Static Analyzer: A Final Project Report by Kevin Kelly

Table of Contents

Introduction

My final year project was to create a Static Analyzer that operated primarily on Linux, being able to disassemble files, detect strings and imported libraries, and measure the entropy of the file. This report will detail what roadblocks I encountered while implementing this project, what I did and didn't achieve, tests I had performed to ensure the project works correctly, and what I would do differently were I to start again.

What I Achieved

When completed the project, I had achieved the following criteria:

- The program can analyze both Windows (PE) and Linux (ELF) executable files.
- The program, using the Capstone Disassembler, can convert the chosen file into assembly language, presented into a table with highlights of code of interest.
- The program can detect all ASCII strings present in the executable, allowing the user to find both the names of variables and functions, as well as what libraries are imported by the program.
- The program can measure the entropy of the provided file, displaying a value ranging from 0 to 8, providing the user a picture of the degree of obfuscation present in the file.

What I Did Not Achieve

While I am satisfied with my project, there are certain elements that I was unable to implement into its final version. This includes:

- A version of the project that can be run on Windows with all the features found on the Linux version.
- The ability to edit and save the selected file. For example, be able to rename variables or functions found in the disassembler to make understanding what is occurring easier.
- Accessibility options for those who benefit from it. This can include an option to change the font text or themes that change the color palette and layout of the program.

Issues I Encountered & How Were They Resolved

As I worked on this project, there were several times where I came across an obstacle that provided a challenge for me to overcome. Some of these obstacles, as well as what I did to solve said problems include:

- My analyzer is compatible with executable files, meaning any other file would to be rejected by the program. My initial plan for this was to check for a ".exe" in the chosen filename, but there was the potential issue of users adding ".exe" to non-executable files to bypass this. Additionally, Linux Operating Systems tend not to make use of file extensions in filenames.
  Solution: When the user selects a file, the first set of bytes are taken in by the program. A check is then preformed to see if it contains the magic numbers seen in either an ELF or PE file. This involves checking for 0x4D and 0x5A in hex for a PE file and checking for 0x7F in hex and "ELF" in Ascii for an ELF file. If the chosen file fails both checks, then the file is rejected, and the user is asked to choose a different file.
- As part of the Detection of Strings and Libraries, users can filter what is displayed via a text box where they can enter keywords. This is done so that the user can look for a specific string or library. However, when implementing this, the lists displaying the found strings and libraries would not be updated unless the user switched tabs and then returned to the "Detect Strings" page, making it unwieldy and troublesome to preform filtration.
  Solution: All the strings/libraries found are stored in arrays, which form the basis of the lists displayed to the user. When the user inputs a filter, the array is changed to only house strings/libraries that contain the filter. The old list is then wiped and replaced with the new array, followed by an update ran through the GUI. This allows the list of detected strings/libraries to update and change as the user enters filters.
- To achieve Disassembly, I made use of the Capstone Disassembler, which is implemented through use of the Capstone Linux .so libraries, and the Java Native Access (JNA) library. However, when I attempted to make use of Capstone, I received a critical error, stating a "difference between the two versions".

  Solution: After attempting to solve the problem through trial and error, I examined the GitHub Issues Page for the Capstone Disassembler, to see if others have had this same problem. I discovered that this issue had yet to be fixed in the latest version of Capstone, and that the recommended solution was to download the previous version. Once this was done, I had the program functioning with all the expected features.

- As stated previously, the Capstone Disassembler requires Java .so libraries to function. When I compiled my project as an executable, these files were not included. This caused the program to crash when it attempted to open a file. **Solution:** When the project is compiled, include the .so libraries as part of the project to be included, ensuring the executable version of the project has the resources required to function.

What I Learned & What I Would Do Differently

This project has been a great experience that has furthered my development and capability in several skills, including:

- Expanded my knowledge of Java and SwingX, the two core components required for the project, such as how they function and operate, how it handles external libraries and what's required of them for inoperability between multiple Operating Systems.
- Learned how to make use of the Capstone Disassembler, which handled the decompiling of the chosen program into assembly language. I had to learn how to link my project with the library, fix any errors that were present, and ensure Capstone had the resources required when exporting the project into an executable format.
- This project required the creation of wireframes as part of the documentation, something I had not done before. Learning to create mockups of how the project will look and function was an important learning experience for me, as creating wireframes is a vital part of the software development lifecycle.

If I were to start this project again from scratch, I would have spent more time creating a wireframe that is concrete and acts as a rough guideline to how I wish the project to look by the end. This would have saved me more time in the long-term, as I would not be attempting to design most of the GUI as I am coding it.

Additionally, I had spent a great amount of time attempting to fix linking errors between my project and Capstone, while not using the GitHub Issues page for Capstone to see how others had overcome the problems I had. Were I to start the project again, I would have definitely have used this tool more to my advantage when dealing with Capstone-related problems, especially considering GitHub is one of the most commonly used tools in the world.

Differences in Design & Additional Research

An end goal for this project would that be it would have versions available for both Windows and Linux. Unfortunately, due to complications regarding Capstone and it is requiring .so Linux libraries to function, I settled on designing this project exclusively for Linux.

When initially designing the GUI for the project, I had planned for the components of the project – General Details, Disassembly, Detect Strings etc. – to all be placed in one page. However, I had found this to cause a lot of visual clutter. Instead, I decided to make use of a "Tabbed Pane", a component in Java that allows pages to be separated into tabs and dedicated a page for each of the components of the project.

As well as Capstone, I had also planned on importing the Bayes Server API, a library that provides an array of functions and tools for statistics, including measuring entropy of a file. However, I decided instead to implement my own function for measuring entropy, rather than importing a library where only a small sliver of its content would be used.

For additional research, as I had decided to implement my own means of measuring entropy, research was conducted on how that could be achieved. Additionally, research through the Capstone documentation and forum to determine the solution to problems I had encountered with its implementation.

Module Descriptions

HomePage.java

This file holds the GUI that asks the user to select a file that they wish to analyze. If a non-executable (PE or ELF) file is selected, it is rejected.

Imported Libraries: Java Swingx (JButton, JFrame, JPanel, JFileChooser) Java IO (file.Files, IOException) Java AWT (BorderLayout, ActionEvent, Action Listener)

MainPage.java

The core of the project. This holds all of the GUI components that present the main functions of the project to the user.

Imported Libraries: Java Swingx (JButton, JFrame, JPanel, JOptionPane, JTable), Java Security (MessageDigest) Java Util (ArrayList) Java AWT (BorderLayout, ActionEvent, Action Listener), Capstone

Methods.java

Holds several functions I added to help in the implementation of the project, such as functions that check if the file provided is a PE or ELF file.

Imported Libraries: Java IO (File, FileInputStream)

Testing

Below are the tests I preformed on my project, to ensure it behaved and responded as expected:

Date	Description	Result
18th of April 2022	When prompted to select a file for analysis, attempted to provide a non-executable. Files tested included images, text files, spreadsheets, and video files	The program rejects these files, as they do not contain the beginning bytes present in either PE or ELF executable files
18th of April 2022	In the section for the detection of strings and libraries, entered long complex strings to see how the program will preform based on this filter	The program filters the strings presented based on the user input. As the filter provided does not appear in any of the strings found, the program returns a blank list
18th of April 2022	In the table that displays the disassembled program code, tested the ability to highlight instructions by selecting different instructions	When a new instruction is clicked, all occurrences of this instruction are highlight, and the previous instruction is reverted to normal