Sumaya Altamimi - 442203026

A picture containing text, clipart

Description automatically generated

Static Analysis Using Spotbugs and PMD

Lab1 – Systems Security

Table of Contents

[Introduction 2](#_Toc62606131)

[Manual Code Review 3](#_Toc62606132)

[Static Analysis 4](#_Toc62606133)

[Tool choices and versions 4](#_Toc62606134)

[SpotBugs 5](#_Toc62606135)

[Approach and steps: 5](#_Toc62606136)

[Installing and running SpotBugs 5](#_Toc62606137)

[Start New project 5](#_Toc62606138)

[Analyze the Results 6](#_Toc62606139)

[SpotBugs Findings: 7](#_Toc62606140)

[1.Bad practice: Method may fail to close stream. 7](#_Toc62606141)

[2.Internationalization: Reliance on default encoding 7](#_Toc62606142)

[3. Misuse of static fields 7](#_Toc62606143)

[4. Null pointer dereference 8](#_Toc62606144)

[5. RunTimeException Capture 8](#_Toc62606145)

[Tool Comparison 10](#_Toc62606146)

[Attachments 11](#_Toc62606147)

[References 12](#_Toc62606148)

# Introduction

For this lab, I will analyze a simple web server written in Java. The analysis will be performed with two tools, SpotBugs v4.0.0 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.

and Flawfinder v2.0.0. As a result, I will build the source for the simple web server in order to analyze it with FindBugs.

# 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,

Text

Description automatically generated

# Static Analysis

# Tool choices and versions

# SpotBugs

## Approach and steps:

### Installing and running SpotBugs

1. Install the tool directly from the command line as 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.

## 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:Text

  Description automatically generated with medium confidence

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

## 3. Misuse of static fields

In the *constructure*, line 29, *dServerSocket* makes instance method that write to a static field.

Text

Description automatically generated

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

Correction: remove static keyword?

## 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

## 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, as shown below:

try {

...

} catch (RuntimeException e) {

throw e;

} catch (Exception e) {

... deal with all non-runtime exceptions ...

}

## 

# Tool Comparison

# Attachments

# 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>