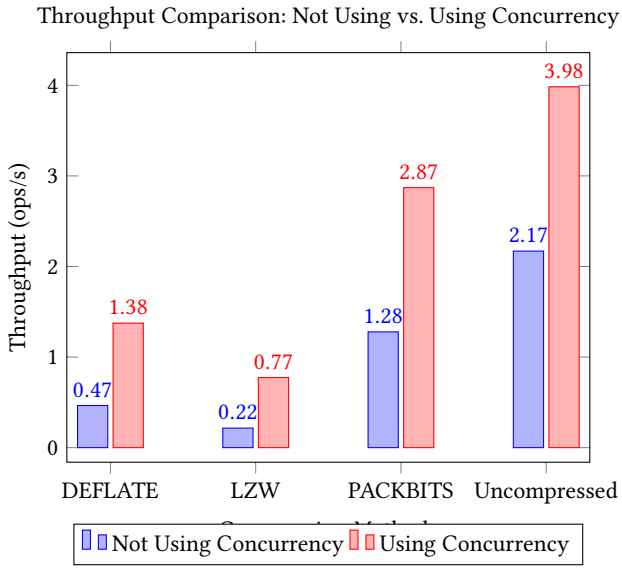



**Figure 4: Comparison of Throughput: Not Using vs. Using Concurrency**

- `OpenJDK Runtime Environment Corretto-8.392.08.1 (build 1.8.0_392-b08)`
- `OpenJDK 64-Bit Server VM Corretto-8.392.08.1 (build 25.392-b08, mixed mode)`

*3.4.2 Results.* To further explore the performance of each compression algorithm, each algorithm was applied separately to the same set of images, conducting a benchmark, that yielded the following results:

**Before Using Concurrency:**

- `TIFF_COMPRESSION_DEFLATE_ADOBE`: Throughput 0.465 ± 0.006 ops/s
- `TIFF_COMPRESSION_LZW`: Throughput 0.216 ± 0.001 ops/s
- `TIFF_COMPRESSION_PACKBITS`: Throughput 1.278 ± 0.053 ops/s
- `Uncompressed`: Throughput 2.170 ± 0.019 ops/s

Upon thorough review, we identified an opportunity for performance enhancement by introducing parallelism into the compression process. The parallel application of compression algorithms resulted in significant throughput improvements:

**After Using Concurrency:**

- `TIFF_COMPRESSION_DEFLATE_ADOBE`: Throughput 1.375 ± 0.012 ops/s (Increase by 196.77%)
- `TIFF_COMPRESSION_LZW`: Throughput 0.774 ± 0.008 ops/s (Increase by 258.33%)
- `TIFF_COMPRESSION_PACKBITS`: Throughput 2.870 ± 0.164 ops/s (Increase by 124.61%)
- `Uncompressed`: Throughput 3.984 ± 0.066 ops/s (Increase by 83.12%)

Beyond benchmark results, we observed a substantial reduction in the tiff test duration, decreasing from 8.4978 ± 0.0407 seconds (±0.48%) to 3.7254 ± 0.128 seconds (±3.44%) by applying compression algorithms to multiple image files in a concurrent way. This notable improvement not only enhances the efficiency of our compression algorithms but also contributes to a more expedited execution of test scenarios.

*3.4.3 Wilcoxon-Mann-Whitney Test and Cliff's Delta.* The Wilcoxon-Mann-Whitney test was performed to compare the two groups: "Not using concurrency" and "Using concurrency."

- Test Statistic ($W$): 3
- P-value: 0.2

**Decision Rule:** If the p-value is less than or equal to the significance level ($\alpha$), typically 0.05, we reject the null hypothesis.

**Interpretation:** Since the p-value (0.2) is greater than 0.05, there is not enough evidence to reject the null hypothesis. We do not conclude a significant difference between the two groups.

*3.4.4 Cliff's Delta Effect Size.* Cliff's Delta was calculated to assess the effect size between the two groups: "Not using concurrency" and "Using concurrency."

- Cliff's Delta: -0.625

**Interpretation:**

- A negative value indicates a tendency for lower values in the "Not using concurrency" group compared to the "Using concurrency" group.
- The magnitude of -0.625 suggests a moderate-sized effect.

**Conclusion:** The negative value of Cliff's Delta, with a moderate magnitude, suggests a moderate tendency for lower values in the "Not using concurrency" group compared to the "Using concurrency" group.

## 4 AUTOMATED GENERATION OF TEST CASES

Automated test generation is a critical aspect of ensuring software dependability and quality. In this context, we employed EvoSuite, a powerful Java-based tool utilizing genetic algorithms for test case generation, in our exploration of enhancing the test suite for the Apache Commons Imaging project.

### 4.1 EvoSuite

Evosuite[4] is a Java automated test generation tool utilizing genetic algorithms to maximize code coverage. Seamlessly integrating with JUnit, it supports parameterized testing, state-based testing, and automatic mock object generation. Evosuite's versatility extends to command line usage and IDE integration, offering an efficient solution for automating high-quality unit tests in Java applications.

Through the utilization of Evosuite, we successfully generated automated tests for the classes within the Apache Commons Imaging project. As assessed by OpenClover[8], a tool offering both line and branch coverage metrics, the outcomes revealed:

*4.1.1* ***Default Coverage Criteria:***

- Code Coverage Boost: Increased from 73.9% to 76.3% with automated tests.
- Tests Generated: A total of 584 tests were automatically generated.

#### 4.1.2 *Branch Coverage Criteria:*

- Code Coverage Boost: Increased from 73.9% to 77.6% with automated tests.
- Tests Generated: A total of 726 tests were automatically generated.

#### 4.1.3 *Wilcoxon Results:*

- Wilcoxon Size: 0.015973411027052232
- Wilcoxon Length: 0.00891931775884413
- Wilcoxon Score: 0.20010965436042927

**Summary**

The experiment involved utilizing Evosuite for automated test generation in the Apache Commons Imaging project, comparing results between Default Coverage Criteria and Branch Coverage Criteria.

#### 4.1.4 *Default Criteria vs. Branch Criteria:*

- **Code Coverage Boost:** The Branch Criteria outperformed Default Criteria, resulting in an increase from 73.9% to 77.6%.
- **Test Generation:** Branch Criteria led to the generation of 726 tests, surpassing the 584 tests generated by Default Criteria.

The Wilcoxon test results indicate statistically significant differences between the two criteria, emphasizing the effectiveness of Branch Coverage Criteria in improving code coverage and test suite quality.

These findings suggest that adopting Branch Coverage Criteria in Evosuite for the Apache Commons Imaging project yields better results compared to the Default Criteria.

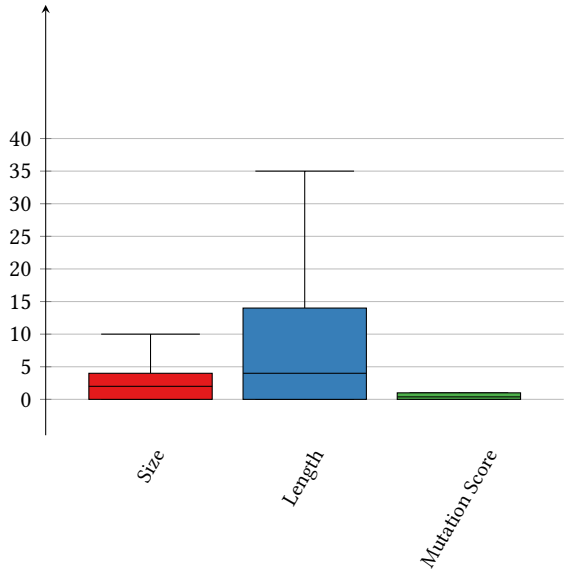



**Figure 5: Results for Size, Length, and Mutation Score for Default Criteria.**

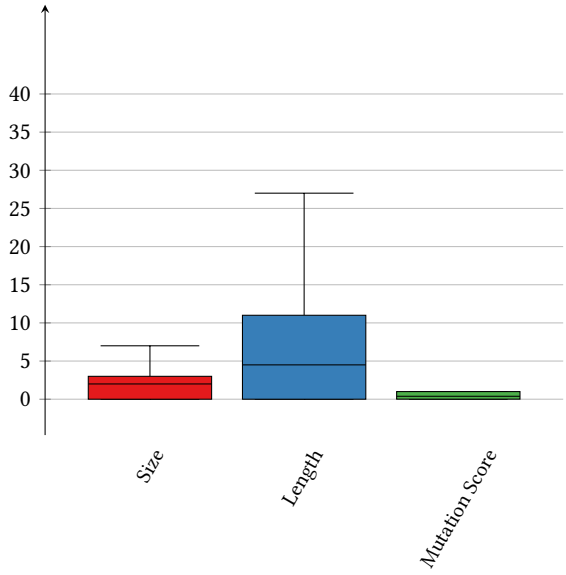



**Figure 6: Results for Size, Length, and Mutation Score for Branch Criteria.**

## 5 SOFTWARE VULNERABILITIES

Software vulnerabilities are weaknesses or flaws in a program that could be exploited by an attacker to compromise the integrity, confidentiality, or availability of the system. These vulnerabilities can exist at various layers of the software, from design errors to implementation flaws. It is crucial to address and fix these vulnerabilities to ensure the security of the software.

Common Types of Vulnerabilities in Apache Commons Imaging:

- ***Data Injection:*** This can occur when external input is not properly filtered or validated, allowing an attacker to inject malicious code.
- ***Buffer Overflow:*** This type of vulnerability occurs when the allotted storage capacity for a buffer is exceeded, enabling an attacker to overwrite surrounding memory.
- ***Inadequate Validation:*** Lack of input validation can lead to security issues. For example, if a user's authenticity is not verified before executing a critical operation.
- ***Authentication and Authorization Issues:*** Failures in authentication or authorization mechanisms may allow unauthorized users to access sensitive resources or functions.
- ***Incorrect Error Handling:*** Improper error handling can expose sensitive information or allow an attacker to discover weaknesses in the system.

### 5.1 OWASP Dependability Checker

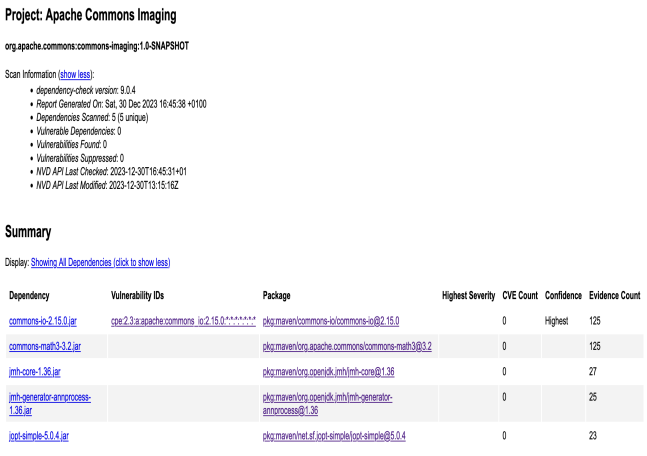
Upon conducting a comprehensive analysis using Owasp Dependency Checker[10] version 9.0.4, the findings reveal crucial insights into the project's dependencies.

### 5.2 Dependency Overview

The following key dependencies have been identified:

- **commons-io-2.15.0.jar**
- **commons-math3-3.2.jar**
- **jmh-core-1.36.jar**
- **jmh-generator-annprocess-1.36.jar**
- **jopt-simple-5.0.4.jar**

These dependencies are integral components influencing the project's structure and behavior.



**Project: Apache Commons Imaging**

org.apache.commons:commons-imaging:1.0-SNAPSHOT

Scan Information (show less):

- dependency-check version: 9.0.4
- Report Generated On: Sat, 30 Dec 2023 16:45:38 +0100
- Dependencies Scanned: 5 (5 unique)
- Vulnerable Dependencies: 0
- Vulnerabilities Found: 0
- Vulnerabilities Suppressed: 0
- NVD API Last Checked: 2023-12-30T16:45:31+01
- NVD API Last Modified: 2023-12-30T13:15:16Z

**Summary**

Display: Showing All Dependencies (click to show less)

| Dependency | Vulnerability IDs | Package | Highest Severity | CVE Count | Confidence | Evidence Count |
|---|---|---|---|---|---|---|
| commons-io-2.15.0.jar | cpe:2.3:a:apache:commons_io:2.15.0:*:*:*:*:*:*:* | pkg:maven/commons-io/commons-io@2.15.0 | | 0 | Highest | 125 |
| commons-math3-3.2.jar | | pkg:maven/org.apache.commons/commons-math3@3.2 | | 0 | | 125 |
| jmh-core-1.36.jar | | pkg:maven/org.openjdk.jmh/jmh-core@1.36 | | 0 | | 27 |
| jmh-generator-annprocess-1.36.jar | | pkg:maven/org.openjdk.jmh/jmh-generator-annprocess@1.36 | | 0 | | 25 |
| jopt-simple-5.0.4.jar | | pkg:maven/net.sf.jopt-simple/jopt-simple@5.0.4 | | 0 | | 23 |

**Figure 7: Owasp DC - HTML Generated Report**

## 5.3 Summary of Findings

- **Commons IO 2.15.0:**
  - Description: The Apache Commons IO library contains utility classes, stream implementations, file filters, file comparators, endian transformation classes, and much more.
  - **Analysis**: The analysis indicates a high-confidence dependency.
- **Commons Math3 3.2:**
  - Description: The Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.
  - **Analysis**: The analysis indicates a high-confidence dependency.
- **JMH Core 1.36:**
  - Description: This library constitutes the JMH Core for Java Microbenchmarking Harness. It is referenced in the project scope of Apache Commons Imaging, version 1.0-SNAPSHOT.
  - **Analysis**: The analysis indicates a high-confidence dependency.
- **JMH Generator Annprocess 1.36:**
  - Description: This dependency represents the JMH Benchmark Generator based on Annotation Processors. Similar to the previous dependency, it is referenced in the project scope of Apache Commons Imaging, version 1.0-SNAPSHOT.
  - **Analysis**: The analysis indicates a high-confidence dependency.
- **Jopt Simple 5.0.4:**
  - Description: Serving as a Java library for parsing command line options, this dependency is referenced in the project scope of Apache Commons Imaging, version 1.36.
  - **Analysis**: The analysis indicates a high-confidence dependency.

## 5.4 Implications

Owasp Dependency Checker does not report any vulnerabilities associated to the dependencies of the project. In the case of presence of vulnerabilities in these dependencies would underscore the critical importance of addressing and mitigating potential security risks. A proactive approach to dependency management and continuous monitoring is recommended to uphold the project's overall integrity and security posture.

These findings serve as a basis for further investigation and remediation efforts to enhance the robustness of the project's software supply chain.

## 5.5 FindSecBugs (SpotBug)

We conducted the analysis with FindSecBugs and identified two security bugs related to hard-coded keys.

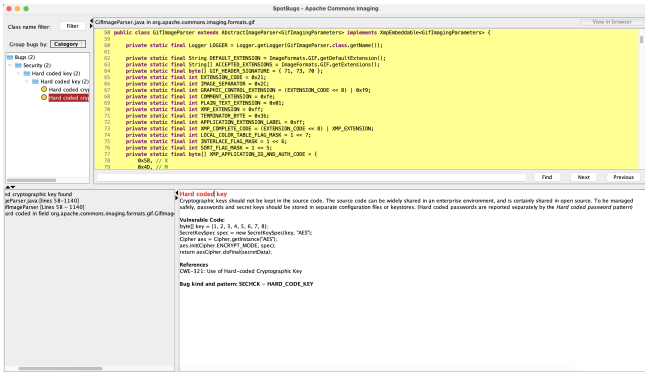



**Figure 8: Report before fix**

FindSecBugs references these types of vulnerabilities as the inappropriate storage of cryptographic keys in the source code. Given that source code can be widely distributed, especially in enterprise or open-source environments, it is not secure to include passwords and secret keys directly in the code. For secure management, passwords and secret keys should be stored in separate configuration files or keystores. (Note: Hard-coded passwords are addressed separately through the Hard-coded password pattern)

# 6 CONCLUSIONS

## 6.1 *Holistic Testing Approach:*

- The integration of CodeCov, Pitest, and JMH benchmarks provides a comprehensive understanding of the Apache Commons Imaging project's testing landscape and performance efficiency.

- CodeCov analysis reveals a commendable overall code coverage, with targeted tests enhancing coverage and demonstrating a commitment to fortifying the software's robustness.
- Pitest analysis showcases strong mutation coverage, emphasizing the effectiveness of targeted tests in detecting controlled mutations.
- JMH benchmarks allowed identifying critical performance improvements. Through the adoption of concurrency, it was possible to significantly enhance throughput and expedite test scenario execution.

### 6.2 *Advanced Test Generation Techniques:*

- Evosuite's integration into the testing workflow proves highly effective, demonstrating its ability to maximize code coverage and generate high-quality automated tests for the Apache Commons Imaging project.
- Branch Coverage Criteria outperforms Default Criteria, resulting in a substantial code coverage boost and the generation of a greater number of tests.
- Statistical analysis, including the Wilcoxon test, supports the conclusion that adopting Branch Coverage Criteria yields superior results, emphasizing the practical relevance of these improvements.

### 6.3 *Dependency Management and Security Practices:*

- The OWASP Dependability Checker analysis provides insights into crucial dependencies of the project, highlighting their significance and indicating high-confidence dependencies.
- No vulnerabilities are reported, affirming the robustness of current dependency management.
- FindSecBugs identifies security bugs related to hard-coded keys, emphasizing the critical need for secure cryptographic key management practices. The recommended approach includes storing passwords and secret keys in separate configuration files or keystores.

### 6.4 *Overall Project Recommendations and Future Research Directions:*

- The project should continue its commitment to comprehensive testing by leveraging CodeCov, Pitest, and JMH benchmarks, addressing specific areas identified for improvement.
- Evosuite's advanced test generation capabilities should be further utilized, with a focus on Branch Coverage Criteria for enhanced code coverage and test quality.
- Dependency management practices are commendable, and ongoing vigilance is recommended to address potential security risks.
- Implementation of FindSecBugs recommendations is crucial, emphasizing the separation of cryptographic keys from the source code to enhance security.

## ACKNOWLEDGMENTS

To the group, for always bringing "mate" while working.

## REFERENCES

[1] Atlassian. 2002. *JIRA*. https://www.atlassian.com/software/jira
[2] CodeCov. 2023. *CodeCov: Leading Test Coverage*. Retrieved December 27, 2023 from https://codecov.io/
[3] Henry Coles. 2013. *PIT - A mutation testing tool for Java and the JVM*. https://pitest.org/
[4] Gordon Fraser and Andrea Arcuri. 2012. *EvoSuite: Automatic Test Suite Generation for Java*. http://www.evosuite.org/
[5] Apache Commons Imaging. 2023. *Apache Commons Imaging: Image I/O and manipulation in Java*. Retrieved December 27, 2023 from https://commons.apache.org/proper/commons-imaging/
[6] JaCoCo. 2023. *JaCoCo - Java Code Coverage Library*. Retrieved December 27, 2023 from https://www.jacoco.org/
[7] JaCoCo. 2023. *JaCoCo Check Mojo Documentation*. Retrieved December 27, 2023 from https://www.jacoco.org/jacoco/trunk/doc/check-mojo.html
[8] OpenClover. [n. d.]. *OpenClover*. https://www.openclover.org/
[9] OpenJDK. 2023. *Java Microbenchmarking Harness (JMH)*. Retrieved December 27, 2023 from https://openjdk.java.net/projects/code-tools/jmh/
[10] OWASP Foundation. 2023. *OWASP Dependency-Check*. https://owasp.org/www-project-dependency-check/ Accessed: January 15, 2024.
[11] SonarSource. 2018. *SonarCloud*. https://www.sonarsource.com/products/sonarcloud/