Sumaya Altamimi - 442203026

Logo, company name

Description automatically generated

A picture containing text, clipart

Description automatically generated

Static Analysis Using Spotbugs and PMD

Lab1 – Systems Security

Table of Contents

[Introduction 2](#_Toc62740707)

[Manual Code Review 2](#_Toc62740708)

[Static Analysis 3](#_Toc62740709)

[Tool choices and versions 3](#_Toc62740710)

[SpotBugs 3](#_Toc62740711)

[Approach and steps: 3](#_Toc62740712)

[Installing and running SpotBugs 3](#_Toc62740713)

[Start New project 3](#_Toc62740714)

[Analyze the Results 4](#_Toc62740715)

[SpotBugs Findings: 5](#_Toc62740716)

[1.Bad practice: Method may fail to close stream. 5](#_Toc62740717)

[2.Internationalization: Reliance on default encoding 5](#_Toc62740718)

[3. Misuse of static fields 6](#_Toc62740719)

[4. Null pointer dereference 6](#_Toc62740720)

[5. RunTimeException Capture 7](#_Toc62740721)

[PMD 8](#_Toc62740722)

[QuickStart 8](#_Toc62740723)

[Analyze the Results 9](#_Toc62740724)

[PMD findings 9](#_Toc62740725)

[1. Assignment To Non Final Static 9](#_Toc62740726)

[2. Close Resource 9](#_Toc62740727)

[3. Literals First In Comparisons 11](#_Toc62740728)

[4. Control Statement Braces 12](#_Toc62740729)

[Code After Modification 13](#_Toc62740730)

[Tool Comparison 14](#_Toc62740731)

[Attachments 14](#_Toc62740732)

[References 15](#_Toc62740733)

# Introduction

For this lab, I will analyze a simple web server written in Java in my macOS. The analysis will be performed first manually and then with two tools, SpotBugs v4.2.0 which is the spiritual successor of FindBugs. It is a tool to find bugs in Java programs. It looks for instances of “bug patterns” or code instances that are likely to be errors. Specifically, it scans bytecode (class files) generated by JDK8 and newer versions.

The second tool that I will use is PMD v6.30.0 which stands for Programming Mistake Detector. There is an add0n to PMD that is called Copy Paste Detector (CPD). Unlike SpotBugs that only works with java, PMD support other languages as well.

As a result, I will build the source for the simple web server in order to analyze it with these tow tools.

# Manual Code Review

After analyzing the web server manually, it seems simple, clear, and good. However, there are few things I noticed, and I think it is better to make some changes to better protect the web server. The first code problem is the *unclosed buffer reader ‘br’, and file reader ‘fr’*. The only closed one was the Output Stream Writer ‘osw’ in line 82.



Closing files is important for the following reasons [1]:

* It impacts the performance with too many open files, slowing down the program.
* Changes to files will not go into effect until after the file is closed, so if we edit, leaves open, and reads a file, we won't see the edits.
* It puts the program in the garbage collectors hands.
* Many more...

The second thing that seems suspicious, the while true in the run method, I think this might impact the performance and there should be another way to process the request.

Text

Description automatically generated

# Static Analysis

# Tool choices and versions

The first tool that I used is SpotBugs v4.2.0

The second tool is PMD v6.30.0

Operating system: MacOS

# SpotBugs

## Approach and steps:

### Installing and running SpotBugs

1. Install the tool directly from the command line as shown below or install it from GitHub using the following link: <https://github.com/spotbugs/spotbugs>.

Text

Description automatically generated

1. Open the tool by writing the tool name ‘*spotbugs’*. The tool GUI will appear, and looks like this:

Graphical user interface, text, application

Description automatically generated

### Start New project

1. Make a new project from file menu and name it *WebServerAnalysis*.
2. Add the source code files ‘*webserver.class’* and click on the Analyze button.

Graphical user interface, application

Description automatically generated

### Analyze the Results

1. The results will appear in the left-hand side as shown below:

Table

Description automatically generated with low confidence

1. Click and read each one to enhance your code.

Graphical user interface, text, application

Description automatically generated

1. If you find bug categories are ambiguous, you can refer to SpotBugs official site[4] ,which include detailed description about each one of them.

## SpotBugs Findings:

## 1.Bad practice: Method may fail to close stream.

As mentioned in Manual Review section, the buffer reader is not closed.

Correction: Close Java.io.reader. Use a try/finally block to ensure that streams are closed before the method returns.

Text

Description automatically generated with medium confidence

## 2.Internationalization: Reliance on default encoding

Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. This bug occurs three times.

* In method *processRequest*, line 49, the use of InputStreamReader: Graphical user interface, text

  Description automatically generated
* In the same method *processRequest*, line 53, the use of OutputStreamWriter:

A screenshot of a computer

Description automatically generated with low confidence

In method *serveFile*, line 103, the use of FileReader:

Correction: Use an alternative API and specify a charset name or Charset object explicitly.

Text

Description automatically generated

## 3. Misuse of static fields

In the *constructor*, line 29, *dServerSocket* makes instance method that write to a static field. Assigning a value to a static field in a constructor could cause unreliable behavior at runtime since it will change the value for all instances of the class.

Text

Description automatically generated

This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

Correction: remove static keyword is one solution. You can either initialize it statically.

Text

Description automatically generated

## 4. Null pointer dereference

In the *processRequest* method, line 62, *request* variable has dereferenced without nullcheck.

Text

Description automatically generated

The result of invoking readLine() is dereferenced without checking to see if the result is null. If there are no more lines of text to read, readLine() will return null and dereferencing that will generate a null pointer exception.

Correction: check the *request* for null before dereferencing it.

Text

Description automatically generated

## 5. RunTimeException Capture

In method *serveFile*, line 106, exception is caught when the exception is not thrown:

Text

Description automatically generated

The method uses a try-catch block that catches Exception objects, but Exception is not thrown within the try block, and RuntimeException is not explicitly caught. It is a common bug pattern to say try { ... } catch (Exception e) { something } as a shorthand for catching a number of types of exception each of whose catch blocks is identical,but this construct also accidentally catches RuntimeException as well, masking potential bugs.

Correction: Either explicitly catch the specific exceptions that are thrown, or to explicitly catch RuntimeException exception, rethrow it, and then catch all non-Runtime Exceptions.

Text

Description automatically generated

# PMD

## QuickStart

* In terminal, run the command ‘brew install pmd’, or simply go to the official page <https://pmd.github.io/> and click download. You can also run the following commands in terminal:
* $ cd $HOME
* $ curl -OL https://github.com/pmd/pmd/releases/download/pmd\_releases%2F6.30.0/pmd-bin-6.30.0.zip
* $ unzip pmd-bin-6.30.0.zip
* $ alias pmd**=**"$HOME/pmd-bin-6.30.0/bin/run.sh pmd"
* $ pmd -d /usr/src -R rulesets/java/quickstart.xml -f text
* Replace the path after -d parameter with your source code path.

## Analyze the Results

* Analysis results will be shown in the terminal as shown below:

Text

Description automatically generated

## PMD findings

## Assignment To Non Final Static

In line 29, Possible unsafe assignment to a non-final static field in a constructor.

Text

Description automatically generated

## Close Resource

In line 35, Ensure that resources like this Socket object are closed after use.

Text

Description automatically generated

In line 47, Ensure that resources like this InputStreamReader object are closed after use.

Text

Description automatically generated with medium confidence

In line 52, Ensure that resources like this OutputStreamWriter object are closed after use.

Text

Description automatically generated

In line 87, Ensure that resources like this FileReader object are closed after use.

Text

Description automatically generated

## Literals First In Comparisons

In line 68, Position literals first in String comparisons.

Text

Description automatically generated

In line 98, Position literals first in String comparisons.

Text

Description automatically generated

Correction: If the variable is null, we won't get a null pointer exception. So, these will be like:

Text

Description automatically generated

Text

Description automatically generated

## Control Statement Braces

In line 94, This statement should have braces.

Text

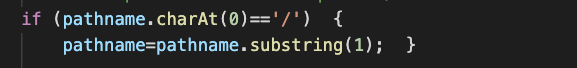
Description automatically generated

In line 99, This statement should have braces.

Text

Description automatically generated

Correction: It is better to have braces for if statements, like this:



Text

Description automatically generated

# Code After Modification

After modifying all problems mentioned by the two tools, there is no bugs as shown below using SpotBugs tool:

Graphical user interface, text, application

Description automatically generated

However, in the case of PMD there is still three unavoidable alarms for closing the resources: buffer reader, socket, and output stream writer as shown below:

Text

Description automatically generated

I assumed these as false alarms or (false negative) alarms. The modified code can be found in attachments as described in the attachment section.

# Tool Comparison

After the Simple Web Server analysis experience and research on these tools, I found SpotBugs more efficient, clear, user friendly and more accurate. Also, for larger projects, SpotBugs will be my first solution. In addition to that, the GUI in SpotBugs, and it is available resources and easy configuration makes it preferable. These are basic requirements in today’s technology. For the bugs description as well, I found SpotBugs more informative. However, it is worth to mention that PMD has a lot of features that I didn’t use and a lot of documentations if someone wants to dive deeper in this. Below are some points to compare between the two tools.

|  |  |  |
| --- | --- | --- |
| Factor | SpotBugs | PMD |
| Does the tool analyze source or binary as input? | Binary | Source Code |
| Which category of tools is it? | Type checking Style checking Program understanding Program verification Property checking Bug finding Security review | Type checking Style checking Program understanding Program verification Property checking Bug finding Security review |
| Example of a finding that is reported by one tool but not the other. | RunTimeException Capture | Control Statement Braces |
| Example of a finding reported by both tools. | Close Resource | |
| RunTimeException Capture | True negative | False positive |
| Control Statement Braces | False positive | True negative |

# Attachments

Along with this report, I included the original source code java file, the modified version of the code in SimpleWebServer folder, and the exported results from each tool after analysis.

# References

1. <https://stackoverflow.com/questions/25070854/why-should-i-close-files-in-python/25070998>
2. <https://github.com/spotbugs/spotbugs>
3. <https://spotbugs.github.io/#using-spotbugs>
4. <https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html>