

Static File Analysis Tool

Design Manual

Brian Tobin

C00216353

Table of Contents

Introduction.....	2
Sequence Diagrams.....	3
Open Application and Select a File	3
Generate Hash and Backup Current File	4
Check if file is packed and unpack	5
Display and select strings	6
Display Hex	7
Display Disassembly.....	7
User Interface Wireframes	8
Main Window	8
Pop Up Windows	9
Main Window Displays	10
Strings	10
Hex	11
Disassembly	11
Checklist	12
Pseudo-code	13
Find Strings from Byte Array	13
Imported DLL's	14
Disassembly	15

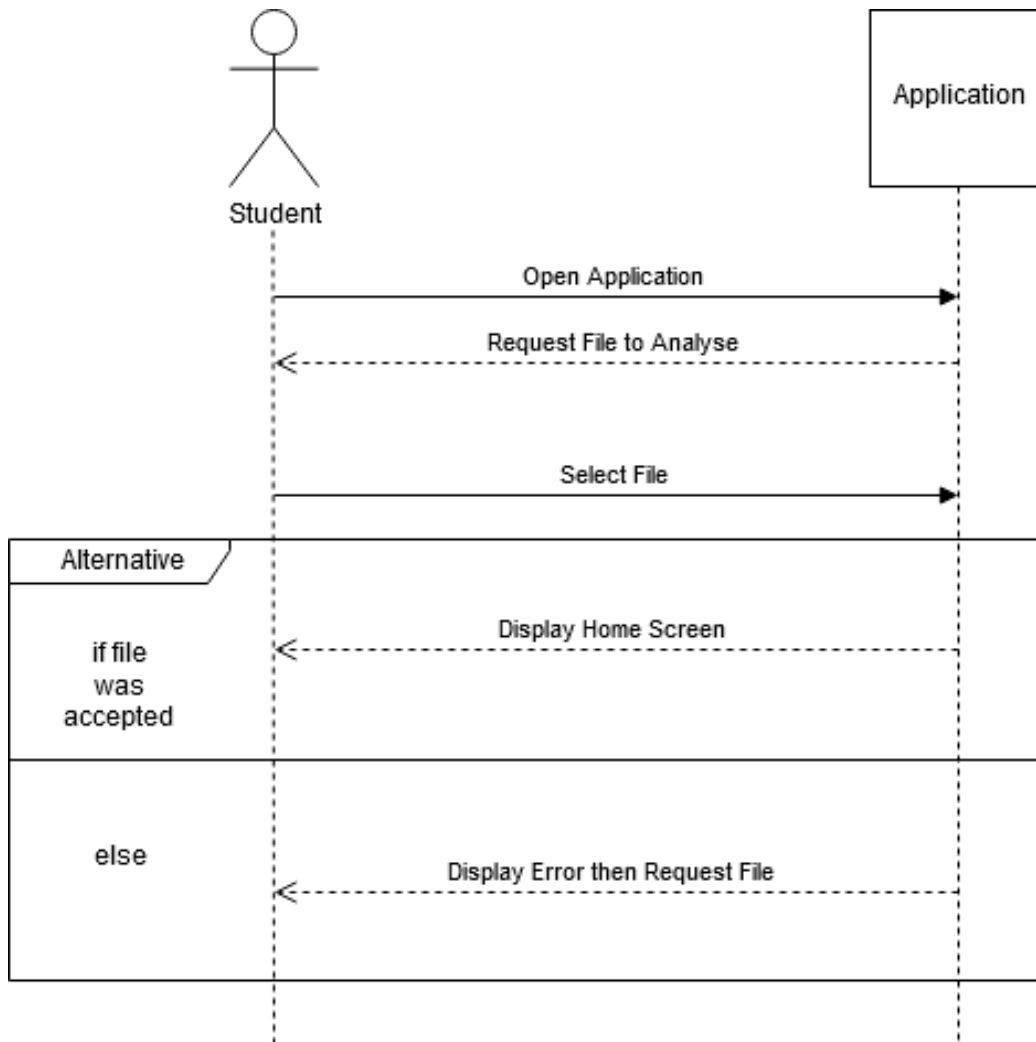
Introduction

This manual will cover the main design features of the application including UML class and sequence diagrams and mockups of the user interface created using Balsamiq. It should give the reader a thorough understanding of how the application should be built, how it is interacted with by the user and what the user interface should look like. The application is a static file analysis tool intended for students learning reverse engineering or malware analysis. The main functionality of this application has been determined in the Functional Specification document. The user interface is kept simple as the application needs to be easy to use.

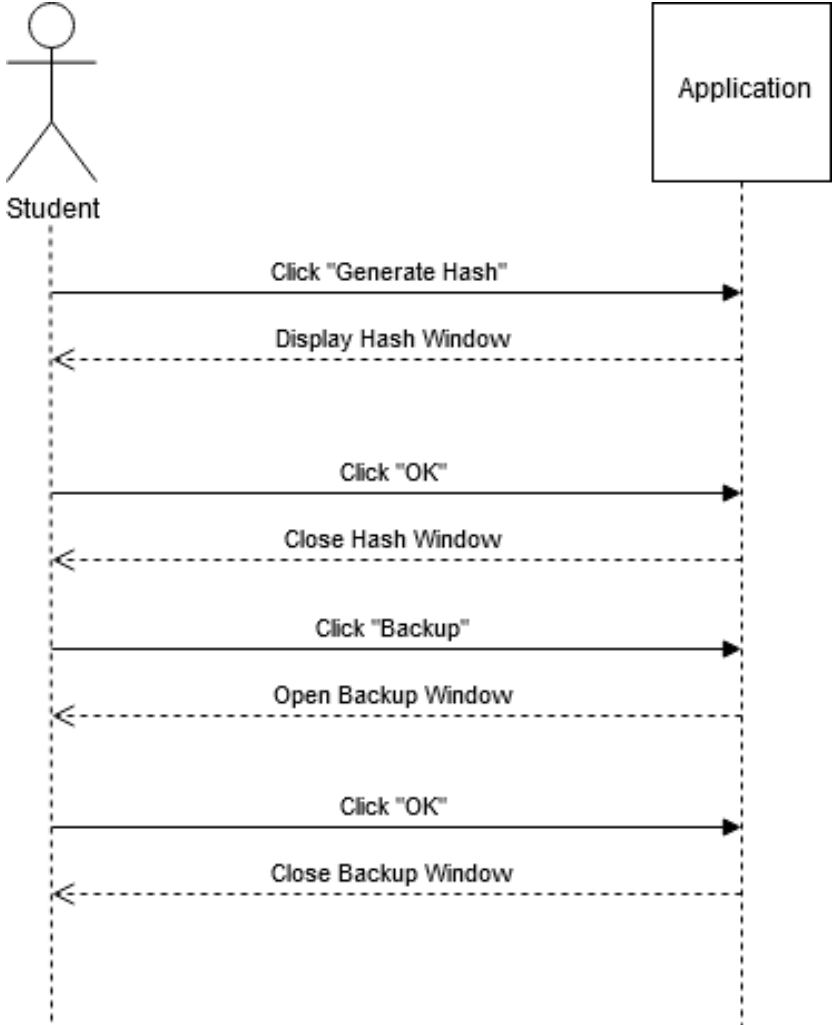
Sequence Diagrams

These sequence diagrams describe the interactions between the user and the application arranged in a time sequence for each main function.

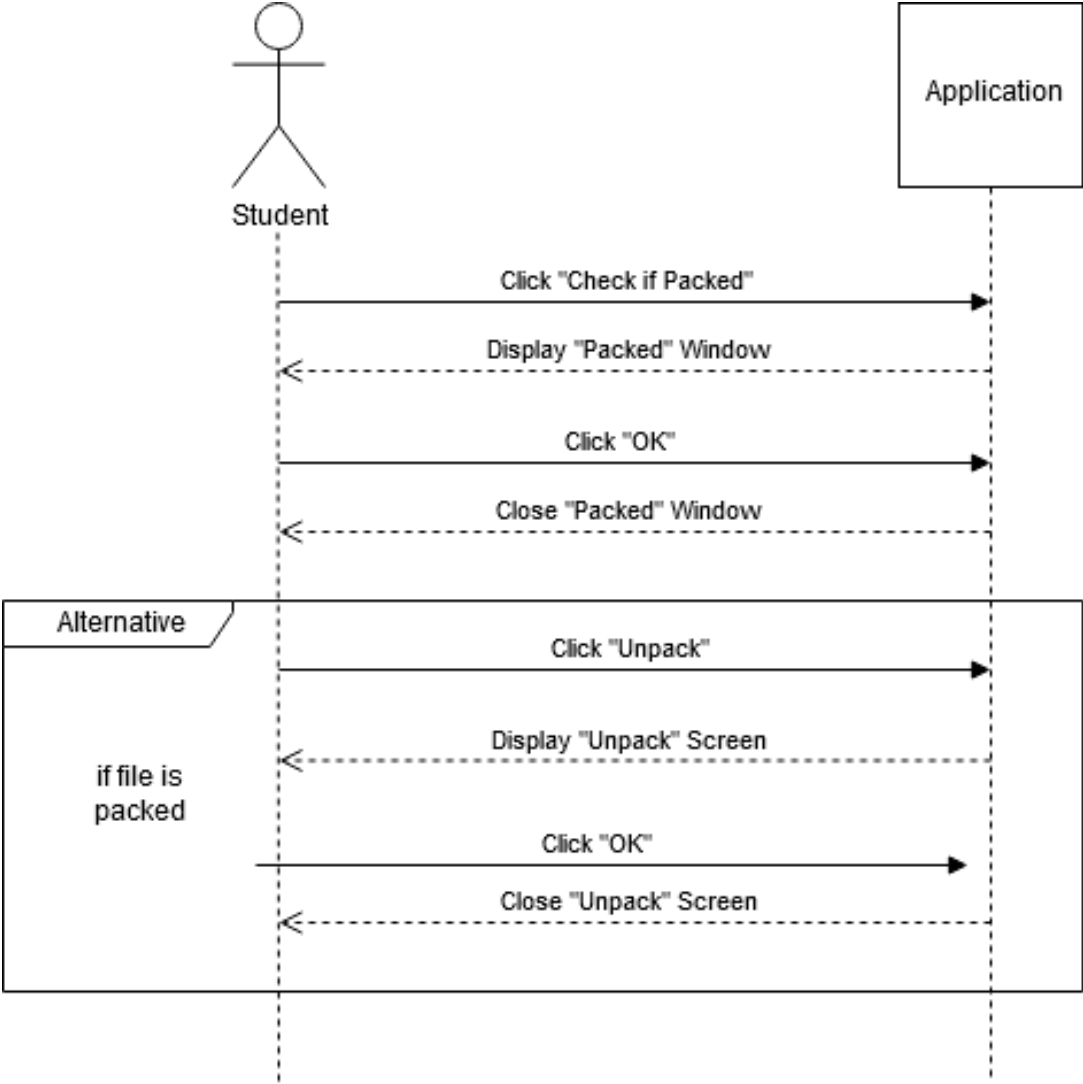
Open Application and Select a File



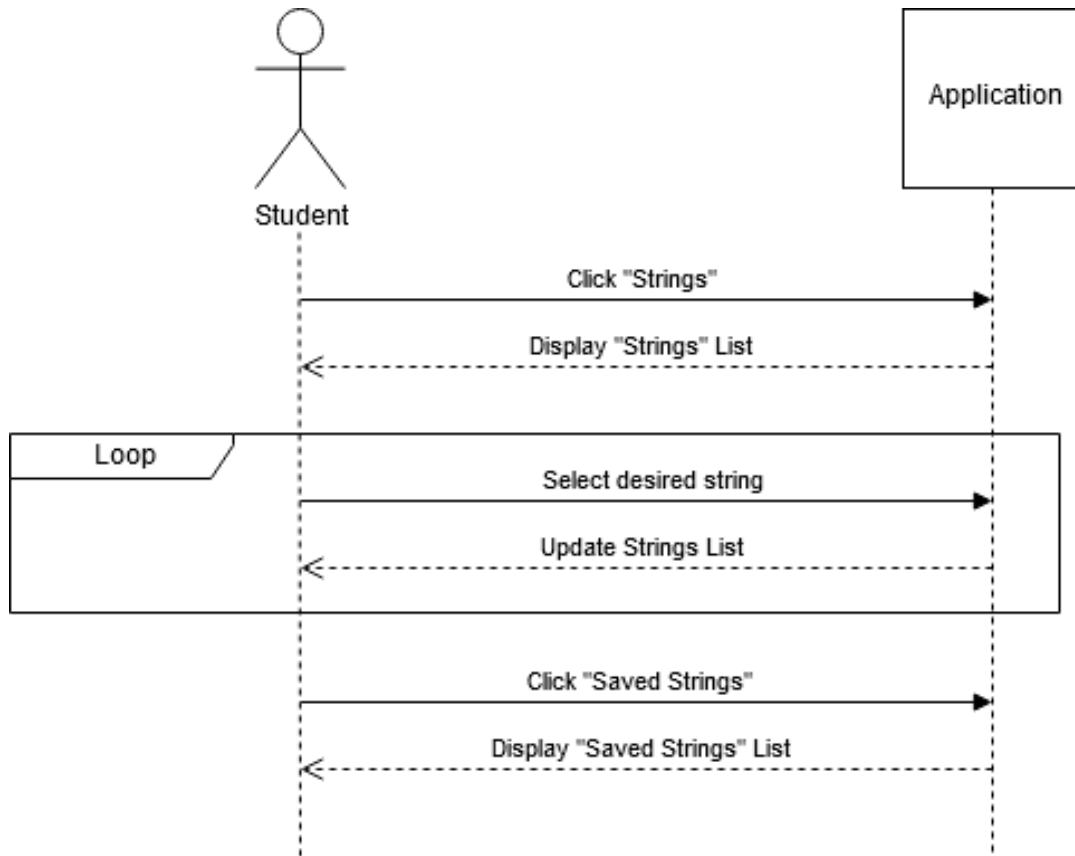
Generate Hash and Backup Current File



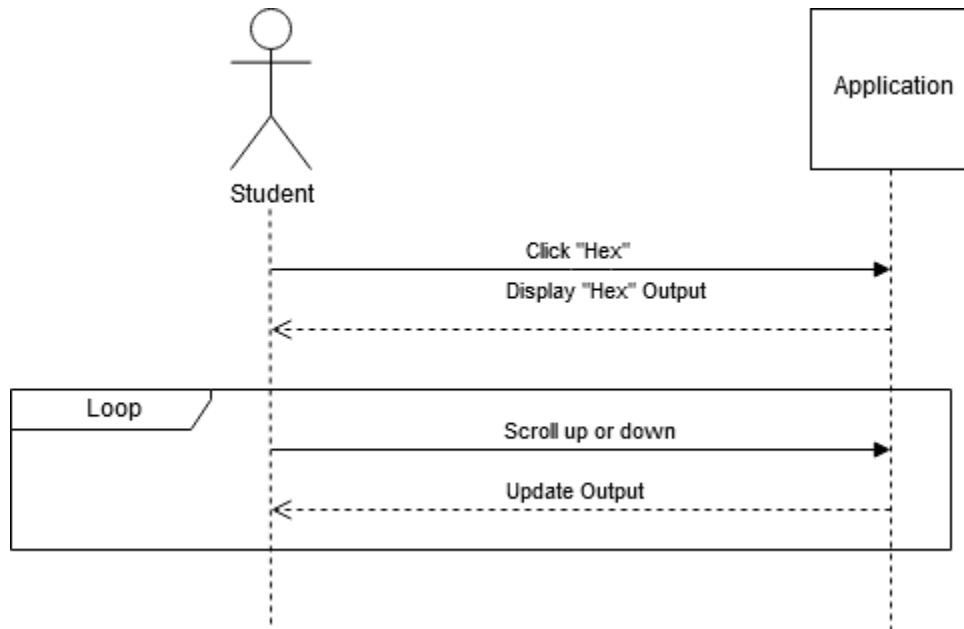
Check if file is packed and unpack



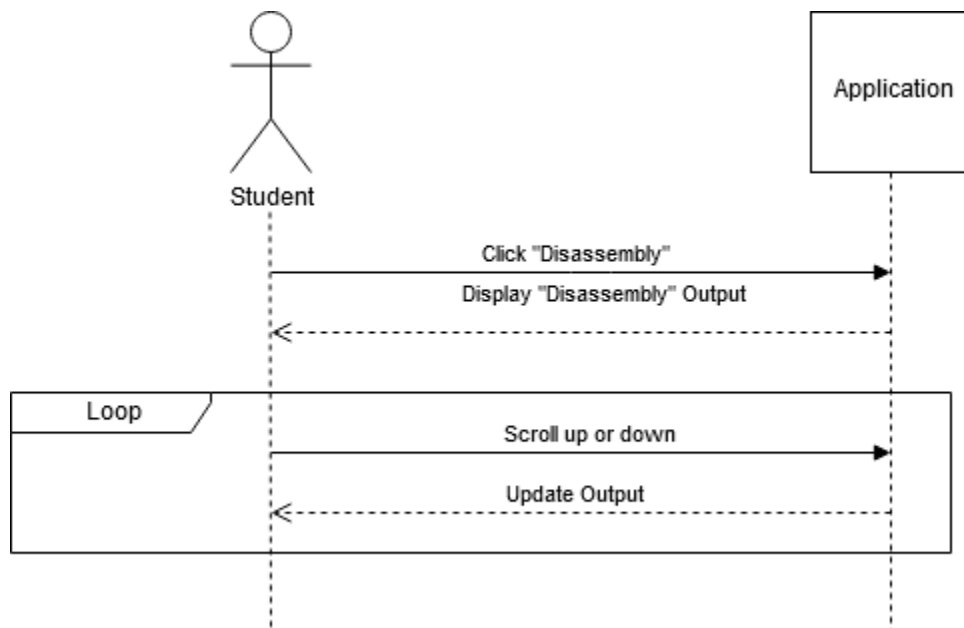
Display and select strings



Display Hex



Display Disassembly



User Interface Wireframes

Main Window

The mockup window below is what the main window could look like. The title of the window contains the name and hash of the current file. A menu bar just below this with the main functions of the application. The “File”, “Strings” and “Packers” options can each have a small menu window as seen below in Figures 2, 3 and 4, while the remaining options will be displayed in the big white space at the center of the main window seen in Figure 1.

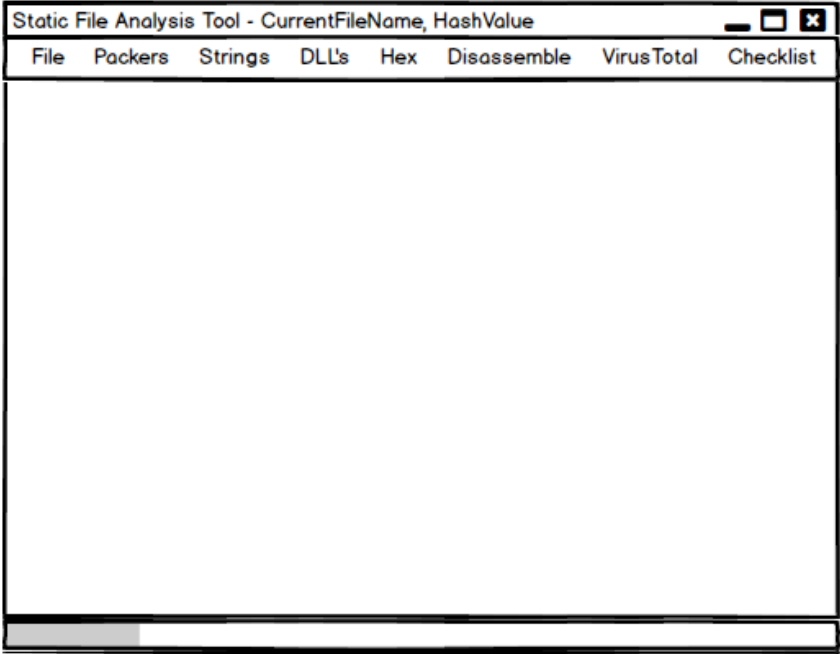


Figure 1: Main window

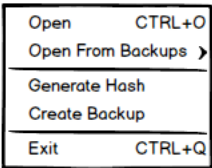


Figure 2: File Menu

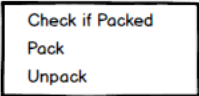


Figure 3: Packer Menu

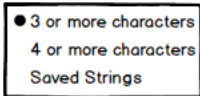


Figure 4: Strings Menu

Pop Up Windows

A few sample windows that will be displayed when the user chooses options from the “File” or “Packers” menus are shown below. They do not provide much information and don’t need to take up the whole main window, so they are just pop ups that can be dismissed by clicking “OK”.

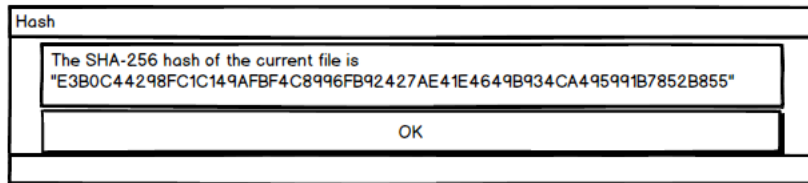


Figure 5: Hash pop up

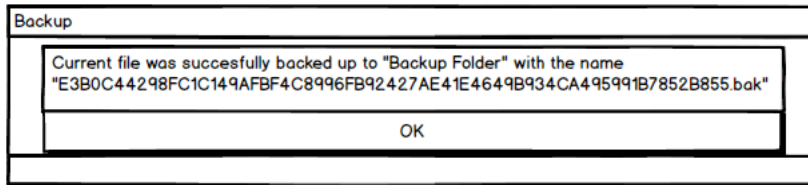
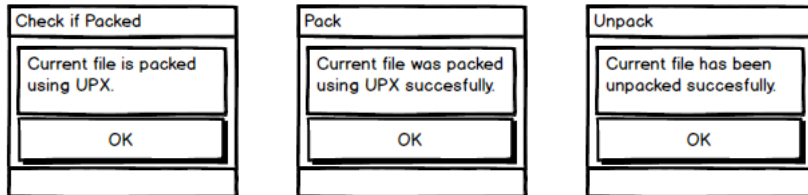


Figure 6: Backup pop up



Figures 7, 8 and 9: Packer pop ups

Main Window Displays

These options from the main windows menu bar will be displayed on the main window. They are much more detailed than the previous options requiring the user to interact with the output more than just clicking “OK”. The user should be able to move between these screens seamlessly with the application remembering how far they scrolled down for example.

Strings

The strings output is a list of the strings found in the current file, allowing the user to select the strings they find interesting. The strings menu allows the user to select between 3 or 4 character minimum strings and the saved strings section containing all the strings they selected.

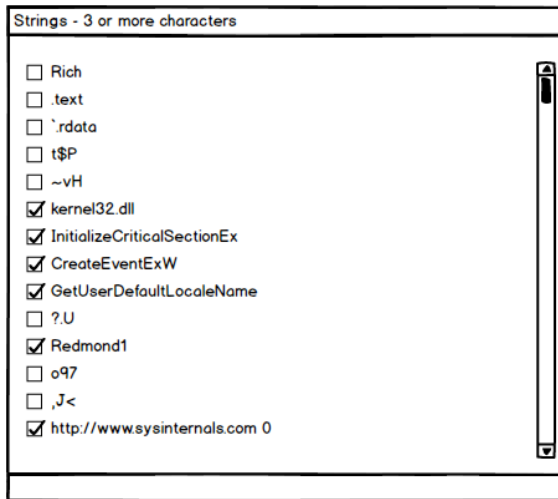


Figure 10: Strings output

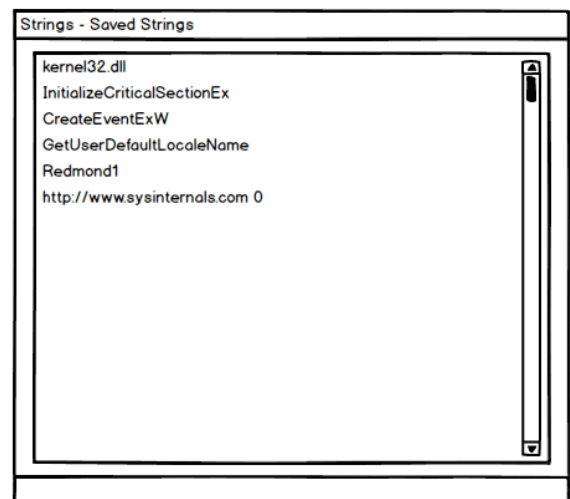


Figure 11: Saved strings

Hex

The hex mode displays the current file in hex on the left and the ASCII value of the hex on the right. The user can scroll through this screen and find sentences made up of the individual strings found earlier.

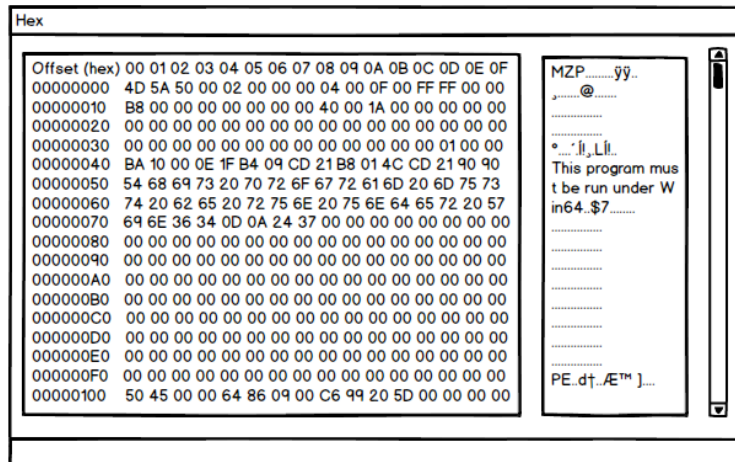


Figure 12: Hex mode

Disassembly

In disassembly mode the current file is disassembled and displayed on the screen. The left of the display shows the hex value and the position it is located in the file. The right of the display shows what the hex value is interpreted as in assembly code.

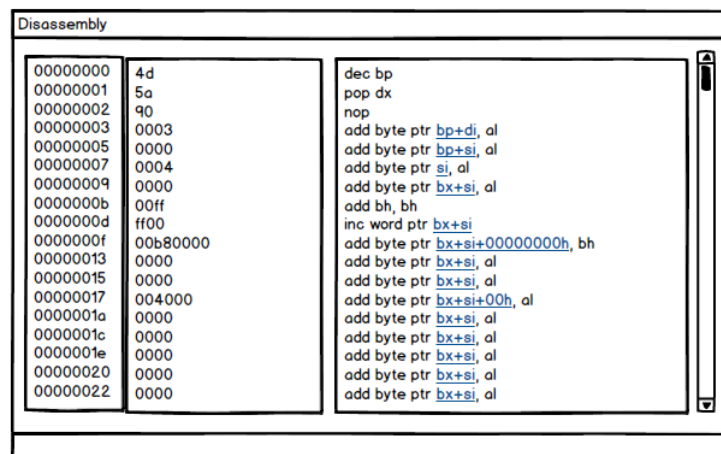
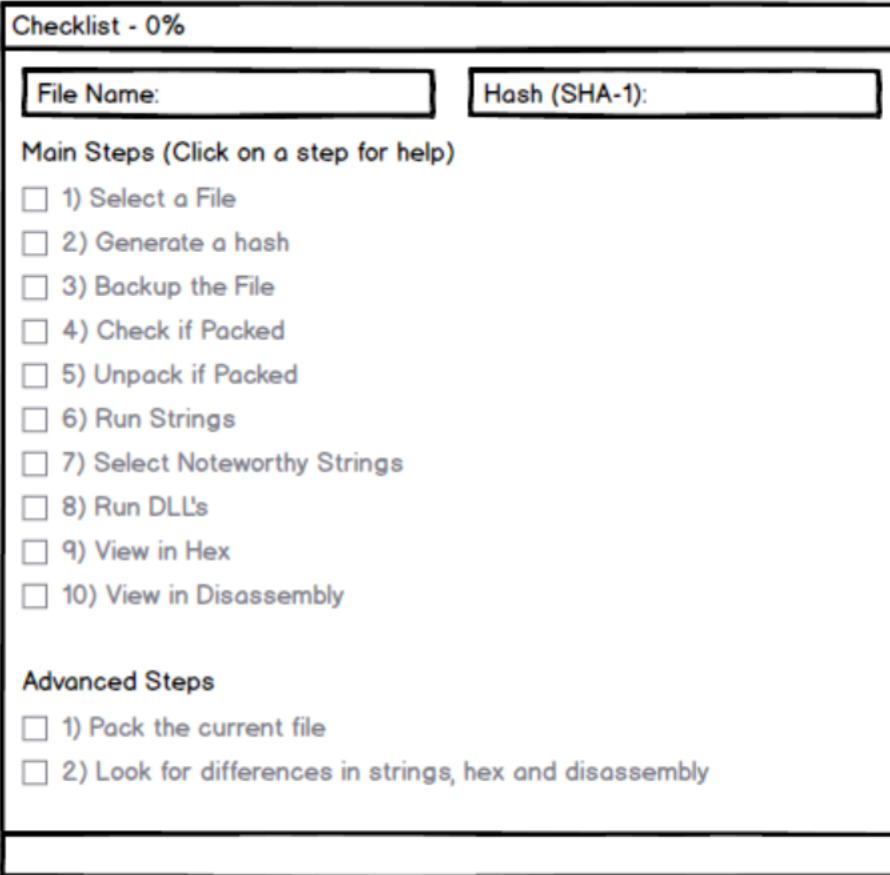


Figure 13: Disassembly mode

Checklist

The checklist is one of the most important screens which guides the user on what to do. This is a very basic example of what they checklist should look like. It should have a list of steps and a way to view more information on each step.



The screenshot shows a window titled "Checklist - 0%". At the top, there are two input fields: "File Name:" and "Hash (SHA-1):". Below these fields, the text "Main Steps (Click on a step for help)" is displayed. Underneath, there is a list of ten main steps, each preceded by an unchecked checkbox. Below the main steps, the text "Advanced Steps" is displayed, followed by two advanced steps, each preceded by an unchecked checkbox.

Checklist - 0%

File Name: Hash (SHA-1):

Main Steps (Click on a step for help)

- 1) Select a File
- 2) Generate a hash
- 3) Backup the File
- 4) Check if Packed
- 5) Unpack if Packed
- 6) Run Strings
- 7) Select Noteworthy Strings
- 8) Run DLL's
- 9) View in Hex
- 10) View in Disassembly

Advanced Steps

- 1) Pack the current file
- 2) Look for differences in strings, hex and disassembly

Figure 14: Checklist

Pseudo-code

Find Strings from Byte Array

This function finds all strings in a file. A file is copied into a byte array. A string is defined as 3 or more characters between 32 and 126 inclusively on the ASCII table.

Initialize tempString to 0

Initialize stringList to 0

For each byte in the byte array

 If the current byte is between 32 and 126

 Append byte to tempString

 Else if tempString length is greater than or equal to 3

 Append tempString to stringList

 Set tempString to null

 Else

 Set tempString to null

Imported DLL's

This function finds the Import Directory Table and Import Address Table physical addresses from the PE header. The IDT and IAT contains relative pointers to the address of the DLL names and function names which are stored as null terminated strings. This function then follows these pointers and inserts the strings into a string list.

Initialize IDTLocation to the 4 byte integer at PE Header Start Location + 128

Initialize pointerTerminator to false

While pointerTerminator is false

 If pointerValue does not equal 0

 Get string at pointerValue

 Else

 Set pointerTerminator to true

Repeat this for the Import Address Table to get the imported function names.

Disassembly

This function turns all the bytes in the code section of a PE file into a list of instruction.

Initialize codeStartLoc to the 4 byte integer at PE text location + 20

Initialize codeEndLoc to the 4 byte integer at PE text location + 8 - codeStartLoc

Initialize instructionOffset to 0

While codeStartLoc + instructionOffset < codeEndLoc

 Initialize instructionComplete to false

 While instructionComplete is false

 Initialize currentByte to byte array at codeStartLoc + instructionOffset

 If currentByte is a prefix byte

 Handle the prefix byte

 Else if currentByte is an opcode byte

 Handle the opcode byte

 Else if currentByte is a mod reg r/m byte

 Handle the mod reg r/m byte

 Else if currentByte is an SIB byte

 Handle the SIB byte

 Else if currentByte is a Displacement byte

 Handle the Displacement byte

 Else if currentByte is an Immediate byte

 Handle the Displacement byte

 Increment instructionOffset