# Final Project Report

# Maldive
# Dynamic Malware Analysis Tool

**Shane Doherty C00249279**

**Supervisor: Joseph Kehoe**

**South East Technological University**

## 1 Table of Contents

## Contents

## 2 Introduction

This report will document the general description of the project, along with specific features that make it up. These features will include a description, and the challenges, limitations and screenshots that are specific to each. Should it be applicable, my own personal thoughts on how I feel about each will be discussed. In addition, the successes achieved within the project are outlined, along with areas that I feel could have been improved upon.

## 3 Project Description

The aim of this project was to develop a malware analysis tool that would dynamically investigate a program to determine if it contains malicious actions. The application takes in either a Portable Executable (PE) file, which is the file format used with .exe files in the Windows Operating System (OS), or a process that is already running on the OS. In the case of a PE file, the process is launched and the analysis tool hooks into it in the same way as the running process. Several methods are then utilised to analyse the targeted process, which are listed below, along with challenges and how I personally feel about the implementation.

### 3.1 Process Hooking

#### 3.1.1 Description

For the program to be able to monitor the activities of a process, it must first know details to identify which process to look at. To achieve this, the program has the capability to launch a process by itself and gather the relevant data when doing so. If the user wants to use a currently running process, a view of each one is shown and can be selected from. In either case, the process ID (PID) that the OS uses is also used in the project for the majority of the functionalities. As multiple processes can be in use by a Windows program, this allows for individual ones to be used and identified.

#### 3.1.2 Limitations

While most programs have full functionality, this is not the case for some system-level processes. They require higher elevation of permissions than can be given to a user program, so certain actions are not possible. In terms of malware analysis, this is not a concern as system processes are considered safe by default.

#### 3.1.3 Challenges

There were no noteworthy challenges to overcome when identifying specific processes to be used in the analysis tool, other than correctly formatting the running processes that are retrieved from Windows. This was a quick feature to implement and development using this method of process hooking was able to begin in a relatively fast amount of time.

### 3.1.4 Screenshots

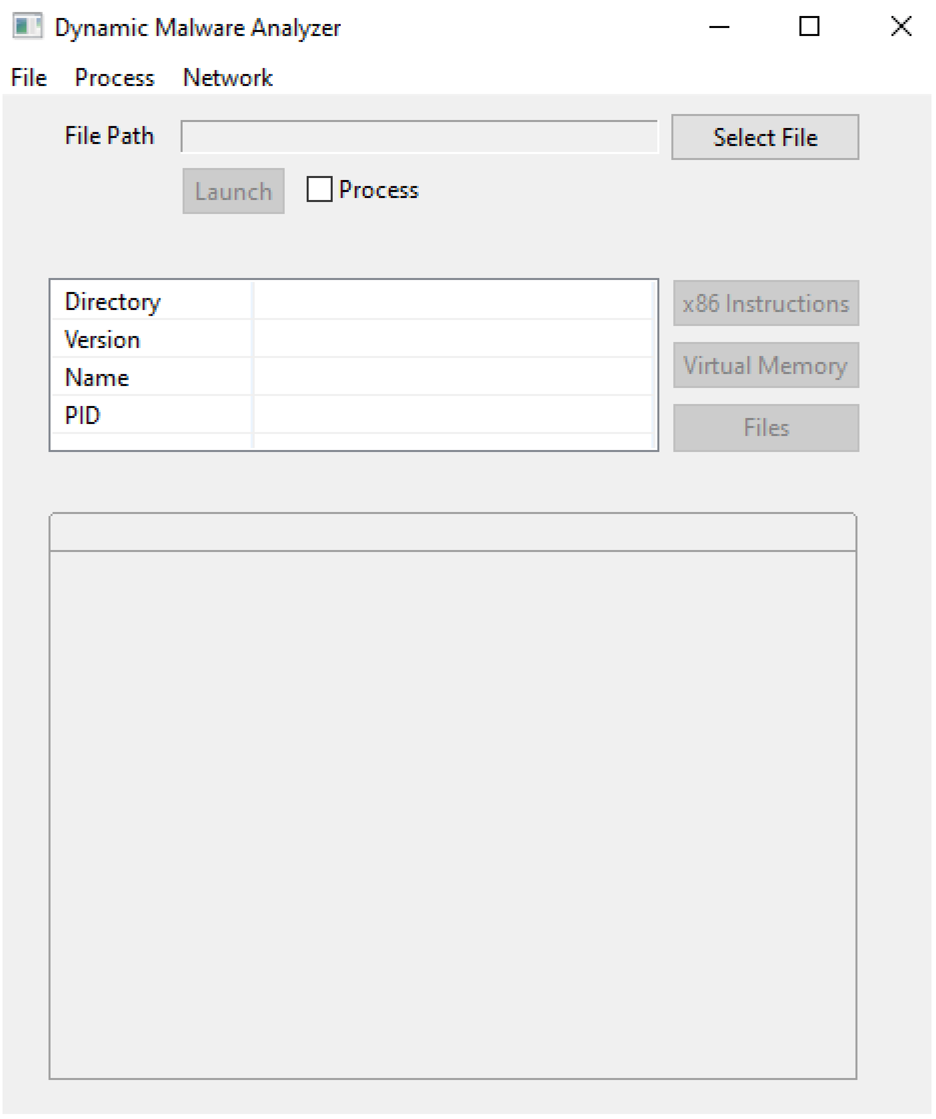
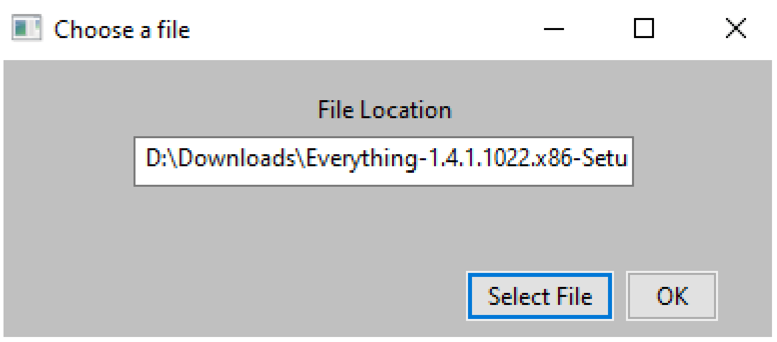


**Fig.1.** Main window without a process attached

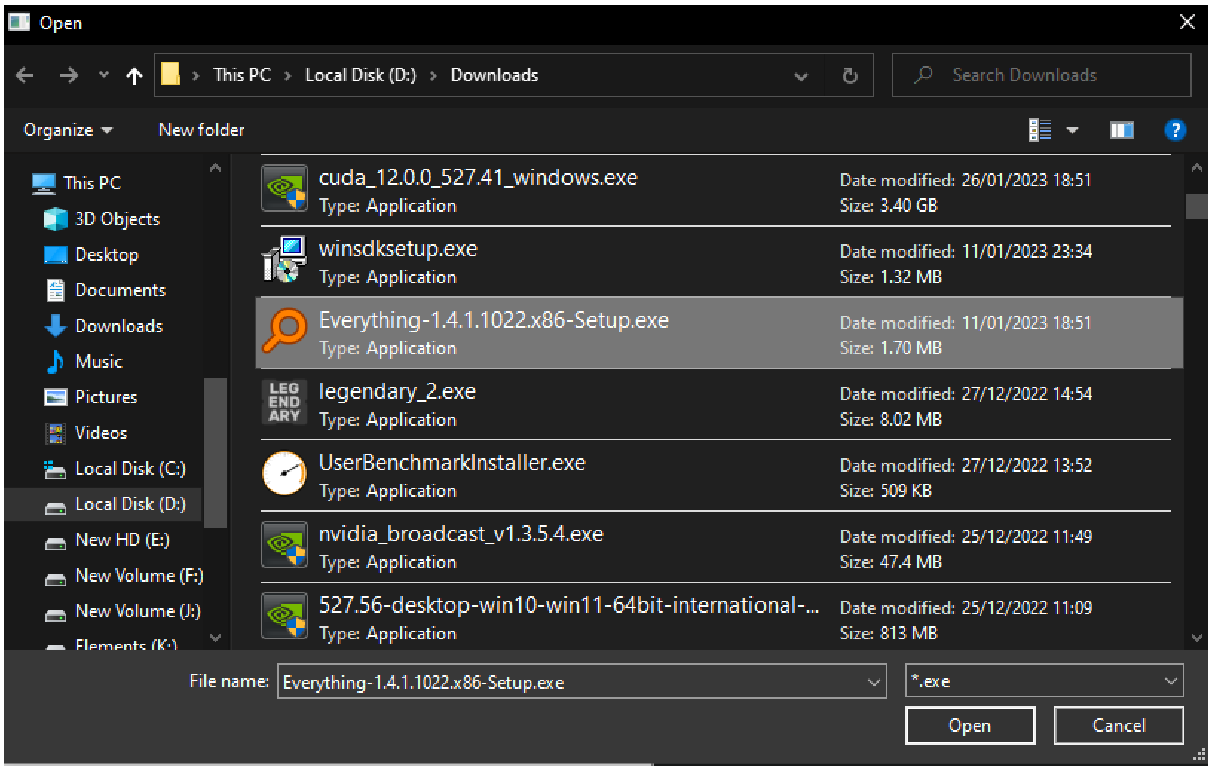
This window is the first screen shown without a process being selected. It presents an option to choose a file, or alternatively a running process can be selected by using the *Process* checkbox.


Choose a file

File Location

D:\Downloads\Everything-1.4.1.1022.x86-Setu

**Fig. 2.** Select file screen


Open

This PC › Local Disk (D:) › Downloads

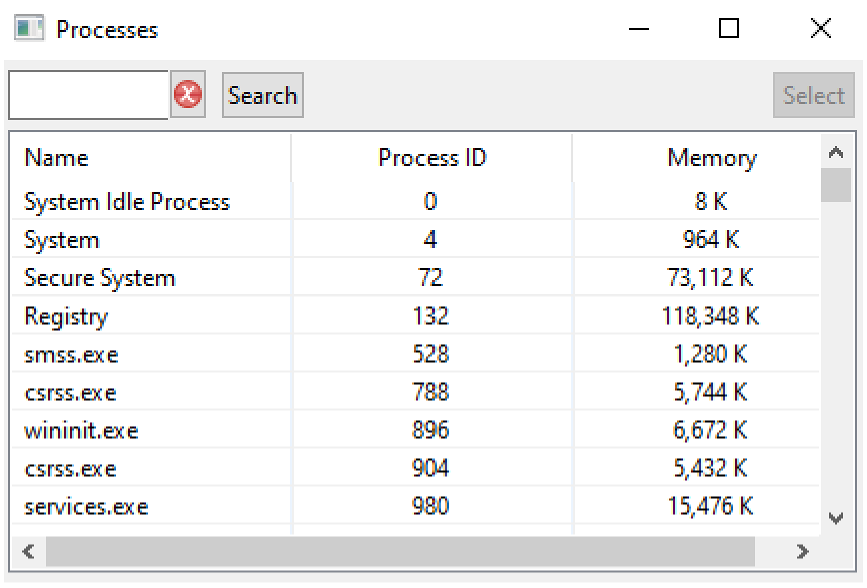
| | | |
|---|---|---|
| cuda_12.0.0_527.41_windows.exe | Type: Application | Date modified: 26/01/2023 18:51 Size: 3.40 GB |
| winsdksetup.exe | Type: Application | Date modified: 11/01/2023 23:34 Size: 1.32 MB |
| Everything-1.4.1.1022.x86-Setup.exe | Type: Application | Date modified: 11/01/2023 18:51 Size: 1.70 MB |
| legendary_2.exe | Type: Application | Date modified: 27/12/2022 14:54 Size: 8.02 MB |
| UserBenchmarkInstaller.exe | Type: Application | Date modified: 27/12/2022 13:52 Size: 509 KB |
| nvidia_broadcast_v1.3.5.4.exe | Type: Application | Date modified: 25/12/2022 11:49 Size: 47.4 MB |
| 527.56-desktop-win10-win11-64bit-international-... | Type: Application | Date modified: 25/12/2022 11:09 Size: 813 MB |

File name: Everything-1.4.1.1022.x86-Setup.exe

*.exe

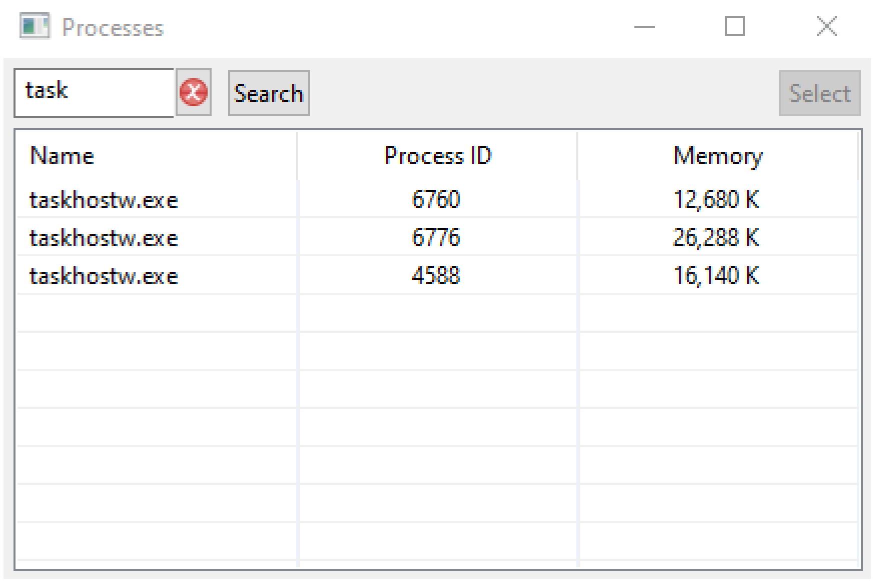
**Fig. 3.** File explorer window to choose a file

Should the user choose to open a file, the *Choose a file* screen is displayed. A file directory may be entered into the *File Location* text field, or the *Select File* button can be used to open the Windows Explorer where a file can be selected from the user's file directory. Once a file is chosen, the *OK* button sends the chosen file back to the main window to prepare a process to be created from the executable.



| Name | Process ID | Memory |
|---|---|---|
| System Idle Process | 0 | 8 K |
| System | 4 | 964 K |
| Secure System | 72 | 73,112 K |
| Registry | 132 | 118,348 K |
| smss.exe | 528 | 1,280 K |
| csrss.exe | 788 | 5,744 K |
| wininit.exe | 896 | 6,672 K |
| csrss.exe | 904 | 5,432 K |
| services.exe | 980 | 15,476 K |

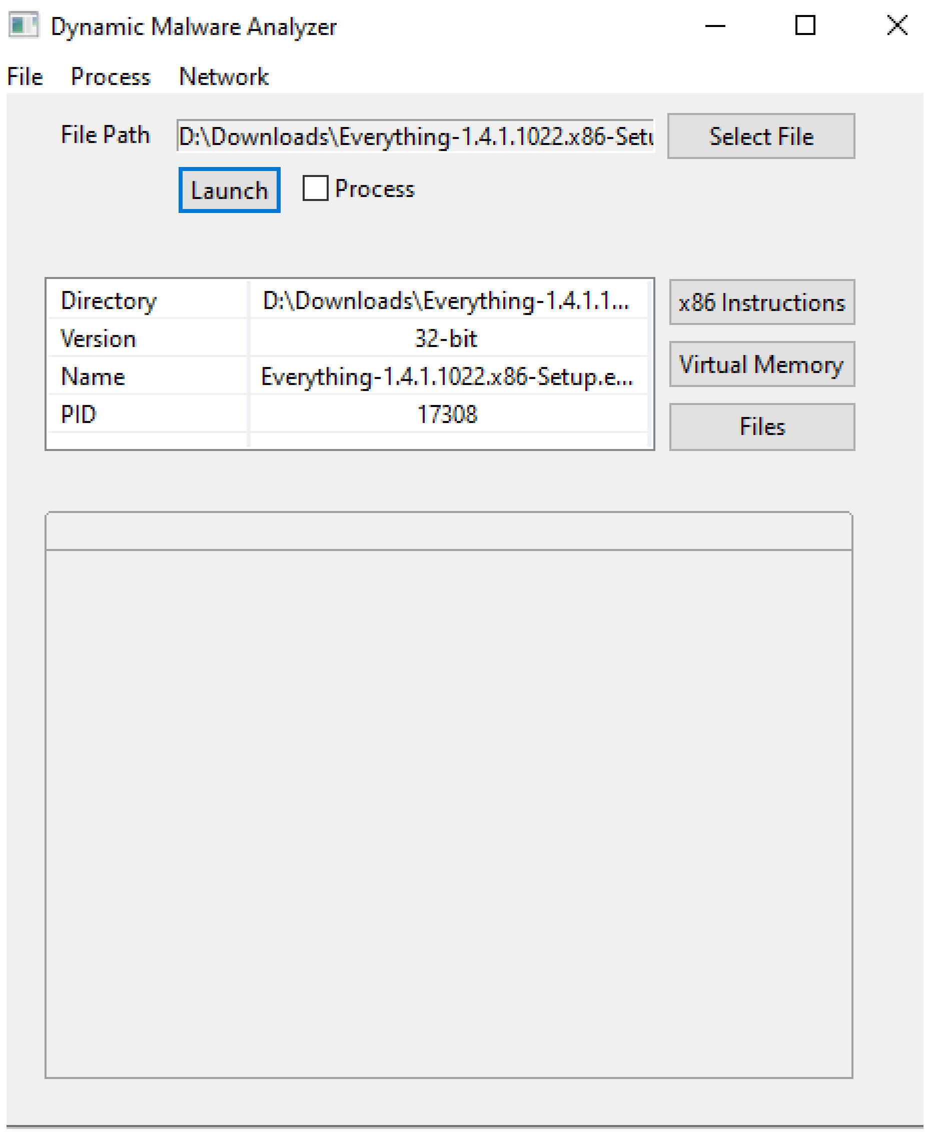
**Fig.3.** Select Process



| Name | Process ID | Memory |
|---|---|---|
| taskhostw.exe | 6760 | 12,680 K |
| taskhostw.exe | 6776 | 26,288 K |
| taskhostw.exe | 4588 | 16,140 K |

**Fig.4.** Select Process, search feature

Should the user instead choose to select a currently running process, a window is shown that displays all processes that the OS is running. This window displays the *Name* that the OS can see for the process, along with the unique *Process ID* (PID) that identifies the specific instance of a process. The *Memory* that the process is currently allocated is also shown, displayed in kilobytes. A search feature is implemented to filter the names of processes. When a process is selected, the PID is sent back to the main window in preparation for the analysis tool to hook into the process.



D:\Downloads\Everything-1.4.1.1022.x86-Setu

| | |
|---|---|
| Directory | D:\Downloads\Everything-1.4.1.1... |
| Version | 32-bit |
| Name | Everything-1.4.1.1022.x86-Setup.e... |
| PID | 17308 |

**Fig.4.** Main window with process attached

When either a file or process is chosen, the option to *Launch* becomes available. This finalises the hooking stream, and details of the file/process are shown. The *Directory* displays the path of the file, which is only displayed if a file is chosen in the select phase. The *Version* field outlines whether the PE file is 32-bit or 64-bit, the data of which is extracted from the file itself. The *Name* shows how the file is described within the file directory, and the *PID* is either the already existing process identifier or the result of the newly created process if a file was chosen. This finalises the hooking stream, and the main functionality of the program can be used.

## 3.2 Instruction View and Modification

### 3.2.1 Description

The first section in the analysis that began development was a view of the instructions that make up a PE file. Executable files on the Windows OS use the x86 instruction set in .exe files, which can contain malicious instructions. This can make analysis on the instructions a useful feature, so the project implements a view of the entire instruction set that makes up an executable.

To read the instructions, the program processes the raw data of the file and extracts the bytes that contain x86 instructions. As the instructions are stored as bytes, just outputting them would yield very little information. To remedy this, the Capstone disassembly library is utilised, which displays the instructions as the human readable opcodes, along with the address the code is positioned in the file, and any additional operands. Once all of the instructions are processed, they are sent to the Graphical User Interface (GUI) to be displayed to the user. For jump instructions that are selected, the section of opcodes that are skipped are highlighted to show where the jump is directed.

To add further utility to the x86 instructions, individual opcodes can be edited by the user to change the functionality of the targeted executable. This allows the user to change individual bytes of the file in a much easier way than editing the file itself. The updated bytes can then be saved as a copy of the file, then run with the changed instructions.

### 3.2.2 Limitations

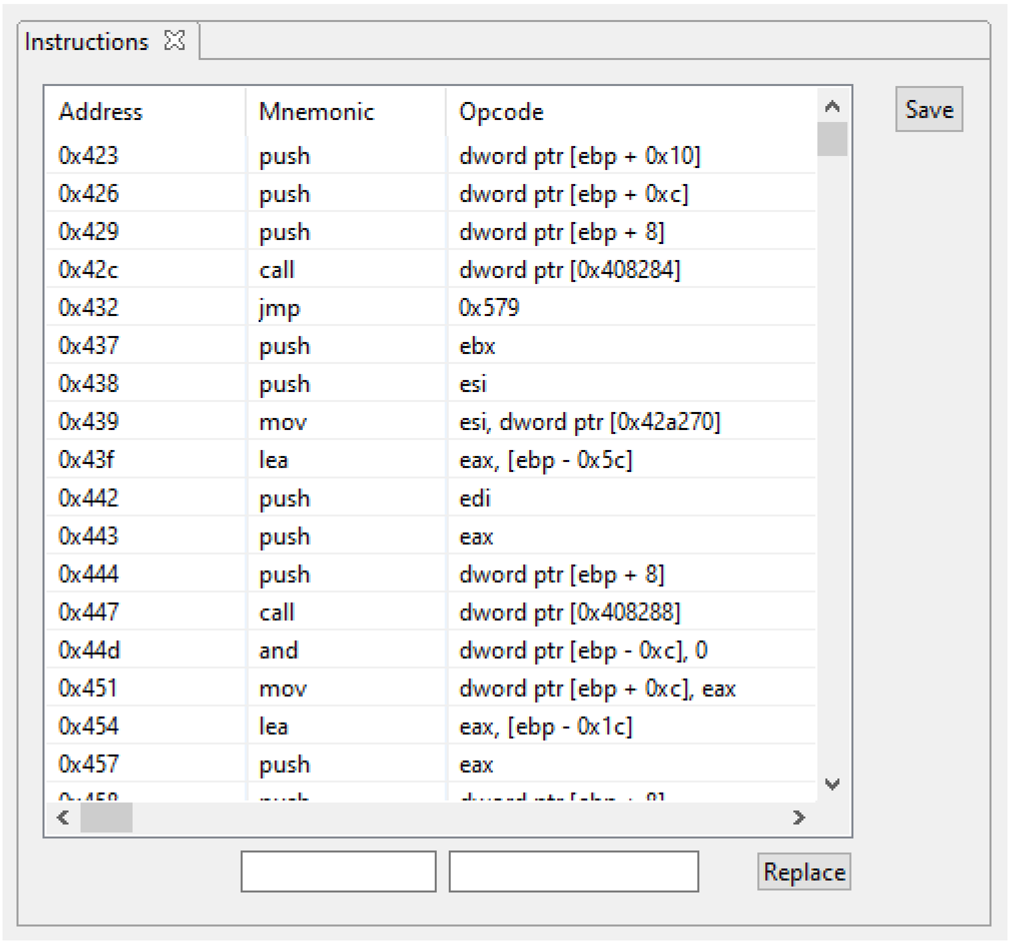
Some limitations are presented with modifying the instructions, as the OS may pick up on the changed file and not allow the file to be run without making changes to its antivirus protections. Another complication that arose is that some executables are actually self-extracting zips, where the executable is a compressed collection of files which is unzipped when run. Attempting to change the instruction set of these files will almost always cause the resulting file to be corrupted and unable to run. It is possible to unzip the .exe file beforehand, however it is not guaranteed to run as intended when executing it in this state.

### 3.2.3 Challenges

Before development into this feature began, there were many things to research to understand how to extract the instructions from an executable file. The documentation for the PE format is very extensive and hard to follow. As the format requires precise pointers and offset incrementations, any misunderstandings with where a section of data is being accessed causes the entire reading of the file to be misaligned and incorrectly labelled by the program. To move on to other features, a library to access the instructions was temporarily implemented, however it would only extract a small section of the instructions. Later on in development, the library was removed and a custom implementation eventually succeeded in correctly extracting the features.

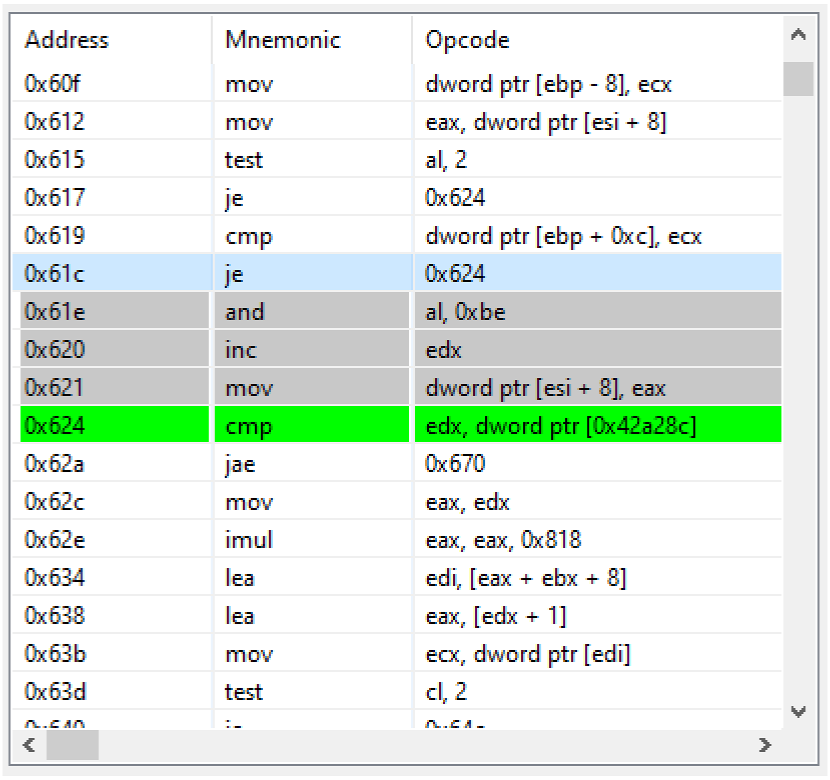
In relation to the instruction replacement, the aim was to have the user type in the human readable opcode into a field in order to replace it. However, The capstone disassembler is a one-way translation that takes bytes and returns the opcode. This meant that turning the opcode back into a byte to be replaced in the file was not possible, and therefore the field for the instructions must only use bytes. This still allows the functionality to be present, but is not as user friendly as was intended.

### 3.2.4 Screenshots



| Address | Mnemonic | Opcode |
|---|---|---|
| 0x423 | push | dword ptr [ebp + 0x10] |
| 0x426 | push | dword ptr [ebp + 0xc] |
| 0x429 | push | dword ptr [ebp + 8] |
| 0x42c | call | dword ptr [0x408284] |
| 0x432 | jmp | 0x579 |
| 0x437 | push | ebx |
| 0x438 | push | esi |
| 0x439 | mov | esi, dword ptr [0x42a270] |
| 0x43f | lea | eax, [ebp - 0x5c] |
| 0x442 | push | edi |
| 0x443 | push | eax |
| 0x444 | push | dword ptr [ebp + 8] |
| 0x447 | call | dword ptr [0x408288] |
| 0x44d | and | dword ptr [ebp - 0xc], 0 |
| 0x451 | mov | dword ptr [ebp + 0xc], eax |
| 0x454 | lea | eax, [ebp - 0x1c] |
| 0x457 | push | eax |

**Fig.5.** x86 Instruction Window



| Address | Mnemonic | Opcode |
|---|---|---|
| 0x60f | mov | dword ptr [ebp - 8], ecx |
| 0x612 | mov | eax, dword ptr [esi + 8] |
| 0x615 | test | al, 2 |
| 0x617 | je | 0x624 |
| 0x619 | cmp | dword ptr [ebp + 0xc], ecx |
| 0x61c | je | 0x624 |
| 0x61e | and | al, 0xbe |
| 0x620 | inc | edx |
| 0x621 | mov | dword ptr [esi + 8], eax |
| 0x624 | cmp | edx, dword ptr [0x42a28c] |
| 0x62a | jae | 0x670 |
| 0x62c | mov | eax, edx |
| 0x62e | imul | eax, eax, 0x818 |
| 0x634 | lea | edi, [eax + ebx + 8] |
| 0x638 | lea | eax, [edx + 1] |
| 0x63b | mov | ecx, dword ptr [edi] |
| 0x63d | test | cl, 2 |

**Fig.6.** Jump instruction highlighting

When the user chooses to view the x86 instructions of a process, this window is shown. It displays the *Address* of the instruction as it appears in the .exe file, the *Mnemonic* that outlines which instruction is being used, along with any additional *Opcodes* that the instruction uses.

The instruction set includes jump instructions that can skip ahead to a different instruction or move backwards to a previous instruction. These can serve as loops or if statements and allow a change of sequence of execution. If any of these jump instructions are selected, the resulting jump is highlighted to where the sequence will continue.

### 3.2.5 Personal Thoughts

In total, the implementation of viewing and modifying the x86 instruction set turned out as I envisioned, showing the user something similar to what the computer sees when it is executing the program. Along with malware analysis, it is useful for those who are interested in low-level machine instructions, being a great tool to tinker with and use to change around some instructions without modifying the original file. If I had more time, I would have liked to look into translating the instructions into a pseudo C++ style format, with if and while statements instead of jump instructions, etc.

## 3.3 Virtual Memory Space Scanner

### 3.3.1 Description

One of the more challenging features to implement was access to a targeted process' virtual memory space. In Windows, each process is given a virtual space to store any memory that a process may need, instead of requiring that all memory be stored in physical memory at all times. This allows the OS to swap in and out memory that needs to be run at any given moment, and also allows two or more processes to use the same address to call a piece of data. Knowing what a process is storing as data can be very useful in analysing it for malicious actions, seeing if it is storing data about the user that it should not be. Therefore, the project incorporates a feature that allows the user to view sections of the memory space that contains readable data, and a search feature to look through the space. Searching provides a basic wildcard implementation to allow for more exact phrasing for what to search.

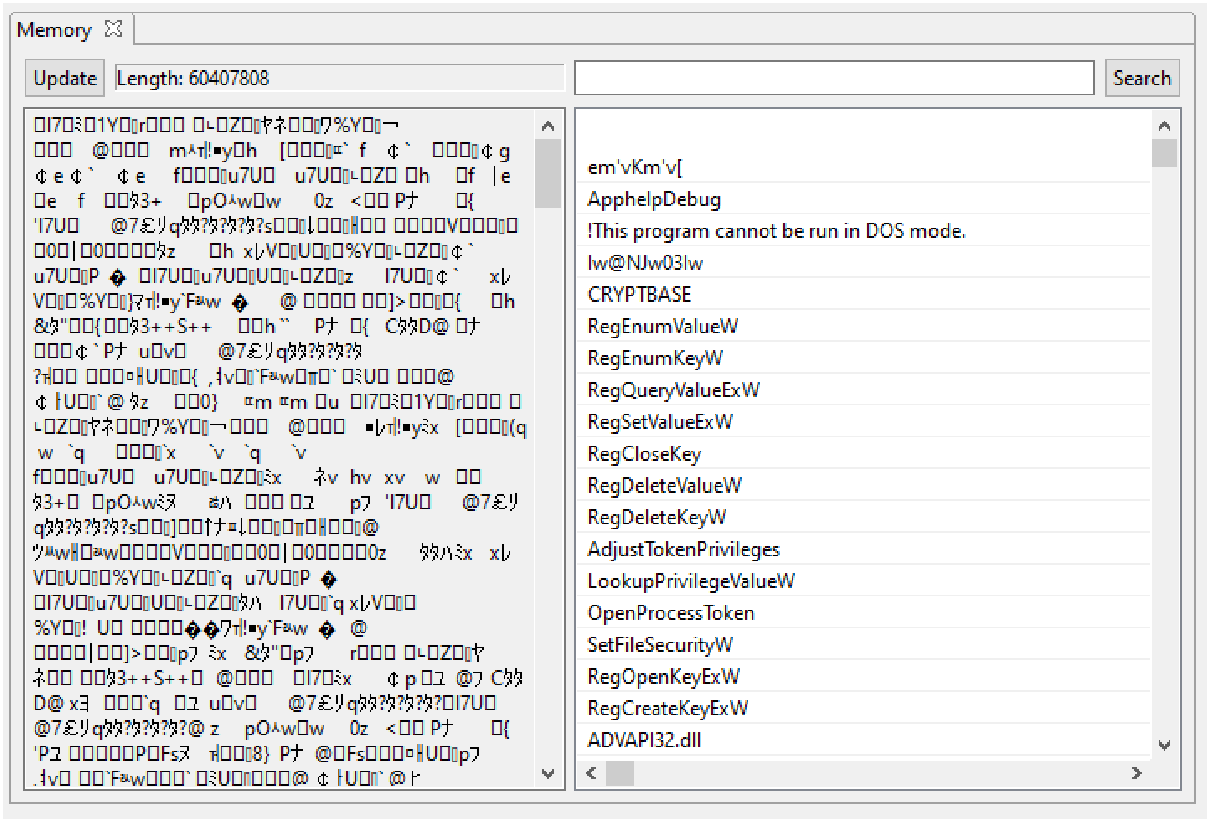
### 3.3.2 Limitations

The virtual memory space of a program can be extremely large in some cases, which brings up many performance issues when loading certain processes. As the virtual memory is much larger than what it holds in physical memory in most cases, attempting to load these types of files resulted in a slowdown of the program, and would sometimes cause a freeze of the entire program for multiple seconds. Multithreading was utilised to attempt to remedy this, splitting the intensive task from the GUI's functionality. This was mostly a success, however it would still cause a less extreme slowdown when moving the large section of data to the GUI. This made it impossible to display the entire memory space in the GUI, instead having to show only a certain section at one time.

### 3.3.3 Challenges

Implementing a view into the virtual memory space was by far the most difficult and time consuming part of the project. The memory is not set up in a way to be read in its entirety, instead it is split into pages that are to be read individually. In order to read these pages, they must all be individually accessed with low level function calls to the *windows.h* API in C++, then sent back to the Java GUI through the Java Native Interface. For each function call, protection values had to be modified, the size of the page accurately being added as an input, then the address values within the page being read to a buffer, along with other smaller considerations. In many cases, the script would simply crash, and in others would output null or default byte values. It took a considerable amount of time to get it in a working state, then additional time to deal with the performance issues relating to the size of the outputted data. This issue does not apply to the search field, as Swing Widget Tool (SWT), the interface used to design the GUI, has a virtual table feature to allow for large amounts of data that can be formatted in a table to be added with little performance impact.

### 3.3.4 Screenshots

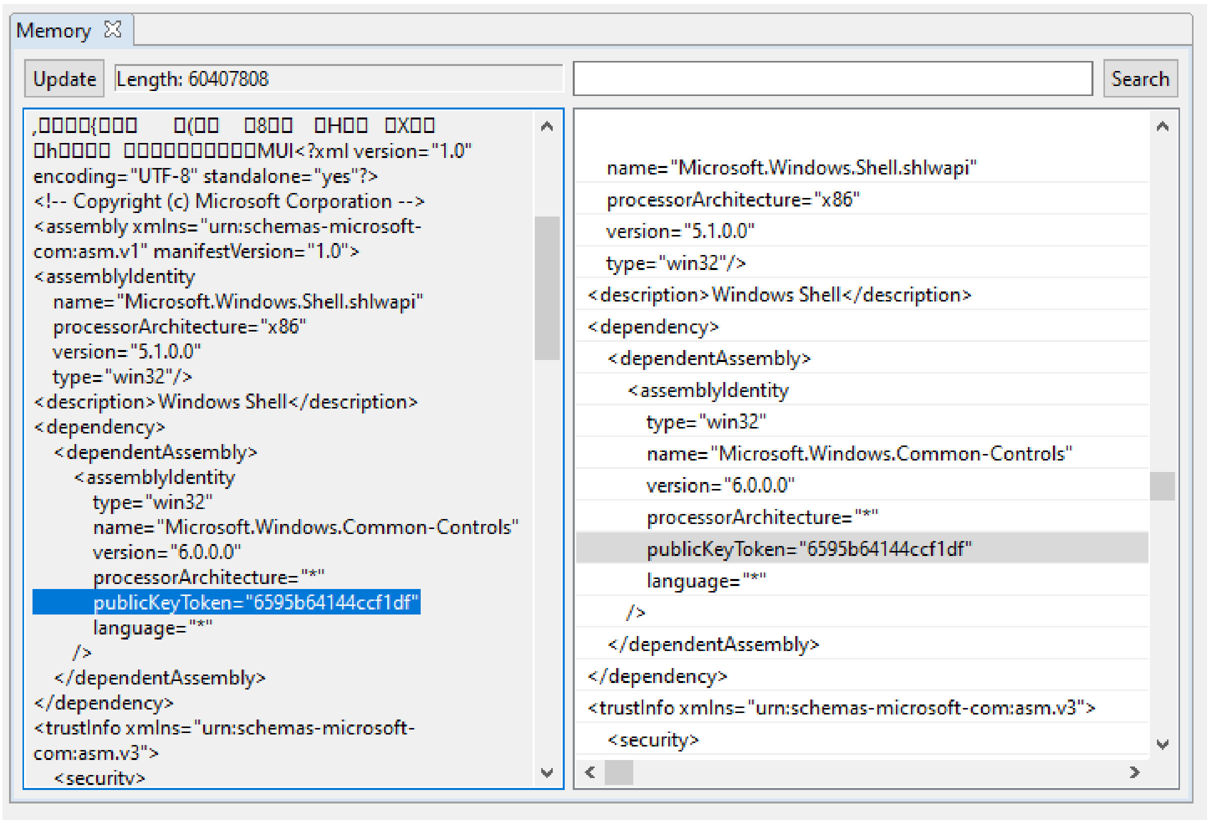


```
[illegible]
```

```
em'vKm'v[
ApphelpDebug
!This program cannot be run in DOS mode.
lw@NJw03lw
CRYPTBASE
RegEnumValueW
RegEnumKeyW
RegQueryValueExW
RegSetValueExW
RegCloseKey
RegDeleteValueW
RegDeleteKeyW
AdjustTokenPrivileges
LookupPrivilegeValueW
OpenProcessToken
SetFileSecurityW
RegOpenKeyExW
RegCreateKeyExW
ADVAPI32.dll
```

**Fig. 7.** Virtual Memory View

This is the view of the virtual memory screen. The field on the left contains the unfiltered ASCII representation of the bytes contained within the memory. In the majority of cases, this will be completely useless sections of uninitialised data that the process is not using. This is the expected behaviour of the program, as every page that is available is read, including the allocated space that is not in use. The functionality comes from the readable text that is extracted from the memory space, shown on the right table. This is the result of processing the virtual memory to attempt to find values, strings, and any other section of data ready to be used by the targeted program. The amount of information that can be gathered with this feature is very extensive, as the table usually contains thousands of values. The *length* value shows the entire size of the virtual memory space, in bytes, that is contained within the raw memory space.

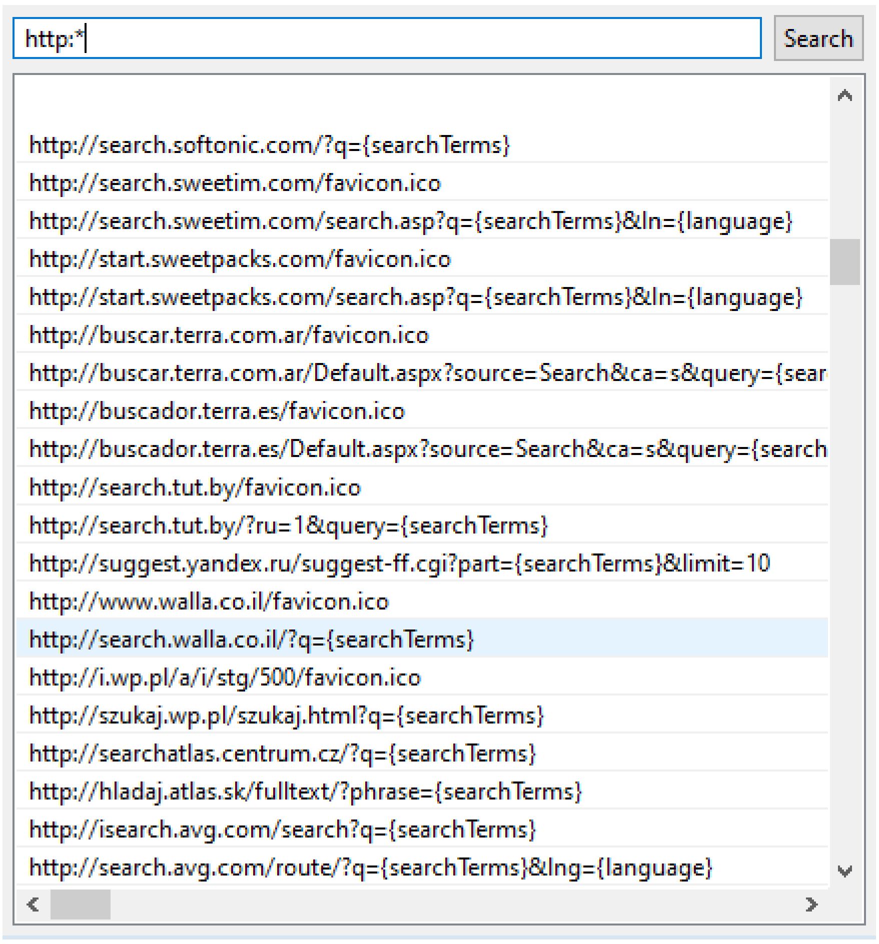


```
,□□□□{□□□    □(□□   □8□□   □H□□   □X□□
□h□□□□  □□□□□□□□□□MUI<?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<assembly xmlns="urn:schemas-microsoft-
com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
   name="Microsoft.Windows.Shell.shlwapi"
   processorArchitecture="x86"
   version="5.1.0.0"
   type="win32"/>
<description>Windows Shell</description>
<dependency>
   <dependentAssembly>
      <assemblyIdentity
         type="win32"
         name="Microsoft.Windows.Common-Controls"
         version="6.0.0.0"
         processorArchitecture="*"
         publicKeyToken="6595b64144ccf1df"
         language="*"
      />
   </dependentAssembly>
</dependency>
<trustInfo xmlns="urn:schemas-microsoft-
com:asm.v3">
   <security>
```

```
   name="Microsoft.Windows.Shell.shlwapi"
   processorArchitecture="x86"
   version="5.1.0.0"
   type="win32"/>
<description>Windows Shell</description>
<dependency>
   <dependentAssembly>
      <assemblyIdentity
         type="win32"
         name="Microsoft.Windows.Common-Controls"
         version="6.0.0.0"
         processorArchitecture="*"
         publicKeyToken="6595b64144ccf1df"
         language="*"
      />
   </dependentAssembly>
</dependency>
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
   <security>
```

**Fig. 8.** Selecting a section of data

The text table allows for selecting individual portions of the memory, which displays the surrounding sections of raw data. This usually results in the left section being much more readable, as it is then accessing a used page of memory. The specific section that was selected by the user is highlighted in this view.



```
http://search.softonic.com/?q={searchTerms}
http://search.sweetim.com/favicon.ico
http://search.sweetim.com/search.asp?q={searchTerms}&ln={language}
http://start.sweetpacks.com/favicon.ico
http://start.sweetpacks.com/search.asp?q={searchTerms}&ln={language}
http://buscar.terra.com.ar/favicon.ico
http://buscar.terra.com.ar/Default.aspx?source=Search&ca=s&query={sear
http://buscador.terra.es/favicon.ico
http://buscador.terra.es/Default.aspx?source=Search&ca=s&query={search
http://search.tut.by/favicon.ico
http://search.tut.by/?ru=1&query={searchTerms}
http://suggest.yandex.ru/suggest-ff.cgi?part={searchTerms}&limit=10
http://www.walla.co.il/favicon.ico
http://search.walla.co.il/?q={searchTerms}
http://i.wp.pl/a/i/stg/500/favicon.ico
http://szukaj.wp.pl/szukaj.html?q={searchTerms}
http://searchatlas.centrum.cz/?q={searchTerms}
http://hladaj.atlas.sk/fulltext/?phrase={searchTerms}
http://isearch.avg.com/search?q={searchTerms}
http://search.avg.com/route/?q={searchTerms}&lng={language}
```

**Fig.9.** Search feature with wildcard

The large amount of text data can be filtered through with the search feature. The user can search by either using exact phrasing, where if the text entered is a subsection of a particular table value, it is shown in the search. In the above example, a wildcard is used, where the text is only shown if it begins with *http*. Common patterns that yield the most information can be found with this feature, should users learn what to look for.

### 3.3.5 Personal Thoughts

The section of code that I produced from working through this problem is perhaps the most proud of a code piece I have ever been. This challenge seemed largely unsolved from what I could find, therefore very little external help was utilised when dealing with it. Ideally, the entire memory space would be accessible to the user in the GUI, however the search feature still makes it a highly usable feature in the program.

## 4 General Issues

There were several areas that provided difficulty during the course of development that were not specific to any one section. These issues remained relevant along with others described above

### 4.1 GUI

The interface chosen to develop the Java GUI was Swing Widget Tool (SWT) for the Eclipse development environment. Overall, it helped extensively with creating the GUI, allowing for a real-time view of how individual components were placed in the window and dragging the positions around within this view proved highly useful. However, some issues while using SWT made it so that it was not as efficient as I would have liked.

The greatest cause of inconvenience was how SWT managed the different layouts that can be applied to a program. These layouts are essential for allowing the window to be resized, moving components accordingly as the window changes in size. The two layouts that were used in the project were the *GridLayout* and *FormLayout*. Attempting to reposition components of the GUi while they were in these layouts would, in the majority of cases, cause every other component to change in size and position. This increased the time required to modify the GUI dramatically as components would have to be repositioned every time a change was made. In rare cases with the form layout, repositioning a widget would cause Eclipse to completely freeze and increase CPU usage to the point where my entire device would slow down until it was forcibly stopped.

Even when the components were laid out correctly, it proved very difficult to have them resize as expected, with usually only one component actually changing size with the window. There was little I could do to change this behaviour, other than arranging the GUI in such a way that the most important aspect of a window was the one to change size.

These complications increased the time required to develop the GUI in areas that it should not have, and without mentioning the actual time to design the GUI. While it was still significantly faster to use this feature than manually writing all of the code for the GUI, these issues served as minor annoyances throughout the development cycle of the project.

## 5 Project Overview

### 5.1 Achieved

The project contains several features that successfully go towards the analysis of programs for malicious actions. It provides methods to give the user a deep dive into the workings of a targeted program by showing them values stored within its memory, instructions that make up its executable compilation, and details specific to the application. These are highly technical aspects that are very difficult to manually access without the help of the analysis tool's capabilities, and they are shown in a way that one with basic technical skills can comprehend and tinker with.

Overall, the main goals set out at the beginning of the project were successfully achieved, and a solid foundation to build on what has been done is laid out, should it be revisited in the future. This is due to how the functionality was developed, consisting of several sets of tools that make up the larger scheme of the project. Splitting it in this way allows for existing tools to be reviewed, modified, and expanded upon while new functionality can be easily applied. Maintenance and supportability were constantly in focus during the development cycle, which conforms to the agile methodology of development.

### 5.2 Not Achieved

The biggest issue that came with developing low-level functionality was the time required to be invested into each section. There was a constant cycle of research, trial and error, and failed implementations before a stable solution could be found. Along with the additional areas to allot time, such as the GUI, performance tuning, documentation, and external factors, the feature set that is supplied in the final release is not as extensive as I would have liked. Many areas that were considered for development had to be scrapped because of this, such as automatic logging of progress to resume analysis after the program had shut down. Given more time, the existing tools could be heavily expanded upon and made even more intuitive, while supporting features would be added to streamline the operability between the tools for a more seamless user experience.

Before the development phase of the project began, the initial plan was to have the program detect the type of malware that is prevalent in a targeted process, whether that be scareware, ransomware, bloatware, etc. This soon turned out to be entirely infeasible within the development timeframe, most likely requiring an extensive use of machine learning and signature detection. Access to relevant malware datasets that would aid in the machine learning process would also be a challenge, with the most extensive ones being privatised by malware protection companies to keep their competition from accessing them. This is specifically mentioned as this functionality was a considerable portion of the project proposal, with the features developed intended to achieve the detection goal. This does not subtract from the ultimate aim of this project, which was to create a tool that dynamically analyses a piece of software for malicious actions. The analysis is still possible, just not as efficient as what could entirely be achieved.

### 5.3 What I Would Change

If I was to restart the project from scratch, I would set the scope of what could be achieved to a more reasonable level. The expectations of what I could produce in the given time applied unneeded pressure from an early stage, and time was spent in these areas that ultimately led to very little progress towards the final product. Setting goals that were more in line with what I could reasonably produce would allow me to expand more on these areas instead of attempting to do too much of everything at once. This comes down to proper planning of a large scale project, in which I feel that I have gained invaluable experience during the course of development.

## 6 Learning Outcomes

When I was considering which project to undertake near the start of the year, I had a conversation with Richard Butler, a lecturer in South East Technological University. I mentioned that I was interested in a project that involved working with technical areas where solutions involved tinkering with low-level data to produce a result. He had recommended working on a malware analysis tool, as this suited what I described perfectly. As much of this project came from researching, among other areas, how an operating system handles processes, memory swapping, and file execution, the depth of understanding with low-level OS management that I have gained is significant. Some of these areas were touched upon during the four years of college that I attended, however this was mostly on a theory level. Being able to properly interact with Windows function calls and handle files down to the byte level gave me a more intricate understanding than a book ever could.

Along with the content of the program, working on a project for such an extended period of time provided me with experience that is more generally suited towards the working environment that is very soon to follow the conclusion of my project. Using an agile methodology to build a project from scratch showed what the type of work that will be expected of me will be when I leave college, knowing how to submit constant iterations of documents and prototypes of software from what I applied during the past college year. These skills will stick with me well into the foreseeable future, and by then I aim to have refined them many times over.

## 7 Conclusion

Overall, I am satisfied with the amount achieved during the project. In general, it was an enjoyable experience working through the problems that came with development, and a lot was learnt by doing so. While the problems encountered meant that the final product was not as fully feature complete as I would have hoped, the work that went into the areas that were developed means that I will look back on this project in a positive light.

## 8 Acknowledgements

I would like to extend my thanks to my supervisor, Dr. Joseph Kehoe, for guiding me through this project during the past few months. The weekly meetings with him provided me with valuable information and directions to take the development of the program. Without his guidance, there would have been many occasions where I would have been completely unsure on what to work on, or areas to research. The materials and guidance provided helped in many ways towards the final submission of the project. Thank you, Joseph!

## 9 Plagiarism Declaration

Declaration

I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
I have provided a complete bibliography of all works and sources used in the preparation of this submission.
I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: Shane Doherty

Student Number: C00249279

[illegible]

Signature: