Static File Analysis Tool

Project Report

Brian Tobin

C00216353

Table of Contents

Introduction

This report gives a description of the project on what it is and what is used for and lists some of the technologies used. The project is then reviewed on the amount achieved and not achieved, problems encountered, testing, what was learned and what would be done differently if the project was started over again. A conclusion is then made on the project overall and how I think the project turned out.

Project Description

Static File Analysis Overview

Static Analysis is usually the first step involved in reverse engineering or malware analysis. A file is analysed using various techniques without ever executing the file. The goal is to gather as much information about the file and come up with a hypothesis on what the file will do when executed.

Project Overview

This project is a tool designed primarily for students that combines some of the basic static analysis techniques and puts them into one easy to use application. The application provides a checklist for the users to follow and is primarily designed for 32 bit PE files. The first thing a user should do is select a file to analyse, then they generate a hash of the file and finally create a backup of the file in a selected location. They can then check if the file is packed using UPX, and if they file has high entropy which may imply that it is packed using a different algorithm. They should then find strings and select any strings that they think might be useful. Next, they can find DLL's and imported functions. These can give very useful hints of some of the higher level things the file will do. Finally they can disassemble the file and use information gathered from previous steps to create a final hypothesis of what the file will do when executed. This tool also provides a hex editor which can be used to get a better understanding of how a file is structured and can also be used to bypass anti-disassembly techniques that a file may implement.

Technologies Used

Qt and C++ for the GUI and backend.

HTML for displaying most text.

Git for version control.

Project Review

Amount Achieved

All of the core functionality was achieved.

- The strings function finds all relevant strings just like other industry strings functions.
- The application can detect, pack and unpack files using UPX on windows only.
- DLL names and function calls can be found and displayed.
- File can be viewed in hex and bytes can be edited.
- The files code can be found, disassembled and displayed.
- A checklist that guides the user is implemented.

Extra usability features were also achieved.

- A search bar was implemented for strings and disassembly.
- Strings can be sorted and unsorted dynamically.
- Strings can be jumped to in the hex editor.
- Jump and call values in disassembly can be followed by clicking on them.
- Disassembly keeps track of each jump the user takes and allows them to jump backwards.
- In disassembly, calls to the data segment display the imported function name rather than the pointer value to this function.
- In disassembly, where an instruction references a string, that string is printed beside the instruction.

Not achieved

There was not enough time to build binaries for Linux. However, as everything is open-source, it should be easy for any Linux users to create their own.

On Linux, the application cannot detect if a file is packed using UPX as the application uses a Windows executable to do this. A Linux UPX executable needs to be added for this functionality.

The disassembler is not complete, it cannot perfectly handle every instruction, but it works well enough for the scope of the project, i.e., students learning the absolute basics of static file analysis. It works perfectly for all simple executable files that were tested.

Problems Encountered

Displaying massive amounts of data in QT objects.

The first major problem encountered was when trying to display a few thousand strings in a QListWidget object provided by QT. It took a few seconds to load all the strings into this object and there was a noticeable delay when using the built in scroll bar. When strings were selected to be saved in the saved strings section, the object became almost unusable with each selected string creating an even bigger delay when trying to do anything.

This issue was fixed by only loading 30 strings at a time into the QListWidget and creating a custom scroll bar that would replace the current strings with the next 30 strings when it is used. This made keeping track of the selected strings much more difficult by needing to keep track of the position of the string in the display and the value of the scroll bar.

This is also the solution that was used for displaying hex and disassembly as the QTextBrowser used for them could not handle the amount of data required.

Sorting strings and keeping track of saved strings

The built in QStringList sorting function was used to sort the strings. As strings were saved based on their display location and the value of the scroll bar, sorting the strings messed up the saving function.

To fix this a swap map was used to keep track of the original location of all strings before sorting. The current location of the string was appended to each string, the sorting function was then run, the locations were removed from each string and the new position was stored in the swap map at the original position. This allows the users to seamlessly move between sorted and unsorted strings while maintain the correct saved strings.

Testing

All features of this tool were tested using every combination of inputs where possible. For example the hex editor was tested using 5 different files. Files with no data, less data than can fit on a row, just enough data that can fit on a row, enough data that requires a scroll bar but does not fill the last row, and finally a file with enough data that requires a scroll bar and fills the last row completely. These files were used to make sure that the system can properly handle all combinations of files without crashing or reading data beyond the file.

Another example is testing each feature under different circumstances such as trying to find strings with no file selected. No strings were displayed and the system did not crash. Then opening a txt file and trying to find DLL's. The system displayed the proper error message of "DLL's not found. This file is not a PE file!" and did not crash.

Disassembly was tested manually for each basic instruction as they were being handled. For example, the first 6 opcodes 00 – 05, are add instructions. When these were being handled, every valid mode of operation was tested for each opcode. So an instruction was tested using indirect addressing mode with 0, 1 or 4 displacement bytes, or register addressing mode, both with and without an SIB byte.

There are some instructions that cannot exist, for example, some opcodes cannot use register addressing mode. These are not specifically handled as they cannot exist in a valid executable file.

Differences from design

The virus total section had to be removed from the design as on further research I found out that it could not be used commercially.

What I learned

During this project I learned a lot about the steps involved in software development. I learned that the research and design steps are extremely important to define the core of an application. Most of the tools I used were new to me, such as Git and Qt. There was a bit of a learning curve initially, but I got used to them after a little while.

Git

I used git as the version control system during the development of this project. I am very familiar with git now and can see the important role that version control software plays in the development process.

Qt

I used the Qt framework to develop the application for this project. It is very useful for creating GUI applications that can be built for the popular desktop operating systems, Windows, Linux and macOS with little to no change to the code. I am much more familiar using Qt and some of the custom objects that it provides.

File structure

This project helped me learn a lot about how files work, how they are stored and how they are split up into different sections. Making a disassembler also gave me a much better understanding of how files are executed such as where the code is stored in a file and where the entry point of the code is located.

What I would do differently if starting again

If I were to start this project again I would spend less time working on the hex editor, potentially even leaving it out altogether and spend that time making the application work for ELF files as well as PE files. I would also spend some of this time working on the user interface, which was left until the end to make sure that the core functionality was working first. Finally, I would make sure that the application runs on Linux.

Conclusion

In conclusion, the core functionality of the application is working and there was enough time to add some extra features that improve the usability of the tool. Not all features were implemented such as creating binaries for Linux, but these were small things that could easily be added in the future and do not affect the functionality of the application. I also learned a lot about the process of developing software from start to finish and how useful version control software can be. I learned a lot about the structure of files and how they are stored and executed as well. I am happy with how this project turned out and believe it has been a worthwhile experience.

Acknowledgements

I would like to thank Dr. Joseph Kehoe for his advice and guidance throughout this project.