Static File Analysis Tool

Functional Specification

Brian Tobin

C00216353

Table of Contents

Introduction

Overview

The product that will be developed is a static file analysis tool for students. It will be designed to help students learn about the basic techniques used during static file analysis. There is no existing tool like this, all the current tools are more specialised towards certain techniques and offer far more in depth features than are required by students. This product will allow students to get everything they need from one product, rather than requiring multiple complex tools. This product will require the ability to take any file as an input, generate a hash of the file and backup the file. It should be able to check if a file is packed using UPX, pack and unpack files using UPX, find strings in the file, find DLL's from the strings output, view the file in hex and disassembled modes. There should be a checklist that the user will follow which will also give a description of what the techniques are and why they are being used.

Business Context

This product could be deployed to almost any English speaking school or college across the world. Windows is the dominant operating system in the world with just under 80% market share and Linux is only around 1.5% and macOS around 15%. The vast majority of students that would want this product will also have access to a Windows computer and the percentage of them having access to Linux computers may be disproportionately higher than the global market share because Linux distributions such as Kali Linux are popular for cyber security.

General Description

Product Functions

The product is a tool to help reverse engineering or malware analysis students learn about static file analysis. It should have a checklist which users can follow in sequence to perform each task. The checklist should have a description about what each task is, why they are done, and help on how to use them. The tool should be able to take a file as an input, get a hash of it, then backup the file. It should be able detect if the file is packed, and attempt to unpack it. The strings function should search the file and output a list of strings, then allow the user to choose noteworthy strings from the list to be saved in a separate window. The tool should find imported DLL names and functions and give a general description of what they are and examples of common DLL's. It should also be able to view the file in hex mode, with decoded text in a column on the right. Finally, it should have a disassembly mode where the disassembled code is laid out in an easy to read way.

User Characteristics and Objectives

The objective of the product is to help students understand the basic principles of static file analysis. The students are expected to have basic skills with using computer applications. The users require the product to be easy to use and to be able to adequately teach them about the various techniques used.

Operational Scenarios

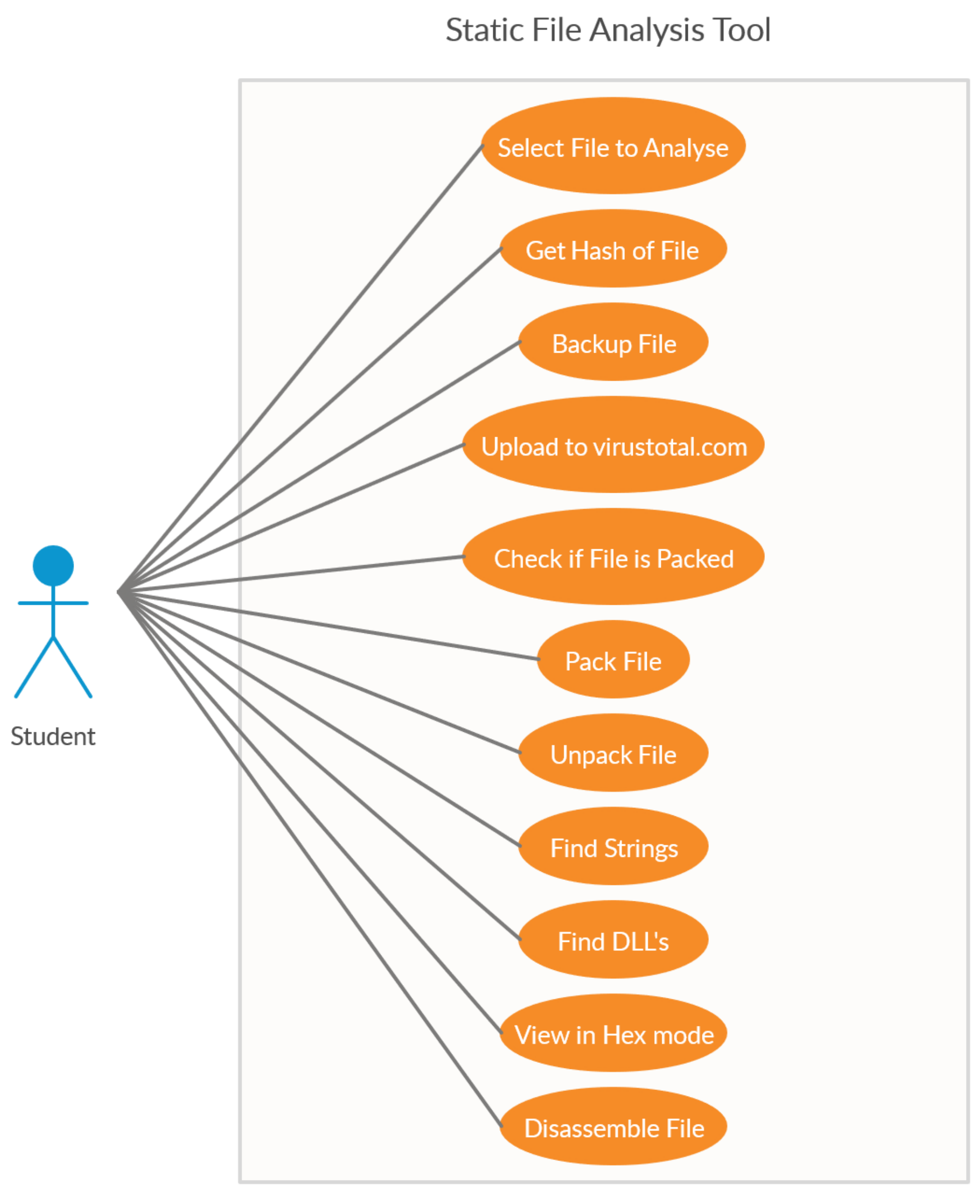

Figure 1: Use Case Diagram

Functional Requirements

Select File to Analyse

Actor

Student

Description

The user must be able to select the file they want to analyse. This should be done by opening the default file explorer for a given operating system and allowing any file to be selected. This use case begins when the program is first run and also when the user clicks the file button to select another file to analyse.

Successful Scenario

1. The user clicks on the “file” button.
2. The user selects the file they want to analyse from the file explorer.
3. The program display a success message.

Alternatives

- 2a. The file is too big (2GB limit).
    - The program displays an error message.
- 2b. There is not enough system memory to copy the selected file into memory.
    - The program displays an error message.
- 2c. No file selected
    - The program displays a message that no file is selected.

Disassemble the File

Actor

Student

Brief Description

The product should be able to disassemble a file into assembly code. This is the final step of static file analysis and by using the information gathering from previous functions, it will be used to try and determine what a file does when executed. This use case begins when the user wants to view the disassembled code of the current file.

Successful Scenario

1. The user clicks on the "disassemble" button.
2. The program disassembles the current file.
3. The disassembled code is displayed on the screen.

Alternatives

- 2a. The current file cannot be disassembled (not a PE file).
  - An error message is displayed to the user.
- 2b. The current file cannot be disassembled (general error).
  - An error message tells user the file cannot be disassembled.
- 2c. No file selected
  - The program displays a message that no file is selected.

Find Strings

Actor

Student

Description

The product should be able to search a file and create a list of strings, where strings are defined as a sequence of alphanumeric characters of at least 3 characters in length and terminated by a null character. The user should be able to select the strings that they think will be useful and these selections should be saved for later. This use case begins when the user wants to view strings in the current file.

Successful Scenario

1. The user clicks on the "strings" button.
2. The program searches through the current file for strings.
3. The program displays the strings on the screen.
4. The user selects the strings they want to keep for later.
5. The program writes these strings in the "saved strings" section.

Alternatives

- 2a. No file selected
    - The program displays a message that no file is selected.

Find DLL’s

Actor

Student

Description

The product should be able to find the DLL names and imported function names that are used by the selected file. This use case begins when the user want to get information on the DLL’s that are used by the current file.

Successful Scenario

1. The user clicks on the “DLL” button.
2. The program finds any DLL’s used by the current file.
3. The program displays the DLL’s on the screen.

Alternatives

- 2a. No DLL’s are found.
    - The program displays a message that no DLL’s were found.
- 2b. No file selected
    - The program displays a message that no file is selected.

Check if the File is packed using UPX

Actor

Student

Description

The product should be able to detect if a file is packed using UPX, which is a free and open-source executable packer. UPX should be used because it is the easiest to implement and is only used to demonstrate how packers can affect analysis by obstructing code. This use case begins when the user wants to check if the current file is packed.

Successful Scenario

1. The user clicks the “packer” button.
2. The program displays the packer context menu.
3. The user clicks the “check if packed” button.
4. The program checks if the file is packed using UPX.
5. The program displays a message that the file is packed or not.

Alternatives

- 4a. No file selected
    - The program displays a message that no file is selected.

Unpack the file

Actor

Student

Description

The product should be able to unpack a file that has been packed using UPX. This will allow the users to get more information from the more critical functions. It will work along with the packer detection function, being used if UPX packing is detected. This use case begins when the user has checked if the current file is packed and now wants to unpack the file.

Successful Scenario

1. The user clicks the “packer” button.
2. The program displays the packer context menu.
3. The user clicks the “unpack” button.
4. The program unpacks the current UPX packed file.
5. The program displays a message that the file was unpacked successfully.

Alternatives

- 4a. The file cannot be unpacked for any reason.
  - The program displays a message that the file cannot be unpacked.
- 4b. No file selected
  - The program displays a message that no file is selected.

View the File in Hex Mode

Actor

Student

Description

The product should have the ability to view a file in hex mode and show the text in a column on the right. This will be used to help the students understand what a file looks like and where the strings are being found. This use case begins when the user wishes to view the current file in hex mode.

Successful Scenario

1. The user clicks on the “hex” button.
2. The program displays the current file in hex mode.
3. The user looks through the output.

Alternatives

- 2b. No file selected
    - The program displays a message that no file is selected.

Pack the file

Actor

Student

Description

The product should be able to pack a file using UPX. This should be used to demonstrate how packing a file can obstruct code and limit what can be found through analysis. The users should be able to pack a file that they created. This use case begins when the user has checked if the file is packed and now wants to pack the file.

Successful Scenario

1. The user clicks the “packer” button.
2. The program displays the packer context menu.
3. The user clicks the “pack” button.
4. The program packs the current file using UPX.
5. The program displays a message that the file was packed successfully.

Alternatives

- 4a. The file cannot be packed for any reason.
  - The program displays a message that the file cannot be packed.
- 4b. No file selected
  - The program displays a message that no file is selected.

Generate Hash of File

Actor

Student

Description

The product must be able to generate a hash value of a given file. The hashing algorithm used does not matter as long as it is relatively new. Hashing algorithms such as MD5 or SHA-1 are popular and will work good enough as the hash will only be used to verify if the file has changed in any way. This use case begins when the user wants to get the hash of the current file.

Successful Scenario

1. The user clicks the “hash” button.
2. The program generates a hash of the current file.
3. The program displays the hash in a pop up box.

Alternatives

- 2a. The hash cannot be generated.
  - The program displays a message that the hash could not be created.
- 2b. No file selected
  - The program displays a message that no file is selected.

Backup the File

Actor

Student

Description

The product must be able to back up the file that the user selects. This is done in case the file is altered which can be verified using the hash that they should have generated if they follow the checklist properly. This use case begins when the user wants to create a backup of the file.

Successful Scenario

1. The user clicks the "backup" button.
2. The program creates a backup of the current file.
3. The program displays a success message.

Alternatives

- 2a. Creating a backup has failed.
  - The program displays a message that the backup has failed.
- 2b. No file selected
  - The program displays a message that no file is selected.

FURPS+

FURPS+ is a model used to identify requirements that are not defined in a use case. It stands for Functionality, Usability, Reliability, Performance and Supportability and the plus is for defining other constraints such as design, implementation or interface constraints. It is useful for defining technical and non-technical requirements of projects that may not be considered otherwise.

Functionality

The product should have a checklist that can be used to know the sequence that the functions should be completed in. It should be a list of all the functions that the program will teach in the order that they should be completed in. It should also have a link for each item that will give the users a description about what the technique is and why they are doing it. The checklist is one of the most important functions and it is difficult to define with a single use case as it is used at many points throughout the application.

Usability

As the application is for students, it should be intuitive and easy to use. Disassembly code is particularly hard to read and should be displayed using colors and spaces between sections to help the users to try and understand it better. There should be a consistent design between all the screens of the application.

Reliability

The application does not have much reliability requirements. It should be able to handle all edge cases and any errors and not crash. There are no external connections and the whole application is standalone. Any files used are all stored locally.

Performance

Most of the functions of this application are fast to perform as long as the file is not huge. Disassembly is the most intensive function and this along with some of the other functions can be executed in the background once a file is selected so they can be displayed near instantly when the user wants to use that function.

Supportability

The application should run on both Windows and Linux. It should be adaptable in the sense that it could be easily updated to support more types of packers or to add new functions for example. It should also be able to be compiled for other operating systems if needed in the future.

+

There may be some extra constraints with interfacing a disassembler and also sending a file to virustotal.com.

Metrics

- The application must at least be able to select a file, perform disassembly, find strings, find DLL’s, check if packed and provide a checklist.
- It must run on Windows at a minimum and Linux if there is enough time.
- It must be easy to use by students.

Testing

To test this application, it should be given to Cybercrime and IT Security students as they are a perfect example of the target users. They should be given a quick introduction to what the application is used for and then be given an hour or so to try out all the different functions. A questionnaire should be filled out by them at the end where they give their opinion on how easy it was to use, how much they feel they learned about static file analysis, and how the application could be improved. These questionnaires can be used to determine how successful this project was, and how it could be improved in the future.

Project Plan

There will be 12 weeks to complete this project, starting on the 6th of January. This is the development plan.

Weeks 1 -2

Get Qt setup and create a basic user interface that functionality can be added to over time. Create the pop up dialog boxes and allow them to be interacted with.

Weeks 3 – 4

Be able to open a file, create a backup and generate a hash. Also be able to detect if the file is packed using UPX and get the entropy.

Weeks 5 – 6

Be able to find strings and also display and edit the bytes in hex edit mode.

Weeks 7 – 8

Be able to find imported DLL's and function names.

Weeks 9 - 10

Be able to disassemble PE files.

Weeks 11 - 12

Create the checklist pages and add any extra functionality if there is time. Polish the project, create final binaries.