Software Dependability Project, supervised by Prof. DARIO DI NUCCI

GHAZI BOUSSIDA, University of Salerno, IT
MOHAMED AZIZ KHITMI, University of Salerno, IT
ABDULWASSIF RASHID, University of Salerno, IT

.

The work accomplished in this project aims for the improvement of the quality of software, its security and its reliability. These goals were achieved through the analysis of code quality, code coverage, test cases, and performance of different software components. The work was done over an existing Apache Commons project, that uses Java as a programming language.

Additional Key Words and Phrases: Programming, Software Analysis,Software Quality, Code Coverage, Testing, Test Cases, Mutation Testing.

1 INTRODUCTION

This work is realized as an end-of-course project for the Software Dependability course at the University of Salerno. In this project, the focus is on the analysis of software quality, of code coverage, of test cases,and of the performance of the project's components.

2 THE PROJECT

2.1 Choice of project

The project of choice for this analysis is Apache Commons Email. This project aims to provide an API for sending emails. It is built on top of the Java Mail API, in the aim of simplifying it. This project can be found on the Apache Commons website https://commons.apache.org/proper/commons-email/. Apache Commons Email is built mainly in Java language, using Maven as a build tool.

2.2 Preparation

In order to start working on this project, the project is forked from the original Github repository https://github.com/apache/commons-email and the contributors are added to be able to collaborate on working on it. The forked repository is available at https://github.com/gboussida/commons-email.

3 SOFTWARE QUALITY ANALYSIS

This section covers the software quality analysis of the project, including the tools used for that matter, primary findings, the process of treating the code to increase code quality, and finally the outcome of it.

3.1 Tools

In this process, SonarCloud was used to analyze the project. SonarCloud automatically scans and analyzes code for bugs, code smells, vulnerabilities, and security issues.

3.2 Findings

The first analysis performed on the project shows that the project contains, ordered by severity, the following issues :

- 4 Security Hotspots,

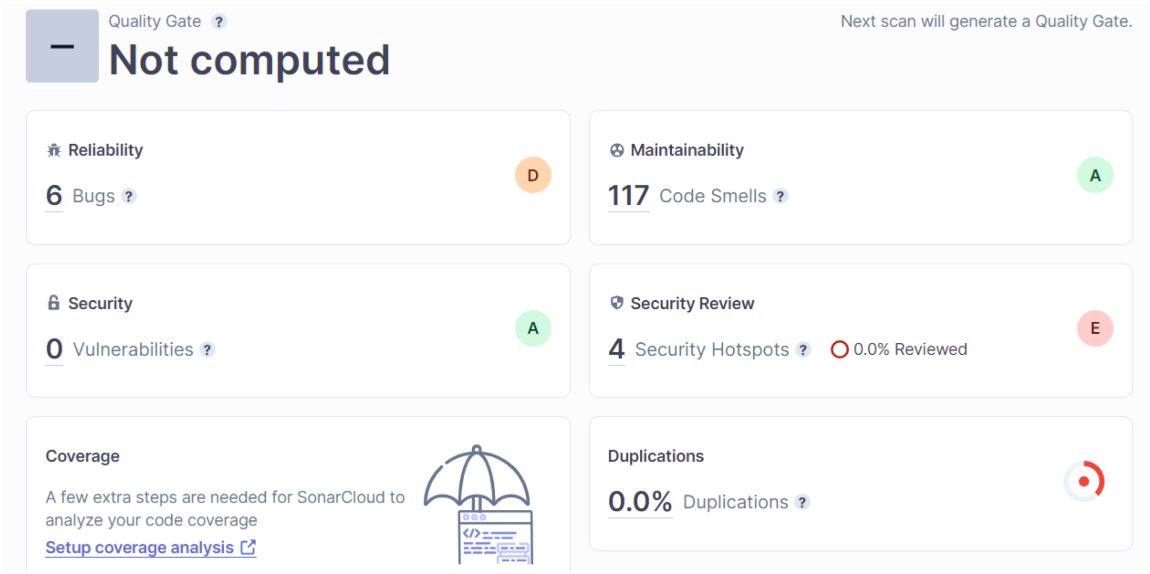
- 6 Bugs,
- 117 Code Smells.

The table 1 categorizes these issues:

Findings	Severity
Security Hotspots	Medium (3)
	Low (1)
Bugs	High (2)
	Medium (4)
Code Smells	High (23)
	Medium (42)
	Low (52)

Table 1. Categorization of code quality analysis findings

The figure 1 shows an overview of the findings.


Quality Gate ?
Not computed
Next scan will generate a Quality Gate.

Reliability
6 Bugs ? D

Maintainability
117 Code Smells ? A

Security
0 Vulnerabilities ? A

Security Review
4 Security Hotspots ? 0.0% Reviewed E

Coverage
A few extra steps are needed for SonarCloud to analyze your code coverage
Setup coverage analysis

Duplications
0.0% Duplications ?

Fig. 1. Results of first analysis using SonarCloud

3.3 Treatment

In an attempt to improve code quality and get rid of the issues found, code refactoring was required.

3.3.1 *Security hotspots.*
To reduce the number of security hotspots present in the code, the following was done:

- Use of `SecureRandom()` to replace `Random()`: This assures the proper encryption of sensitive data when generating a random variable.
- Use of Https to replace Http: This assures the encryption of normal HTTP requests and responses.

Two of the remaining hotspots were related to the use of regex which is vulnerable to polynomial runtime due to backtracking. The regex in this case uses pattern for extracting HTML tags from the email. These hotspots were reviewed as safe because the input is not user-controlled and is not restricted to a small number of characters. In addition, attempting to refactor them lead to multiple dependency issues.

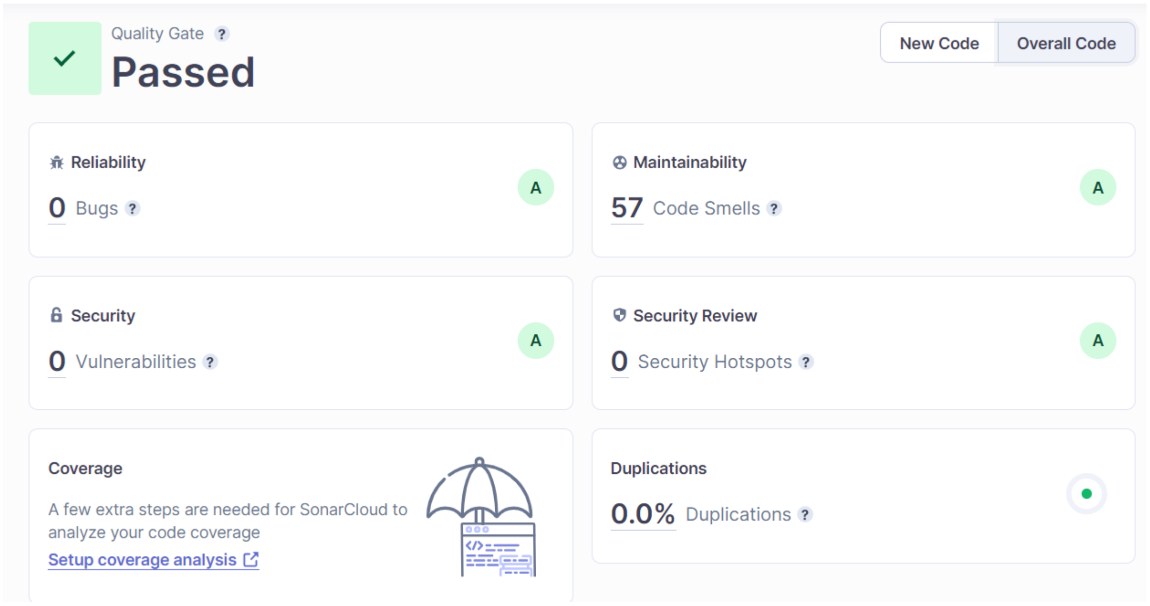
3.3.2 *Bugs.*
To treat the bugs highlighted by the code quality analysis, the refactoring included the following:

- Definition of a constant instead of duplicating literals.
- Refactoring of complex nested `IF`s to reduce complexity and overload.
- Replacement of `System.err` by a logger.
- Removed the synchronization using "out" which is a method parameter, and should not be used for synchronization.

3.3.3 *Code Smells.*
The treatment of code smells mainly involved removing redundant code, simplifying substring logic, adding a validation check to ensure the valid format of the Email and removing deprecated code.

3.4 Outcome

The treatments performed in section 3.3 lead to the results shown in figure 2. These results show that all the bugs and security hotspots are successfully treated. The remaining code smells are mainly linked to the cognitive complexity of some methods that contain multiple IF statements that are needed and cannot be modified.



Quality Gate ?
Passed

Reliability
0 Bugs ? A

Maintainability
57 Code Smells ? A

Security
0 Vulnerabilities ? A

Security Review
0 Security Hotspots ? A

Coverage
A few extra steps are needed for SonarCloud to analyze your code coverage
Setup coverage analysis

Duplications
0.0% Duplications ?

Fig. 2. Overview of the code quality analysis performed after treatment

4 CODE COVERAGE ANALYSIS

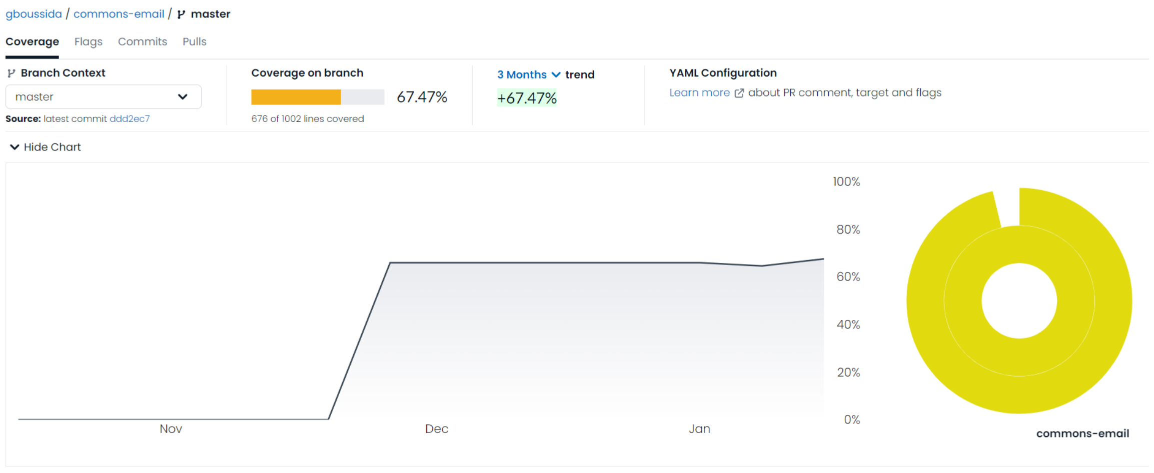
This section covers the code coverage analysis of the project. It presents tools used to conduct this analysis, the code coverage ratio of the main branch, then the code coverage by file.

4.1 Tools

Codecov is used to achieve the code coverage analysis. Codecov is a code coverage reporting solution that gives developers actionable insights to deploy reliable code. Setting up Codecov requires adding it to the Github actions workflow yaml file using the Codecov token.

4.2 Code Coverage of the project

The analysis shows that the coverage on the main branch is equal to 67.47%. Generally, a good coverage ratio between 60% and 90% is considered good. The figure 3 shows a display of the codecov dashboard.


gboussida / commons-email / master

Coverage Flags Commits Pulls

Branch Context: master

Source: latest commit ddd2ec7

Coverage on branch 67.47%

676 of 1002 lines covered

3 Months trend +67.47%

YAML Configuration

Learn more about PR comment, target and flags

Hide Chart

Fig. 3. Overview of Codecov dashboard

4.3 Code Coverage by file

The table2 shows the code coverage ratio by file.

File name	Coverage
src/main/java/org/apache/commons/mail/EmailAttachment.java	100%
src/main/java/org/apache/commons/mail/ DefaultAuthenticator.java	100%
src/main/java/org/apache/commons/mail/SimpleEmail.java	100%
src/main/java/org/apache/commons/mail/util/ MimeMessageParser.java	85.87%
src/main/java/org/apache/commons/mail/ Email.java	76.29%
src/main/java/org/apache/commons/mail/ ImageHtmlEmail.java	75.61%
src/main/java/org/apache/commons/mail/EmailUtils.java	73.13%
src/main/java/org/apache/commons/mail/resolver/ DataSourceCompositeResolver.java	68.42%
src/main/java/org/apache/commons/mail/resolver/ DataSourceClassPathResolver.java	64.29%
src/main/java/org/apache/commons/mail/resolver/ DataSourceUrlResolver.java	64%
src/main/java/org/apache/commons/mail/resolver/ DataSourceFileResolver.java	61.9%
src/main/java/org/apache/commons/mail/util/ IDNEmailAddressConverter.java	60.87%
src/main/java/org/apache/commons/mail/resolver/ DataSourceBaseResolver.java	60%
src/main/java/org/apache/commons/mail/MultiPartEmail.java	53.17%
src/main/java/org/apache/commons/mail/HtmlEmail.java	49.03%
src/main/java/org/apache/commons/mail/util/ MimeMessageUtils.java	38.89%
src/main/java/org/apache/commons/mail/ EmailException.java	36.84%

Table 2. Code coverage ratios by file

5 MUTATION TESTING BY PI TEST

Mutation testing, facilitated by PITest, is a method utilized to evaluate the effectiveness of a test suite by injecting simulated defects, referred to as mutations, into the code. Subsequently, it verifies whether the existing tests can identify and capture these mutations. PIT, an acronym for "Mutation Testing with JUnit and TestNG," is a widely-used mutation testing framework designed for Java.

We initiated the testing process by installing the PIT Plugin and adding the JUnit 5 dependency to the pom.xml file. Following this addition, We proceeded to build the system and executed the following command:

mvn org.pitest:pitest-maven:mutationCoverage

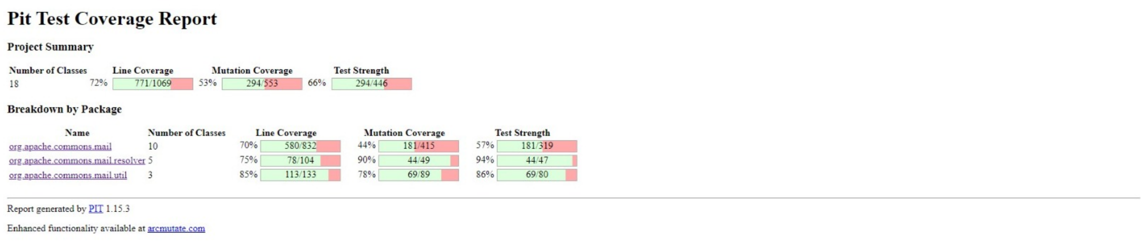
The execution of this command yielded two distinct outcomes:

(1) Mutations introduced in my project
(2) A comprehensive mutation testing report

The detailed report, outlining the results of the mutation testing, is accessible in the **target/pit-report** directory.

5.1 Results and Findings

This is detail view of report



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
18	72%	771/1069	53%	294/553	66%	294/446

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.apache.commons.mail	10	70%	580/832	44%	181/415	57%	181/319
org.apache.commons.mail.resolver	5	75%	78/104	90%	44/49	94%	44/47
org.apache.commons.mail.util	3	85%	113/133	78%	69/89	86%	69/80

Report generated by PIT 1.15.3

Enhanced functionality available at arcmutate.com

Fig. 4. Pi Test Coverage Results

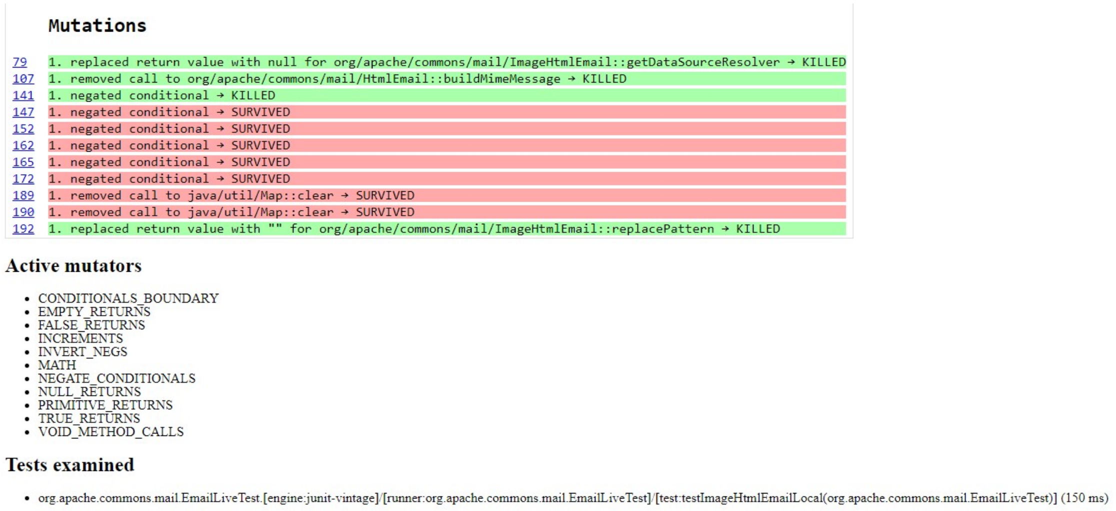
For more on general side, as per the summary, through PI test we mutated the whole project. we mutated 553 lines of code and our test case was only able to kill 294 mutations. 259 lines were not killed by test cases these lines were either ignored or were not covered. 107 lines were not covered by test cases and 152 lines survived that means test cases ran on these lines but was unable to detect the line mutation and survived.

5.2 Overall Result

As we can see from the above report generated that some of the code lines were killed during the mutation testing and some of the code lines survived the the mutation testing. The test is examined is the example of mutation testing on one test class Email.

It's important to note that the CONDITIONALSBOUNDARY mutation operator is just one of several mutation operators used in mutation testing. Others include operators for modifying arithmetic operations, logical operations, and method calls, among others. Each operator targets a specific aspect of the code to evaluate the thoroughness of the test suite.

Empty Returns: Mutation Operator: This operator involves replacing the return statement of a method with an empty return or a return of a default or neutral value



Mutations

```
79  1. replaced return value with null for org/apache/commons/mail/ImageHtmlEmail::getDataSourceResolver → KILLED
107 1. removed call to org/apache/commons/mail/HtmlEmail::buildMimeMessage → KILLED
141 1. negated conditional → KILLED
147 1. negated conditional → SURVIVED
152 1. negated conditional → SURVIVED
162 1. negated conditional → SURVIVED
165 1. negated conditional → SURVIVED
172 1. negated conditional → SURVIVED
189 1. removed call to java/util/Map::clear → SURVIVED
190 1. removed call to java/util/Map::clear → SURVIVED
192 1. replaced return value with "" for org/apache/commons/mail/ImageHtmlEmail::replacePattern → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- org.apache.commons.mail.EmailLiveTest.[engine:junit-vintage]/[runner:org.apache.commons.mail.EmailLiveTest]/[test:testImageHtmlEmailLocal(org.apache.commons.mail.EmailLiveTest)] (150 ms)

Fig. 5. Pi Test Coverage Report

False Returns: Mutation Operator: This operator involves changing the return value of a method to a boolean false.

In both cases, the idea is to introduce artificial mutations into the code and observe how well the existing test suite can identify these changes. If the test suite can catch the mutations, it suggests that the tests are effective in ensuring the correct behavior of the methods. If mutations go undetected, it may indicate areas where the test suite needs improvement, potentially by incorporating additional test cases that account for variations in return values.

These mutation operators are part of a broader set of operators used in mutation testing, each targeting specific aspects of the code to assess the thoroughness of the testing process.

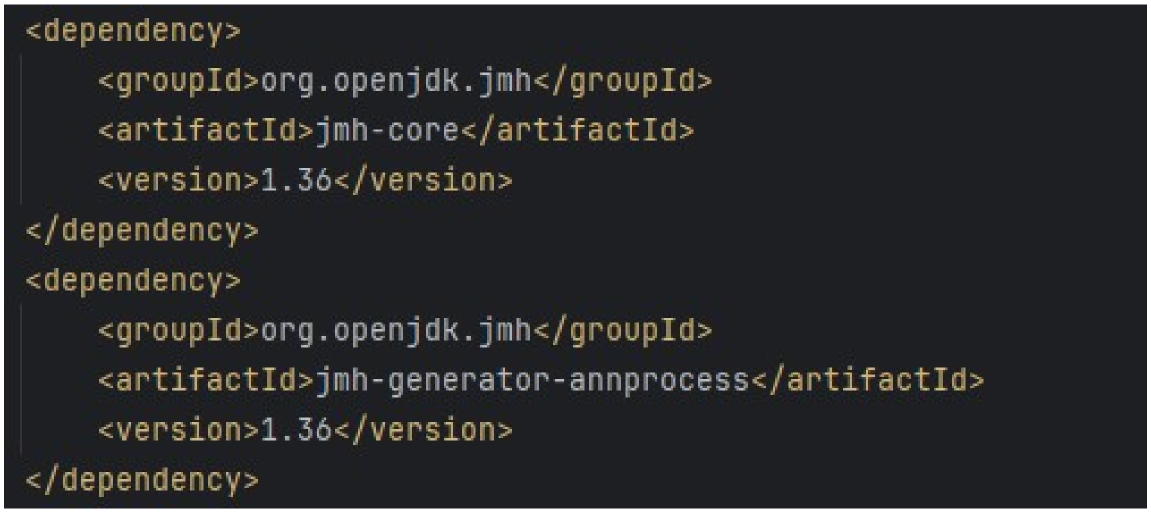
6 JAVA MICROBENCHMARK HARNESS

In this section, we are going to write simple performance tests with JMH (Java Microbenchmark Harness), we will look at the some of the most greedy methods of our project and apply some micro-benchmarks to stress them.

6.1 Installation and Configuration

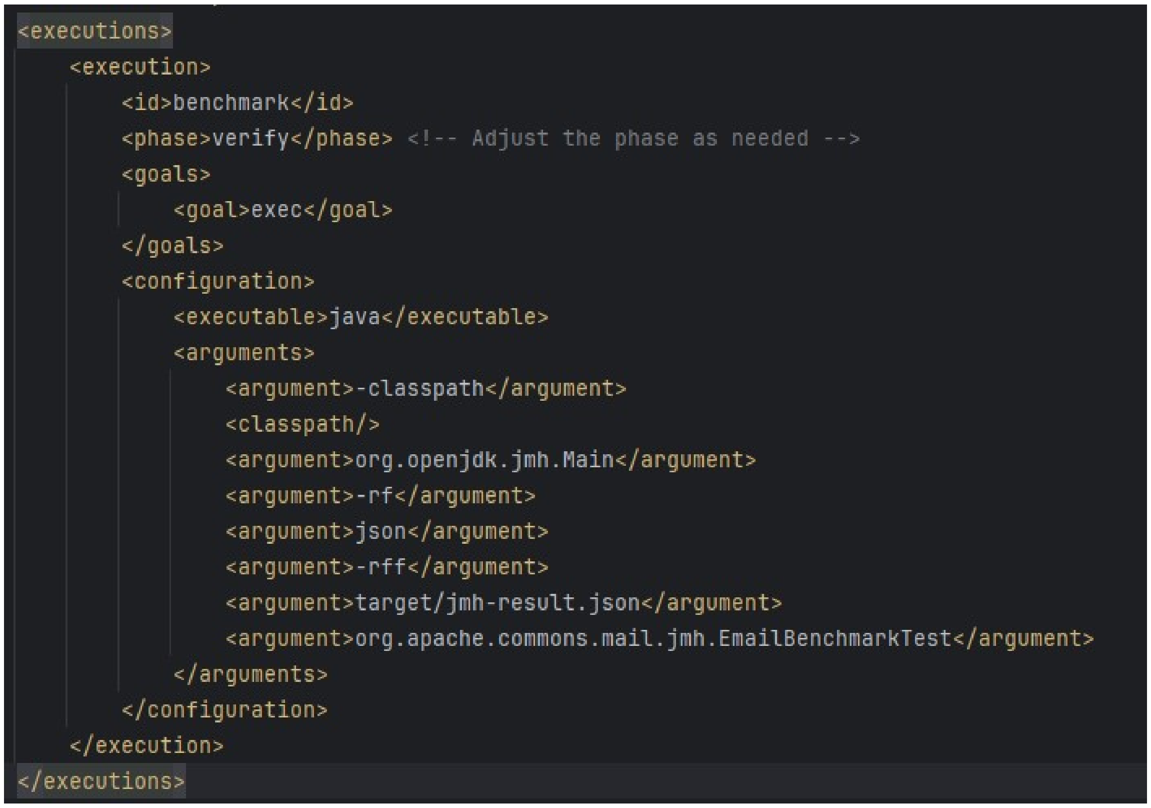
Before we write and run our code, we must install JMH plugin, we installed our JMH plugin using Maven:

After installing the JMH dependency, to specify the class for running the performance test and storing the results in a JSON format file we must configure:



```xml
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-core</artifactId>
    <version>1.36</version>
</dependency>
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-generator-annprocess</artifactId>
    <version>1.36</version>
</dependency>
```

Fig. 6. JDK Dependency installation.



```xml
<executions>
    <execution>
        <id>benchmark</id>
        <phase>verify</phase> <!-- Adjust the phase as needed -->
        <goals>
            <goal>exec</goal>
        </goals>
        <configuration>
            <executable>java</executable>
            <arguments>
                <argument>-classpath</argument>
                <classpath/>
                <argument>org.openjdk.jmh.Main</argument>
                <argument>-rf</argument>
                <argument>json</argument>
                <argument>-rff</argument>
                <argument>target/jmh-result.json</argument>
                <argument>org.apache.commons.mail.jmh.EmailBenchmarkTest</argument>
            </arguments>
        </configuration>
    </execution>
</executions>
```

Fig. 7. JDK benchmark configuration.

6.2 Writing Benchmarking Code

Before designing some simple performance tests, we must look for some of the most greedy methods in this project:

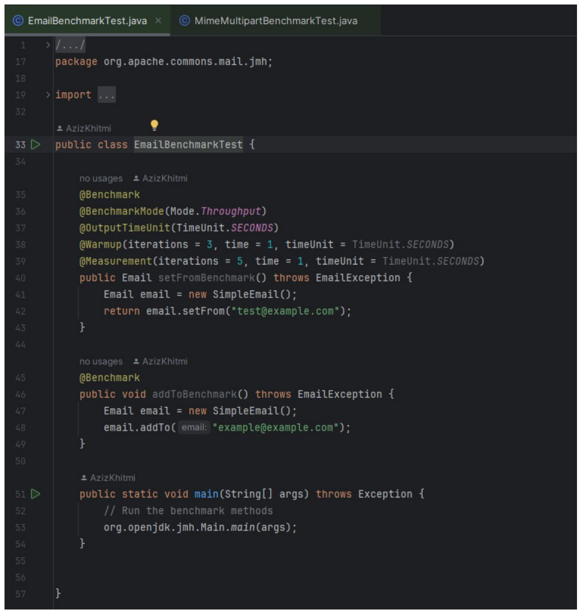
(1) **MimeMultipart.addBodyPart(Part p)** This method adds a new body part to the MimeMultipart. The body part can be a text part, a multipart, or an attachment. This method is greedy because it will add the body part to the end of the MimeMultipart, even if the MimeMultipart already contains other body parts. This method is located in the ***Multipart*** class. It is used to add a new body part to the Multipart. The body part can be a text part, a multipart, or an attachment.

(2) **MimeBodyPart.setText(String text)** This method sets the text content of the MimeBodyPart. The text content is the plain text that will be displayed in the email message. This method is greedy because it will replace the existing text content, even if the MimeBodyPart already has text content. This method is located in the ***MimeBodyPart*** class. It is used to set the text content of the MimeBodyPart. The text content is the plain text that will be displayed in the email message.

(3) **Email.setFrom(Address address)** This method sets the from address of the email message. The from address is the address that will be displayed as the sender of the email message. This method is greedy because it will replace the existing from address, even if the email message already has a from address. This method is located in the ***Email*** class. It is used to set the from address of the email message. The from address is the address that is displayed as the sender of the email message.

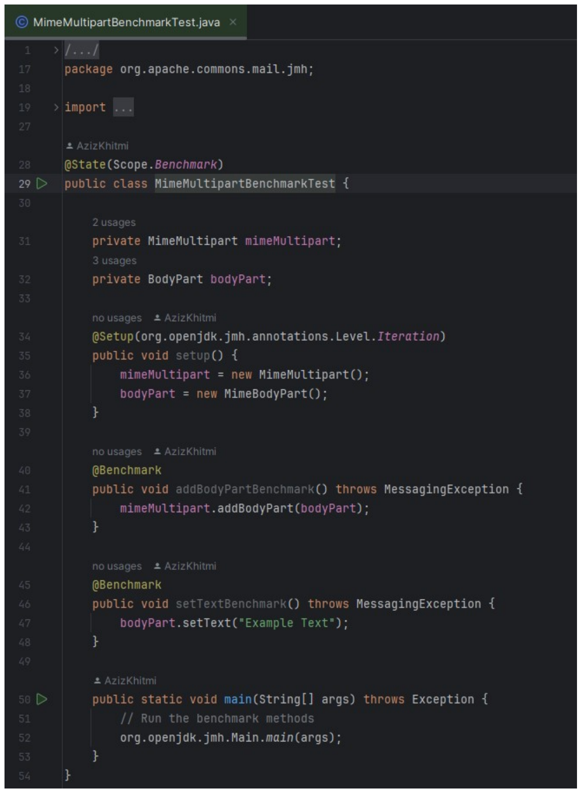
(4) Email.addTo(Address address) This method adds a new recipient to the email message. The recipient type can be TO, CC, or BCC. This method is greedy because it will add the recipient to the end of the list of recipients, even if the email message already has recipients of the same type. This method is also located in the ***Email*** class. It is used to add a new recipient to the email message. The recipient type can be TO, CC, or BCC.

After determining which greedy methods we want to run performance tests on, we have written two separate test code files, **'EmailBenchmarkTest.java'** for testing the greedy methods mentioned in the **'Email.java'** class, and **'MimeMultipartBenchmarkTest.java'** for testing the greedy methods mentioned in **'MultiPartEmail.java'** as follows:


EmailBenchmarkTest.java × MimeMultipartBenchmarkTest.java

```java
/.../
package org.apache.commons.mail.jmh;

import ...

AzizKhitmi
public class EmailBenchmarkTest {

    no usages  AzizKhitmi
    @Benchmark
    @BenchmarkMode(Mode.Throughput)
    @OutputTimeUnit(TimeUnit.SECONDS)
    @Warmup(iterations = 3, time = 1, timeUnit = TimeUnit.SECONDS)
    @Measurement(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS)
    public Email setFromBenchmark() throws EmailException {
        Email email = new SimpleEmail();
        return email.setFrom("test@example.com");
    }

    no usages  AzizKhitmi
    @Benchmark
    public void addToBenchmark() throws EmailException {
        Email email = new SimpleEmail();
        email.addTo( email: "example@example.com");
    }

    AzizKhitmi
    public static void main(String[] args) throws Exception {
        // Run the benchmark methods
        org.openjdk.jmh.Main.main(args);
    }


}
```

Fig. 8. Benchmark code for EmailBenchmarkTest.java


MimeMultipartBenchmarkTest.java ×

```java
/.../
package org.apache.commons.mail.jmh;

import ...

AzizKhitmi
@State(Scope.Benchmark)
public class MimeMultipartBenchmarkTest {

    2 usages
    private MimeMultipart mimeMultipart;
    3 usages
    private BodyPart bodyPart;

    no usages  AzizKhitmi
    @Setup(org.openjdk.jmh.annotations.Level.Iteration)
    public void setup() {
        mimeMultipart = new MimeMultipart();
        bodyPart = new MimeBodyPart();
    }

    no usages  AzizKhitmi
    @Benchmark
    public void addBodyPartBenchmark() throws MessagingException {
        mimeMultipart.addBodyPart(bodyPart);
    }

    no usages  AzizKhitmi
    @Benchmark
    public void setTextBenchmark() throws MessagingException {
        bodyPart.setText("Example Text");
    }

    AzizKhitmi
    public static void main(String[] args) throws Exception {
        // Run the benchmark methods
        org.openjdk.jmh.Main.main(args);
    }
}
```

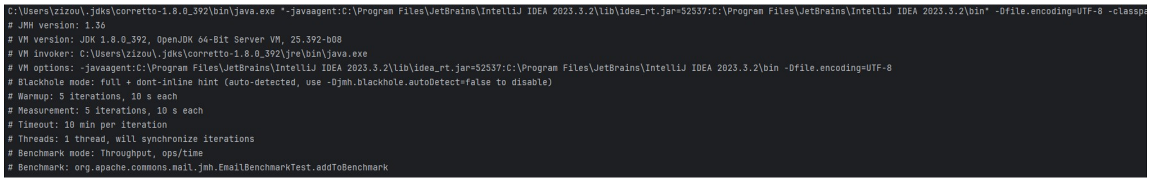
Fig. 9. Benchmark code for MultiPartEmail.java

- **@Benchmark:** This annotation is used to mark a method as a benchmark that should be measured. JMH will treat methods annotated with @Benchmark as the target code for performance measurement.
- **@State:** The @State annotation is used to specify the scope of the state object for benchmark methods. It can be either Scope.Benchmark (shared among all threads) or Scope.Thread (each thread gets its own instance).
- **@Warmup and @Measurement:** These annotations allow you to control the warm-up and measurement iterations for your benchmarks. You can specify the number of iterations, time, or operations for warm-up and measurement phases.

6.3 Results

Running JMH tests for performance involves benchmarking our Java code to measure its execution time, throughput, and other relevant performance metrics. The process involves careful design of benchmark methods using JMH annotations, execution of the benchmarks, and analysis of the results.

These were the results after running our JMH performance test:

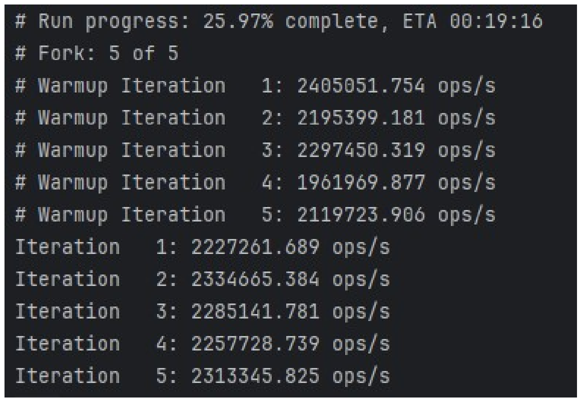


```
C:\Users\zizou\.jdks\corretto-1.8.0_392\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.2\lib\idea_rt.jar=52537:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.2\bin" -Dfile.encoding=UTF-8 -classpa
# JMH version: 1.36
# VM version: JDK 1.8.0_392, OpenJDK 64-Bit Server VM, 25.392-b08
# VM invoker: C:\Users\zizou\.jdks\corretto-1.8.0_392\jre\bin\java.exe
# VM options: -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.2\lib\idea_rt.jar=52537:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.2\bin -Dfile.encoding=UTF-8
# Blackhole mode: full + dont-inline hint (auto-detected, use -Djmh.blackhole.autoDetect=false to disable)
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: org.apache.commons.mail.jmh.EmailBenchmarkTest.addToBenchmark
```

Fig. 10. JMH startup.

(1) **Benchmark Execution:** JMH executes the benchmark methods that you've annotated with @Benchmark. Each benchmark is run multiple times to ensure accurate measurements.
(2) **Warm-up Iterations:** JMH performs warm-up iterations before the actual measurements. During warm-up, the JVM has an opportunity to optimize the code and reach a stable state. The warm-up helps eliminate the impact of JIT (Just-In-Time) compilation and other factors that can affect the initial performance.
(3) **Measurement Iterations:** After warm-up, JMH executes the benchmark methods for the specified number of measurement iterations. These iterations are the actual measurements used for performance analysis.
(4) **Metrics and Statistics:** JMH collects various metrics, including average execution time, throughput, and other statistical data. These metrics provide insights into the performance characteristics of your code.

In JMH (Java Microbenchmarking Harness), the run progress results include various parameters that provide information about how the benchmark is executed:

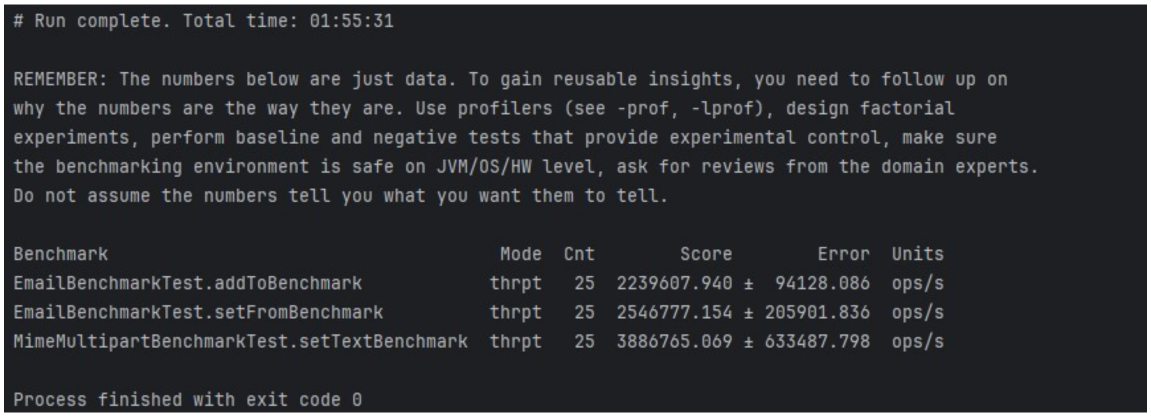


```
# Run progress: 25.97% complete, ETA 00:19:16
# Fork: 5 of 5
# Warmup Iteration   1: 2405051.754 ops/s
# Warmup Iteration   2: 2195399.181 ops/s
# Warmup Iteration   3: 2297450.319 ops/s
# Warmup Iteration   4: 1961969.877 ops/s
# Warmup Iteration   5: 2119723.906 ops/s
Iteration   1: 2227261.689 ops/s
Iteration   2: 2334665.384 ops/s
Iteration   3: 2285141.781 ops/s
Iteration   4: 2257728.739 ops/s
Iteration   5: 2313345.825 ops/s
```

Fig. 11. JMH run progress.

(1) **Fork (5 of 5)**: JMH runs benchmarks in multiple forks to account for JVM warm-up effects and to reduce the impact of external factors on the results.
(2) **Warmup Iteration**: Warmup iterations are the initial iterations of the benchmark that are used to allow the JVM to reach a stable state before actual measurements begin.
(3) **Iteration (5)**: The total number of iterations for the benchmark. After the warmup iterations, JMH performs the specified number of iterations to measure the actual performance of the benchmark.

After our run progress is complete, we will receive a breakdown of the results returned by our benchmark performance test:



```
# Run complete. Total time: 01:55:31

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

Benchmark                                Mode  Cnt        Score        Error  Units
EmailBenchmarkTest.addToBenchmark       thrpt   25  2239607.940 ±  94128.086  ops/s
EmailBenchmarkTest.setFromBenchmark     thrpt   25  2546777.154 ± 205901.836  ops/s
MimeMultipartBenchmarkTest.setTextBenchmark  thrpt   25  3886765.069 ± 633487.798  ops/s

Process finished with exit code 0
```

Fig. 12. JMH benchmark results.

- **Benchmark:** The name of the benchmark method or class.
- **Mode:** The mode in which the benchmark is run. Common modes include "avgt" (average time) and "thrpt" (throughput).
- **Cnt (25):** The number of forks. In this example, the benchmark is executed twenty-five times.
- **Score:** The average time or throughput score for the benchmark.

- **Error:** The error margin associated with the score.
- **Units (s/op):** The units in which the score is measured (seconds per operation in this case).

7 EVOSUITE

In this section, we are going to run a Evosuite test.

Installation: To set up EvoSuite, We initiated the process by cloning the EvoSuite repository onto my system. Following the cloning step, We manually added the evosuite-1.2.0.jar file to the EvoSuite folder to meet the necessary requirements.

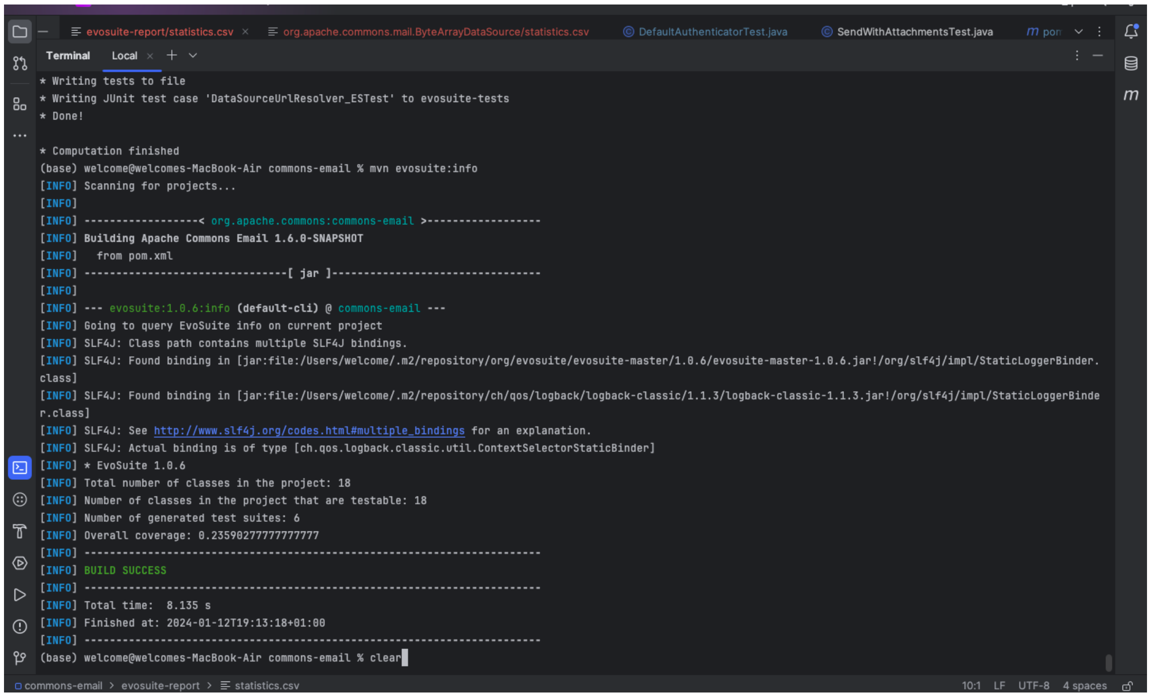
Test Generation: Once the installation was completed successfully, We utilized EvoSuite to generate tests. We selected the target class for test generation, opting for fileupload.java in this instance. In the command, We specified the path to the evosuite-1.2.0.jar, the class Fully Qualified Name (FQN), and the project path. As a result, the generated tests were stored under the /evosuite-test/ directory.

Compilation: To make these generated tests usable, compilation was necessary. EvoSuite tests have dependencies on JUnit 4+ and Hamcrest 1.3+. To fulfill these dependencies, We manually downloaded the required dependencies, such as JUnit 5.9.1 and Hamcrest 2.2, from the Maven Central Repository. We then compiled the tests using the appropriate paths, resulting in the generation of .class files stored in the same folder as the corresponding Java file.

Here is the command we use to run the test:

java -jar tools/evosuite-1.2.0.jar -generateTests -target target/classes

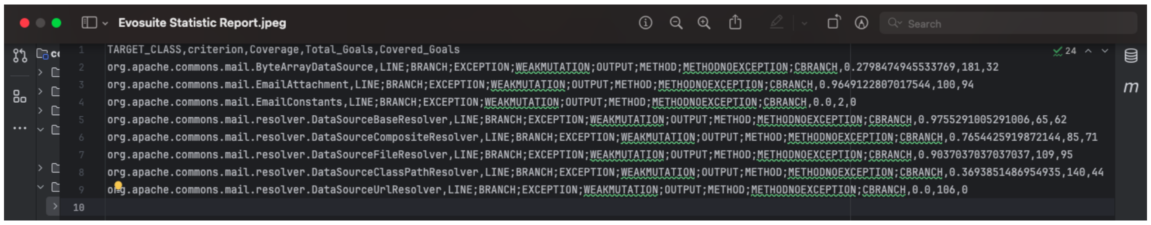
7.1 Terminal Result



```
* Writing tests to file
* Writing JUnit test case 'DataSourceUrlResolver_ESTest' to evosuite-tests
* Done!

* Computation finished
(base) welcome@welcomes-MacBook-Air commons-email % mvn evosuite:info
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< org.apache.commons:commons-email >-----------------
[INFO] Building Apache Commons Email 1.6.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- evosuite:1.0.6:info (default-cli) @ commons-email ---
[INFO] Going to query EvoSuite info on current project
[INFO] SLF4J: Class path contains multiple SLF4J bindings.
[INFO] SLF4J: Found binding in [jar:file:/Users/welcome/.m2/repository/org/evosuite/evosuite-master/1.0.6/evosuite-master-1.0.6.jar!/org/slf4j/impl/StaticLoggerBinder.
class]
[INFO] SLF4J: Found binding in [jar:file:/Users/welcome/.m2/repository/ch/qos/logback/logback-classic/1.1.3/logback-classic-1.1.3.jar!/org/slf4j/impl/StaticLoggerBinde
r.class]
[INFO] SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
[INFO] SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]
[INFO] * EvoSuite 1.0.6
[INFO] Total number of classes in the project: 18
[INFO] Number of classes in the project that are testable: 18
[INFO] Number of generated test suites: 6
[INFO] Overall coverage: 0.23590277777777777
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.135 s
[INFO] Finished at: 2024-01-12T19:13:18+01:00
[INFO] ------------------------------------------------------------------------
(base) welcome@welcomes-MacBook-Air commons-email % clear
```

Fig. 13. Test case Generation via Evosuite

After running the command we get a folder of statistics report in Evosuite folder.



```
TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals
org.apache.commons.mail.ByteArrayDataSource,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.2798474945533769,181,32
org.apache.commons.mail.EmailAttachment,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9649122807017544,100,94
org.apache.commons.mail.EmailConstants,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.0,2,0
org.apache.commons.mail.resolver.DataSourceBaseResolver,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9755291005291006,65,62
org.apache.commons.mail.resolver.DataSourceCompositeResolver,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.7654425919872144,85,71
org.apache.commons.mail.resolver.DataSourceFileResolver,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9037037037037037,109,95
org.apache.commons.mail.resolver.DataSourceClassPathResolver,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.3693851486954935,140,44
org.apache.commons.mail.resolver.DataSourceUrlResolver,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.0,106,0
```

Fig. 14. Results Generated

7.1.1 *Statistic report.*

The provided statistics report from EvoSuite includes coverage metrics for different classes in the Apache Commons Email library. Here's a brief explanation of the key fields:

- TARGETCLASS: org.apache.commons.mail.ByteArrayDataSource
- criterion: LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH
- Coverage: 27.98
- TotalGoals: 181

- CoveredGoals: 32

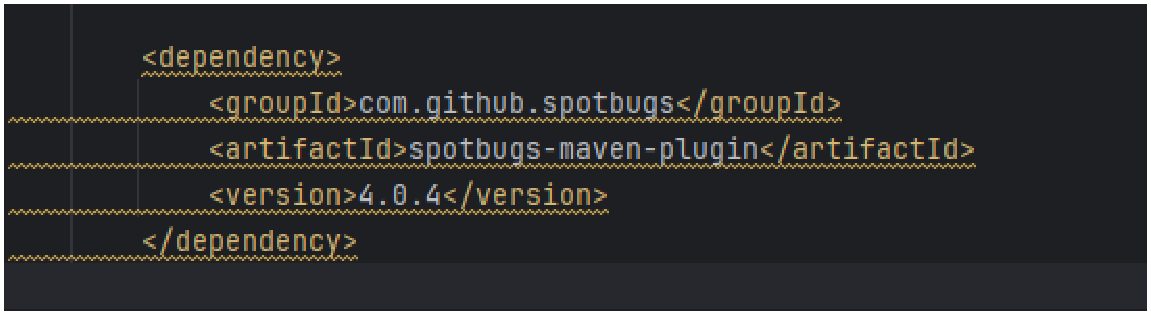
This suggests that EvoSuite, based on the applied coverage criteria, achieved approximately 27.98 coverage for the ByteArrayDataSource class, covering 32 out of 181 coverage goals. Similar interpretations can be made for the other classes listed in the report. The coverage percentages provide insights into how well the automatically generated tests cover different aspects of the code.

8 STATIC SECURITY ANALYSIS

In this last section, we focus on finding security bugs using "FindSecBugs" and detect dependency versions having vulnerabilities using "OWASP DC".

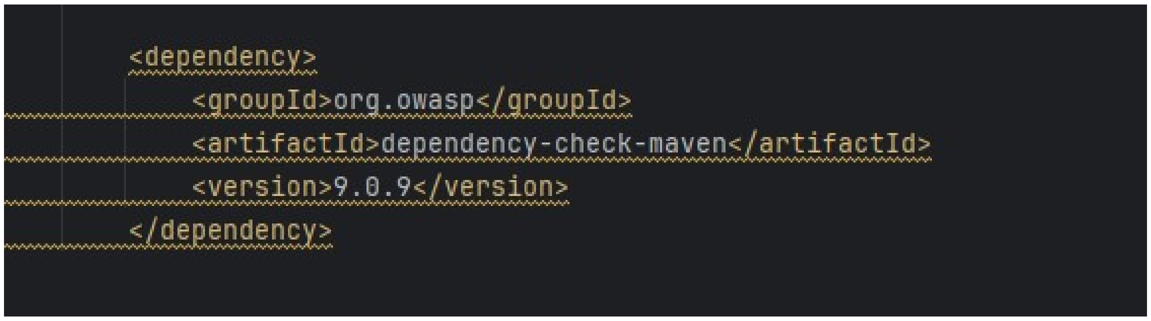
8.1 Installation and Configuration

First, we install the dependencies for "FindSecBugs" and "OWASP DC" through pom.xml (using Maven), as follows:



```
    <dependency>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>4.0.4</version>
    </dependency>
```

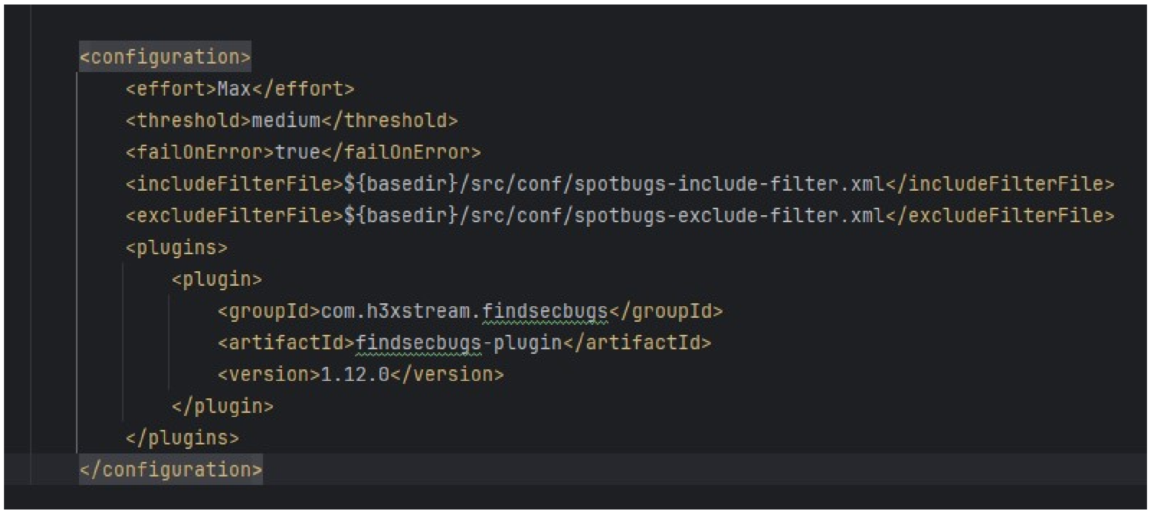
Fig. 15. Installation of "FindSecBugs"



```
    <dependency>
        <groupId>org.owasp</groupId>
        <artifactId>dependency-check-maven</artifactId>
        <version>9.0.9</version>
    </dependency>
```

Fig. 16. Installation of OWASP DC.

For FindSecBugs tool, we write a configuration that is designed to enhance static analysis of the codebase by using both SpotBugs and Find Security Bugs plugins, and it provides fine-grained control over the analysis settings and filters, as the figure below:



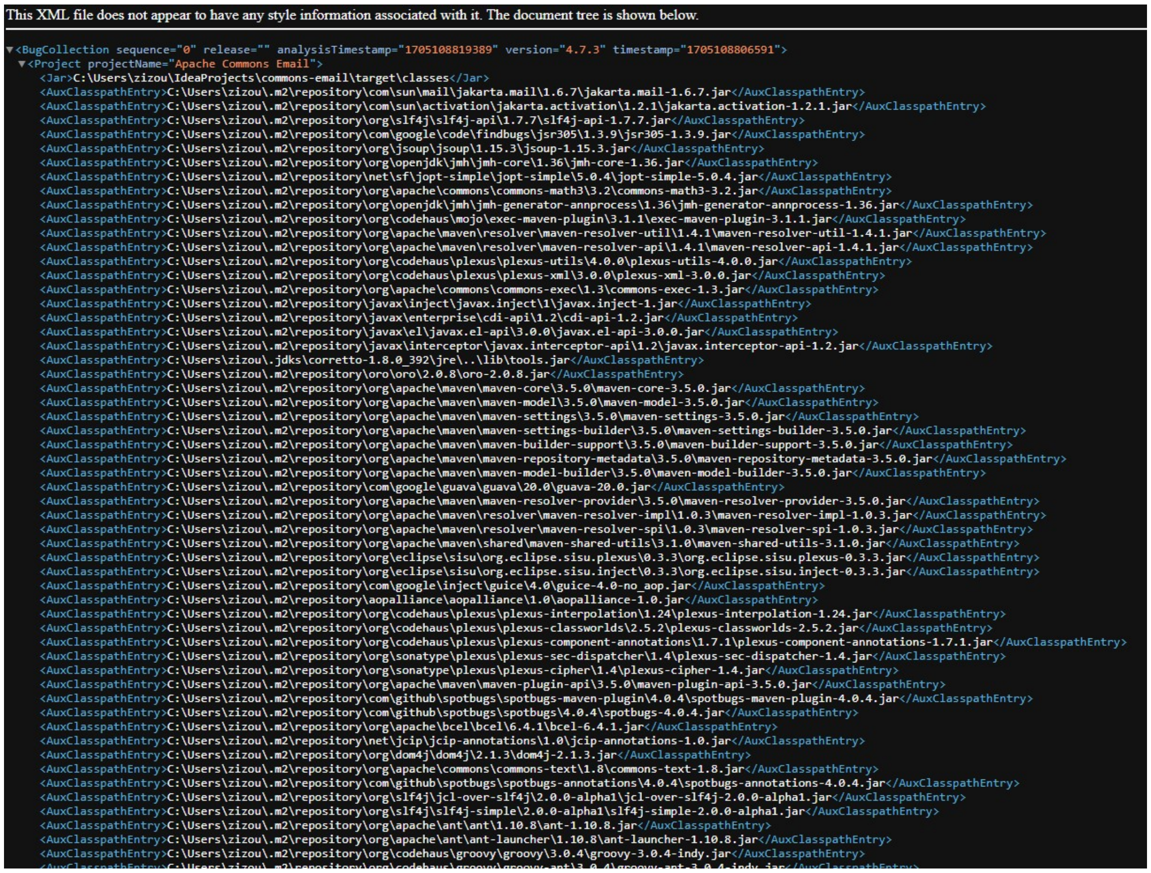
```
<configuration>
    <effort>Max</effort>
    <threshold>medium</threshold>
    <failOnError>true</failOnError>
    <includeFilterFile>${basedir}/src/conf/spotbugs-include-filter.xml</includeFilterFile>
    <excludeFilterFile>${basedir}/src/conf/spotbugs-exclude-filter.xml</excludeFilterFile>
    <plugins>
        <plugin>
            <groupId>com.h3xstream.findsecbugs</groupId>
            <artifactId>findsecbugs-plugin</artifactId>
            <version>1.12.0</version>
        </plugin>
    </plugins>
</configuration>
```

Fig. 17. FindSecBugs Configuration.

(1) `<effort>Max</effort>`: Specifies the analysis effort level. In this case, it's set to "Max," indicating that the analysis should use the maximum effort to find potential issues.
(2) `<threshold>medium</threshold>`: Sets the threshold for reporting issues. Issues with severity equal to or higher than the specified threshold will be reported. In this case, the threshold is set to "medium."
(3) `<failOnError>true</failOnError>`: If set to `true`, the build will fail if any bugs or issues are found during the analysis. This is a way to enforce code quality standards.
(4) `<includeFilterFile>${basedir}/src/conf/spotbugs-include-filter.xml</includeFilterFile>`: Specifies the path to an XML file containing include filters. Include filters define which classes or packages to include in the analysis. In this case, it's pointing to a file named `spotbugs-include-filter.xml` located in the `src/conf` directory.
(5) `<excludeFilterFile>${basedir}/src/conf/spotbugs-exclude-filter.xml</excludeFilterFile>`: Similar to the include filter, this specifies the path to an XML file containing exclude filters. Exclude filters define which classes or packages to exclude from the analysis.

8.2 Results

8.2.1 ***FindSecBugs.*** Now that we installed FindSecBugs tool, we can run our results either using Maven command "**mvn spotbugs:spotbugs**" for results generated in XML file or "**mvn spotbugs:gui**" for results shown in GUI, below is a figure presenting the results in XML file:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
▼<BugCollection sequence="0" release="" analysisTimestamp="1705108819389" version="4.7.3" timestamp="1705108806591">
  ▼<Project projectName="Apache Commons Email">
     <Jar>C:\Users\zizou\IdeaProjects\commons-email\target\classes</Jar>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\sun\mail\jakarta.mail\1.6.7\jakarta.mail-1.6.7.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\sun\activation\jakarta.activation\1.2.1\jakarta.activation-1.2.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\slf4j\slf4j-api\1.7.7\slf4j-api-1.7.7.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\google\code\findbugs\jsr305\1.3.9\jsr305-1.3.9.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\jsoup\jsoup\1.15.3\jsoup-1.15.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\openjdk\jmh\jmh-core\1.36\jmh-core-1.36.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\net\sf\jopt-simple\jopt-simple\5.0.4\jopt-simple-5.0.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\commons\commons-math3\3.2\commons-math3-3.2.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\openjdk\jmh\jmh-generator-annprocess\1.36\jmh-generator-annprocess-1.36.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\mojo\exec-maven-plugin\3.1.1\exec-maven-plugin-3.1.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\resolver\maven-resolver-util\1.4.1\maven-resolver-util-1.4.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\resolver\maven-resolver-api\1.4.1\maven-resolver-api-1.4.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\plexus\plexus-utils\4.0.0\plexus-utils-4.0.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\plexus\plexus-xml\3.0.0\plexus-xml-3.0.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\commons\commons-exec\1.3\commons-exec-1.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\javax\inject\javax.inject\1\javax.inject-1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\javax\enterprise\cdi-api\1.2\cdi-api-1.2.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\javax\el\javax.el-api\3.0.0\javax.el-api-3.0.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\javax\interceptor\javax.interceptor-api\1.2\javax.interceptor-api-1.2.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.jdks\corretto-1.8.0_392\jre\..\lib\tools.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\oro\oro\2.0.8\oro-2.0.8.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-core\3.5.0\maven-core-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-model\3.5.0\maven-model-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-settings\3.5.0\maven-settings-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-settings-builder\3.5.0\maven-settings-builder-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-builder-support\3.5.0\maven-builder-support-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-repository-metadata\3.5.0\maven-repository-metadata-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-model-builder\3.5.0\maven-model-builder-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\google\guava\guava\20.0\guava-20.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-resolver-provider\3.5.0\maven-resolver-provider-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\resolver\maven-resolver-impl\1.0.3\maven-resolver-impl-1.0.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\resolver\maven-resolver-spi\1.0.3\maven-resolver-spi-1.0.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\shared\maven-shared-utils\3.1.0\maven-shared-utils-3.1.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\eclipse\sisu\org.eclipse.sisu.plexus\0.3.3\org.eclipse.sisu.plexus-0.3.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\eclipse\sisu\org.eclipse.sisu.inject\0.3.3\org.eclipse.sisu.inject-0.3.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\google\inject\guice\4.0\guice-4.0-no_aop.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\aopalliance\aopalliance\1.0\aopalliance-1.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\plexus\plexus-interpolation\1.24\plexus-interpolation-1.24.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\plexus\plexus-classworlds\2.5.2\plexus-classworlds-2.5.2.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\plexus\plexus-component-annotations\1.7.1\plexus-component-annotations-1.7.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\sonatype\plexus\plexus-sec-dispatcher\1.4\plexus-sec-dispatcher-1.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\sonatype\plexus\plexus-cipher\1.4\plexus-cipher-1.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\maven\maven-plugin-api\3.5.0\maven-plugin-api-3.5.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\github\spotbugs\spotbugs-maven-plugin\4.0.4\spotbugs-maven-plugin-4.0.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\github\spotbugs\spotbugs\4.0.4\spotbugs-4.0.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\bcel\bcel\6.4.1\bcel-6.4.1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\net\jcip\jcip-annotations\1.0\jcip-annotations-1.0.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\dom4j\dom4j\2.1.3\dom4j-2.1.3.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\commons\commons-text\1.8\commons-text-1.8.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\com\github\spotbugs\spotbugs-annotations\4.0.4\spotbugs-annotations-4.0.4.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\slf4j\jcl-over-slf4j\2.0.0-alpha1\jcl-over-slf4j-2.0.0-alpha1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\slf4j\slf4j-simple\2.0.0-alpha1\slf4j-simple-2.0.0-alpha1.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\ant\ant\1.10.8\ant-1.10.8.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\apache\ant\ant-launcher\1.10.8\ant-launcher-1.10.8.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\groovy\groovy\3.0.4\groovy-3.0.4-indy.jar</AuxClasspathEntry>
     <AuxClasspathEntry>C:\Users\zizou\.m2\repository\org\codehaus\groovy\groovy-ant\3.0.4\groovy-ant-3.0.4-indy.jar</AuxClasspathEntry>
```

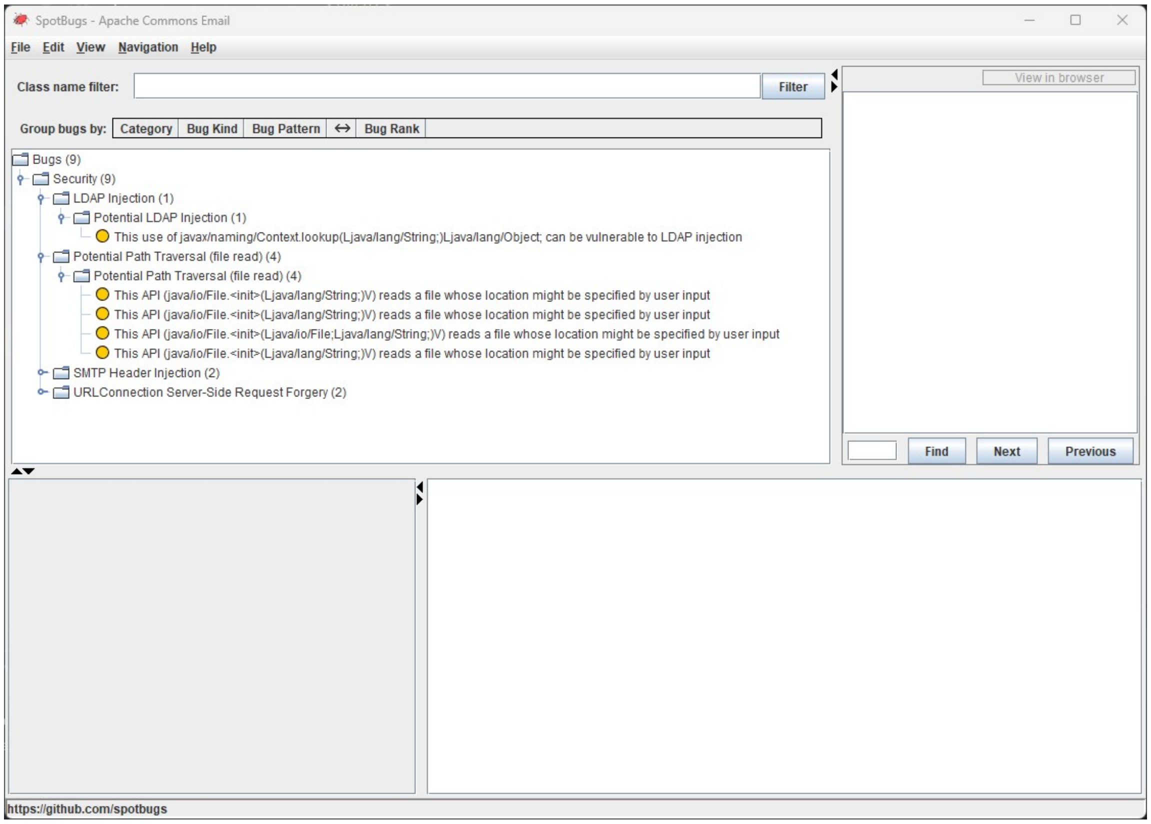
Fig. 18. Spotbugs results in XML.

These are some results we can find on the XML file:

(1) **Bug Instances:** Two instances of the bug category "SECURITY" related to "Potential Path Traversal (file read)" have been identified. The instances provide details about the location in the code, the method involved, and information about the potential vulnerability.
(2) **Bug Categories:** The file includes a bug category related to "SECURITY" with a general description.
(3) **Bug Patterns:** The file defines several bug patterns related to security issues, such as LDAP Injection, SMTP Header Injection, Potential Path Traversal, and URLConnection Server-Side Request Forgery (SSRF) and File Disclosure. Each bug pattern includes a short description, details about the vulnerability, vulnerable code examples, and potential solutions or countermeasures.
(4) **Bug Codes:** Bug codes are associated with specific bug patterns, providing a standardized way to reference and categorize them.
(5) **Errors:** There's information about missing classes (1 missing class named "apply").
(6) **FindBugs Summary:** A summary of the analysis is provided, including the number of packages, total classes, priority 2 bugs, total bugs, and other relevant statistics.

(7) **File Stats and Package Stats:** Detailed statistics are provided for each file and package, including bug counts, file sizes, and other metrics.

We can also interpret the results from a GUI:



```
Bugs (9)
  Security (9)
    LDAP Injection (1)
      Potential LDAP Injection (1)
        This use of javax/naming/Context.lookup(Ljava/lang/String;)Ljava/lang/Object; can be vulnerable to LDAP injection
    Potential Path Traversal (file read) (4)
      Potential Path Traversal (file read) (4)
        This API (java/io/File.<init>(Ljava/lang/String;)V) reads a file whose location might be specified by user input
        This API (java/io/File.<init>(Ljava/lang/String;)V) reads a file whose location might be specified by user input
        This API (java/io/File.<init>(Ljava/io/File;Ljava/lang/String;)V) reads a file whose location might be specified by user input
        This API (java/io/File.<init>(Ljava/lang/String;)V) reads a file whose location might be specified by user input
    SMTP Header Injection (2)
    URLConnection Server-Side Request Forgery (2)
```

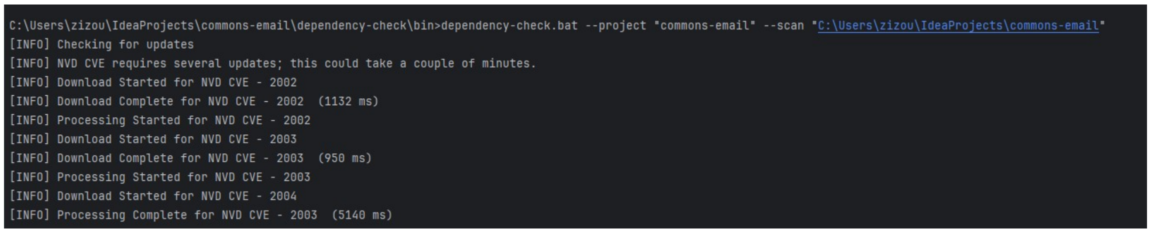
Fig. 19. Spotbugs results in GUI.

The SpotBugs results in Apache Common Emails indicate the presence of various types of bugs, with a focus on security-related issues. Here's a breakdown of the findings:

(1) **Total Bugs: 9** - The tool identified a total of 9 bugs in the codebase.
(2) **Security Issues: 9** - All the identified bugs are categorized as security issues, emphasizing the importance of addressing these concerns promptly.
(3) **Specific Security Vulnerabilities:**
   - LDAP Injection (1): One bug relates to LDAP injection, a type of security vulnerability where untrusted input can manipulate LDAP queries.
   - Potential Path Traversal (1): Another bug involves a potential path traversal issue, indicating a risk of unauthorized access to file system paths.
   - SMTP Header Injection (2): Two bugs are related to SMTP header injection, which can lead to email-based attacks by manipulating email headers.

- URL Connection Server-Side Request Forgery (2): Two bugs point to URL connection server-side request forgery, highlighting potential risks related to manipulating server-side requests through URL connections.

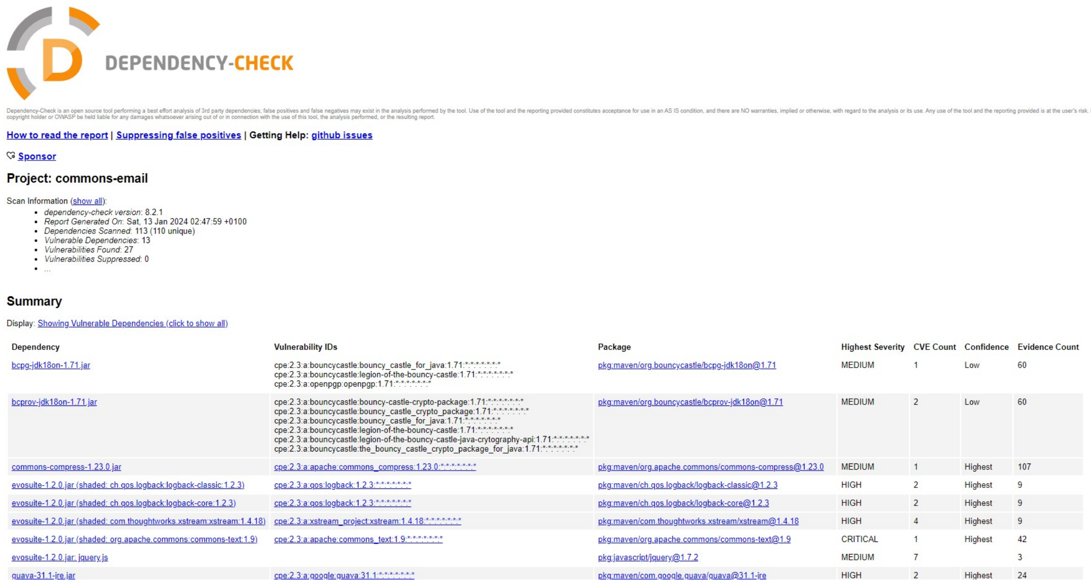
8.2.2 ***Dependency Check***. After installing OWASP DC, we can run our dependency check using the bat file, like shown in the figure:



```
C:\Users\zizou\IdeaProjects\commons-email\dependency-check\bin>dependency-check.bat --project "commons-email" --scan "C:\Users\zizou\IdeaProjects\commons-email"
[INFO] Checking for updates
[INFO] NVD CVE requires several updates; this could take a couple of minutes.
[INFO] Download Started for NVD CVE - 2002
[INFO] Download Complete for NVD CVE - 2002  (1132 ms)
[INFO] Processing Started for NVD CVE - 2002
[INFO] Download Started for NVD CVE - 2003
[INFO] Download Complete for NVD CVE - 2003  (950 ms)
[INFO] Processing Started for NVD CVE - 2003
[INFO] Download Started for NVD CVE - 2004
[INFO] Processing Complete for NVD CVE - 2003  (5140 ms)
```

Fig. 20. Dependency Check scan on common-emails.

After running a scan, Dependency-Check generates a report highlighting any identified vulnerabilities, including details about the vulnerabilities, affected components, and recommended remediation:



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: commons-email

Scan Information (show all):
- *dependency-check version*: 8.2.1
- *Report Generated On*: Sat, 13 Jan 2024 02:47:59 +0100
- *Dependencies Scanned*: 113 (110 unique)
- *Vulnerable Dependencies*: 13
- *Vulnerabilities Found*: 27
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bcpg-jdk18on-1.71.jar	cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle:1.71:*:*:*:*:*:*:* cpe:2.3:a:openpgp:openpgp:1.71:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcpg-jdk18on@1.71	MEDIUM	1	Low	60
bcprov-jdk18on-1.71.jar	cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:the_bouncy_castle_crypto_package_for_java:1.71:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcprov-jdk18on@1.71	MEDIUM	2	Low	60
commons-compress-1.23.0.jar	cpe:2.3:a:apache:commons_compress:1.23.0:*:*:*:*:*:*:*	pkg:maven/org.apache.commons/commons-compress@1.23.0	MEDIUM	1	Highest	107
evosuite-1.2.0.jar (shaded: ch.qos.logback:logback-classic:1.2.3)	cpe:2.3:a:qos:logback:1.2.3:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-classic@1.2.3	HIGH	2	Highest	9
evosuite-1.2.0.jar (shaded: ch.qos.logback:logback-core:1.2.3)	cpe:2.3:a:qos:logback:1.2.3:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.2.3	HIGH	2	Highest	9
evosuite-1.2.0.jar (shaded: com.thoughtworks.xstream:xstream:1.4.18)	cpe:2.3:a:xstream_project:xstream:1.4.18:*:*:*:*:*:*:*	pkg:maven/com.thoughtworks.xstream/xstream@1.4.18	HIGH	4	Highest	9
evosuite-1.2.0.jar (shaded: org.apache.commons:commons-text:1.9)	cpe:2.3:a:apache:commons_text:1.9:*:*:*:*:*:*:*	pkg:maven/org.apache.commons/commons-text@1.9	CRITICAL	1	Highest	42
evosuite-1.2.0.jar: jquery.js		pkg:javascript/jquery@1.7.2	MEDIUM	7		3
guava-31.1-jre.jar	cpe:2.3:a:google:guava:31.1:*:*:*:*:*:*:*	pkg:maven/com.google.guava/guava@31.1-jre	HIGH	2	Highest	24

Fig. 21. Dependency-Check report.

The Dependency-Check report typically includes information about the identified vulnerabilities in the project's dependencies, we can note some of the most common elements found in a Dependency-Check report:

(1) Vulnerability Details: The scan has identified several dependencies with known vulnerabilities, which are categorized by severity, confidence, and evidence count. The summary table encapsulates the critical data points associated with each vulnerable dependency:

- The dependencies ***bcpg-jdk18on-1.71.jar*** and ***bcprov-jdk18on-1.71.jar*** are reported with medium severity vulnerabilities, albeit with low confidence. The evidence count is significantly high, suggesting multiple references that corroborate the findings.
- A critical vulnerability has been detected within ***commons-compress-1.23.0.jar***, which carries the highest confidence level. The CVE count for this library is 1, implying a singular but critical exposure.
- Dependencies such as ***evosuite-1.2.0.jar*** manifest multiple vulnerabilities with high severity and the highest confidence level. These vulnerabilities are supported by a lower evidence count but represent a significant security risk.

(2) Recommendations: Based on the Dependency-Check findings, it is recommended that we:
- Prioritize the review of dependencies marked with 'Critical' and 'High' severity.
- Validate the vulnerabilities with 'Low' confidence ratings against additional security sources to ascertain their relevance and potential impact.
- Update or replace vulnerable dependencies with secure versions or alternatives where possible.
- Implement a continuous monitoring strategy for the project's dependencies to proactively manage emerging vulnerabilities.

(3) Risk Metrics:
- CVSS Score: Common Vulnerability Scoring System (CVSS) score, providing a numeric representation of the vulnerability's severity.
- CWE: Common Weakness Enumeration identifier, categorizing the type of vulnerability.

(4) Summary Information:
- Total Number of Vulnerabilities: An overview of the total number of vulnerabilities found in the project.
- Breakdown by Severity: A summary categorizing vulnerabilities by severity levels.

(5) Suppression and False Positives:
- Suppressed Findings: Information about any vulnerabilities that have been marked as false positives or deliberately suppressed.

In conclusion, the incorporation of FindSecBugs and Dependency-Check enhances our software dependability. FindSecBugs identifies code-level vulnerabilities early on, with customizable settings for focused analysis. Dependency-Check ensures the security of third-party components by providing actionable reports on vulnerabilities. Together, they establish a proactive security framework, reducing the risk of breaches and contributing to overall software reliability.

9 CONCLUSION

Finally, our software dependability project has involved a systematic approach to software testing, encompassing tasks ranging from meticulous code analysis to comprehensive system tests. Through this process, various bugs were identified, and performance optimizations were implemented to enhance the overall reliability of the software. The findings not only contribute to the immediate goal of improving software dependability but also provide insights for future software development projects. As we conclude this phase, attention is directed towards ongoing advancements in technology, suggesting a continual need for refined testing methodologies and adaptability to emerging challenges in the field.