

# Static File Analysis Tool

## Research Manual

Brian Tobin

C00216353

## Abstract

Static file analysis is the first step taken when trying to reverse engineer a file. This is where you attempt to find out what a file does, without ever executing the file. Static file analysis tools are more complex than needed for students studying reverse engineering and malware analysis. This report reviews some common static file analysis tools and the techniques they use. It also covers how students want a tool that is easy to use, and the technologies I will use to develop that tool.

## Table of Contents

Abstract.....	i
Table of Contents.....	ii
Introduction.....	iv
Static File Analysis .....	1
What is static file analysis? .....	1
Techniques Used .....	1
Hashing.....	1
VirusTotal.....	2
Packed Files.....	3
Strings.....	4
Portable Executable Dynamic Linking.....	5
Disassembly.....	6
Conclusion.....	8
Existing Products .....	9
PEiD .....	9
Hiew .....	11
010 Editor.....	12
Detect It Easy .....	13
Dependency Walker .....	14
What Reverse Engineering Students Want In An Application? .....	15
Conclusion.....	15
Front End Technologies.....	16
What are front-end technologies? .....	16
Qt.....	16
What is Qt? .....	16
Why use Qt? .....	17
Pros and Cons of using Qt.....	17
Swing.....	18
What is Swing?.....	18
Why use Swing? .....	18
Pros and Cons of using Swing.....	18
Xamarin.....	19
What is Xamarin?.....	19
Why use Xamarin? .....	19

Pros and Cons of using Xamarin .....	19
Conclusion.....	20
Back End Technologies .....	21
What are back-end technologies?.....	21
C++.....	21
What is C++? .....	21
Why use C++? .....	21
Pros and Cons of using C++ .....	21
Java.....	22
What is Java? .....	22
Why use Java? .....	22
Pros and Cons of using Java .....	22
C# .....	23
What is C#? .....	23
Why use C#?.....	23
Pros and Cons of using C# .....	23
Conclusion.....	24
Summary and Conclusion .....	25
Summary .....	25
Conclusion.....	25
Bibliography .....	26

## Introduction

There are many different tools used during static file analysis. These tools can be very powerful and complex. Students studying reverse engineering and malware analysis must use these tools to help them learn. This is a problem because the tools are designed with functionality in mind, not necessarily usability. It can be hard to learn the basics when all the tools are so advanced.

This report begins by looking at what features a static file analysis tool would require, then looks at some existing products and what students want in a tool. The technologies that could be used to create this tool are then reviewed. Finally, a conclusion is made about how this tool will be developed.

# Static File Analysis

## What is static file analysis?

Static file analysis is usually the first step taken during reverse engineering or malware analysis where a file analyst is trying to find out what a particular file does and how it does it. It is static because the file is never actually executed unlike dynamic file analysis where the file is executed and monitored in a safe environment. Static file analysis is straightforward in that you can follow a checklist to make sure you do everything you need to. It may not be enough for complicated files but it is a good starting point to get an idea of what a file does. (Sikorski & Honig, 2012; Ninja, 2015)

## Techniques Used

### Hashing

The first thing that should be done is to generate a hash of the file (Yusirwan, et al., 2015). A hash is a value calculated using the exact contents of the file. It is “unique” and if any part of the file is changed, the hash will be a completely different value.

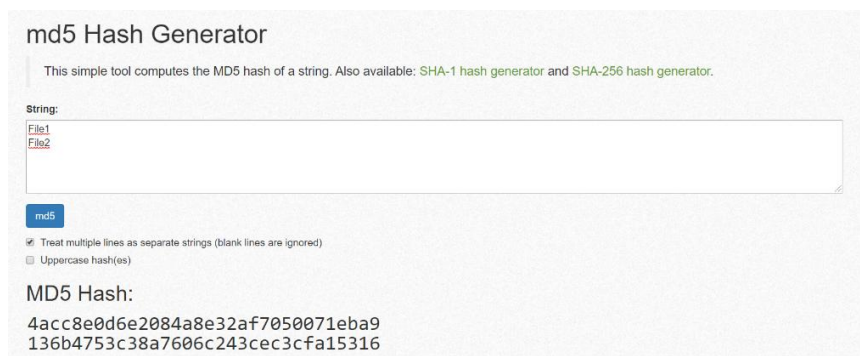


Figure 1: Screenshot of MD5 hash generator on miraclesalad.com (Miracle Salad, 2019)

In Figure 1, we can see the results of hashing the words “File1” and “File2” using the MD5 hashing algorithm. “File1”, gives a value of “4acc8e0d6e2084a8e32af7050071eba9”, and “File2” gives a value of “136b4753c38a7606c243cec3cfa15316”. The only difference between the words was a ‘1’ changed to a ‘2’, but the hash value is completely different. This can be used to determine if the file has been changed in any way. The file should also be backed up in case it is altered which you can verify with the hash and then restore the file.

## VirusTotal

The first thing to do for malware analysis is to upload the file to a site called virustotal.com. This website will scan the file using over 70 antiviruses and other tools giving the best chance of detecting common malware (VirusTotal, n.d.).

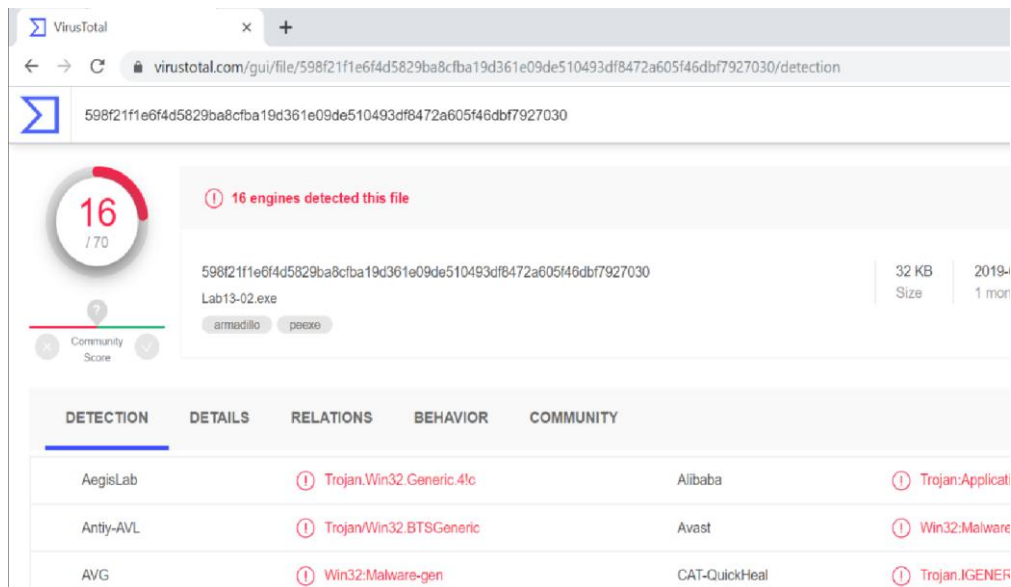


Figure 2: Screenshot of a VirusTotal.com file scan.

In Figure 2, we can see part of the detailed report that virustotal.com gives. The file name is “Lab13-02.exe” and a hash is given above it. It shows that 16 out of the 70 detection tools detected it as a problem and it appears to be a Trojan. While VirusTotal is great for malware, it is not as useful for general file analysis.

Sections					
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5
.text	4096	192512	99328	8	9c0c236a88a5e4a88674c3e4dbe094b3
.rdata	196608	8192	2048	7.76	c37be3c6a936f1e63ca5373e5b7465eb
.data	204800	102400	17920	7.98	293a60ea1b5c03bd9fc941490eac2153
.aspack	307200	8192	5120	6.44	c19bf9e631572104703cedec8c4f9ec75
.data	315392	4096	0	0	d41d8cd98f00b204e9800998ecf8427e

Imports	
+ advapi32.dll	
+ kernel32.dll	
+ user32.dll	

Figure 3: More details on VirusTotal.com.

Figure 3 is another screenshot from VirusTotal showing some of the details about the file section sizes and some of the DLL’s the file imports. This is some of the only useful information we can get about non-malware files from VirusTotal.

## Packed Files

Executable files can be packed or compressed to make the file smaller or to obfuscate the code to make it harder to reverse engineer or analyse (Sikorski & Honig, 2012). The code is compressed and then packaged with the decompression code in one executable file. Most common packer formats can be detected by a cross-platform tool called Detect It Easy. UPX is a common packer that can be easily unpacked; if you unpack the file, it may get rid of some of the obfuscation, allowing you to find out even more information about the files functionality (Ninja, 2015).

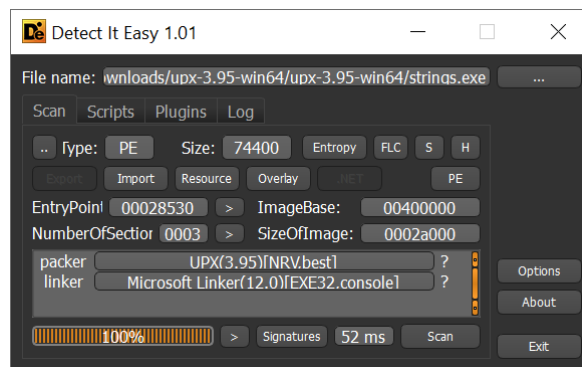


Figure 5: Screenshot I took, of Detect It Easy showing a UPX packed file (NTInfo, 2019).

In Figure 5, I have selected a file called “strings.exe” that I packed myself, and in the packer box, we can see that the file is packed using UPX version 3.95.

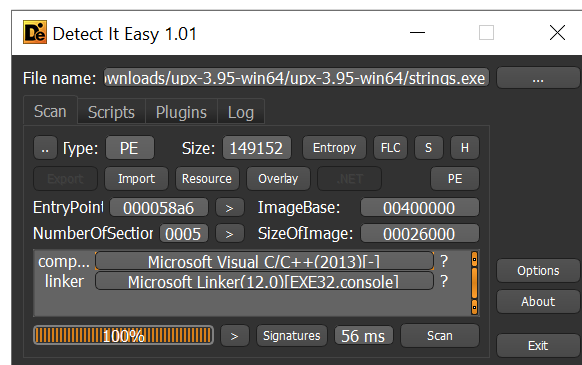


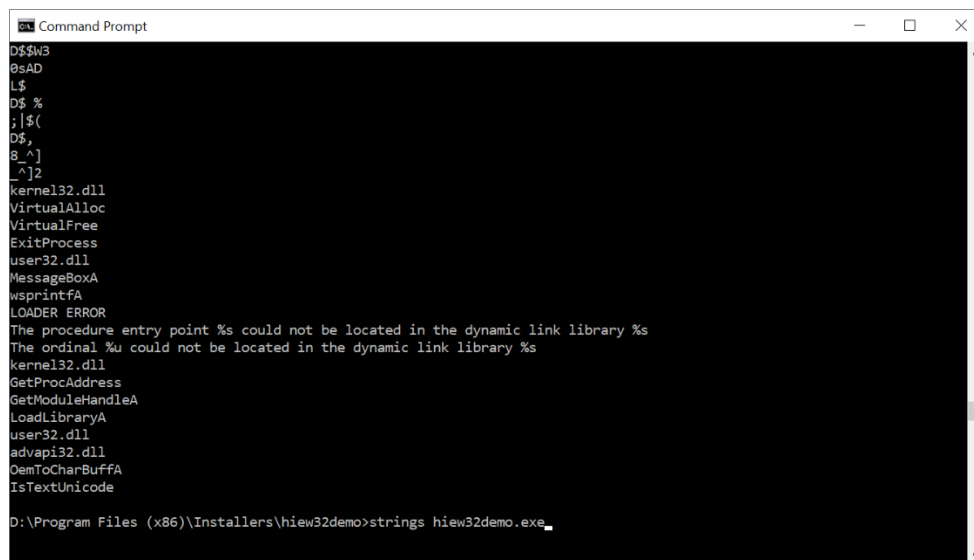
Figure 6: Another screenshot of Detect It Easy showing an unpacked file.

In Figure 6, I unpacked the same file and now we can see that Microsoft Visual C/C++ 2013 compiled it and it is no longer packed. I ran the strings command on both the packed and unpacked versions and got 2631 and 2449 strings respectively. The packed file gave more strings but it was mostly false readings, while the unpacked file gave fewer strings but had a lot more useful information.



## Strings

The next step would be to search the file for strings that could give you hints about the functionality of the file (Ninja, 2015). A string is a group of alphanumeric characters that might be human-readable. Strings are stored with a null value at the end so they can be identified when needed. They can be searched for by using the strings program, which will go through a hex dump of the file and try to find null-terminated strings of at least 3 or 4 characters in length depending on the implementation of the strings command (Sikorski & Honig, 2012).



```
Command Prompt
D$$WB
0sAD
L$
D$ %
;|$(
D$,
8_^]
L^]2
kernel32.dll
VirtualAlloc
VirtualFree
ExitProcess
user32.dll
MessageBoxA
wsprintfA
LOADER ERROR
The procedure entry point %s could not be located in the dynamic link library %s
The ordinal %u could not be located in the dynamic link library %s
kernel32.dll
GetProcAddress
GetModuleHandleA
LoadLibraryA
user32.dll
advapi32.dll
OemToCharBuffA
IsTextUnicode
D:\Program Files (x86)\Installers\hiew32demo>strings hiew32demo.exe_
```

Figure 4: Screenshot I took, running strings on an executable file.

We can see in Figure 4 that strings found the names of the linked libraries: kernel32.dll, user32.dll and advapi32.dll. It also found the functions VirtualAlloc, VirtualFree and ExitProcess among others. These can give hints about the functionality of the program. Strings can also give many false positives like the first few strings we see in Figure 4, these should be filtered out manually before analysing the good strings. When analysing the good strings we are looking for things like DLL names, functions or IP addresses. (Ninja, 2015)

## Portable Executable Dynamic Linking

Windows uses DLL's, which are part of a shared library of common code that can be used by many applications at the same time. It is possible to get the names of some DLL's by running the strings command on a file, but Dependency Walker is a tool that will build a hierarchical tree diagram of all the DLL's and functions used in a Portable Executable file. This is used to get a better idea of what the file will do when executed. (Sikorski & Honig, 2012)

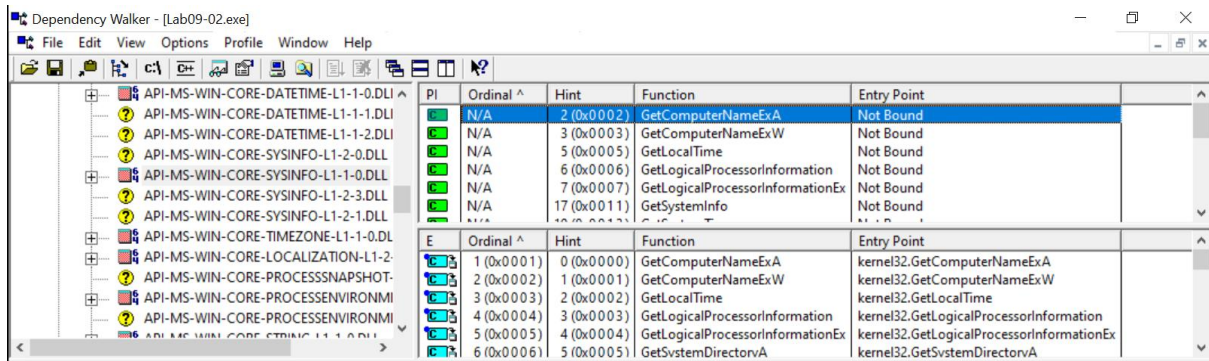
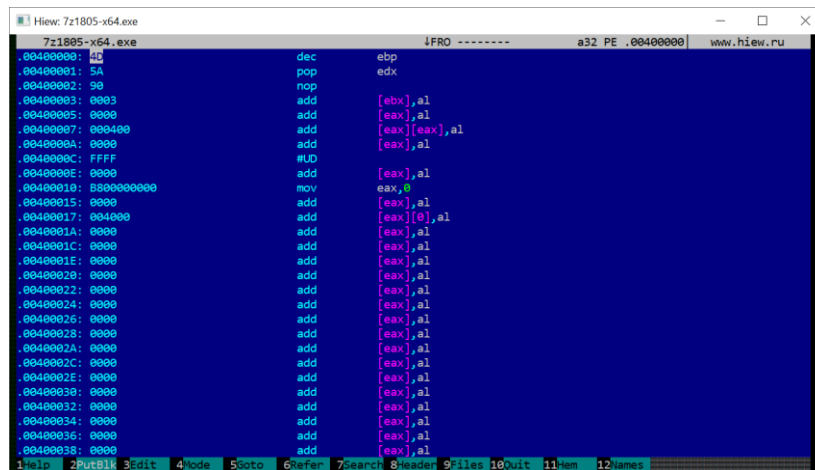


Figure 7: Screenshot of Dependency Walker (Dependency Walker, 2015)

In Figure 7, we can see a screenshot of Dependency Walker that has built a picture of the DLL's used by the malicious file "Lab09-02.exe". In one of the DLL's, the function "GetComputerNameExA" is highlighted, we can tell that it along with the other functions below are gathering information about the computer. The file being analysed may be sending this information back to the creator of this malicious file, but further searching would be required to determine exactly what it is doing.

## Disassembly

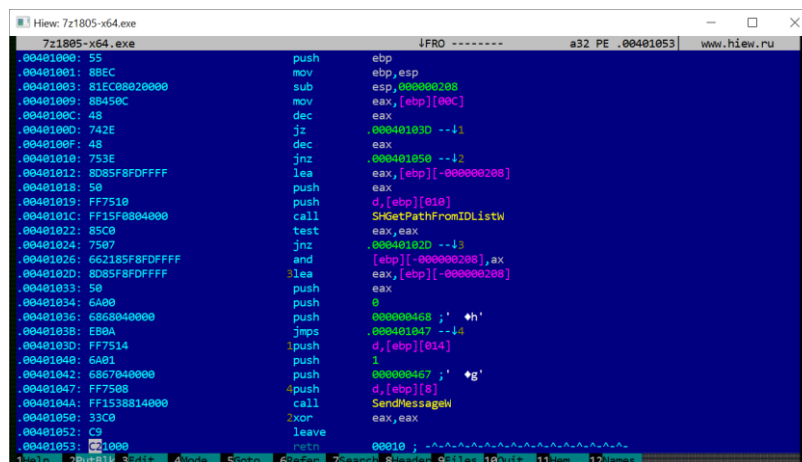
When the previous techniques have been used to try to get an understanding of what the program does, we can then disassemble the file. A disassembler will try to recreate the assembly code of the file, which is not very human-readable. This is why we try to understand what the program does before looking at this code. (Yusirwan, et al., 2015)



```
7z1805-x64.exe
00400000: 4D      dec     ebp
00400001: 5A      pop     edx
00400002: 90      nop
00400003: 0003   add     [ebx],al
00400005: 0000   add     [eax],al
00400007: 000400 add     [eax],[eax],al
0040000A: 0000   add     [eax],al
0040000C: FFFF   #UD
0040000E: 0000   add     [eax],al
00400010: 88000000 mov    eax,0
00400015: 0000   add     [eax],al
00400017: 004000 add     [eax][0],al
0040001A: 0000   add     [eax],al
0040001C: 0000   add     [eax],al
0040001E: 0000   add     [eax],al
00400020: 0000   add     [eax],al
00400022: 0000   add     [eax],al
00400024: 0000   add     [eax],al
00400026: 0000   add     [eax],al
00400028: 0000   add     [eax],al
0040002A: 0000   add     [eax],al
0040002C: 0000   add     [eax],al
0040002E: 0000   add     [eax],al
00400030: 0000   add     [eax],al
00400032: 0000   add     [eax],al
00400034: 0000   add     [eax],al
00400036: 0000   add     [eax],al
00400038: 0000   add     [eax],al
```

Figure 8: Screenshot of a disassembled executable file I took using Hiew (Suslikov, 2019)

Figure 8 shows how some disassemblers are not completely accurate and try to interpret the entire file as code. On the left side we have the memory address and value at that address e.g. memory address: ‘.00400000’, with the hex value: ‘4D’. This value is the start of the file and is the number that says this file is a PE file, but the disassembler is interpreting it as the instruction ‘dec ebp’. The disassembler should only disassemble bytes in the code section and should default to the code start location to be easier to use.



```
7z1805-x64.exe
00401000: 55      push   ebp
00401001: 8BEC   mov    ebp,esp
00401003: 81EC002000 sub   esp,00000200
00401009: 8B450C mov    eax,[ebp+00C]
0040100C: 48      dec    eax
0040100D: 742E   jz     .0040103D --41
0040100F: 48      dec    eax
00401010: 753E   jnz   .00401050 --10
00401012: 8D5F8FDFFF lea   eax,[ebp-00000208]
00401018: 50      push  eax
00401019: FF7510 push  d,[ebp+010]
0040101C: FF15F0804000 call  SHGetPathFromIDListW
00401022: 85C0   test  eax,eax
00401024: 7507   jnz   .0040102D --43
00401026: 662185F8FDFFF and   [ebp-00000208],ax
0040102D: 8D5F8FDFFF lea   eax,[ebp-00000208]
00401033: 50      push  eax
00401034: 6A00   push  0
00401036: 6680400000 push  000000468 ;' *h'
00401038: EB9A   jmps  .00401047 --14
0040103D: FF7514 push  d,[ebp+014]
00401040: 6A01   push  1
00401042: 6687040000 push  000000467 ;' *g'
00401047: FF7508 push  d,[ebp+08]
0040104A: FF1538140000 call  SendMessageW
00401050: 33C0   xor   eax,eax
00401052: C9      leave
00401053: 0000   ret    0
```

Figure 9: Another screenshot of Hiew, showing proper code

In figure 9, we have another screenshot of the same file just further down, which appears to have some valid code. This disassembled code appears to be from the code section of the file and contains some useful information such as the names of some imported DLL functions and some strings. This can now be analysed using prior knowledge of what we think the program is doing.

## Conclusion

In conclusion, static file analysis is the first basic step taken when trying to reverse engineer or analyse a computer file. There is only a handful of techniques that are used during static file analysis. The first step is to hash and backup the file, and then if it is believed to be malware, it should be uploaded to VirusTotal.com. The next step is to try to detect if the file is packed, and attempt to unpack it. Then we run the strings command on the file to gather a list of strings, and then look for interesting strings like function names or IP addresses. Then for PE files, we can use Dependency Walker to build a diagram of the DLL's and functions used. Finally, we can disassemble the file and using the information we gathered from the previous steps, to try to work out what the file is doing.

## Existing Products

I researched some existing products that are used for static file analysis to see what features they offer and gave some pros and cons of each product. The products needed to cover all the techniques used in static file analysis. I thought that this would help me to define what my project will require and where I can improve on previously existing products.

### PEiD

PEiD is a tool for Windows that can detect common packers, cryptors and compilers for Portable Executable files. It also has a disassembler and gives smaller details like the entry point of the program. (Aldeid, 2013)

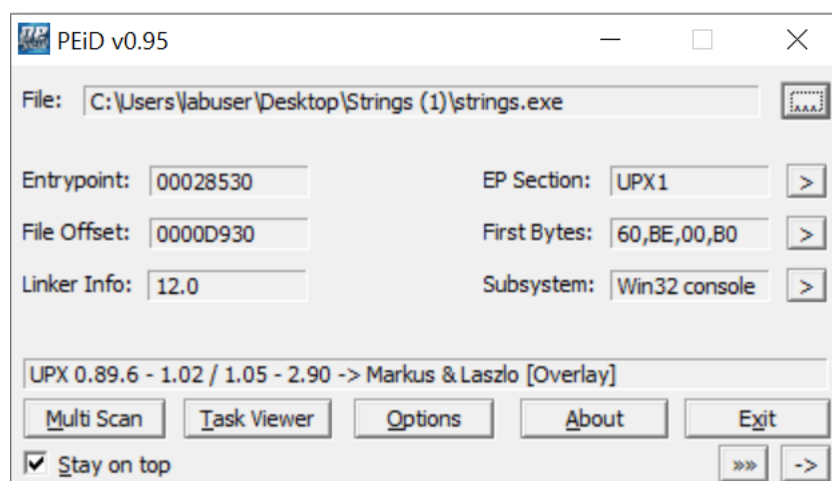


Figure 10: Screenshot of PEiD main window (SOFTPEDIA, 2018)

In Figure 10, we can see a screenshot of PEiD. It shows that the strings.exe file is packed using UPX. I think that the layout looks good and is easy to use, although it is not very clear that the '>' symbol beside the "First Bytes" box is what opens up the disassembler.

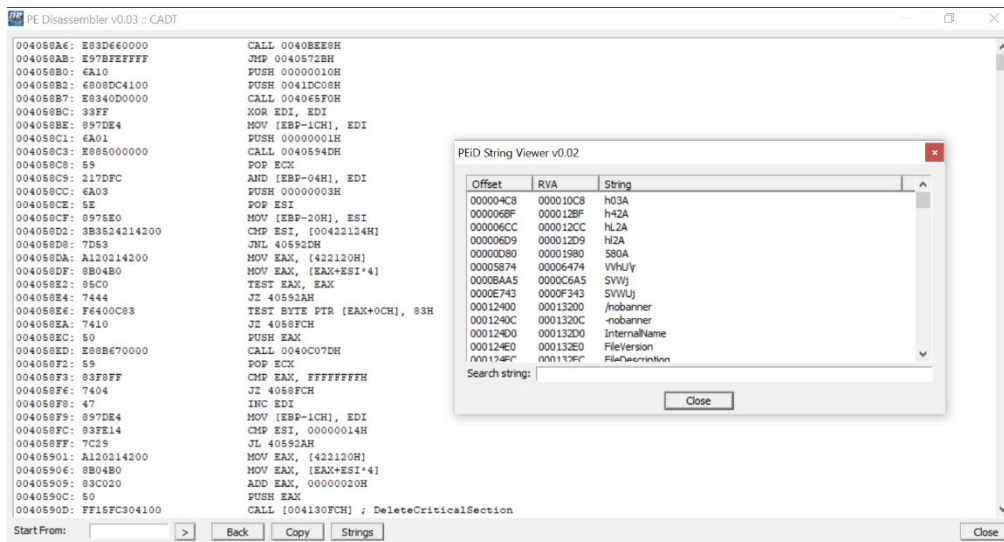


Figure 11: Screenshot of PEiD's disassembler and strings windows

In Figure 11, we can see the disassembler window. It is all black text on a white background, and I think this could be improved by adding some colour to the code, which would make it easier to read. The strings window is also accessed here and it gives the location of each string, with the ability to search for a string as well.

Pros:

- Relatively simple program that can show disassembly, strings and if file is packed.

Cons:

- Runs only on Windows and only works for PE files.
- No hex editor.
- Black and white colour scheme looks flat.

## Hiew

Hiew is a hex editor for Windows that is often used for static file analysis. It is command line based and doesn't look very good but has lots of features like the ability to view and edit files in text, hex and disassembled code modes. It has a built in x86-64 disassembler and assembler and many other advanced features like an encryption/decryption system and support for many different modules. (Suslikov, 2019)



```
Reset Password  JFRO ----- a64 M64.00000001`000010C4|Hiew 8.51 (c)SEN
.000010C4: 6A00          push     0
.000010C6: 4889E5       mov     rbp, rsp
.000010C9: 4883E4F0    and     rsp, 0F0 ; 'E'
.000010CD: 488B7D08    mov     rdi, [rbp][8]
.000010D1: 488D7510    lea    rsi, [rbp][010]
.000010D5: 89FA       mov     edx, edi
.000010D7: 83C2
.000010DA: C1E20
.000010DD: 4801F
.000010E0: 4889D
.000010E3: EB04
.000010E5: 4883C
.000010E9: 48833
.000010ED: 75F6
.000010EF: 4883C
.000010F3: E3080
.000010F8: 89C7
.000010FA: E36D5
.000010FF: F4          hlt
.00001100: 55          3push  rbp
.00001101: 4889E5       mov     rbp, rsp
.00001104: 5D          pop    rbp
.00001105: E98A560000  jmp    .00000001`00006794 --↓4
1Help  2  3  4GoHdr  5Entry  6SecTbl  7SymTbl  8Header  9  10LdCmd
```

Figure 12 shows a screenshot of the Hiew hex editor interface. The main window displays assembly code with addresses and instructions. A dialog box is open in the center, showing file metadata: Magic: FEEDFACF, CPU: X86\_64, Machine: LIB64|I386\_ALL, Filetype: EXECUTE, Load commands count: 28, Load commands size: 00001028/4136, Flags: 00200085, and Reserved: 00000000. The interface is dark-themed with a blue background for the code area.

Figure 12: Hiew sample taken from hiew.ru (Suslikov, 2019)

In figure 12, we can see a sample of what Hiew looks like. I think that it looks very old and is not very intuitive. It is running in a console, so you can only use a keyboard for input and not the mouse.

Pros:

- Can disassemble files.
- Many features used for static file analysis such as, viewing files in hex and text with the ability to search.

Cons:

- Runs in a console, making it harder to interact with than a GUI application.
- Only works on Windows.
- Can't unpack files, limiting its functionality.



## 010 Editor

010 Editor is a cross-platform text editor that supports multiple formats. It can use binary templates to parse a file into a hierarchical structure to make it easier to read binary files. It is designed to be file editing software and supports hex editing which can be used during static analysis but it does not offer much more features for file analysis. (SweetScape Software Inc., 2019)

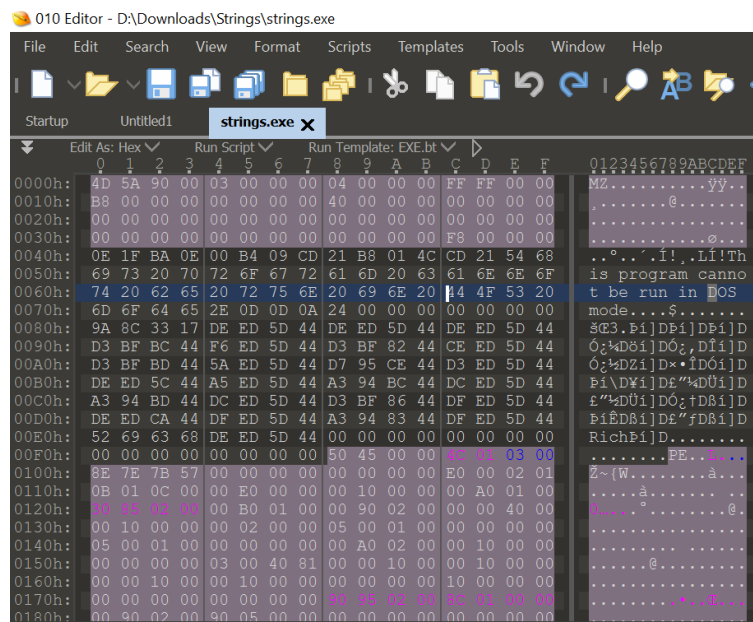


Figure 13: Screenshot of 010 Editor viewing strings.exe in hex mode (SweetScape Software Inc., 2019)

In Figure 13, we can see a screenshot I took of 010 Editor opening a file in hex mode. It also shows the respective text in the column to the right of the hex.

Pros:

- Lots of hex editing features.
- Cross-platform.

Cons:

- File editing tool rather than a static analysis tool.
- No disassembler.
- Can't unpack files.

## Detect It Easy

Detect It Easy is a cross-platform packet identifier that is used to determine file types. It has an open architecture of signatures, allowing the community to add new more complex detection algorithms. This means the software can live on when the old algorithms become irrelevant without the support of the original developer. (NTInfo, 2019)

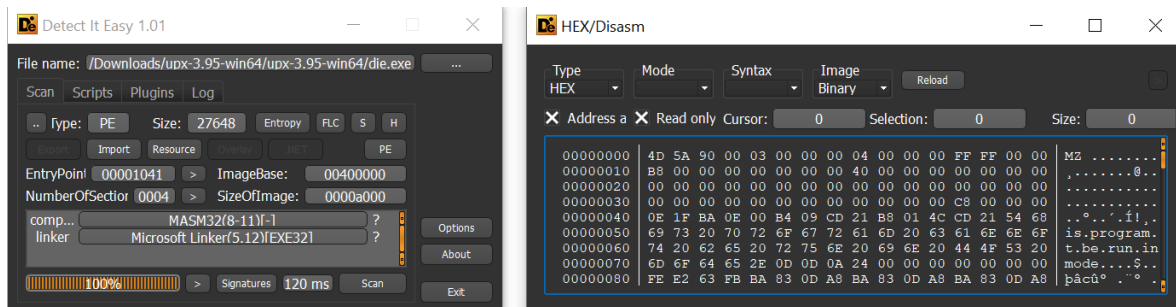


Figure 14: Screenshot of Detect It Easy (NTInfo, 2019)

In Figure 14, we can see a screenshot of Detect It Easy with the main window on the left and hex editor on the right. In my opinion, the main screen is very cluttered and the window is very small and cannot be resized. I think if the window was bigger, the layout could be designed to be more intuitive. I think the hex editor on the right looks good and is easy to use however, when the window is open, you can no longer use the main window until the hex editor is closed. This means you will need to close the hex editor just to use the search function, which is annoying.

Pros:

- Cross-platform.
- GUI based and easier to use than console programs.
- Has hex editor.

Cons:

- Has a lot more features than is required for students learning about static file analysis.
- GUI is cluttered and not very user-friendly.
- Cannot run strings command without external script.
- Only one window can be used at a time, you cannot use the hex editor and search at same time.

## Dependency Walker

Dependency Walker is a tool for Windows that builds a hierarchical tree diagram of dependent modules for any Windows module e.g. executable and DLL files. (Dependency Walker, 2015)

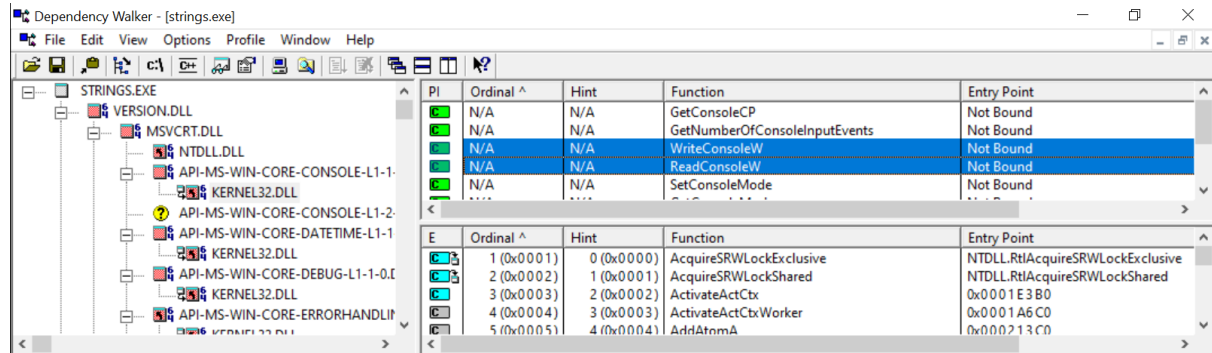


Figure 15: A Screenshot of Dependency Walker looking at strings.exe (Dependency Walker, 2015)

In Figure 15, we can see some of the DLL's being used by the strings.exe file. It calls "KERNEL32.dll" using the functions "WriteConsoleW" and "ReadConsoleW", which would suggest that the program would be reading from and writing to the console.

Pros:

- Can give a more in depth view of the functions that are being used from the DLL's.

Cons:

- Runs on Windows only.
- Its static analysis techniques are limited to finding dependent modules.

## What Reverse Engineering Students Want In An Application?

From personal experience of being a student learning about reverse engineering and malware analysis, I know that we don't need to go into huge detail of all the techniques used during static file analysis; we just need to understand the basic ideas first. The tools needed by students do not have to be very powerful and only require basic features, and they should look good and be easy to use. For example, Detect It Easy can do most of the required things but it can't run strings and the GUI is cluttered and doesn't work very well. Dependency Walker only does one thing and Hiew can't unpack files and uses a console interface making it hard to use.

## Conclusion

Many tools already exist for static file analysis such as Hiew, PEiD, Dependency Walker, and Detect It Easy. These are very powerful tools for what they are used for but are also complicated and not very user friendly. There is no existing software that offers all the basic static file analysis features in one place that is also very user-friendly. You will need multiple powerful programs just to use the most basic features from each. For students learning about static file analysis, it would be easier for them to have all the basic tools required in one easy to use program.

## Front End Technologies

### What are front-end technologies?

Front-end is a term used when referring to what an end user of an application will see. This will usually be a webpage or a Graphical User Interface (GUI). There are many different tools that can be used to create front-end user interfaces. The examples I am going to cover are Qt, Swing and Xamarin. The purpose of having a user-interface is to allow a person to interact with a program. A good interface will be intuitive and easy to use.

### Qt

#### What is Qt?

Qt is a free and open-source toolkit used to develop cross-platform GUI applications. It has built libraries that can integrate natively with different operating systems. Qt Creator is an IDE that implements the Qt toolkit allowing the code written in it to be compiled for various operating systems such as Window, Linux, Android and iOS. (The Qt Company, 2019)

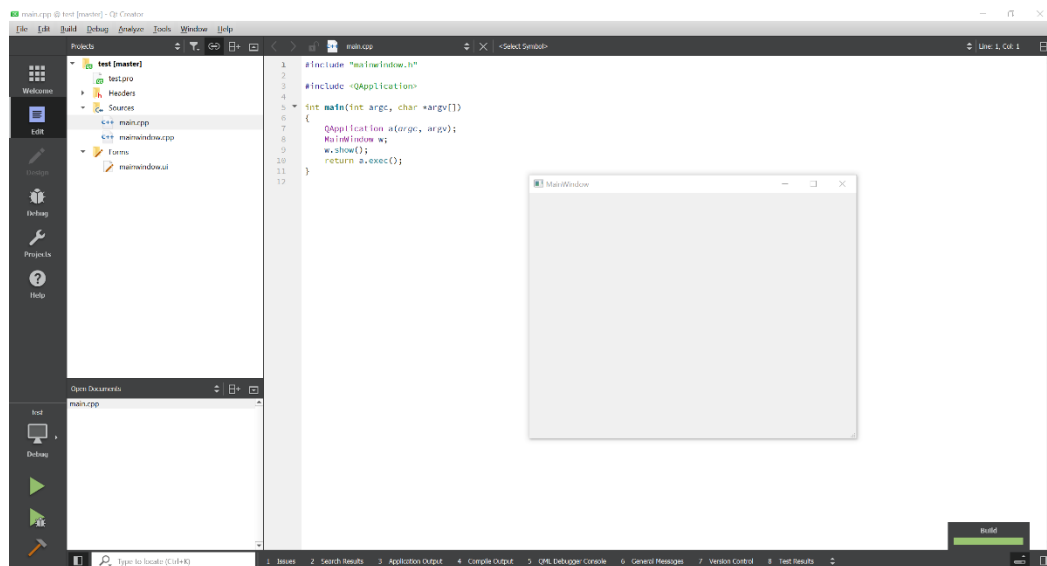


Figure 16: Screenshot of Qt Creator and an empty window (The Qt Company, 2019)

In Figure 16, we can see the Qt Creator IDE with the default code used to open up an application with a blank window.

## Why use Qt?

By using Qt, my application could be cross-platform and run on both Windows and Linux without changing any code. Windows and Linux are the main operating systems that students will be using, so having my application built for both will be ideal. It also uses C++, which is a language that I have used before. Qt Creator also has built in support for Git, which is a version control system used by almost every software development team currently.

## Pros and Cons of using Qt

### Pros:

- Easy way to make program cross-platform, as code can be compiled for lots of different operating systems.
- No GUI problems across platforms, unlike some other tools.

### Cons:

- I have never used it before; I will need to learn it, taking time away from project development.
- My project must conform to their GPL or LGPL licensing.

## Swing

### What is Swing?

Swing is a GUI toolkit used for Java applications. It is built on top of AWT, which is the original GUI toolkit for Java. The GUI must be designed by code only; there is no visual designer like some of the new GUI building toolkits. (JavaTpoint, 2018)

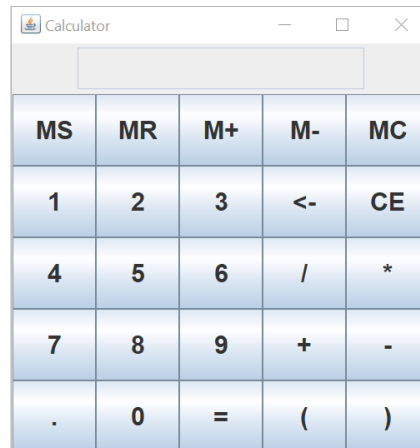


Figure 17: Screenshot of a calculator that I made using Swing

In Figure 17, we can see a screenshot of a calculator application that I made using Swing. It does not look very good and has been succeeded by JavaFX.

### Why use Swing?

I have used Swing before when developing a calculator in Java, so there is much less of a learning curve if I choose to use Swing, which would give more time to spend on developing the project.

### Pros and Cons of using Swing

Pros:

- Easy for me to put a basic GUI together.

Cons:

- Will make cross-platform development much more difficult, introducing unnecessary GUI problems across different operating systems.
- The GUI will not look very good.

## Xamarin

### What is Xamarin?

Xamarin is an open source platform that uses C# for the front and back end code, and is built on top of the .NET framework. It is mostly designed for developing mobile apps for Android and iOS, but also supports macOS and some other less popular operating systems. Xamarin can be used to make applications for Windows, but it is designed for mobile apps and may limit desktop functionality. It also cannot be used to build applications for Linux. (Microsoft, 2019)

### Why use Xamarin?

I would use Xamarin if I wanted to develop an application for Android and iOS devices, but a static analysis tool is much better suited for a desktop environment.

### Pros and Cons of using Xamarin

#### Pros:

- Cross-platform across mobile operating systems, Android, iOS

#### Cons:

- Will not create apps for Linux.
- Static analysis tools are much more useful on desktops.



## Conclusion

In conclusion, I believe the best choice for me is to use Qt and QT Creator for the front-end development of my static file analysis tool. Qt can compile a project across both Windows and Linux with little to no changes to the code, and the GUI should look the same across both platforms. The only cons I can see are that I will need to learn how to use Qt, but this should not be a problem as there is lots of support available online. Qt Creator also supports Git, which will be good for me to get more experience using.

I do not think Swing is the best choice for me because of the problems creating GUI's for both Windows and Linux. In addition, the fact that Swing is old and the GUI will not look as good as it could with newer tools like Qt.

I think Xamarin would be a good choice for developing mobile applications but a static file analysis tool is much more useable on a desktop environment.

## Back End Technologies

### What are back-end technologies?

Back-end is a term used when referring to all the things that are happening in the background that the user does not see. For example, if a user runs the strings command that we have seen previously, they will receive an output of strings. They do not see how the program is getting the strings, which is all done in the background. There are many different programming languages that can be used for the back-end. The languages I will be covering are C++, Java and C#.

### C++

#### What is C++?

C++ is a programming language that was created as an extension to C to add object-oriented features. It is a low-level language and compiles directly to machine code. It is used where speed is important, is also one of the most popular languages, and has lots of support. In C++, memory is managed manually unlike some higher-level languages which use automatic garbage collection. This makes programming in C++ more complex. (Wikipedia, 2019)

#### Why use C++?

I have used C++ previously, so I can spend more time creating my project and won't need to spend time learning how to use it. It is also the main programming language implemented in Qt, which is the tool I would like to use for front-end development.

#### Pros and Cons of using C++

Pros:

- I have used it before, very little learning curve.
- It is the main language used by Qt.

Cons:

- More work involved in programming e.g. garbage collection.

## Java

### What is Java?

Java is a programming language designed to be easy to use and has an object-oriented model. Java programs are also very portable and will run on anything that has a Java Virtual Machine installed. It has a built in garbage collector, which will free up memory from objects that are no longer in use. This is one of the things that makes development easier. It is also popular for mobile apps, with the Android operating system being built in Java. (Wikipedia, 2019)

### Why use Java?

I have the most experience with Java over other programming languages. In my opinion, it is the easiest language that I have used. Java has a built in GUI toolkit called Swing. I have some experience making applications with Swing.

### Pros and Cons of using Java

#### Pros:

- Java is the language that I have the most experience using.

#### Cons:

- Not officially supported in Qt.
- There are unnecessary problems introduced when developing for multiple platforms using Swing.

## C#

### What is C#?

C#, pronounced C-Sharp, is an object-oriented programming language that runs on the .NET framework and was created by Microsoft. It is a high-level language that compiles to Common Language Runtime, which is a virtual machine that executes .NET programs. C# is mostly used to develop applications and games for Windows. (Microsoft, 2015)

### Why use C#?

C# is a high-level language and so it should be easy to use. I could use C# if I wanted to develop a mobile app, and I could use it with Xamarin to make the app cross-platform.

### Pros and Cons of using C#

#### Pros:

- Good for creating applications for Windows or mobile.

#### Cons:

- Not easy to create cross-platform programs for both Windows and Linux.
- I do not have much experience using C#.

## Conclusion

In conclusion, I believe the best choice for me is to use C++ for the back-end development of my static file analysis tool. C++ is the main programming language used by Qt, which is the tool I would like to use for the front-end development of my project. I have also used C++ before, I do not find it much harder than other languages, and even though there will be a bit more work involved, it should not have much of an impact overall.

I do not think that I should use Java because it is not officially supported by Qt and it think that Qt will be the better choice over Swing. I found Java and C++ to feel similar enough anyway and am comfortable using either.

C# is a good language for creating Windows or mobile apps but it does not work well for what I need.

## Summary and Conclusion

### Summary

In this report, the main areas looked at static file analysis, Existing Products, Front End Technologies and Back End Technologies. In static file analysis, I gave an overview of what it is and explained the techniques used for: Hashing, VirusTotal, Packed Files, Strings, Portable Executable DLL's and Disassembly. In Existing Products, I reviewed some common applications used during static file analysis and discussed what students want in an application. In Front End and Back End Technologies, explained what they are and reviewed Qt, Swing, Xamarin, C++, Java and C#, giving an overview, why I should use them and some pros and cons of each.

### Conclusion

In conclusion, I found out what a static file analysis tool must be able to do and what students learning about reverse engineering and malware analysis want in a tool. Looking at existing products, they do not suit the needs of students very well. They are unnecessarily complicated, and students would benefit from using a tool that is easier to use and only has the features they need. I want to create this tool, and after reviewing some front and back-end technologies, I have decided to use Qt and the Qt Creator IDE to develop the GUI interface and C++ for the back-end development. It will be cross-platform, working on both Windows and Linux.

## Bibliography

- Aldeid, 2013. *PEiD*. [Online]  
Available at: <https://www.aldeid.com/wiki/PEiD>  
[Accessed 07 October 2019].
- Dependency Walker, 2015. *Dependency Walker 2.2*. [Online]  
Available at: <http://www.dependencywalker.com>  
[Accessed 06 October 2019].
- JavaTpoint, 2018. *Java Swing Tutorial*. [Online]  
Available at: <http://javatpoint.com/java-swing>  
[Accessed 12 October 2019].
- Microsoft, 2015. *Introduction to the C# Language and the .NET Framework*. [Online]  
Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>  
[Accessed 22 October 2019].
- Microsoft, 2019. *Xamarin*. [Online]  
Available at: <http://dotnet.microsoft.com/apps/xamarin>  
[Accessed 16 October 2019].
- Miracle Salad, 2019. *md5 Hash Generator*. [Online]  
Available at: <https://www.miraclesalad.com/webtools/md5.php>  
[Accessed 03 October 2019].
- Ninja, S., 2015. *Static Malware Analysis*. [Online]  
Available at: <https://resources.infosecinstitute.com/malware-analysis-basics-static-analysis/>  
[Accessed 01 October 2019].
- NTInfo, 2019. *Detect It Easy*. [Online]  
Available at: <https://ntinfo.biz/index.html>  
[Accessed 05 October 2019].
- Sikorski, M. & Honig, A., 2012. Exploring Dynamic Linked Functions with Dependency Walker. In: *Practical Malware Analysis*. San Francisco: William Pollock, pp. 16-17.
- Sikorski, M. & Honig, A., 2012. Finding Strings. In: *Practical Malware Analysis*. San Francisco: William Pollock, pp. 11-13.
- Sikorski, M. & Honig, A., 2012. Packed and Obfuscated Malware. In: *Practical Malware Analysis*. San Francisco: William Pollock, pp. 13-14.
- Sikorski, M. & Honig, A., 2012. *Practical Malware Analysis*. San Francisco: William Pollock.
- SOFTPEDIA, 2018. *PEiD*. [Online]  
Available at: <https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml>  
[Accessed 07 October 2019].

Suslikov, E., 2019. *Hiew*. [Online]

Available at: <http://www.hiew.ru>

[Accessed 06 October 2019].

SweetScape Software Inc., 2019. *010 Editor*. [Online]

Available at: <https://www.sweetscape.com/010editor/>

[Accessed 10 October 2019].

The Qt Company, 2019. *Qt*. [Online]

Available at: <http://qt.io>

[Accessed 11 October 2019].

VirusTotal, n.d.. *How it works*. [Online]

Available at: [support.virustotal.com/hc/en-us/articles/115002126889-How-it-works](https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works)

[Accessed 01 October 2019].

Wikipedia, 2019. *C++*. [Online]

Available at: [http://en.wikipedia.org/wiki/C%2B%2B%](http://en.wikipedia.org/wiki/C%2B%2B%2B)

[Accessed 17 October 2019].

Wikipedia, 2019. *Java (programming language)*. [Online]

Available at: [http://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Java_(programming_language))

[Accessed 19 October 2019].

Yusirwan, S., Prayudi, Y. & Riadi, I., 2015. Implementation of Malware Analysis using Static and Dynamic Analysis Method. *International Journal of Computer Applications*, 117(6), pp. 11-15.