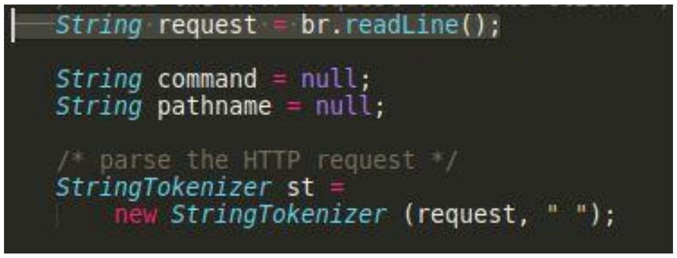
Understanding and Using Static Code Analysis Tools

Austin Hansen

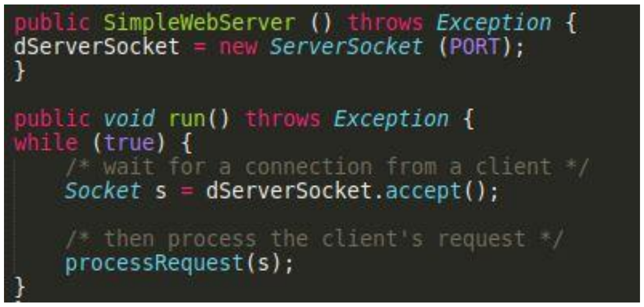
1001530325

Manual Analysis- Part 1



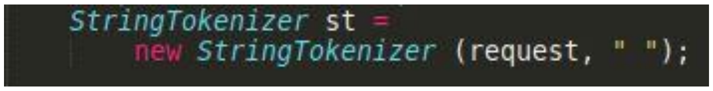
```java
String request = br.readLine();

String command = null;
String pathname = null;

/* parse the HTTP request */
StringTokenizer st =
    new StringTokenizer (request, " ");
```

The line with "String request = br.readLine();" (line 56), could have knock on security issues. While Java is a strongly typed language and is less vulnerable to say a buffer overflow, we still expose the system to an unnecessary risk by not validating the contents of the request variable.



```java
public SimpleWebServer () throws Exception {
dServerSocket = new ServerSocket (PORT);
}

public void run() throws Exception {
while (true) {
    /* wait for a connection from a client */
    Socket s = dServerSocket.accept();

    /* then process the client's request */
    processRequest(s);
}
```

This is a nitpick, but since this is a server, we should have some encryption present. By having no encryption, we open the server up to any number of attacks using a man in the middle approach. (lines 28 - 39)



```java
StringTokenizer st =
    new StringTokenizer (request, " ");
```

A related issue from the first security issue, is that this line (line 62) does not check the input, not only that, but it does not check if the variable request is NULL, which will result in undefined behavior.

Tool Choices

The tools utilized in this lab were:

- Findbugs (version 4.2.3) with FindSecBugs (version 1.11.0) extension
- RATS (version 2.4)

Tool Invocation Process

FindBugs and FindSecBugs:

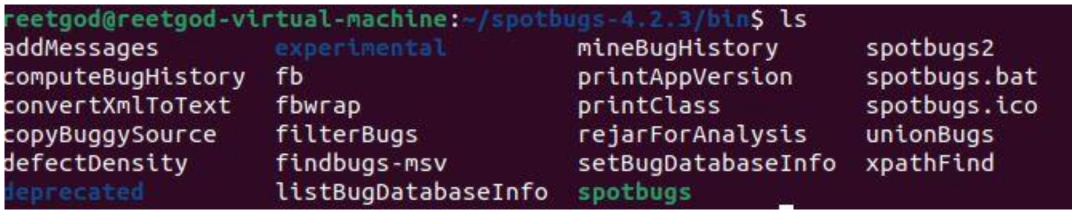
Overall Settings:

- This static analysis tool has the FindSecBugs plugin enabled, it also has another plugin that is not enabled called fb-contib. It is also set to maximum sensitivity.

Downloading:

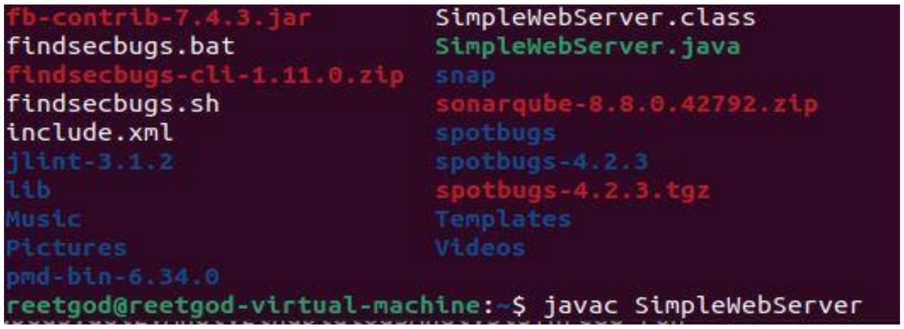
1) Go to https://github.com/spotbugs/spotbugs/releases and select spotbugs-4.2.3-source.tar
2) Click on spotbugs-4.2.3-source.tar, use the command gunzip -c spotbugs-4.2.3.tgz | tar xvf – to untar it.
3) Next, go to https://github.com/find-sec-bugs/find-sec-bugs/releases and select findsecbugs-cli-1.11.0.zip.
4) Download it, and then use the command unzip findsecbugs-cli-1.11.0.zip to unzip the file.

Invocation:

1) Navigate to /spotbugs-4.2.3/bin. This can be done by using a command such as cd /spotbugs-4.2.3/bin.


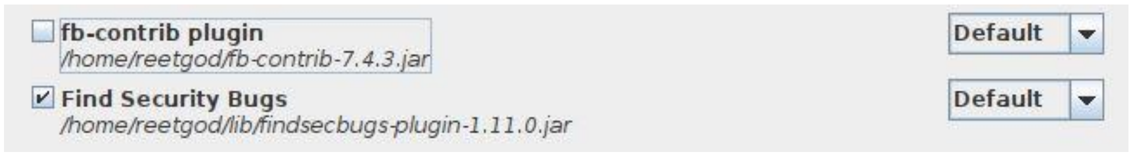
```
reetgod@reetgod-virtual-machine:~/spotbugs-4.2.3/bin$ ls
addMessages          experimental          mineBugHistory      spotbugs2
computeBugHistory    fb                    printAppVersion     spotbugs.bat
convertXmlToText     fbwrap                printClass          spotbugs.ico
copyBuggySource      filterBugs            rejarForAnalysis    unionBugs
defectDensity        findbugs-msv          setBugDatabaseInfo  xpathFind
deprecated           listBugDatabaseInfo   spotbugs
```

2) Run spotbugs, you may have to change file permissions to be able to run it, this can be done by using chmod +x <filename>.
3) After running spotbugs, ensure that the java program that you wish to examine has a java.class file, in other words ensure that the java program has been compiled. If it is not compiled use the command javac <program.java>



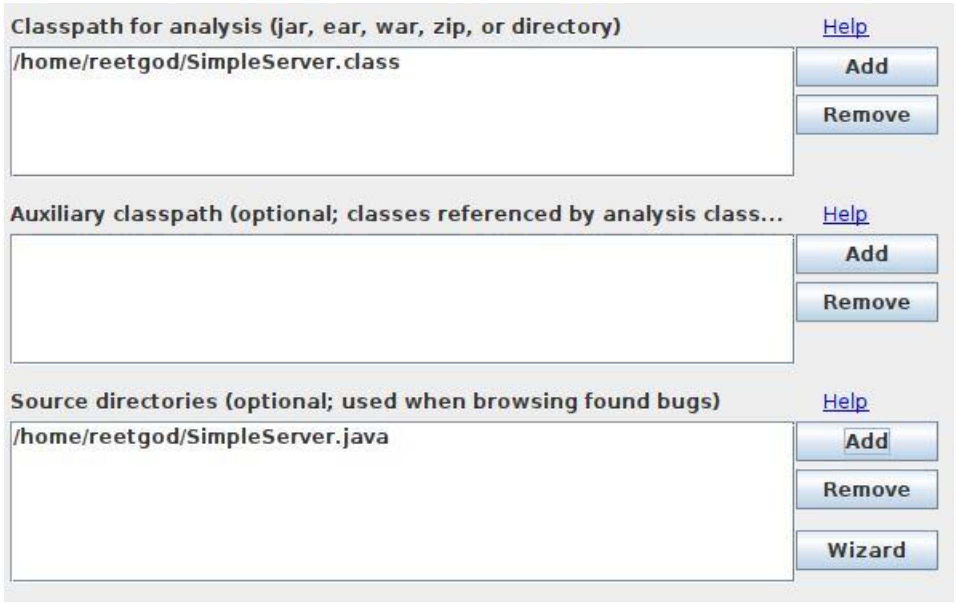
```
fb-contrib-7.4.3.jar           SimpleWebServer.class
findsecbugs.bat                SimpleWebServer.java
findsecbugs-cli-1.11.0.zip     snap
findsecbugs.sh                 sonarqube-8.8.0.42792.zip
include.xml                    spotbugs
jlint-3.1.2                    spotbugs-4.2.3
lib                            spotbugs-4.2.3.tgz
Music                          Templates
Pictures                       Videos
pmd-bin-6.34.0
reetgod@reetgod-virtual-machine:~$ javac SimpleWebServer
```

4) Now, in the SpotBugs GUI go to preferences, and ensure that FindSecBugs is listed as a plugin. If it is not, find it in the directory you unzipped FindSecBugs in.



☐ fb-contrib plugin /home/reetgod/fb-contrib-7.4.3.jar	Default
☑ Find Security Bugs /home/reetgod/lib/findsecbugs-plugin-1.11.0.jar	Default

5) After verifying the existence of FindSecBugs, select create new project, and select the java program that we wish to examines java.class file for the upper portion, including the path that is used to get to it, and put the path of the program.java file in the bottom portion of the create project window.

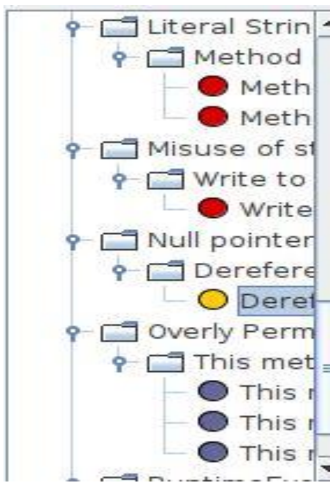


Classpath for analysis (jar, ear, war, zip, or directory)
/home/reetgod/SimpleServer.class

Auxiliary classpath (optional; classes referenced by analysis class...

Source directories (optional; used when browsing found bugs)
/home/reetgod/SimpleServer.java

6) After this is done, SpotBugs will begin analysis. We can now see possible flaws in our program.



Analysis varies from mild (a warning) to dangerous with mild being blue and dangerous being red.

SonarQube

Overall settings:

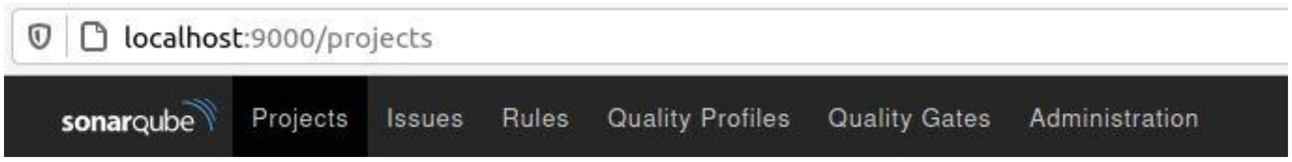
- There are no plugins used for this static code analysis tool, it has the default rules set provided upon download.

Downloading:

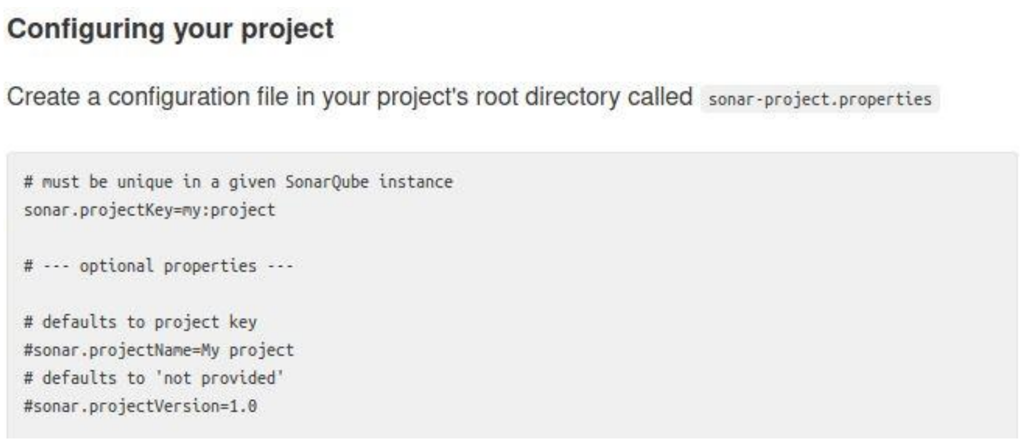
1) Go to https://docs.sonarqube.org/latest/setup/get-started-2-minutes/ to download SonarQube
2) Unzip the downloaded file, then run sonarqube.sh to run the server.

Invocation:

1) After running the server, either connect to http://localhost:9000 through the command line, or type it into your browser.



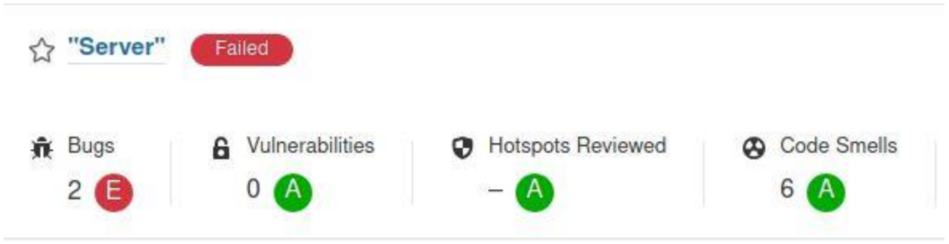
2) Start a new project and name it. Then follow the complicated process of uploading your code to the server via sonarqube scanner. To make it easy to follow along, here is an article that explains the process in more clear terms than their documentation: https://www.c-sharpcorner.com/article/step-by-step-sonarqube-setup-and-run-sonarqube-scanner/


Configuring your project

Create a configuration file in your project's root directory called `sonar-project.properties`

```
# must be unique in a given SonarQube instance
sonar.projectKey=my:project

# --- optional properties ---

# defaults to project key
#sonar.projectName=My project
# defaults to 'not provided'
#sonar.projectVersion=1.0
```

3) After going through the somewhat lengthy scanning process, SonarQube automatically performs the highest setting of analysis on the given code. Here is an outline of the results from SimpleScanner.java:


"Server" Failed

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells
2 E	0 A	– A	6 A

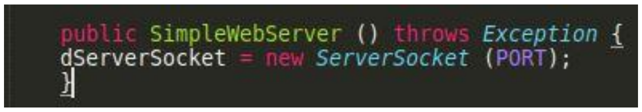
They have quite strict standards. It surprisingly lists 0 vulnerabilities. Additionally, sensitivity of this tool requires the input of predefined rules, which may explain why it lists zero vulnerabilities. Code Smells are the equivalent of warnings, or a notification of bad syntactical practice.

Comparison of SpotBugs and SonarQube

How each tool analyzes and their category:

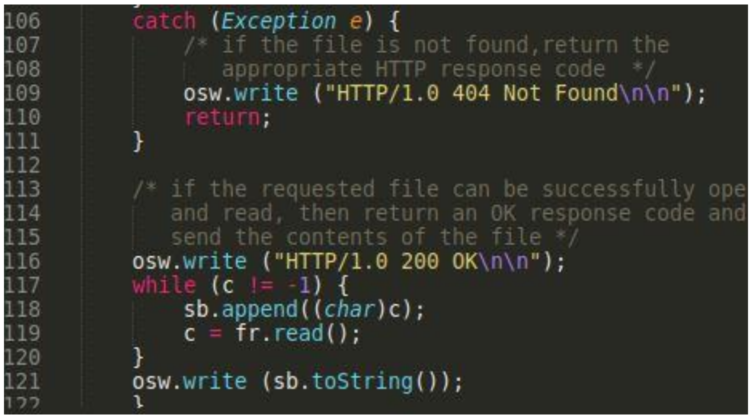
To compare these two tools, we must first understand how each static analysis tool performs its said analysis. As per details in the assignment, as well as the documentation for SpotBugs, we know that SpotBugs analyzes Java Binary data, but what does SonarQube use? SonarQube uses the opposite approach, it uses code syntax, constrained by rules to perform its analysis, in other words it takes source code as input. The category of tools that SpotBugs with the FindSecBugs plugin attached to it, that best fits it are: Security Review and Bug Finding. SonarQube's best fitting categories are: Security Review, Bug Finding, Property Checking, and to a far lesser degree Program Understanding.

Two Example of a problem reported only by SpotBugs:



```
public SimpleWebServer () throws Exception {
dServerSocket = new ServerSocket (PORT);
}
```

SpotBugs throws a warning, saying that this server should be encrypted.

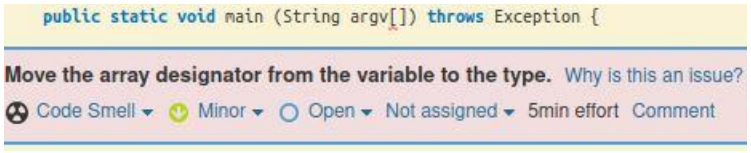


```
    catch (Exception e) {
        /* if the file is not found,return the
           appropriate HTTP response code  */
        osw.write ("HTTP/1.0 404 Not Found\n\n");
        return;
    }

    /* if the requested file can be successfully ope
       and read, then return an OK response code and
       send the contents of the file */
    osw.write ("HTTP/1.0 200 OK\n\n");
    while (c != -1) {
        sb.append((char)c);
        c = fr.read();
    }
    osw.write (sb.toString());
```

Spot Bugs also warns that osw.write may not clean up properly after it is done writing, more details in the result file attached to this assignment.

* There was more reported by SpotBugs than Sonarqube.

An Example of a problem reported only by Sonarqube:



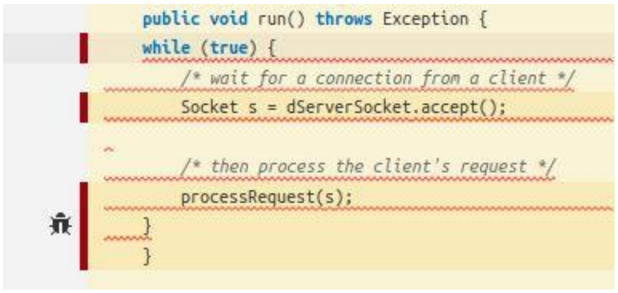
```
public static void main (String argv[]) throws Exception {
```

Move the array designator from the variable to the type. Why is this an issue?

Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 5min effort Comment

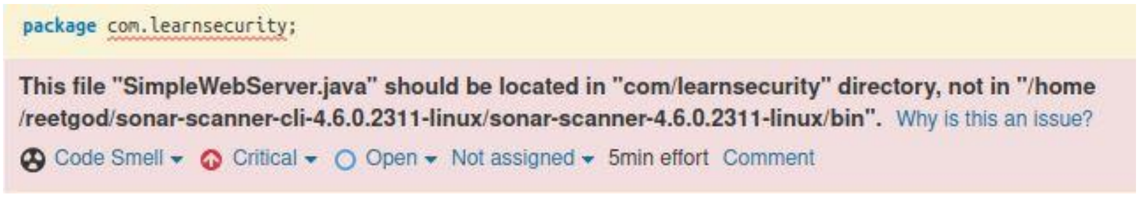
This is a warning that is more geared towards good practice, it is not necessarily bad that we define argv with [] but is not good either.

What do the tools agree on (True Negatives)?



```
public void run() throws Exception {
    while (true) {
        /* wait for a connection from a client */
        Socket s = dServerSocket.accept();

        /* then process the client's request */
        processRequest(s);
    }
}
```

Both tools think that these lines (28-40) are an issue, with Sonarqube questioning its lack of an end statement, and SpotBugs saying that it should be of concern that this is not encrypted and that it will have a knock-on effect after getting input. Same lines, different issues.

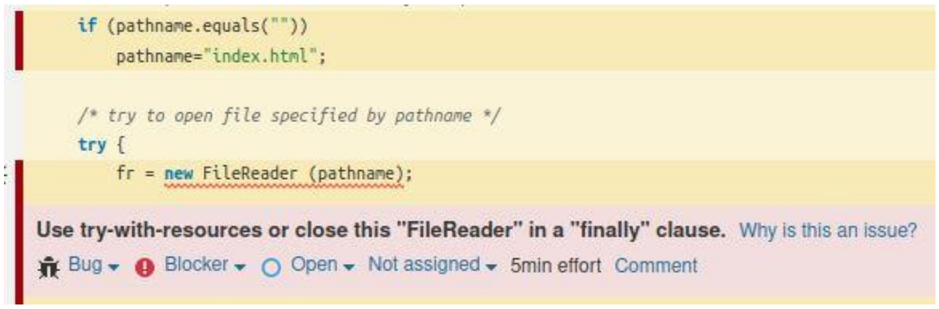


```
package com.learnsecurity;
```

This file "SimpleWebServer.java" should be located in "com/learnsecurity" directory, not in "/home /reetgod/sonar-scanner-cli-4.6.0.2311-linux/sonar-scanner-4.6.0.2311-linux/bin". Why is this an issue?

Code Smell ▾ Critical ▾ Open ▾ Not assigned ▾ 5min effort Comment

Each tool very much dislikes this line, we can see why Sonarqube dislikes it, but what about SpotBugs? To quote the tool “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. Use an alternative API and specify a charset name or Charset object explicitly.”



```
if (pathname.equals(""))
    pathname="index.html";

/* try to open file specified by pathname */
try {
    fr = new FileReader (pathname);
```

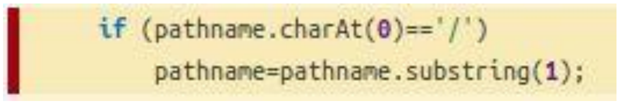
Use try-with-resources or close this "FileReader" in a "finally" clause. Why is this an issue?

Bug ▾ Blocker ▾ Open ▾ Not assigned ▾ 5min effort Comment

Again, we have lines that neither tool likes, but for different reasons. We can see what Sonarqube says, but what does SpotBugs say? Once more, to quote the tool “Method makes literal string comparisons passing the literal as an argument”.

False Positives:

* Almost all discernable false positives came from Sonarqube. While this may seem questionable at first, this is due to it requiring defined rules when performing its analysis. Thus, all false positives are predicated upon whether these rules are defined or not, at least in Sonarqube.

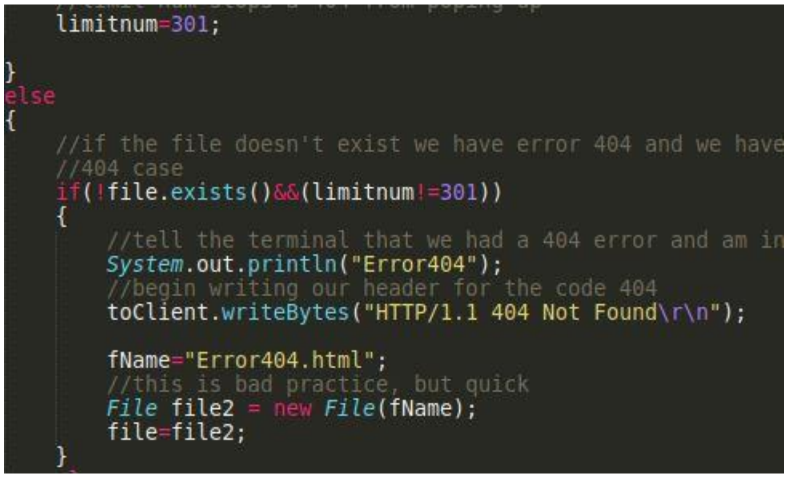


```
if (pathname.charAt(0)=='/')
    pathname=pathname.substring(1);
```

This line is listed as red, yet has nothing wrong with it, it is simply red, because it has no definite rule to check it for correctness.

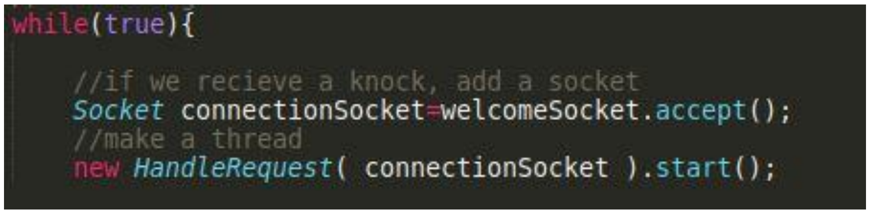
* As far as I can tell, SpotBugs did not throw up any false positives. I'd consider fixing or fix most of what it said. Also, Sonarqube require that I pay to print out a report for it, so I will instead combine several screenshots. Apologies for the inconvenience.

Manual Analysis- Part 2



```
    limitnum=301;

}
else
{
    //if the file doesn't exist we have error 404 and we have
    //404 case
    if(!file.exists()&&(limitnum!=301))
    {
        //tell the terminal that we had a 404 error and am in
        System.out.println("Error404");
        //begin writing our header for the code 404
        toClient.writeBytes("HTTP/1.1 404 Not Found\r\n");

        fName="Error404.html";
        //this is bad practice, but quick
        File file2 = new File(fName);
        file=file2;
    }
```

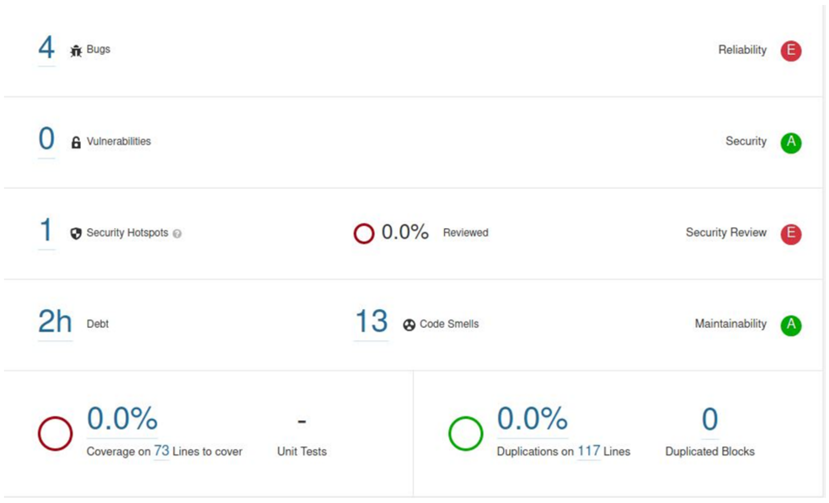
The if statement here is bad logic, as it will always evaluate to true when invoked.



```
while(true){

    //if we recieve a knock, add a socket
    Socket connectionSocket=welcomeSocket.accept();
    //make a thread
    new HandleRequest( connectionSocket ).start();
```

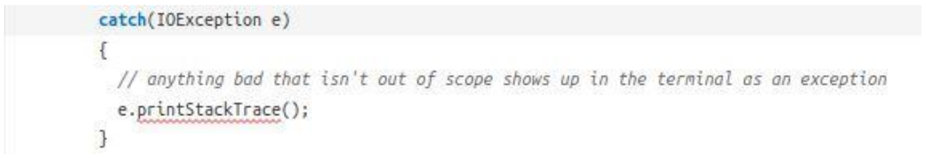
This will likely throw up warnings from both tools, it does not account for a case where the loop is exited and then the socket is closed.

Results - Part 2



Here is a summary of the results from sonarqube, it is a good reflection of the state of this rushed code.

The security hotspot mentioned in the results sample is:



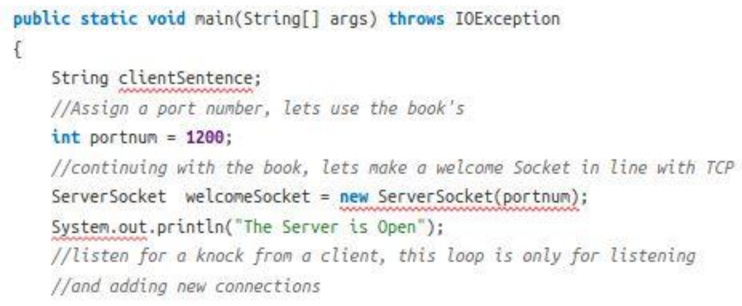
```
        catch(IOException e)
        {
          // anything bad that isn't out of scope shows up in the terminal as an exception
          e.printStackTrace();
        }
```

It was put in for debugging purposes, perfect example of what rushed code can do to overall security.

Fixes – Part 2

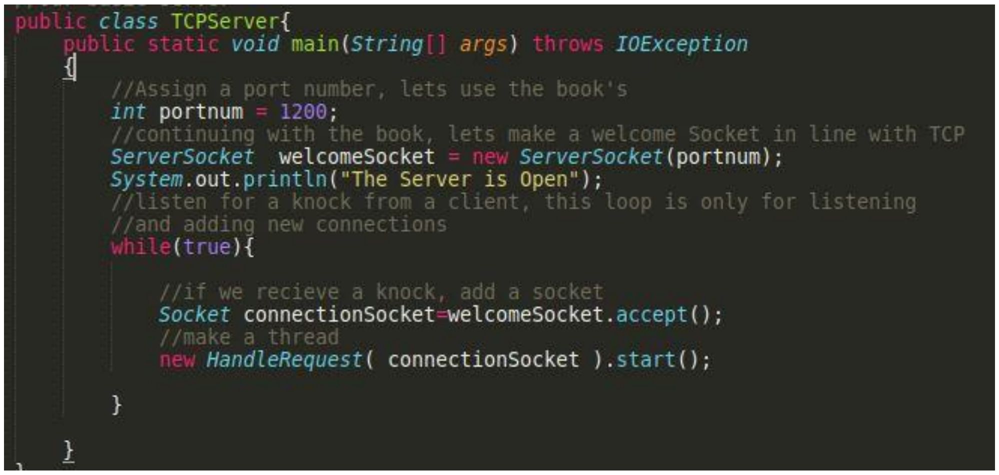
3 fixes:

1) Remove extraneous values:



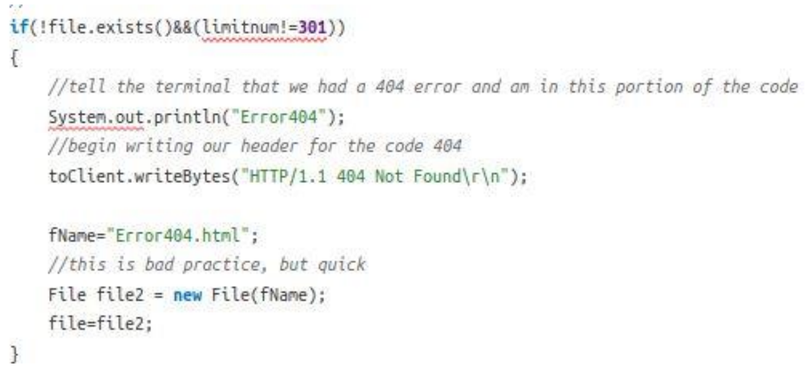
```
public static void main(String[] args) throws IOException
{
    String clientSentence;
    //Assign a port number, lets use the book's
    int portnum = 1200;
    //continuing with the book, lets make a welcome Socket in line with TCP
    ServerSocket  welcomeSocket = new ServerSocket(portnum);
    System.out.println("The Server is Open");
    //listen for a knock from a client, this loop is only for listening
    //and adding new connections
```

To



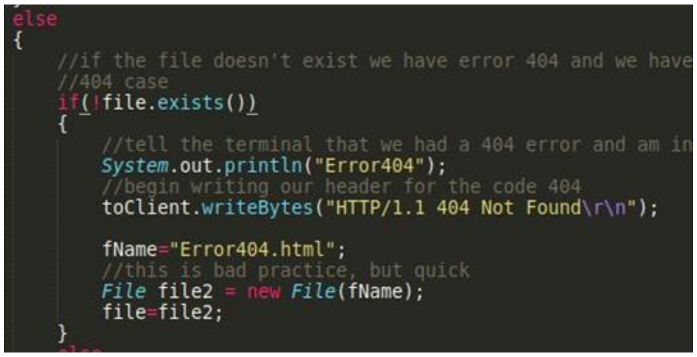
```
public class TCPServer{
    public static void main(String[] args) throws IOException
    {
        //Assign a port number, lets use the book's
        int portnum = 1200;
        //continuing with the book, lets make a welcome Socket in line with TCP
        ServerSocket  welcomeSocket = new ServerSocket(portnum);
        System.out.println("The Server is Open");
        //listen for a knock from a client, this loop is only for listening
        //and adding new connections
        while(true){

            //if we recieve a knock, add a socket
            Socket connectionSocket=welcomeSocket.accept();
            //make a thread
            new HandleRequest( connectionSocket ).start();

        }

    }
```

2) Fix Bad Logic:



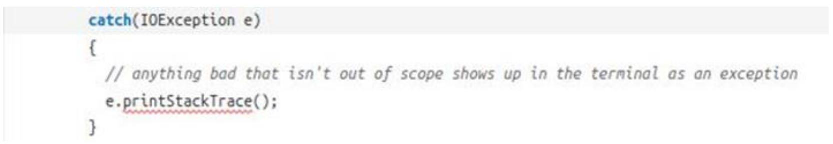
```
if(!file.exists()&&(limitnum!=301))
{
    //tell the terminal that we had a 404 error and am in this portion of the code
    System.out.println("Error404");
    //begin writing our header for the code 404
    toClient.writeBytes("HTTP/1.1 404 Not Found\r\n");

    fName="Error404.html";
    //this is bad practice, but quick
    File file2 = new File(fName);
    file=file2;
}
```

To:



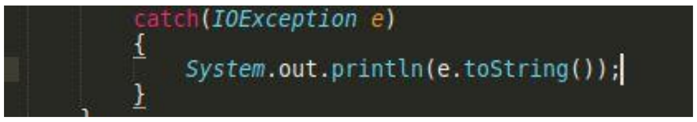
```
else
{
    //if the file doesn't exist we have error 404 and we have
    //404 case
    if(!file.exists())
    {
        //tell the terminal that we had a 404 error and am in
        System.out.println("Error404");
        //begin writing our header for the code 404
        toClient.writeBytes("HTTP/1.1 404 Not Found\r\n");

        fName="Error404.html";
        //this is bad practice, but quick
        File file2 = new File(fName);
        file=file2;
    }
```

3) Fix Security Hotspot



```
catch(IOException e)
{
  // anything bad that isn't out of scope shows up in the terminal as an exception
  e.printStackTrace();
}
```

To



```
catch(IOException e)
{
    System.out.println(e.toString());
}
```