<u>Project Plan: A Function Specification Report by Kevin Kelly</u>

Table of Contents

Introduction 2
Functionality of the Tool 2
GUI Mockup 4
  Home Page 4
  General Details 4
  Disassembly 5
  Detect Strings 5
  Measure Entropy 6
Metrics 7
References 7

Introduction

This paper will go into greater detail on how each functionality of my Static Analysis Tool will operate, a mock-up of how the GUI will look, and specific goals I wish to achieve to consider my project complete.

Functionality of the Tool

My project is a Static Analysis tool for Linux. To complete the project, the following functionalities must be implemented:

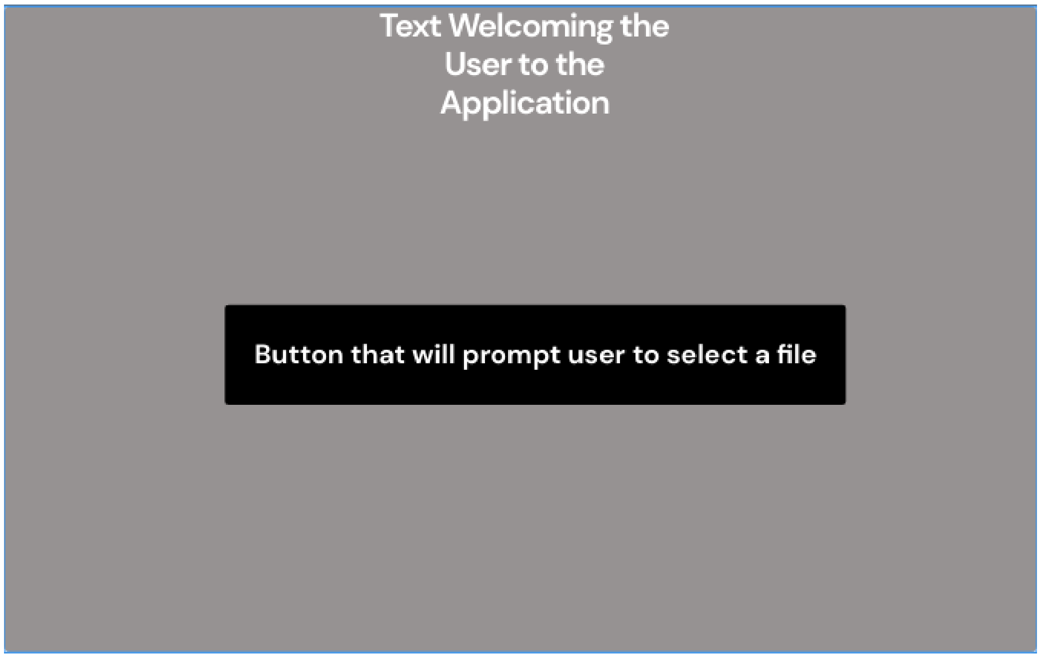
- **Disassembler:** File disassembly is taking an executable file's machine code and transforming it into human-readable assembly language, so that the user can determine the program's purpose. This is done by first reading in the file and determining it's compiler. It then goes through the file instruction by instruction. In Intel architecture, it first looks for the Instruction Prefix bytes. A maximum of four can be present in an instruction, with one byte each. These prefixes can be used to override the sizes of operands and instructions, as well as controlling loops. Next to the prefix bytes is the Opcode, 1 or 2 bytes that determines the operation to be performed. Following the Opcode are the optional Mod R/M bytes (determines what registers or memory locations the instruction is to use), the SIB byte (determines the address in memory of the operands used) and the Displacement and Immediate bytes. The disassembler repeats this process for each instruction until the end of the file is reached (Radhakrishnan, 2010).
- **Detect Strings:** Strings in the code can be vital in determining a code's purpose, as they can be used for variables, function names or imported libraries. In many coding languages, a string is an array of characters from a character set (ASCII, Unicode etc.) that is terminated by a null character, indicating an end of the string. When going through the file to be analyzed,

the tool can keep track of sections of the code that fits the above description, presenting the list of Strings to the user when analysis is finished.

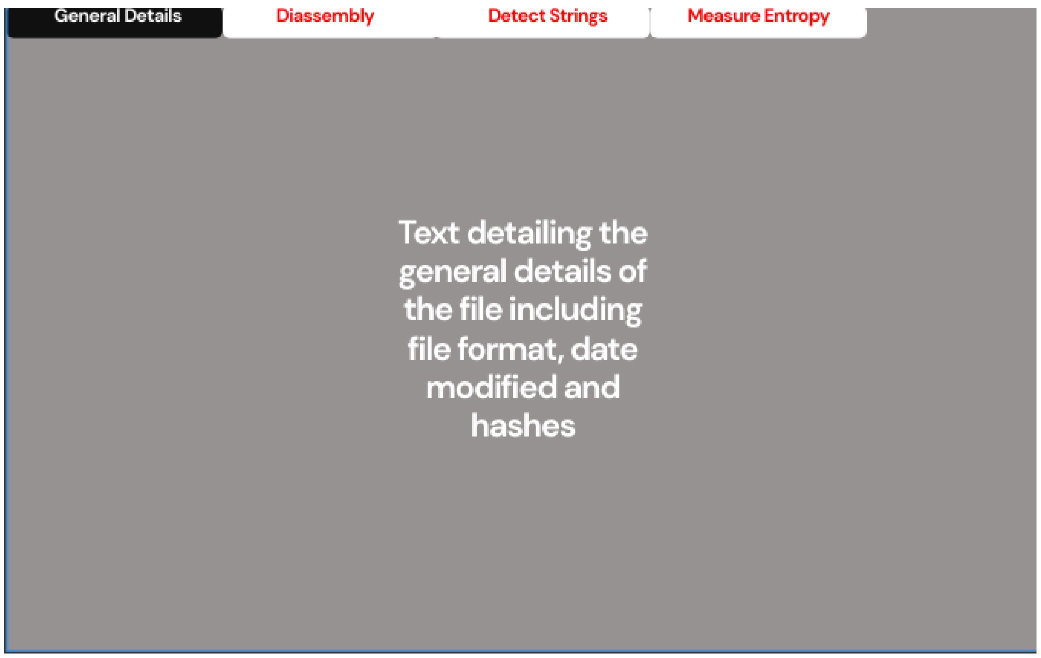
- **Measure Entropy:** To measure a file's entropy is measure the disorder or randomness, two processes must be achieved: File Segmentation and Sequence Comparison and involves two files. File segmentation is where each file is divided into a sequence of segments. The Sequence Comparison is where the two sequences are taken and the Levenshtein Distance, a measure of the difference of the sequences between the two is calculated. This resulting value is expressed as a percentage and can be used to measure entropy. This method is known as the similarity index (Baysa, 2012).

GUI Mockup

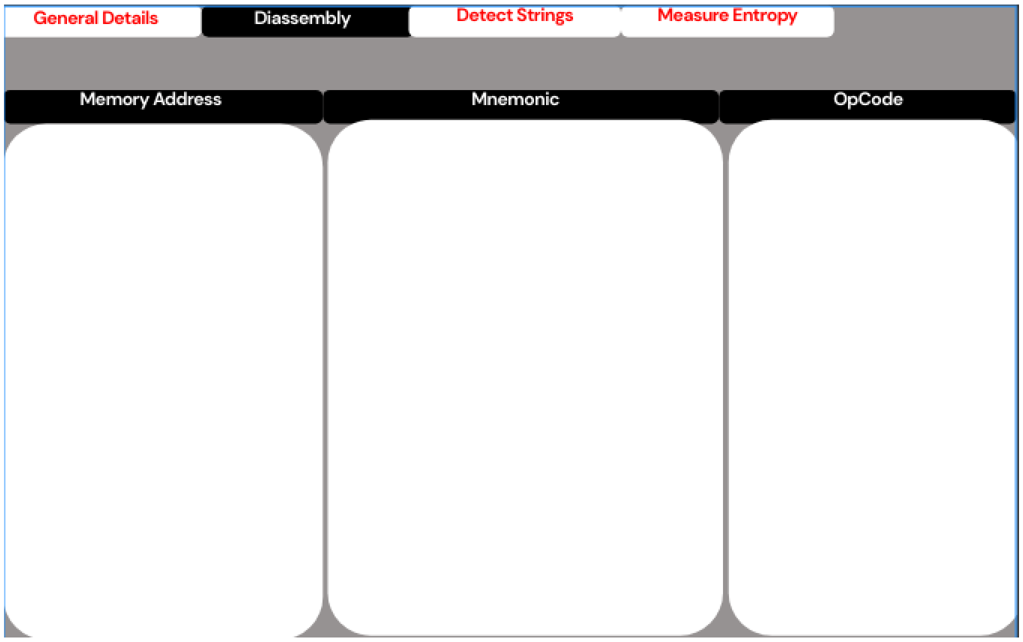
Home Page


Text Welcoming the User to the Application
Button that will prompt user to select a file

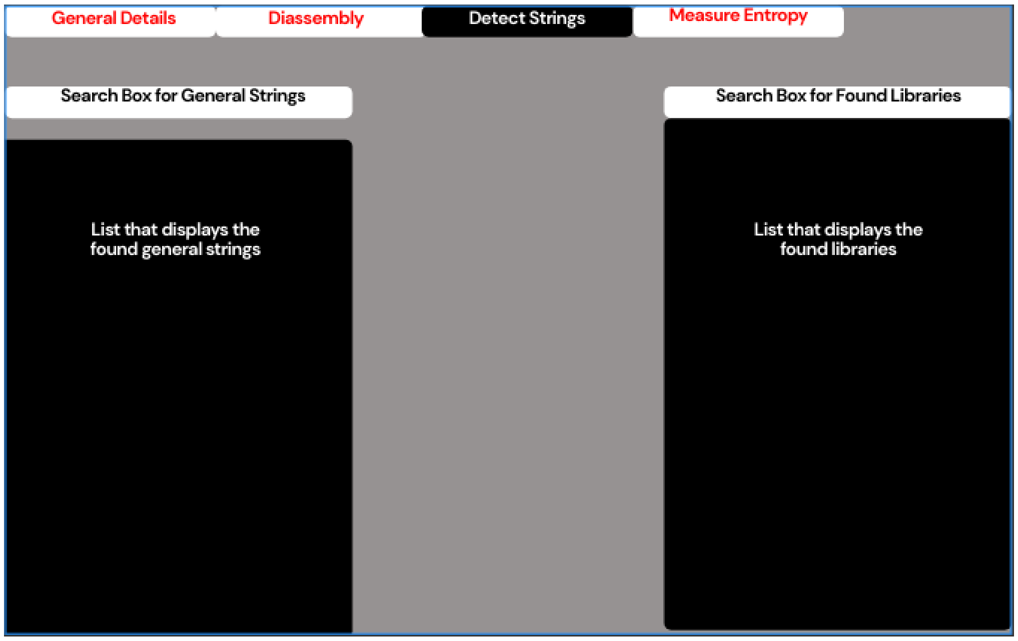
General Details


General Details
Diassembly
Detect Strings
Measure Entropy
Text detailing the general details of the file including file format, date modified and hashes

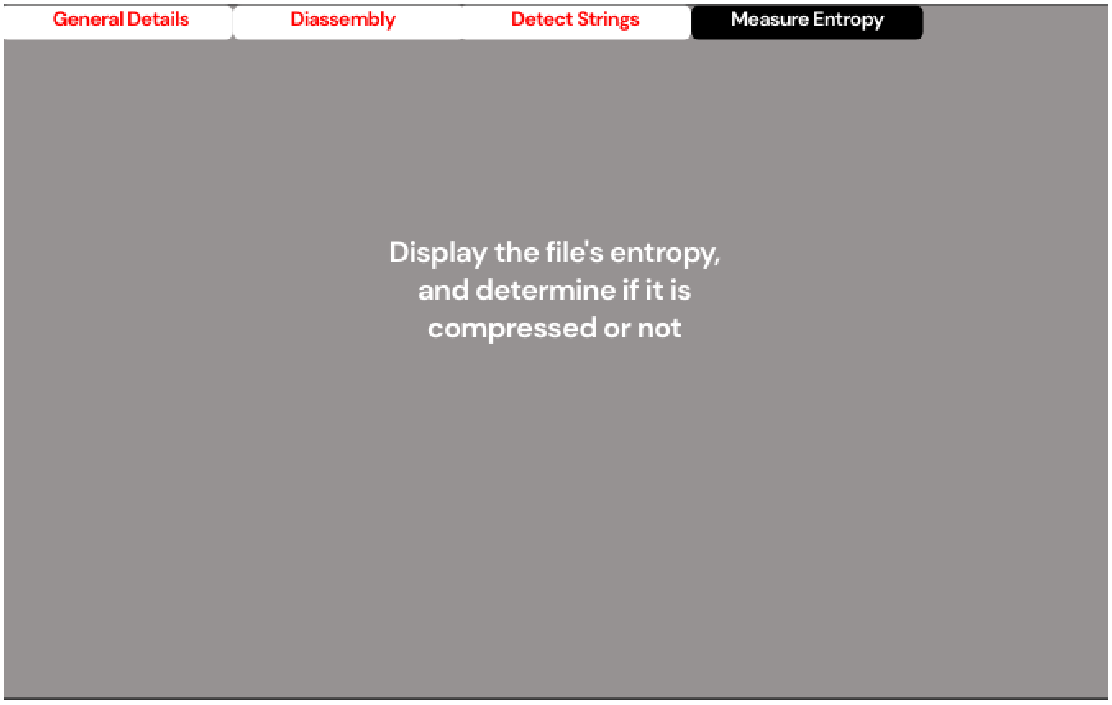
Disassembly



Detect Strings


General Details
Diassembly
Detect Strings
Measure Entropy
Search Box for General Strings
Search Box for Found Libraries
List that displays the found general strings
List that displays the found libraries

Measure Entropy



Display the file's entropy, and determine if it is compressed or not

Metrics

- For the project to be considered complete, it must be fulfill the following criteria:
- The program can run on Linux, with an easy-to-use GUI.
- The program can preform disassembly from machine code of a given file into assembly language, with details including the compiler used and date of creation.
- The program can go through the file and find all ASCII patterns, thus detecting strings and imported libraries.
- The program can express the entropy of the file in a given percentage.

References

Baysa, D. (2012). *Structural Entropy and Metamorphic Malware .* San Jose State University .

Radhakrishnan, D. (2010). *Approximate Disassembly.* San Jose State University .