Static Analysis Project: A Research Report by Kevin Kelly

Table of Contents

What is Static Analysis?

Static Analysis, as defined by NIST's Computer Security Resource Center, is "Detecting software vulnerabilities by examining the app source code and binary and attempting to reason over all possible behaviors that might arise at runtime. [CITATION Nis21 \l 6153]" It is a means of dissecting a file's contents, examining its workings to determine if the file is potentially malicious. Unlike its counterpart,

dynamic analysis, static analysis conducts its investigation without executing the instructions provided by the code. The earliest known tool for Static Analysis was Lint, invented by Stephen C. Johnson in the 1970s [CITATION Owl12 \l 6153]

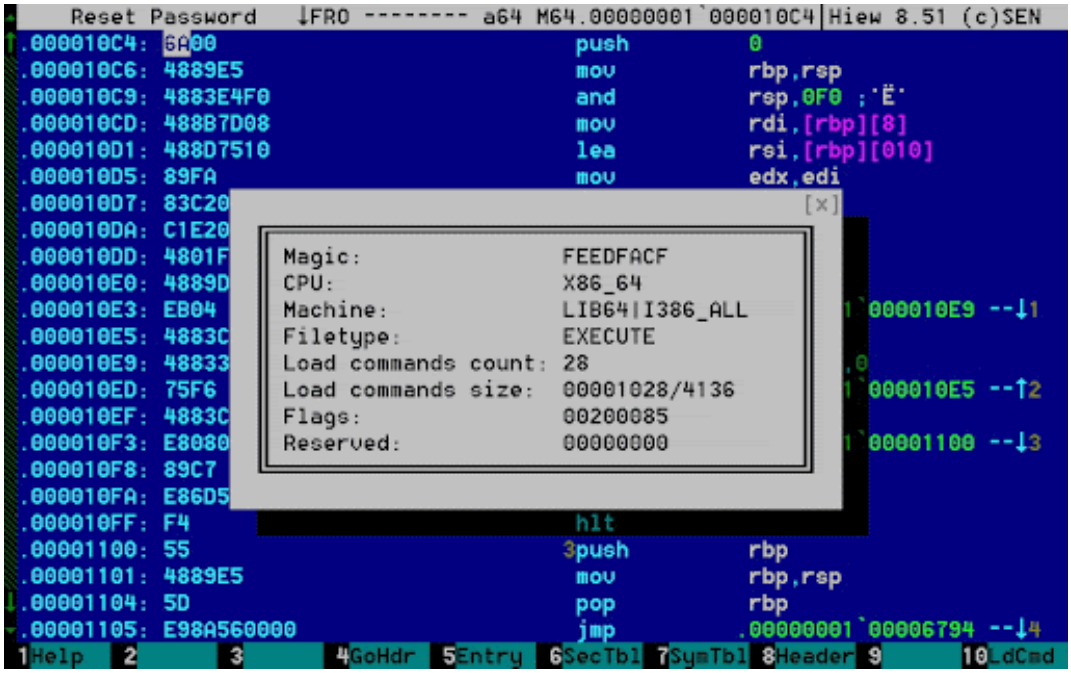
The features that a Static Analysis Tool tends to provide are as follows:

- **Disassembler:** A disassembler can be used to take an executable file's machine and reverse the assembly process to transform it into readable assembly language code. This can be used by the user to determine the file's purpose and if it can potentially cause harm to the system.
- **Detect Strings:** As part of the disassembler process, Static Analysis detects strings that are found in the code. This can provide very useful clues in determining the meaning behind the code. For example, if a tool detects the strings "FindFirstFile" and "FindNextFile", these could mean the malware is "ransomware". A tool detects Strings by searching for patterns of contiguous printable ASCII characters that end with a null character.
- **Find Imports:** A subset of Detecting Strings, strings ending in ".dll" on Windows or ".so" in Linux are compiled libraries that the file is importing to make use of its functions. These libraries can be checked against a lookup table to determine what the malware is making use of to achieve its objective. For example, "wininet.dll" is an API that allows a program to interact with HTTP and FTP protocols [CITATION Mic20 \l 6153]. If a static analysis tool finds this imported in the program, it can be determined it has network functionality.
- **Measure Entropy:** Entropy in the context of programming is a measure of a file's randomness. Creators of malicious programs attempt to disguise the danger of their work by encrypting or compressing their code, thus introducing entropy [CITATION Whi19 \l 6153]. By measuring a file's entropy, an analysist can detect suspicious sections of the code that must be decrypted or examined to determine the meaning behind it. A file's entropy can range from 0, not random, to 8, completely random. A file's entropy is determined by the variability of bits per byte.

Existing Tools

There are several tools available on the Internet that can perform static analysis. By examining their features and their limitations, useful information can be gleaned from this research that can dictate the working of the project.

Hacker's View (Hiew)



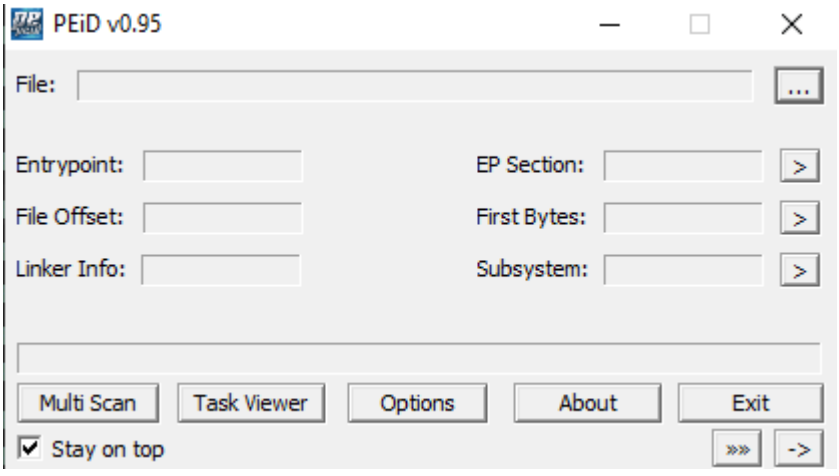
Advantages	Disadvantages
Boasts several useful features such as x86-64 disassembler, built in 64bit decrypt and block operations	Not all features are available for free, with some locked behind a payment
A popular tool used by many, meaning tutorials and documentation of other's use of it is plentiful	Simple and basic GUI, meaning it can be difficult to use for some
Available in both English and Russian	Is currently only available on Windows Operating Systems

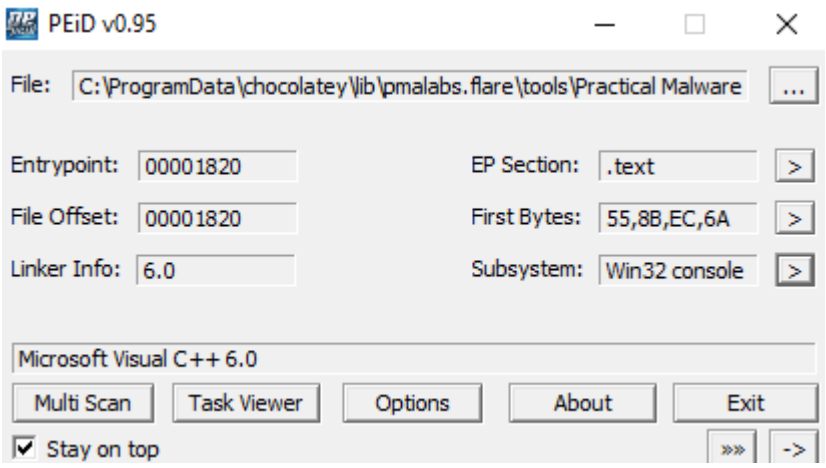
PEview

Advantages	Disadvantages
A tool dedicated to analyzing files	Can only be used regarding

in the Portable Executable (PE) that many malwares make use of	Windows, does not offer support for the Linux equivalent to PEs, Executable and Linkable Files (ELF)
It is a lightweight program, making it suitable for systems with limited resource availability	
Is free and easy to use, making it suitable for beginners	Light in features, meaning it cannot perform in-depth analysis

PEiD





Advantages	Disadvantages
Specializes in detecting if a PE file has been packed/compressed, as well as detecting how it was compiled	Does not offer support in static analysis beyond detection of compliment and packing
Is available in full for free	The website where it was originally hosted was discontinued, must be downloaded via a third-party site

From examining existing tools on the market, several issues arise:

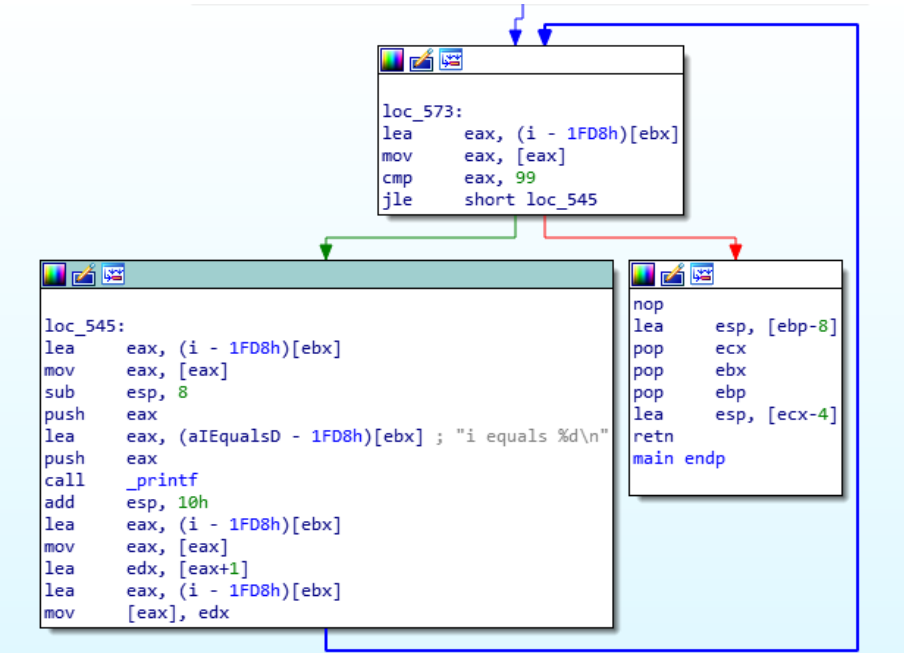
- The tools available may not be beginner friendly or difficult to use, for the GUI tends to be basic
- Each tool seems to specialize in one aspect of static analysis. For example, Hiew is used as a disassembler while PEiD is used for examining compression. There isn't a "one size fits all" tool available.
- The tools available appear to prioritize working on Windows Operating Systems. Linux appears to be often regarded as secondary or an afterthought.

From this, I can determine the objectives I wish my static analysis tool can achieve:

A static analysis tool that is executable on both Linux and Windows, supports a multitude of features that are critical in static analysis, including disassembler to assembly and pseudo-code, examination of

PE and ELF files and detection of compression or packing, all presented in an attractive, easy-to-use GUI.

A tool that I consider to be the paragon of what I aspire the project to achieve is the tool IDA. IDA is a static analysis tool that can perform disassembly on binary files to assembly language, but does so in a clear, easy-to-read GUI.



The screenshot showcases IDA being used on a simple C program that consists of a for loop. From the GUI, we can see through the arrows where the for loop begins, what occurs if the loop is true or false (green and red arrows respectively) and where the loop returns to it's beginning.

IDA also allows the user to rename variables found in the assembly to help understand what is occurring in the code. IDA's well-made GUI makes understanding assembly, something many can find daunting, a much easier process, and I hope to achieve a similar presentation and GUI in my project as well.

PEs, ELFs, and the Linux Technology Stack

A mandatory requirement of this project is a static analysis tool that is operable on Linux. Therefore, an understanding of the Linux Technology Stack and how it handles files and malware is critical in the success of this project.

Executable and Linkable Files (ELFs)

ELF is a popular file format in Linux that is used for binary files, shared libraries, core dumps etc. A ELF consists of the following format:

- An ELF Header
- Zero or more Segments
- Zero or more Sections

A Segment in an ELF is used during runtime, while a Section is relevant during link time. An ELF format can be read in Linux using the *readelf* CLI command [CITATION You20 \l 6153].

ELF files in it's Header, Segments and Sections uses it's own Data Types, specified as possible:

- **HalfWord:** 8-bit unsigned integer
- **Word:** 16-bit unsigned integer
- **Address:** Specifies addresses in memory. Can be 32-bit or 64-bit
- **Offset:** Specifies offsets in memory. Can be 32-bit or 64-bit

An ELF Header contains the metadata of the file, such as type of file, machine type, the offsets in memory of where the Program Headers and Section Headers, detailing the file's Segments and Sections respectively, as well as entries detailing the number of headers and their sizes.

At the beginning of the ELF file is the "EI_NIDENT" a series of 16 bytes that specify how the file is to be parsed. The beginning of EI_NIDENT of an ELF file always begins with the hex value 7F followed by 45, 4C, and 46, ASCII encoding of E, L, and F respectively. This is done to specify the file is an ELF file.

The following is the output of the ELF header of the Linux "ls" command, using *readelf –h /bin/ls:*

```
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x4049a0
Start of program headers:          64 (bytes into file)
Start of section headers:          124728 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         9
Size of section headers:           64 (bytes)
Number of section headers:         29
Section header string table index: 28
```

As stated previously, Program Headers specify the Segments of the ELF file. They are located together in an array in the ELF file.

Program Headers will define the follow characteristics of it's respective Segment:

- Type of Segment
- Where the Segment is located in the ELF file
- It's Virtual and Physical Addresses
- The segment's permissions (Read, Write, Execute)

Below is a screenshot of the segments of the Linux *ls* command, obtained using the command *readelf --segments /bin/ls*

```
Elf file type is EXEC (Executable file)
Entry point 0x4049a0
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000001da64 0x000000000001da64  R E    200000
  LOAD           0x000000000001de00 0x000000000061de00 0x000000000061de00
                 0x0000000000000800 0x0000000000001568  RW     200000
  DYNAMIC        0x000000000001de18 0x000000000061de18 0x000000000061de18
                 0x00000000000001e0 0x00000000000001e0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x000000000001a5f4 0x000000000041a5f4 0x000000000041a5f4
                 0x0000000000000804 0x0000000000000804  R      4
.
```

There are three main Segments in an ELF File

- **Code Segment:** Contains all code
- **Data Segment:** Contains all data
- **Stack Segment:** Memory Enclosing Stack

Similar to Segments, Sections are defined by Section Headers, whose location in the ELF file and quantity can be found in the ELF header.

A Section Header defines the following characteristics of a Section:

- Section Name (Not a string, but rather an offset into the Section Header String table, a series of bytes that are terminated by a null byte)
- Type of Section
- Flags (3 Types: SHF_WRITE determines that is a segment is created from this section, it should be writable. SHF_ALLOC determines that this Section should be loaded into memory, and SHF_EXECINSTR determines the section contains executable code)
- Location and size of the Section in the ELF file
- Specifications if the Section has any alignment constraints
- Any additional/miscellaneous information

Common Sections seen in the ELF file include:

- **.text:** Contains code
- **.data:** Where global variables and tables exist
- **.bss:** Contains uninitialized variables and array

Portable Executable Files (PEs)

Portable Executables is a common file format seen in Windows. It is the most common format for Windows Executables but is also seen in object code and Dynamic Link Libraries (DLLs). Static analysis tools such as the previously discussed PEView and PEiD are dedicated to analyzing PEs.

The structure of a PE file is as follows:

- **MS-DOS Header:** This is the same header present in the MS-DOS Operating System since Version 2. This is present so that, if an incompatible file is read by the Operating System, it can print out the string “This program cannot be run in DOS mode”. The MS-DOS Header is in the first 64 bytes of the PE file. Without the

MS-DOS Header as the first part of the PE file, the Operating System will fail to load the file into memory.

- **DOS Stub:** This contains the string that the program cannot be run in DOS mode.
- **PE Header:** Using the structure type IMAGE_NT_HEADER, the PE Header consists of three main components:
    - **Signature:** A four byte signature that is used to showcase that this file is a PE. In a PE file, this is the values 50 and 45, P and E in hex respectively, followed by two null bytes.
    - **IMAGE_FILE_HEADER:** Consists of 20 bytes that details basic information about the PE file such as the number of sections the PE file contains and the type of architecture it was programmed for.
    - **IMAGE_OPTIONAL_HEADER:** Contains far more important and detailed information about the PE file in comparison to the IMAGE_FILE_HEADER. It consists of 224 bytes.

The information regarding the PE file found in the IMAGE_OPTONAL_HEADER includes:

- **Magic:** Field that informs the OS that this executable is intended for either 32-bit or 64-bit systems, represented by the hex values 0x10b and 0x20b respectively.
- **AddressOfEntryPoint:** Indicates the location of the entry point for the application, as well as the location of the end of the PE file's Import Address Table (IAT), a table consisting of the DLLs the PE file uses to achieve it's purpose.
- **BaseOfCode:** Pointer to the beginning of the file's code.
- **BaseOfData:** Pointer to the beginning of the file's data section.
- **DllCharacteristics:** Contains properties of the DLLs used in conjunction with the PE file.
- **SizeofImage:** States how much space in memory to be reserved for the loaded executable file image.
- **SizeofHeaders:** Indicates how much space is used by the MS-DOS Header, IMAGE_FILE_HEADER, IMAGE_OPTIONAL_HEADER and Section Headers.
- **CheckSum:** A value used to verify the validity of the executable at load time.

As with ELFs, PE files contain Sections, which are loaded sequentially into memory. PE Sections have Section Headers that describes the properties of it's respective Section, including:

- Name
- Physical and Virtual Addresses
- Size and Pointer to Raw Data

Common Sections include:

- **.text:** Contains executable code
- **.data:** Contains initialized data
- **.bss:** Contains uninitialized data
- **.rdata:** Lists the Windows API and DLLs used by the PE file.

Linux Shared Object Files (SO)

A Shared Object (SO) File is a compiled library that is equivalent to a Windows Dynamic Link Library (DLL). It provides functionality to an executable. As mentioned previously, imports from libraries are a key factor in static analysis in determining a malware's purpose (for example, importing a library that can operate with HTTP protocols reveals the malware has network functionality). As this project's objective is to create a static analysis tool for Linux, an understanding of SO files is crucial to success. A SO file follows the file format of a ELF file.

SO files are dynamically loaded upon run time [CITATION Kno17 \l 6153], and the naming convention for said files begins with "lib" and end with the ".so" extension.

Standard SO libraries are typically found in the "/usr" or "/usr/lib" directories on a Linux system. When a program is loaded, the dynamic link loader first checks the environmental variable LD_LIBRARY_PATH, which specifies one or more paths to directories to search for the libraries required for the application to run successfully. This is searched before checking for any path specified in the ELF header of the application or in the standard libraries. This can prove to be a security risk. If the LD_LIBRARY_PATH variable is changed, it can be used to load malicious or dangerous code upon attempting to execute a program [CITATION DCC21 \l 6153].

Dynamic-Link Libraries (DLLs)

Dynamic-Link Libraries are the Windows equivalent to Linux Shared Object Files. They are modules containing functions and data designed to be used by other applications to achieve their purpose, and analyzing what DLLs are imported by a program provides especially useful clues in static analysis in determining the program's purpose.

DLLs define two types of functions: internal functions that are meant to be called by the DLL they are defined in, and external functions designed to be called by other applications and DLLs [CITATION Mic21 \l 6153]

When a program wishes to make use of a function in a DLL, there are two methods of calling it:

- In load-time dynamic linking, the program makes use of an import library, containing the information on where the systems' DLLs are located, to find the needed DLL and it's functions to call as if they originate from the application.
- In run-time dynamic linking, the program calls the function "LoadLibrary" or "LoadLibraryEx", functions that load the specified DLL into the specified address space in memory, followed by the function "GetProcAddress" which obtains the memory addresses of the required functions [CITATION Mic211 \l 6153].

Deciding the Best Language

In any undertaking involving programming, choosing what programming language to create the project is in is a vital decision, akin to choosing the best tools for a construction project.

In my situation, a project to create a static analysis tool, the following factors will dictate my choice:

- What experience do I have with the language? Am I comfortable in my knowledge of it to implement my project in?
- What libraries are available that can assist me in fulfilling the mandatory requirements of the project? Are these libraries available for free or at a cost?

- What options does the language have in terms of creating GUIs? One of my goals for this project is to make the tool user-friendly, so an easy-to-use and understand GUI is essential.
- Is the language portable? Can it operate on multiple architectures or Operating Systems (e.g. across Windows and Linux)

The following chapter will be research into potential candidates for the chosen language, its advantages and disadvantages, and my reasoning for my chosen language.

Python

Python, as defined by the “Python.org” website, is an “interpreted, object-oriented, high-level programming language with dynamic semantics [CITATION Pyt21 \l 6153].” It is a popular choice for programmers with a focus on readability and ease-of-use.

Advantages of Python

- With prioritization on readability, dynamic semantics and a syntax based on the English language, Python is an easy-to-use, easy-to-understand language. Combined with the fact I have undergone a Python training course, as well as real-world experience with the language in my work placement, I am confident with the idea of programming the project in Python.
- It is a portable language, operable across Windows and Linux [CITATION W3S21 \l 6153].
- Python offers great support for installing libraries. PIP is a tool in Python that can be used to install packages from the Python Package Index. Packages contain the code required for a module, which consists of Python libraries [CITATION W3S211 \l 6153]. This means I can easily install libraries to assist me in my project.
- There are several Python GUI frameworks available, such as PySimpleGUI, Wax, and Tkinter [CITATION Tow20 \l 6153]. This offers a great amount of choice in deciding how I wish to design the project’s GUI.

Disadvantages of Python

- Python is an interpreted language. While this offers its advantages, a particular fault with this setup is that it means

Python is slower in comparison to compiled languages such as Java [CITATION Gee21 \l 6153].

- Python is a high-level language, it does not work closely with the hardware. Static analysis is a low-level subject as one of the key components of it is to disassemble machine code into assembly language. This contrast can be a source of problems when implementing the project.

Java

Java is defined as a “general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. [CITATION Gur21 \l 6153]”.It is a programming language built off C# and C++.

Advantages of Java

- Java is the programming language I have the most experience in. Therefore, it is the language I am most comfortable with.
- Java can operate on multiple platforms, including Windows and Linux.
- Java is a compiled language. This means it can have greater speed in comparison to Python, an interpreted language.
- Swing is a toolkit used to design GUIs in Java. It is a toolkit I have great experience, using it to implement GUIs for previous projects.

Disadvantages of Java

- Java, unlike Python, does not have a focus on readability or syntax based on the English language. This can result in a more difficult process in implementing my vision of the project in Java, resulting in slower progress.
- Java is a slower language in comparison to it’s C#/C++ counterparts.
- Due to Java programs running on the Java Virtual Machine, meaning it can consume a higher-than-average amount of memory. This can clash with one of the end goals of the project: to create a market-releasable project. Higher memory usage means users with lower-end systems may not be able to use the tool (for example, a malware analyst may wish to use this tool to

analyze a file on an old test computer but is unable to due to the memory requirements).

C++

C++ is an objected-oriented language that is a superset of the C language. C++ alongside C now are considered low-level languages in comparison to modern standards [CITATION Cou20 \l 6153]

Advantages of C++

- C++ is a low-level language, meaning it operates closely with the hardware. This can prove to be useful in this project, as one of the components of static analysis, disassembly, is the act of converting machine code into low-level assembly language.
- C++ is operable on both Windows and Linux.

Disadvantages of C++

- While I do have experience with C++, it pales in comparison to the other previously mentioned languages. Therefore, I do not consider myself comfortable in my knowledge of the language.
- C++ is an older language compared to Python and Java. Therefore, it does not contain the more modern conveniences or readability that many have come to expect. Therefore, it can be a difficult language to implement the project in.
- I have little experience in working with GUIs in C++, what I have done using C++ GUIs have been simple diagrams that I found difficult to program. Therefore, I do not see C++ as a viable option for designing an easy-to-use GUI.

Conclusion

Taking each language's advantages and disadvantages into account, I have decided that Java will be the language I will use to code this project. It is the language I have the most experience in out of the languages I researched, it contains an extensive number of libraries

that can help me in implementing the necessary criteria, and I have experience creating GUIs in Java for previous projects.

Libraries with Java

Libraries are an essential aspect in implementing the project. They provide the code needed to add important features to the project to fulfill essential criteria.

Libraries that can be used to help implement the project include:

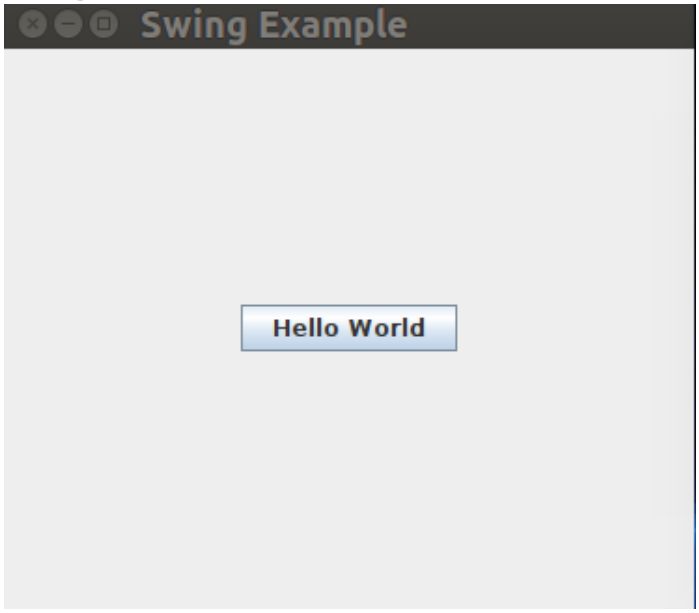
- **Capstone:** A lightweight, open source API that is used for file disassembly. File disassembly plays a large part in static analysis so a library to help incorporate this into my project would be incredibly useful. Reasons as to why Capstone would be appropriate for my project is that it is lightweight, architecture-neutral, and supported on Windows and Linux, helping to fulfill the criteria that the project operates on both platforms. Additionally, it is suited for malware analysis, able to detect actions preformed by malware in x86 architecture [CITATION Cap20 \l 6153].
- **Bayes Server:** Bayes Server provides APIs and Libraries for AI and statistics in several languages, including Java. One of the tools provided by the API allows the program to measure entropy. Bayes Server's technology is used by many companies including Mitsubishi, HP and the Australian and Canadian Government, making me confident in using their work to develop my project [CITATION Bay21 \l 6153]. Additionally, documentation of their API is present, which will be useful if I come across a roadblock when using it.

Deciding the Best GUI Toolkit

One of my goals for this project is to create an easy-to-use GUI that presents information to the user in a clear, understandable format, thus avoiding the user from becoming confused or frustrated. Therefore, what GUI Toolkit I decide to implement the GUI in is an important decision. Java provides several Toolkits to utilize in creating GUIs. The following chapter will be discussing the potential choices for

the Toolkit, and reaching my conclusion at the end. Each Toolkit discussed will be paired with a screenshot of a simple GUI containing a button built using said Toolkit.

Java Swing/SwingX




```
public class SwingExample extends JFrame{

    //JButton
    private JButton helloButton = new JButton("Hello World") ;

    public SwingExample() {

        super("Swing Example") ;
        setLocation(500, 200);

        //Panel for the Button
        JPanel buttonPanel = new JPanel() ;
        buttonPanel.add(helloButton) ;

        //Create a Panel for the Frame
        JPanel thePanel = new JPanel() ;
        thePanel.setLayout(new GridBagLayout() ) ;
        thePanel.add(helloButton, new GridBagConstraints()) ;
        thePanel.setSize(400, 400);


        add(thePanel) ;
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
```

Java Swing is a GUI Toolkit that's part of the Java Foundation Classes that is used to design window-based applications. GUIs designed in Swing consist of a collection of Swing components, from simple components such as buttons (JButton) and labels (JLabels) to more

complex examples such as Tabbed Panes (JTabbedPane) [CITATION Sec21 \l 6153].

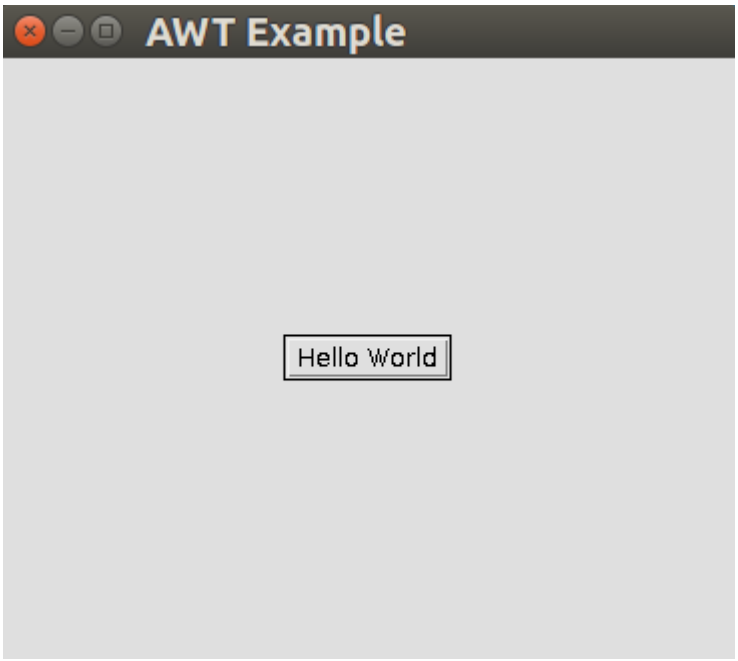
Advantages of Swing

- Java Swing is the GUI Toolkit I have the most experience with. I used it to implement projects in the past and find it easy to use and flexible.
- Swing components are lightweight. They leave a small impact on the systems' memory.
- Swing is platform-independent. Its components stay relatively the same from one machine to another. This is essential in the project criteria of having it be cross-platform [CITATION Sec21 \l 6153].

Disadvantages of Swing

- All of Swings components are drawn. This can result in a slower program in comparison to Toolkits such as AWT.
- It may require a separate JAR file in order to be functional [CITATION Sha19 \l 6153].

Java AWT




```
public class AWTExample extends Frame {
    //Button
    private Button theButton = new Button("Hello World") ;

    public AWTExample() {
        super("AWT Example") ;

        setLocation(500, 200) ;
        setSize(400, 400) ;

        Panel thePanel = new Panel() ;
        thePanel.setLayout(new GridBagLayout() ) ;
        thePanel.add(theButton, new GridBagConstraints()) ;
        thePanel.setSize(400, 400);

        add(thePanel) ;
        pack();
        setVisible(true) ;
    }
```

Java AWT or Abstract Window Toolkit is an early GUI Toolkit in Java. Similarly to Swing, it is used to design window-based applications. The AWT package provides classes to create a GUI, such as Label, TextArea and List [CITATION Jav21 \l 6153].

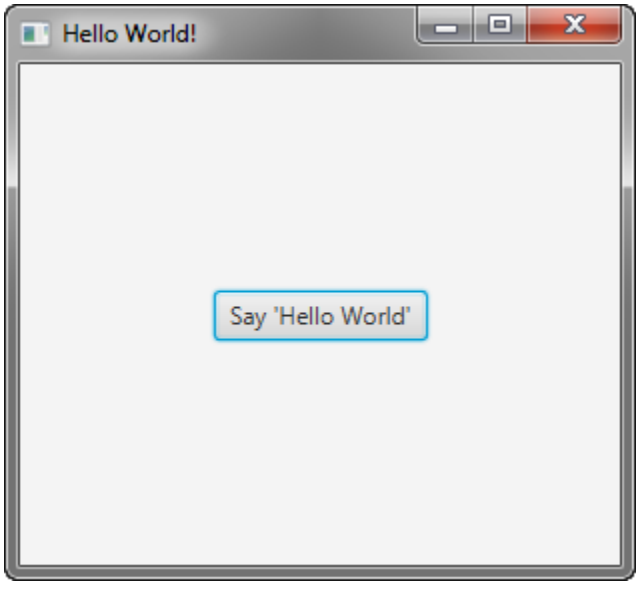
Advantages of AWT

- It is a stable toolkit that will rarely result in crashes. This can be useful in the context of static analysis when handling sensitive or malicious files.
- AWT components are implemented locally by the Operating System. This means the majority of the code is loaded as the system starts, resulting in fewer startup events.

Disadvantages of AWT

- AWT is a Toolkit I do not have much experience with, meaning I would have trouble implementing my intended vision of the project using it.
- AWT is platform dependent, it's components can change from machine to machine. This can cause issues when attempting to make my project able to run on both Windows and Linux.
- In comparison to Swing, AWT does not contain as many components to create a GUI. This makes it a less flexible Toolkit to create a project in.

JavaFX



[CITATION Ora14 \l 6153]

```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

[CITATION Ora14 \l 6153]

JavaFX is “an open source, next generation client application platform for desktop, mobile and embedded systems built on Java”. Not only can it be used for developing web applications, but also web applications to be run across multiple devices, such as Desktop and Mobile.

Advantages of JavaFX

- The components that make up a JavaFX can be styled using CSS, a language I have experience with. This means I would have greater flexibility in how the GUI is designed compared to other options.
- JavaFX provides a high number of graphical effects and animations when designing a GUI. These could prove to be useful when aiming for my goal of an easy-to-use GUI (For example, using animations to help guide the user and point out important aspects of the disassembled file)

Disadvantages of JavaFX

- The startup time for an application using JavaFX is long, especially in comparison to Swing.
- I do not have any prior experience with JavaFX. I would have to learn the Toolkit before I can use it for the implementation of the project, which will take time away from designing the project optimally.

Conclusion

Considering the options available, I've decided I will use Swing/SwingX to create the GUI for the project. It is the Toolkit I have the most experience with, therefore it is a tool I am comfortable with. Additionally, it provides several powerful components that will help fulfill my goal of designing a easy-to-use GUI.

Conclusion

In order to implement this project, I will:

- Create a Static Analysis Tool for Linux. This tool must be able to disassemble files, measure entropy, find what libraries are imported, and detect Strings. This is all done to allow the user to determine the functionality and potential impact of a file without executing it.
- Code the project using the Java Programming Language
- Incorporate the GUI using the Java Swing/SwingX GUI Toolkit
- Focus on ensuring the tool is easy to understand and use

This report has been essential in broadening my understanding regarding Static Analysis, the Linux Technology Stacks, and the choices a person would consider when deciding a programming language. The knowledge accumulated will be invaluable in ensuring my project is well-implemented and a success.

References

Bayes Server, 2021. *Bayes Server.* [Online]
Available at: https://www.bayesserver.com/
[Accessed 17 November 2021].

Capstone, 2020. *Capstone The Ultimate Disassembler.* [Online]
Available at: https://www.capstone-engine.org/
[Accessed 17 November 2021].

CourseReport, 2020. *A Guide to Low Level Programming for Beginners.* [Online]
Available at: https://www.coursereport.com/blog/a-guide-to-low-level-programming-for-beginners
[Accessed 3 November 2021].

DCC Computing Center, 2021. *LD_LIBRARY_PATH - or: How to get yourself into trouble!.* [Online]
Available at: https://www.hpc.dtu.dk/?page_id=1180
[Accessed 3 November 2021].

GeeksForGeeks, 2021. *Disadvantages of Python.* [Online]
Available at: https://www.geeksforgeeks.org/disadvantages-of-python/
[Accessed 3 November 2021].

Guru99, 2021. *What is Java? Definition, Meaning & Features of Java Platforms.* [Online]
Available at: https://www.guru99.com/java-platform.html
[Accessed 3 November 2021].

JavaTPoint, 2021. *Java AWT Tutorial.* [Online]
Available at: https://www.javatpoint.com/java-awt
[Accessed 8 November 2021].

Knowledge, 2017. *What is the Difference Between .so and .a files in Linux.* [Online]
Available at: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8svSAC
[Accessed 3 November 2021].

Microsoft, 2020. *About WinINet.* [Online]
Available at: https://docs.microsoft.com/en-us/windows/win32/wininet/about-wininet
[Accessed 3 November 2021].

Microsoft, 2021. *About Dynamic-Link Libraries.* [Online]
Available at: https://docs.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries
[Accessed 3 November 2021].

Microsoft, 2021. *Dynamic-Link Libraries (Dynamic-Link Libraries).* [Online]
Available at: https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries
[Accessed 3 November 2021].

NIST, 2021. *Static Analysis.* [Online]
Available at: https://csrc.nist.gov/glossary/term/static_analysis
[Accessed 3 November 2021].

Oracle, 2014. *Hello World, JavaFX Style.* [Online]
Available at: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/hello_world.htm
[Accessed 17 November 2021].

Owlapps, 2012. *Lint (software).* [Online]
Available at: http://www.owlapps.net/owlapps_apps/articles?id=18692&lang=en
[Accessed 3 November 2021].

Python, 2021. *What is Python? Executive Summary.* [Online]
Available at: https://www.python.org/doc/essays/blurb/
[Accessed 3 November 2021].

Section, 2021. *Introduction to Java Swing.* [Online]
Available at: https://www.section.io/engineering-education/introduction-to-java-swing/#differences-between-java-swing-and-java-awt
[Accessed 8 November 2021].

Sharma, S., 2019. *Introduction To Swing in Java.* [Online]
Available at: https://www.c-sharpcorner.com/UploadFile/fd0172/introduction-of-swing-in-java/
[Accessed 8 November 2021].

Towards Data Science, 2020. *Top 10 Python GUI Frameworks for Developers.* [Online]
Available at: https://towardsdatascience.com/top-10-python-gui-frameworks-for-developers-adca32fbe6fc
[Accessed 3 November 2021].

W3Schools, 2021. *Introduction to Python.* [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 3 November 2021].

W3Schools, 2021. *Python PIP.* [Online]
Available at: https://www.w3schools.com/python/python_pip.asp
[Accessed 3 November 2021].

Whiteheart, 2019. *Entropy Analysis: A critical test for malware's.* [Online]
Available at: https://whiteheart0.medium.com/entropy-analysis-a-critical-test-for-malwares-69939f5b8b1
[Accessed 3 November 2021].

YouTube, 2020. *In-depth ELF - The Extensible & Linkable Format.* [Online] Available at: https://www.youtube.com/watch?v=nC1U1LJQL8o [Accessed 3 November 2021].