

**Secure Programming CA3  
Static Code Analysis**

**James Finglas – B00094138**

Department of Informatics

School of Informatics and Engineering

Institute of Technology Blanchardstown

Dublin 15.

**BN311 Bachelor of Science in Computing**

**Digital Forensics & Cyber Security**

**Penetration Testing**

**28/10/2018**

Contents

[Abstract 3](#_Toc532338998)

[Scope 3](#_Toc532338999)

[Introduction 3](#_Toc532339000)

[Investigation of code 4](#_Toc532339001)

[FindBugs 4](#_Toc532339002)

[FindBugs Fixes 8](#_Toc532339003)

[Visual code Grepper V2.1.0 9](#_Toc532339004)

[Conclusions 16](#_Toc532339005)

[References 16](#_Toc532339006)

Abstract

This report documents my investigation in a Java based program in an attempt to determine if the program contains any potential security vulnerabilities which may lead to the user’s system potentially being compromised.

# Scope

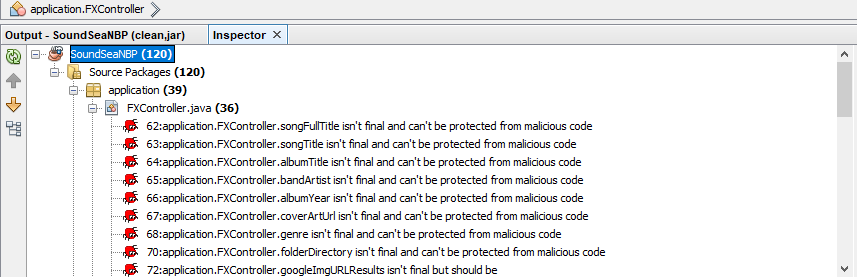
My chosen program is ‘SoundSea’ **[1]**, a java-based program which allows users to download songs from the internet via Pleer.com. Searches are performed through iTunes. My chosen IDE is NetBeans. The Brief requires that 2 different static code analysers be used and that 2 fixes be implemented.  
  
 I chose to use the ‘Find Bugs’ security plugin for net beans as my first static code analyser. This seemed a natural choice due to its integration into the IDE which I had familiarized myself with this year.  
  
My second choice was ‘Visual Code Grepper V2.1.0’ **[2]**. This is an interesting project, as it is stand alone and present issues in what I would consider to be a much more old-school manner. However, installation was simple, and utilization also was simple, merely requiring the user to point at the directory and scan.  
  
In an attempt to restrict myself to code relevant to our course material, I have tried to restrict myself to what I consider to be security flaws which could lead to system exposure.

Introduction

Whilst the program is not a web app, it does contain web-based functionality, so I shall be seeking bug related to the sending and receiving of data. Essentially what I hope to find is flaws that may facilitate ‘man in the middle attacks’.

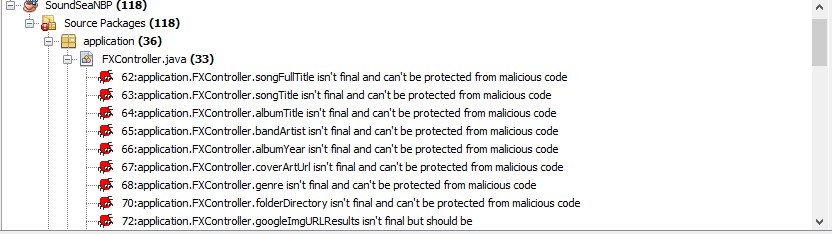
Investigation of code

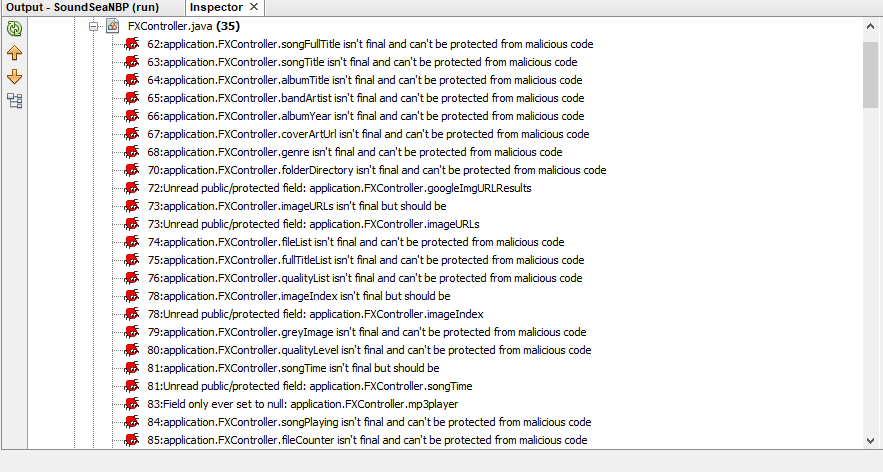
## FindBugs



*Here we see the program running with our FindBugs scan results displayed.*

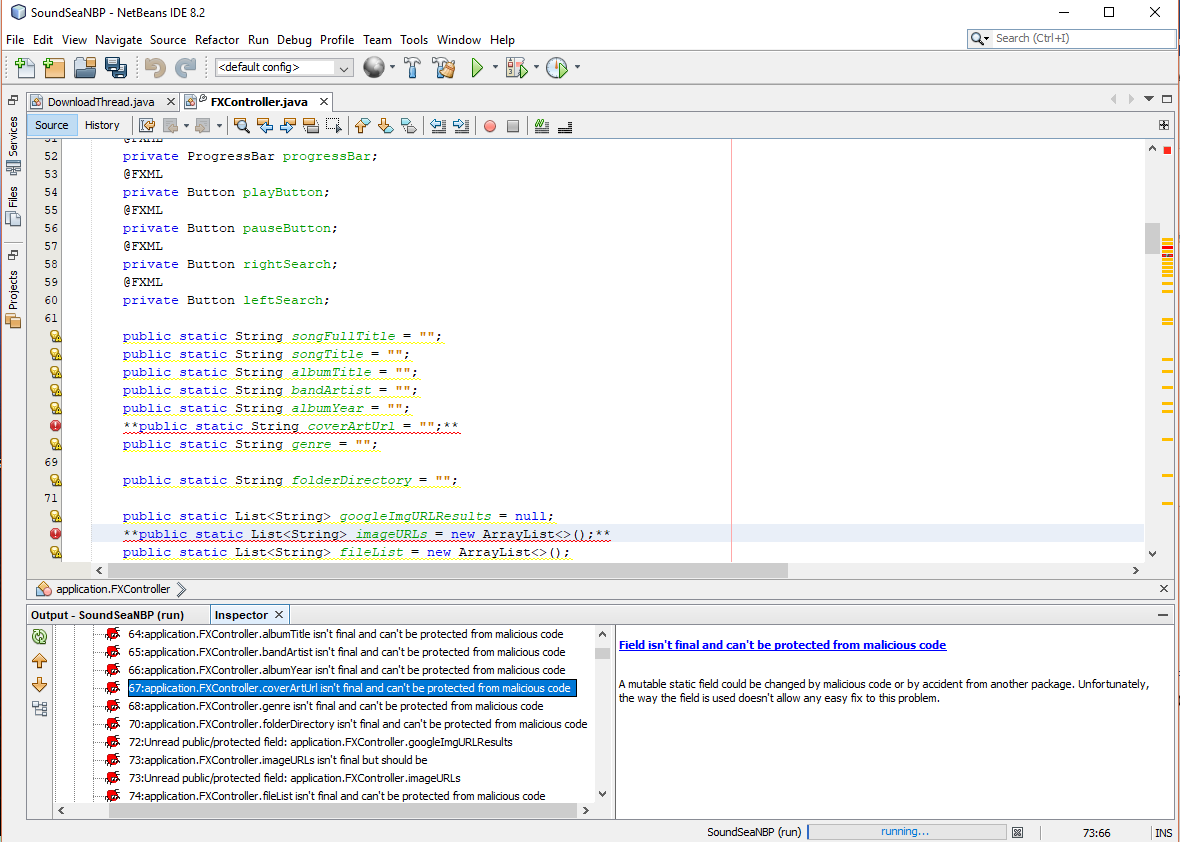
My initial scan revealed 120 bugs as seen above. A closer inspection reveals many trivial bugs such as variables not being set to final, which might allow malicious code to change these variables.





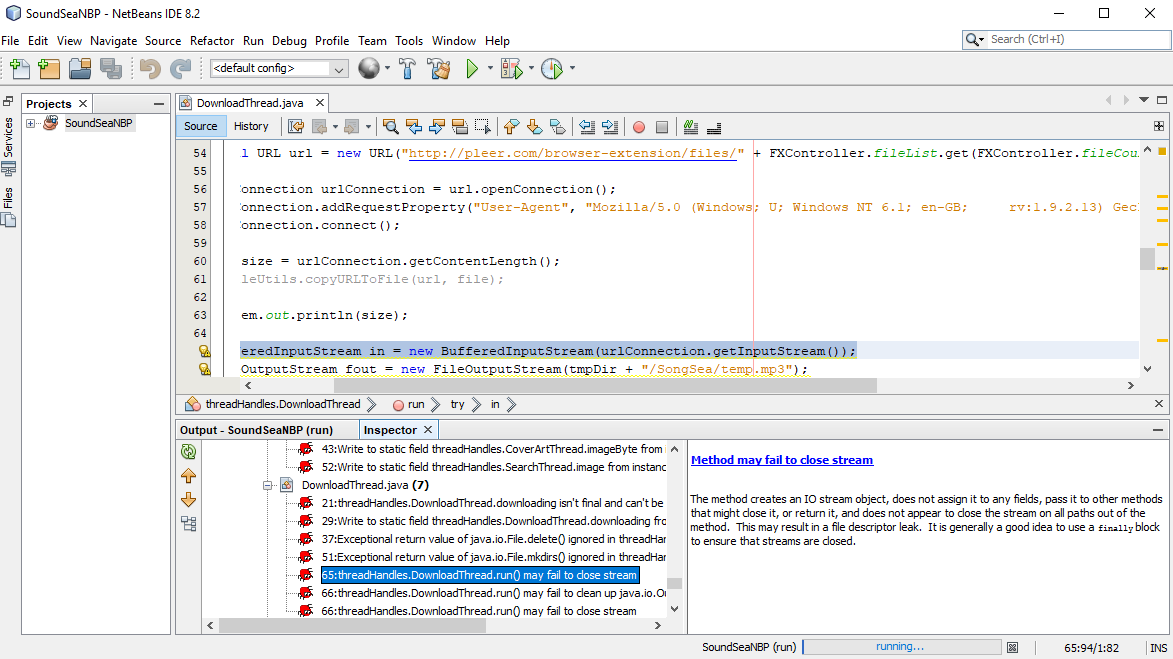
*Examples of trivial bugs found.*

Two of these stood out to me as being potentially dangerous to the system. Two variables, utilizing the URL class have not been set to final, meaning there is potential for them to be overwritten with new values. This should be fixed as externals URLs could potentially be used to transfer data onto the system. However, as the variable at line 67 is being both declared and initialized it cannot be set to final. The second potential bug ties directly into this.



*Line 67 and 73 highlighting non-final URL variables declared.*

The Second potential flaw I discovered is a file descriptor leak. This run() call opens an IOStream object and may not close it down. What we see here is the potential for a data stream to be intercepted via a man in the middle program such as burp suite. Should the threadHandles/DownloadThread.run() fail to terminate, a child process could be started via the parent ID due the file descriptor leak.

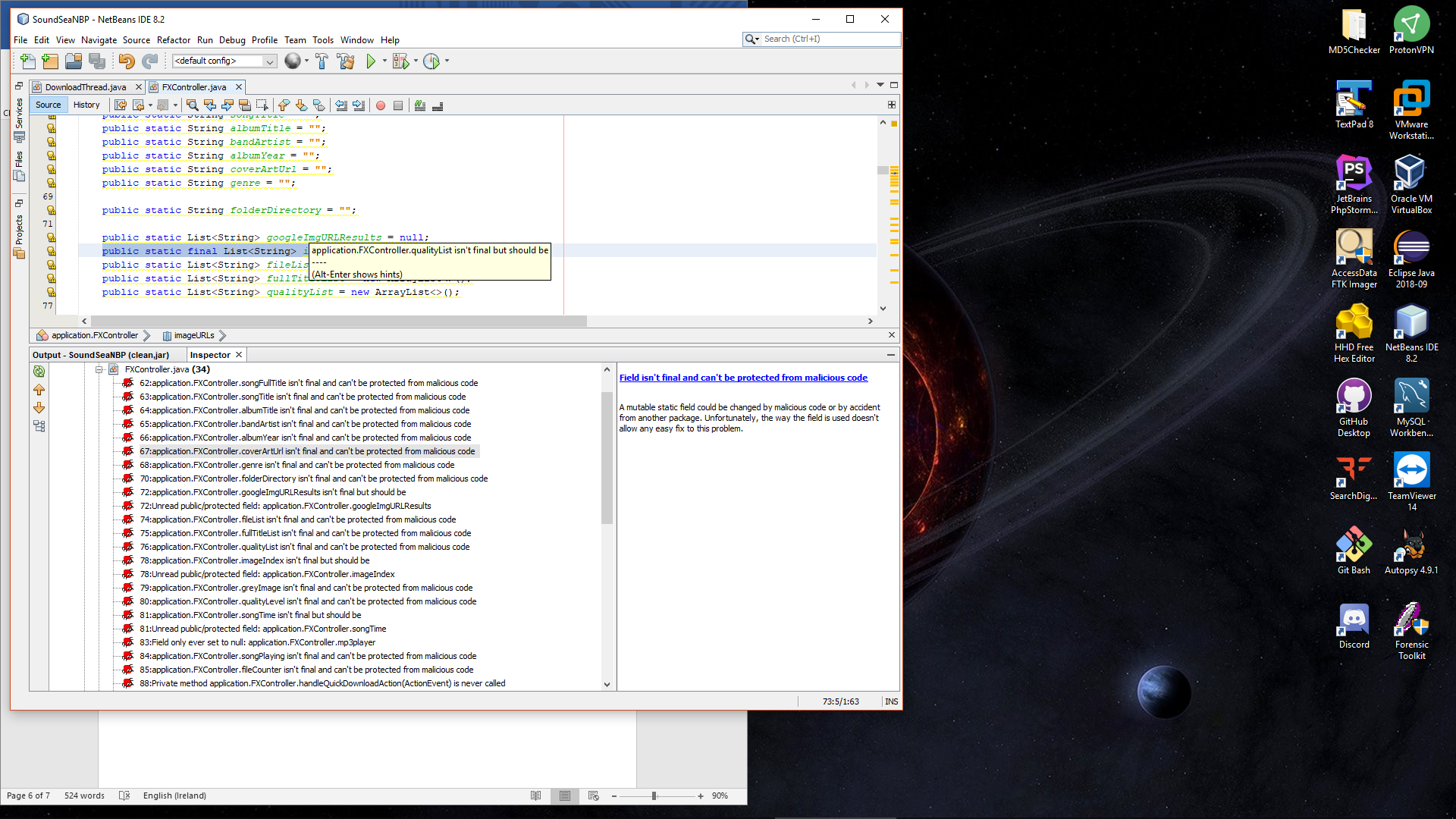


*Here we see the input stream flaw displayed at line 65.*

Combining this with the earlier bug, a script cold be made, and returned via man in the middle attack, to reinitialize a URL variable, and call the threadHandles/DownloadThread.run() to download a malicious program and run it. This is obviously, potentially very dangerous, particularly if the parent program is running as admin. A reverse shell could in theory be initiated.  
  
Along-side this potential flaw, the retention of streams can often lead to Dos crashes of programs. This is not a security concerns, however it’s a major functionality flaw so this bug being fixed will address several issues at once.

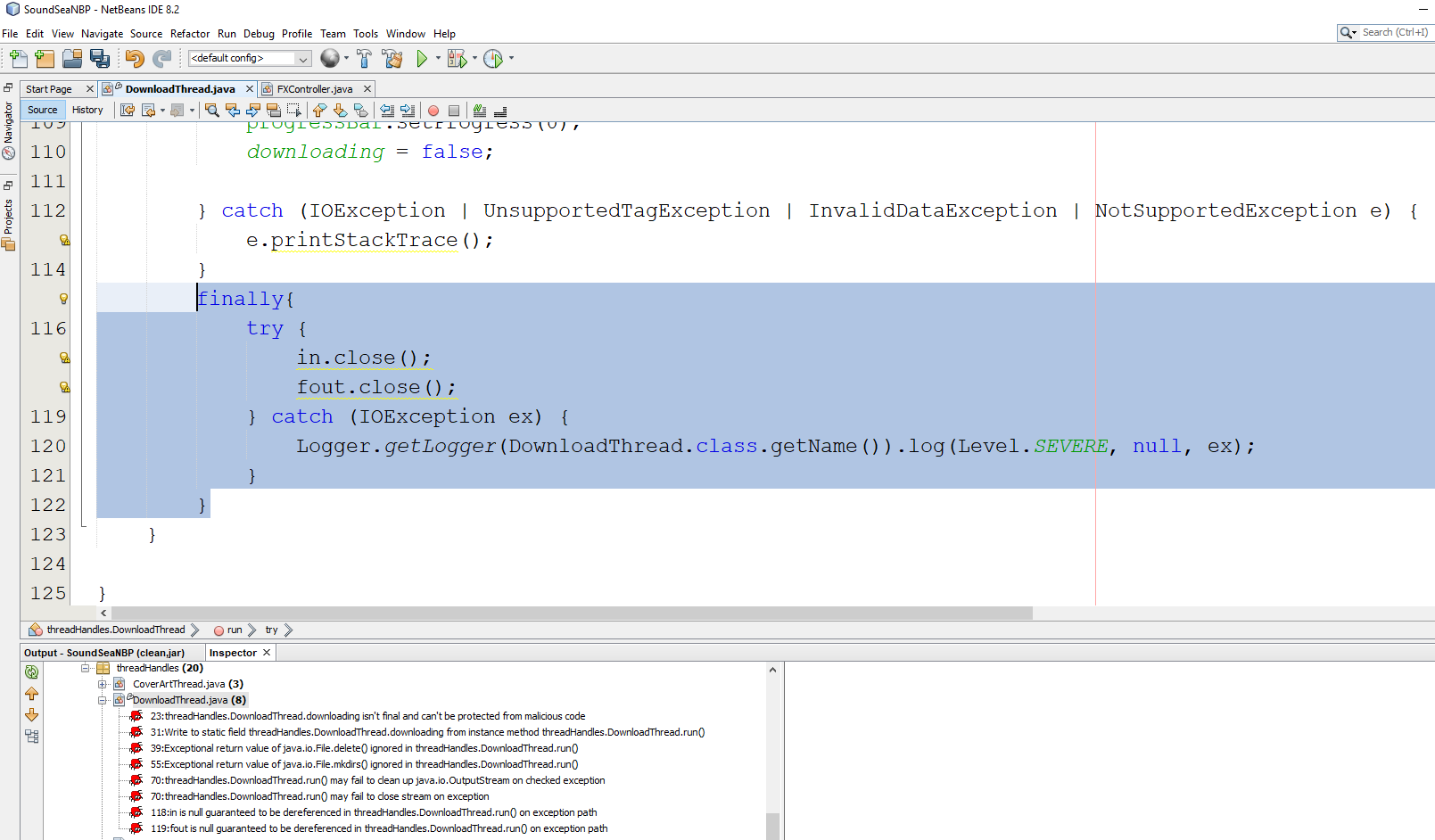
FindBugs Fixes

The first is relatively simply, requiring us to simply set the URL variables to final.

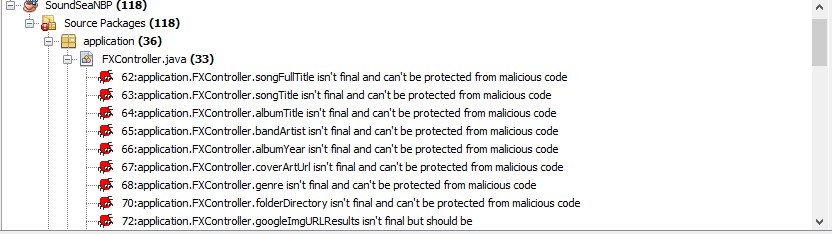


*Here we see FindBugs re scanned and the Issue previously detected at line 73 is now gone.*

The second fix required a finally to be added to the run() function which incorporated a close() call for input stream object. The also had to be wrapped in a try/catch, and I elected to divert the stack trace to the programs internal log file. Purely to add an extra layer of protection I also added this close() call to the output stream.



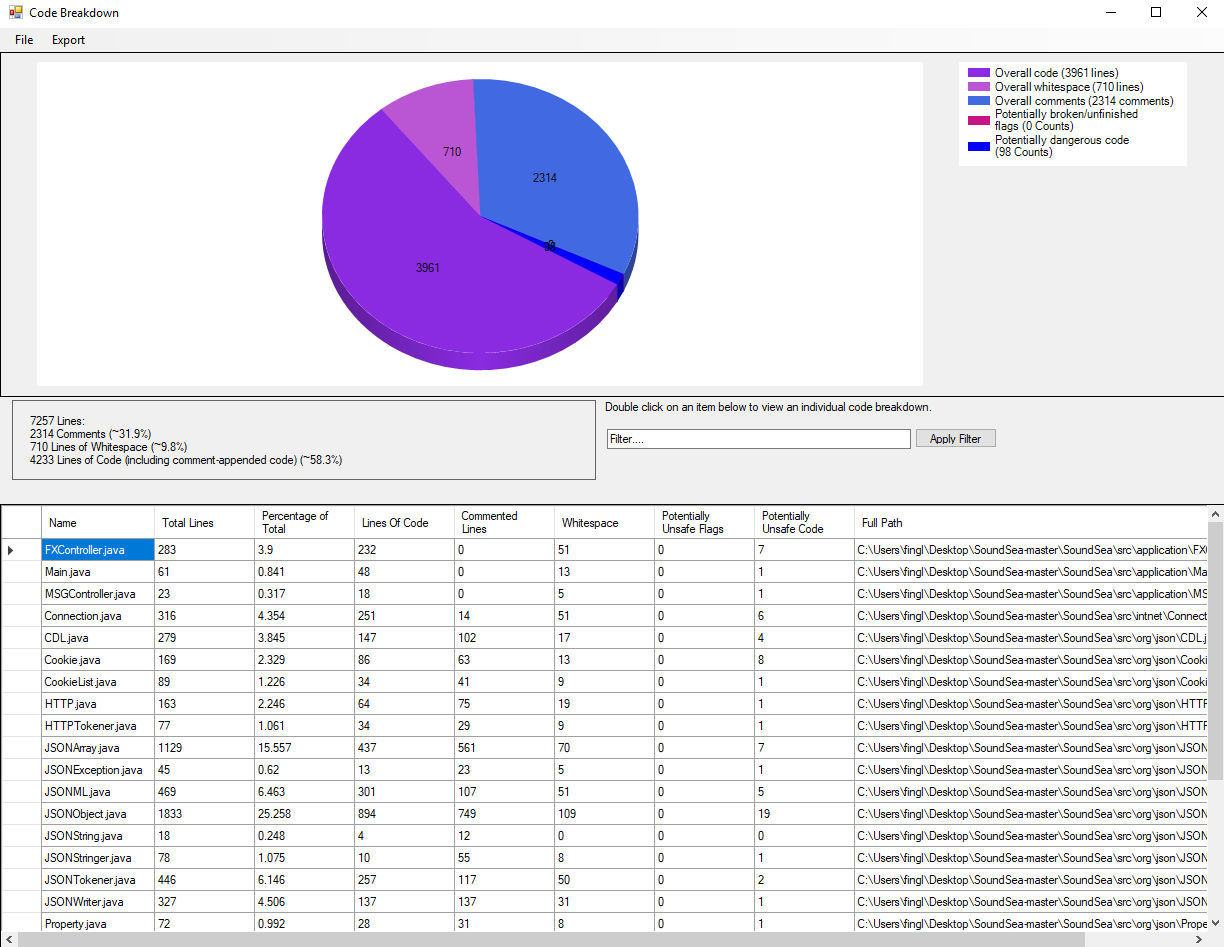
*Here we see that the addition of the finally wrapped in a try/catch including the close functions for the input and output streams has removed the problem at line*

**

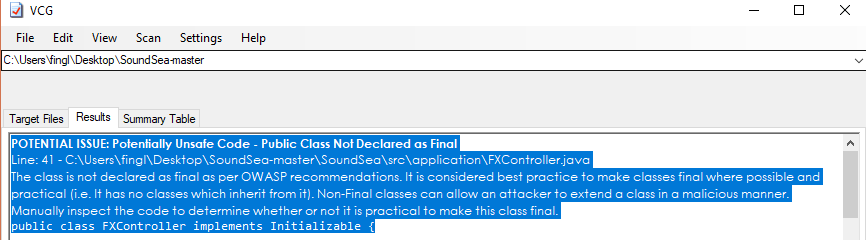
*Here we have confirmation that the two bugs have been rectified.*

## Visual code Grepper V2.1.0

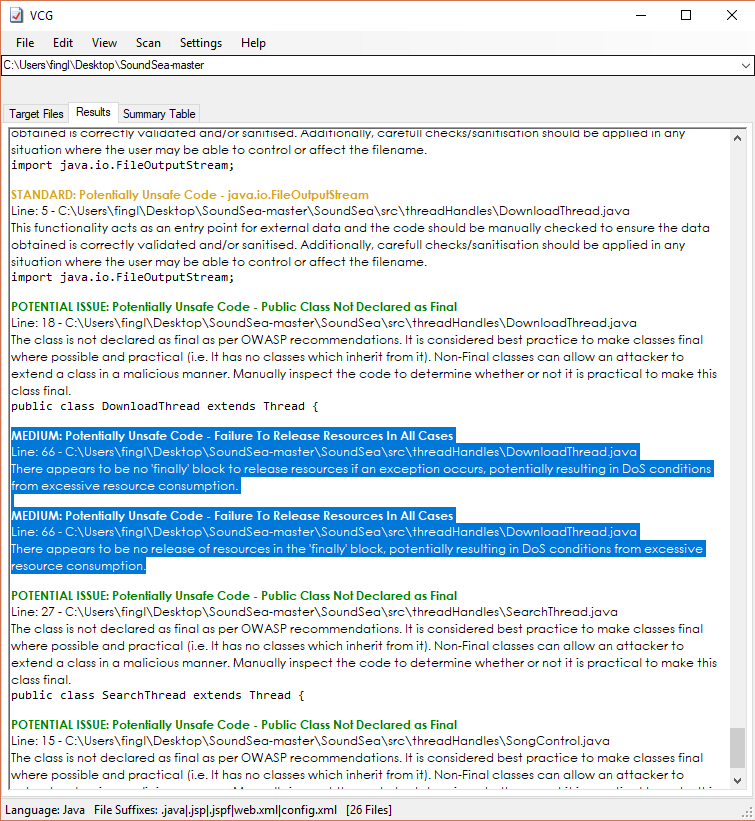
This program turned out to be quite interesting and I must admit I found myself wishing wistfully I had this over the last few years while coding with eclipse. Having already selected my issues, it simply remains to demonstrate the process of fixing the issues via this code analyser.  
  
As mentioned previously this is a stand-alone analyser, however it does come with some interesting features.



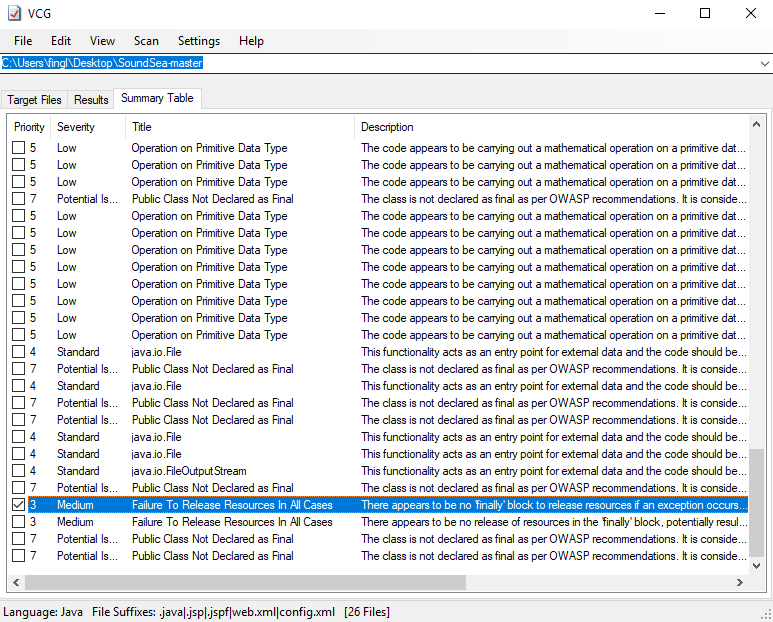
*Here we see a breakdown of our code, and our issues. This is particularly useful in working out which sections of our program our particularly flawed, such as logins, database connections or web connections.*

**

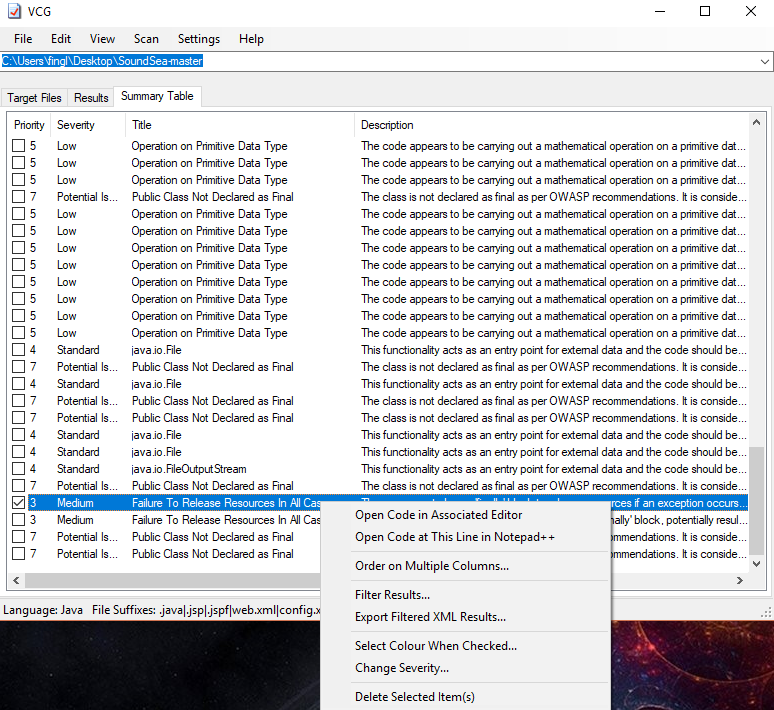
*A difference here, in that our non-final variables are not distinguished individually, instead the entire classed is highlighted as being non- final. This is bad as the class being final would break several important functions of the program.*



*Here we see one of our earlier identified IO buffer stream issues identified via Visual Code Grepper.*

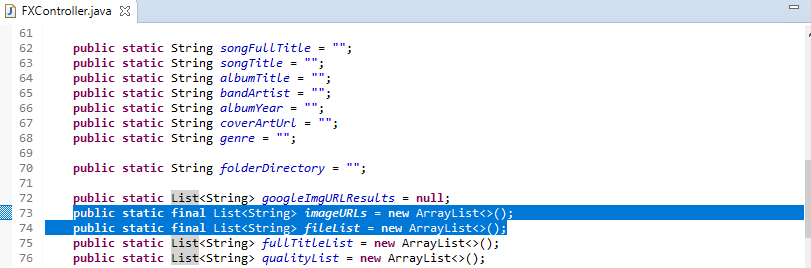
Along-side our slightly retro, yet nice visual representation of our errors, we are also presented with a summary table 

*Visual Code Grepper summary table highlighting one of our IO stream issues.*

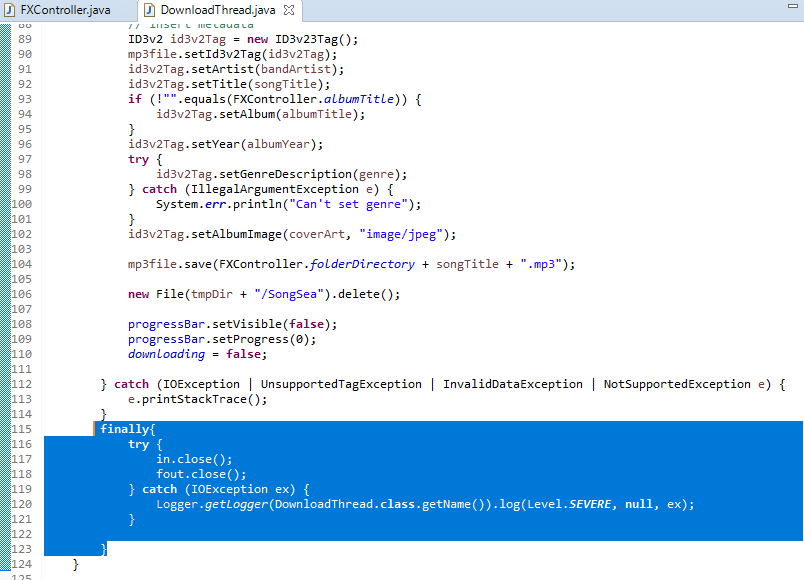
**

*Here we see the other features which along us to link directly to our IDE of choice to fix our bugs.*

The final thing to highlight regarding this program is the ability to open the code directly in our IDE of choice. We are also given a few additional nice features such as filters, the option to change severity, export the results and delete items from our scan. All thing considered, this is a reasonably fleshed out code analyser. All that is left is to fix our problems and rescan our code.

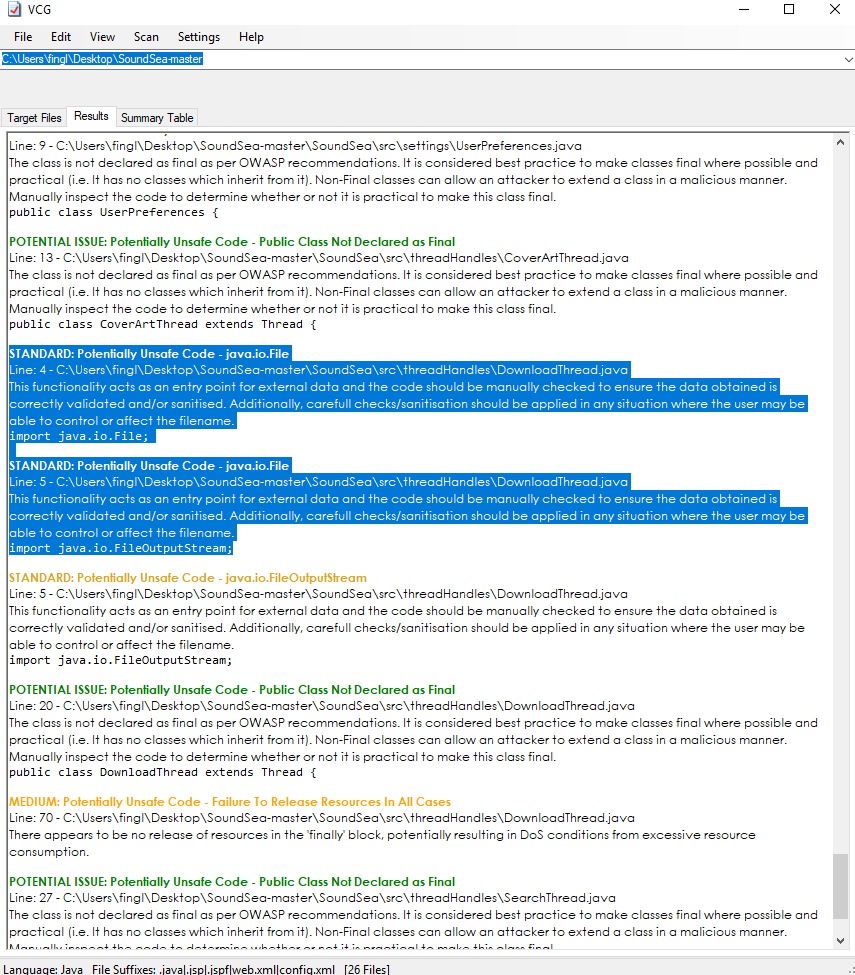


*Here we see our first fix implemented via eclipse, making the variables final.*

**

*Here we have our second fix implemented, closing our input and output streams, inside of our try/catch and logging errors to the log file.*

Lastly, we need to rescan our code. Our first fix will not show up, as the entire class is being highlighted rather than individual variables. However, our second fix shows is indicated, merely be the absence of issues documented din our previous scan.



*Here we see that while we still get warning about inputs not being sanitized,*

*our previously indicated potential Dos issues have been resolved.*

# Conclusions

The integration of find bugs makes it a valuable tool, particularly when developing web apps. The ability to see your issues and fix them on the same screen is obviously a huge time saver. It also cuts back on resources overheads in cases when thin clients are being used in a large team environment.  
  
However, the asthenic leaves something to be desired. And the feed-back is frankly, quite minimal. The explanations of the bugs certainly leave a lot lacking in terms of technical specifics and as such I could myself referencing the java API a lot more in investigation bugs.  
  
Visual Code Grepper on the other hand, seems as though it might be a little out of data. Its last update was three years ago and naturally a lot of java JDK and SE versions have been released since then.  
  
That being said, the program is easy to install, use is a breeze. Feedback is excellent and the ability to link directly to our IDE of choice is fantastic. However, the ability to track issues via class and jump directly to them along-side its continued development makes Find-Bugs by far and away the stand out go to tool between these too tools.

# References

1. Sacert (Stephen Kang), with contributions from: iamaamir, malindaWMD, bryant1410 and mayankTiwari-ideata.  
     
   https://github.com/sacert/SoundSea.
2. NCCGROUP, with contributions from: N1ckDunn.

https://github.com/nccgroup/VCG.

# 