**REsearch Report**

(Security, 2023)

***Student ID: C00276038 Student’s Name: Fionn Finane Course: 4th Year Cybercrime and IT Security BSC (Honours)***



**Table of Contents**

[**Introduction to Malware Analysis** 2](#_Toc182194296)

[**What is Malware?** 2](#_Toc182194297)

[**What is Malware Analysis?** 2](#_Toc182194298)

[**What does it involve?** 2](#_Toc182194299)

[**What are the prerequisites before diving into Malware Analysis?** 3](#_Toc182194300)

[**What is the purpose of Malware Analysis?** 3](#_Toc182194301)

[**Malware Types** 4](#_Toc182194302)

[**Downloaders and Loaders** 4](#_Toc182194303)

[**Backdoors** 4](#_Toc182194304)

[**RATs** 5](#_Toc182194305)

[**Botnets** 5](#_Toc182194306)

[**Reverse Shell** 6](#_Toc182194307)

[**Credential Stealers** 7](#_Toc182194308)

[**GINA INTERCEPTION** 7](#_Toc182194309)

[**Static Analysis** 10](#_Toc182194310)

[**PE File Format** 10](#_Toc182194311)

[**Linking Libraries & Set of Functions** 11](#_Toc182194312)

[**Linux ELF Brief File Structure** 11](#_Toc182194313)

[**Tools** 11](#_Toc182194314)

[**Hashing** 12](#_Toc182194315)

[**What is Hashing?** 12](#_Toc182194316)

[**Components Of Hashing** 13](#_Toc182194317)

[**Wrap it up** 14](#_Toc182194318)

[**Signatures** 14](#_Toc182194319)

[**When Analysing Malware** 14](#_Toc182194320)

[**Host-based Signatures** 15](#_Toc182194321)

[**Network Signatures** 15](#_Toc182194322)

[**Wrap it up** 15](#_Toc182194323)

[**File Entropy** 15](#_Toc182194324)

[**Packed Files** 16](#_Toc182194325)

[**The Unpacking Stub** 16](#_Toc182194326)

[**Unpacked Files** 18](#_Toc182194327)

[**Similar Products / Tools** 19](#_Toc182194328)

[**Guided Walkthrough** 23](#_Toc182194329)

[**Python** 26](#_Toc182194330)

[**Pleading the case for Python** 26](#_Toc182194331)

[**Python (Malware Analysis) Libraries** 27](#_Toc182194332)

[**Angr (Open-Source Binary Analysis Toolkit)** 29](#_Toc182194333)

[**What is Angr?** 29](#_Toc182194334)

[**What is the function of Angr?** 30](#_Toc182194335)

[**Flask (Web-Framework)** 31](#_Toc182194336)

[**Pleading the case for Flask Web (Minimalistic) Framework** 31](#_Toc182194337)

[**Conclusion** 32](#_Toc182194338)

[**References** 34](#_Toc182194339)

# **Introduction to Malware Analysis**

Before properly introducing this chapter's leading content and scope, such as the primary focus on providing an "Introduction to Malware Analysis", it would make sense to highlight what malware is briefly.

What do we mean by Malicious?

Before we can study the Malware Analysis process, we need to clarify the malware.

## **What is Malware?**

Malware is defined as any software or program that performs an undisclosed action that could lead to intentional or unintentional damage or harm to the user, computer or network. (Honig M. S., Malware Analysis Primer, 2012)

Typical examples include viruses, trojan horses, keyloggers, backdoors, rootkits, ransomware, worms, bloatware, adware, scareware, and spyware, all examples of malicious software or malware. (Honig M. S., Malware Analysis Primer, 2012)

Now that we have a clearer idea of what malware is, we can get into the meat of this chapter, that is, the following questions we hope to answer in this chapter.

What is Malware Analysis? What does it involve? What are the prerequisites before diving into malware analysis? One more thing could be highlighting the purpose or benefit of malware analysis. (Honig M. S., Malware Analysis Primer, 2012)

## **What is Malware Analysis?**

The examination, processing, and deconstruction of malware are done by identifying its origin, purpose, behaviour, severity of impact and even potential threat or impact of a malicious program or malware sample. (Honig M. S., Malware Analysis Primer, 2012)

## **What does it involve?**

Malware Analysis involves a primary process called Reverse Engineering, which breaks down malware to gain insight into the design, functionality and architecture through disassemblers or reverse engineering tools. (Honig M. S., Malware Analysis Primer, 2012)

These tools are used primarily in one of the two primary techniques: Static or Dynamic Analysis. (Honig M. S., Malware Analysis Primer, 2012)

They are the go-to methods that malware analysts use in their examination and documentation process, which we will cover in later chapters, mainly when talking about "Static Analysis ", and we will highlight some popular tools that are used in this area when studying and examining the ranges of different malware samples we encounter throughout this document. (Honig M. S., Malware Analysis Primer, 2012)

## **What are the prerequisites before diving into Malware Analysis?**

It is beneficial to have a familiarity with the primary operating systems, especially Windows and Linux, which are the operating systems that will be highlighted more due to how standard malware samples are found on Windows in comparison to Linux or Mac OS environments.

Another is a foundational knowledge of one or more programming languages, especially Python, C, C++, and Assembly, as these languages are popular with malware authors when developing malicious code.

Lastly is a basic familiarity with networking methodologies and terminologies, particularly when it comes to understanding TCP and UDP, understanding of forms of data and the importance of knowing the OSI model as well as Internet Protocol for interrupting and tracing data transmitted across the network, which we will need to know when we are running malware samples in our sandboxes, observing its behaviour and primary functionality.

## **What is the purpose of Malware Analysis?**

To answer this question, we must adopt a detective's mindset before engaging in the Process. (Honig M. S., Malware Analysis Primer, 2012)

We will start by determining and providing the information needed to respond to a scenario such as a network intrusion as an example of a potential threat scenario. (Honig M. S., Malware Analysis Primer, 2012)

The main objective of malware analysis is to determine what happened and ensure that we have the location of all infected computers and files. (Honig M. S., Malware Analysis Primer, 2012)

We aim to ensure intervention is made earlier to mitigate and decrease the total impact scope, such as the damage caused by a particular malware we are investigating. (Honig M. S., Malware Analysis Primer, 2012)

Malware Analysis differs from Reverse Engineering simply because the objective or goal of Malware Analysis is primarily to understand its purpose and functionality and generally not to reconstruct the malware. (Honig M. S., Malware Analysis Primer, 2012)

An important point also is that we are finding identifiable traits or hints to indicate the origin, purpose, category of malware classification, and functionality, and we disassemble or deconstruct in hopes of increasing the efficiency of antivirus, especially anti-malware tools or software to catching different malware types that are particularly difficult or usually slip through the cracks. (Honig M. S., Malware Analysis Primer, 2012)

So, malware analysis and reverse engineering analyse the binary code of any file we receive. However, translating from machine code to assembly code is the disassembly process. (Honig M. S., Malware Analysis Primer, 2012)

Nevertheless, reverse engineering is a broad terminology, so there is overlap but a difference between the two. (Honig M. S., Malware Analysis Primer, 2012)

Now that we understand malware analysis, malware, and how reverse engineering connects to malware analysis, let us investigate the malware types or classifications we will encounter in this next chapter. (Honig M. S., Malware Analysis Primer, 2012)

# **Malware Types**

Here in this chapter, we are, as mentioned in the first chapter, "Introduction to Malware Analysis", going to examine the behaviour of malware in a wide range of malware classifications. (Honig M. S., 2012)

We are starting first with Downloaders and Loaders. (Honig M. S., 2012)

## **Downloaders and Loaders**

These malware types are the most frequently encountered malware families throughout this chapter, as these are some of the primary malware delivery methods. (Honig M. S., 2012)

We do not want to say it is self-explanatory because it is not entirely self-explanatory. (Honig M. S., 2012)

However, the delivery mechanism used here will determine the distinction between Downloaders and Launchers. (Honig M. S., 2012)

With that said, downloaders must download another part of malware from the Internet and execute it once it is delivered to the local system, often involving the malware being packaged with an exploit. (Honig M. S., 2012)

To clarify, this will commonly involve using Windows API known as "URLDownloadtoFileA", which proceeds with calling "WinExec" to download and execute the new malware. (Honig M. S., 2012)

Also, under the alias of a loader, it is defined as any executable that self-extractions its malware, installing it immediately or requiring future convert execution. (Honig M. S., 2012)

Examples would include executable or DLL in its resource section to set things so that the malicious nature is concealed from a user, containing the malware they are designed to load. (Honig M. S., 2012)

If the resource section is compromised or encrypted, the malware will have to extract the resource section before loading. (Honig M. S., 2012)

They often use resource-manipulation API functions such as "Find Resource", "Load Resource", and "Size of Resource". (Honig M. S., 2012)

## **Backdoors**

A Backdoor is a form of malware designed to give attackers the ability to gain discrete remote access to their victim's machine. (Honig M. S., 2012)

Backdoors are found to be the most sought-after forms of malware, coming in different ways and ranging in degrees of capabilities. (Honig M. S., 2012)

The code for the Backdoor will typically pose the complete set of capabilities. (Honig M. S., 2012)

Because of the use of a Backdoor, attackers usually do not need to download additional malware or malicious code. (Honig M. S., 2012)

Do you know we can determine which of these features is implemented by a backdoor by looking at the Windows functions it uses? (Honig M. S., 2012)

Backdoors utilise communication over the Internet in many ways. (Honig M. S., 2012)

However, the most desirable method is over port 80 using HTTP protocol because HTTP is the go-to protocol for outgoing network traffic. (Honig M. S., 2012)

It is an excellent opportunity for malware to exploit, enabling it to blend in with outgoing traffic. (Honig M. S., 2012)

Here are the standard set of capabilities, such as the ability to manipulate registry keys, list display windows, search files, and create directories, as well as other functions we could find interesting to investigate for further research. (Honig M. S., 2012)

Here is the standard possible Process for a backdoor that might be encountered (Honig M. S., 2012)

The malware will spawn two threads after the "CreateProcess" function call. (Honig M. S., 2012)

The first thread reads the socket and writes to the "stdout" pipe. (Honig M. S., 2012)

We often find that these threads manipulate the data using data encoding. (Honig M. S., 2012)

Note that we can reverse engineer the encoding and decoding routines used by the threads to decide packet captures containing encoded sessions. (Honig M. S., 2012)

### **RATs**

Here is our first encounter with backdoors, the malware known as RATs, which stands for Remote Administration Tool, used to remotely manage another computer or multiple computers. (Honig M. S., 2012)

RATs will often be utilised in a targeted attack for a specific task or even a series of attacks, such as exfiltrating information or laterally travelling across the network. (Honig M. S., 2012)

As an example of a RAT's Network Structure, we might find the server running on the victim host infected with malware embedded on their machine. (Honig M. S., 2012)

The client is operating and executing remotely as the command-and-control unit under the complete control of the attacker. (Honig M. S., 2012)

The servers reach out to the client to establish a connection and, as a result, are now taken over by the attacker. (Honig M. S., 2012)

The RAT will standardly communicate over widely used ports like 80 (HTTP) or 443 (HTTPS). (Honig M. S., 2012)

A popular remote administration tool, Poison Ivy RAT is utilised and controlled through shellcode plugins and is useful when running for testing or analysis purposes. (Honig M. S., 2012)

### **Botnets**

The Botnet is characterised as a collection of compromised hosts known as Zombies, controlled by a single entity, generally through a server known as the Botnet Controller. (Honig M. S., 2012)

Our goal as the Botnet is to spread and multiply, taking control of as many hosts as possible, to create an army of zombies, and ultimately to create a massive network to perform further or multiple malicious attacks such as spreading worms, viruses, creating phishing campaigns and commonly even perform a distributed denial of service attack, that can result in taking an online service or website offline just by orchestrating the Botnet's Zombies to attack all at once at the same time. (Honig M. S., 2012)

To differentiate between RATs and Botnets, starting with RATs, which are characterised as taking over on a per-victim basis due to the nature of RATs requiring the attacker to interact with the host directly, is the more intimate approach when compared to Botnets, which all can be quickly compromised at the same time or all at once. (Honig M. S., 2012)

Botnets will be utilised for mass-scale attacks. (Honig M. S., 2012)

RATs are limited due to the requirement to infect hosts one at a time, resulting in fewer hosts. (Honig M. S., 2012)

### **Reverse Shell**

A Reverse Shell is a connection made from an infected machine that provides attackers SSH or remote-level access to this machine. (Honig M. S., 2012)

Reverse Shells are generally standalone malware that is typically part of sophisticated backdoors, so it can be one of the two typical uses of Reverse Shell, or both are mutually exclusive. (Honig M. S., 2012)

Reverse Shells are so powerful that attackers use this connection to their victim's computer to execute commands with privileged access to their victim's local machine. (Honig M. S., 2012)

We will accomplish this by using two types of reverse shells we will standardly encounter. These are Windows Reverse Shells and Netcat Reverse Shells, starting first with Windows Reverse Shells. (Honig M. S., 2012)

#### **Windows Reverse Shells**

Windows Reverse Shells are attackers that employ two simple malware coding implementations for reverse shells on Windows using cmd.exe, basic and multi-threaded. (Honig M. S., 2012)

The primary method is popular among malware authors since it is easier to write and generally works just as well as the multi-threaded technique. (Honig M. S., 2012)

It involves a call to "Create Process" and manipulating the "STARTUPINFO" structure passed to "Create Process". (Honig M. S., 2012)

First, a socket is created, and a connection to a remote server is established. (Honig M. S., 2012)

That socket is then tied to the Standard Streams of (Standard Input, Standard Output and Standard Error) for cmd.exe. (Honig M. S., 2012)

#### **Reverse Shells (Netcat)**

Netcat is a multi-purpose networking (TCP/IP) tool. (Honig M. S., 2012)

One of these capabilities is to set up in listening mode. (Honig M. S., 2012)

We make it act as a server, which can be modified and used to develop a reverse shell by ensuring Netcat executes both the attacker's and the victim's machines. (Honig M. S., 2012)

Attackers are generally known to utilise Netcat and even package into other malware. (Honig M. S., 2012)

**A demonstration of how this process may work will be displayed below.** (Honig M. S., 2012)

1. Start by setting the Netcat with the switch -l to listening mode and set the port with "-p (Honig M. S., 2012) [number] – [remote\_machine\_ip\_address]". (Honig M. S., 2012)
2. Next, the victim's machine connects and provides the shell. (Honig M. S., 2012)

nc [remote\_machine\_ip\_address ] 80 -e (executable switch) cmd.exe (programs or executable ) (Honig M. S., 2012)

Number = port number such as 80 for HTTP protocol. (Honig M. S., 2012)

Remote\_machine\_ip\_address = 12345. (Honig M. S., 2012)

Create Process runs cmd.exe with its window suppressed to hide it from the victim. (Honig M. S., 2012)

The multi-threaded version of a Windows Reverse Shell involves the creation of a socket, two pipes and two threads. (Honig M. S., 2012)

The following are good API calls: Create Thread and Create Pipe. (Honig M. S., 2012)

Malware Authors sometimes use this method to manipulate or encode the data coming in or going out over the socket. (Honig M. S., 2012)

The Create Pipe can tie read and write ends together to a pipe such as Standard Input or stdin and Standard Output or stout. (Honig M. S., 2012)

The Create Process method can pipe the Standard Streams instead of directly to the sockets. (Honig M. S., 2012)

**Credential Stealers**

It contains the strategies that attackers will likely go through to achieve their objective or steal credentials, known as GINA Interception, primarily with the three types of malware: (Honig M. S., 2012)

1. Programs are characterised as waiting for users to log in to exfiltrate their credentials. (Honig M. S., 2012)
2. Programs that dump information stored on Windows, such as password hashes that (Honig M. S., 2012) need to be cracked offline, are known as Hash Dumping. (Honig M. S., 2012)
3. Programs that function to identify and log keystrokes are known as keyloggers. (Honig M. S., 2012)

### **GINA INTERCEPTION**

Within Windows XP, Microsoft develops the Graphical Identification and Authentication or GINA system to allow legitimate third parties to perform their custom login process by adding authentication using Radio Frequency Identification tokens or smart cards. (Honig M. S., 2012)

These third-party support mechanisms were even discovered or made known to malware authors, who found that this could allow them to easily intercept and load their credential stealers to exfiltrate user credentials. (Honig M. S., 2012)

GINS was implemented in DLL, e.g., msgina.dll and loaded by the Winlogon executable during the login process. (Honig M. S., 2012)

Winlogon also works for third-party customisation implemented in DLLs, loading them between Winlogon and GINA DLL. (Honig M. S., 2012)

It behaves as another type of simulated attack, a man-in-middle attack. (Honig M. S., 2012)

It is worth noting that Windows is easy to use, providing the registry location where we will find the third-party DLLs that Winlogon also loads. (Honig M. S., 2012)

HKLM\SOFTWARE\Microsoft\Windows NT \Current Version\WinLogon\GinaDLL (Honig M. S., 2012)

For example, we found a malicious file, fsgina.dll, installed in the exact location in the registry and acting as a GINA interceptor. (Honig M. S., 2012)

The main-in-the-middle attack is being executed through the Logon credentials, which are flowing within the system utilising the malicious file sitting in between the WinLogon.exe and the msgina.dll, and here we can see that the malware can capture all submitted user credentials being sent to the system utilised for authentication. (Honig M. S., 2012)

Moving on to the next chapter, we will introduce Static analysis in all its glory.

# **Static Analysis**

We mentioned this distinction between Dynamic and Static Analysis because it was essential to clarify. (Honig M. S., Basic Static Techniques, 2012)

However, knowing the distinction already highlighted in our first chapter briefly, which is defining the malware analysis process, we will finally highlight and talk about Static Analysis here in this chapter. (Honig M. S., Basic Static Techniques, 2012)

Static analysis involves examining the assembly instructions, looking for the functions and sections containing elements of the overall code structure of a file with the blueprint we call Portable Executables that are utilised by Dynamic Link Libraries and Executable Files in Windows and other formats such as present in some legacy systems through examination of the malware's code structure without having to run it in a controlled or isolated environment such as virtual machines through utilising hypervisors creating our malware lab, as we would be required to set up in the case of a Dynamic Analysis being conducted and to execute this type of malware analysis successfully. (Honig M. S., Basic Static Techniques, 2012)

## **PE File Format**

The Portal Executable file format utilised in Windows contains object code, DLLs and executables. (Honig M. S., Basic Static Techniques, 2012)

PE File format will contain the data structure with critical information necessary for the Windows OS Loader. (Honig M. S., Basic Static Techniques, 2012)

Windows OS Loader is responsible for managing the wrapped executable code. (Honig M. S., Basic Static Techniques, 2012)

Most of the executable code coming from each file that Windows loads is contained in PE File Format, resulting in this being a commonly used file structure. (Honig M. S., Basic Static Techniques, 2012)

Rarely, but legacy file formats still allow malware to occur. (Honig M. S., Basic Static Techniques, 2012)

**Here is the standard Portable Executable File Format Header information we will encounter** (Honig M. S., Basic Static Techniques, 2012)

1. The Imports are the functions from other libraries that a given malware will likely utilise. (Honig M. S., Basic Static Techniques, 2012)
2. The Exports house the functions within the malware that are to be called by other libraries. (Honig M. S., Basic Static Techniques, 2012)
3. The Time Date Stamp stores the time from when the program was compiled. (Honig M. S., Basic Static Techniques, 2012)
4. The Sections state the name of each section within the file containing the sizes on disk and memory. (Honig M. S., Basic Static Techniques, 2012)
5. The Subsystem states whether the program is defined as one of the following will be found here: Command Line Interface (CLI) or Graphical User Interface (GUI) application. (Honig M. S., Basic Static Techniques, 2012)
6. The Resources contain the strings, icons, menus and any other information in the file. (Honig M. S., Basic Static Techniques, 2012)

## **Linking Libraries & Set of Functions**

Linked Libraries and lists of functions are the most valuable and exciting information when analysing executables, mainly the imports that utilise functions inherited from another program or multiple programs. (Honig M. S., Basic Static Techniques, 2012)

These functions are commonly imported coding libraries that usually inherit libraries from multiple programs, not just singular programs, accomplished by linking. (Honig M. S., Basic Static Techniques, 2012)

Linking is the method that can provide a connection to the primary executable, which, when utilised, enables linked imports for the programmer's programs to be easily streamlined, removing the need for the programmer to deal with multiple programs individually it would be tedious for a programmer to apply current operations for many programs repeatedly.

When dealing with coding libraries, believe it or not, there is more than one form of linking, usually during runtime, implemented statically or dynamically, in other words, DLLs. (Honig M. S., Basic Static Techniques, 2012)

Dynamic Linked Libraries are commonly used to dynamically link host Operating System findings for the required libraries during program loading. (Honig M. S., Basic Static Techniques, 2012)

The program will call a function that executes within its library, referred to as the linked library function. (Honig M. S., Basic Static Techniques, 2012)

## **Linux ELF Brief File Structure**

The Executable and Linkable Format, or ELF, is interesting because it has a unique file format structure incorporating functionalities similar to those of both Dynamic Link Libraries and the Executable Files derived from Windows, except the ELF binary, are indeed structured differently to Windows Portable File Format that is utilised by EXEs or Executable Files, with sections similar to its own that we may have guessed, it is known as the ELF Header containing basic information related to the binary as well as the indexes of the Program Header Table (PHT) and the Section Header Table (SHT), moving onto PHT that provides the descriptions of how and where ELF File's data info in memory that is to be loaded, SHT interesting that is a more an additional feature and not an essential component within the ELF File structure that will contain the mapping of data for purposes of assisting when debugging and last component is known as Data housing the entire binary's data of which responsible for pointing to the PHT and SHT coming solely from this section. (Mattick, 2023)

The tools that are among the most used for static analysis are listed below.

## **Tools**

1. **Dependency Walker** builds a hierarchical tree diagram of all dependent modules representing various information we expect to obtain from any binary file. (Software C. , 2024)
2. **PEView** isa tool that specialisesin viewing PE structures. It has an editing feature to modify PE headers for learning purposes or to fix invalid PE files.

Use the tool to view Imported DLLs and functions of any Windows files.

Find out what functions are exported and what offsets those that start execution.

It is also a fantastic tool for beginners who want to familiarise themselves with the PE file structure. (CNET, 2012)

1. **PEBrowse Professional** is both a disassembler and a static analysis tool for Windows executables.

The PEBrowse disassembler allows us to open and examine executes without loading parts of any active processes, such as the debugger.

Applications, system DLLs, and device drivers are all candidates for offline analysis, optionally using PEBrowse programs. (CNET, 2013)

1. **PEiD** isutilised primarily to quickly identify if a given PE file is indeed obfuscated or not and identify its file location utilising the file's Entry Point to scan using Normal or Deep mode to generate results due to PEiD's error control method, all with user-friendly Application utilised for Windows environment. (SOFTPEDIA, 2018)
2. **Resource Hacker** is the go-to resource editor compatible with 32-bit and 64-bit, designated for the Windows operating system, which functions as both a compile and decompile executable with an excellent GUI interface application that allows modification of resources within an executable. (Johnson, 2023)
3. **PE Explorer** is a program that allows for extracting significant information, such as Windows applications and libraries, so we do not need to examine the source code.

Once we have selected the file we wish to examine, PE Explorer will analyse finding resource information, and PE Header information can be obtained from a PE file.

From here, the tool will allow us to explore the specific elements within an executable file. (Software H. , 2024)

# **Hashing**

So, we will move on to hashing, not just as we know from previous chapters, which is utilised to verify integrity, particularly in ensuring intact file integrity for our use case, which is malware classification, examination and fingerprinting captured for any given malware we encounter as malware researchers. (GeeksforGeeks, What is Hashing?, 2024)

We can see how powerful hashing algorithms can be, so much so that it is often utilised in not just cryptography but also to verify and safely cross-examine malware samples received to give ourselves a better understanding of its malware signatures, which will hopefully identify and classify the malware group we are dealing with and VirusTotal is a valuable and brilliant tool for automating and shortcutting this for us. (GeeksforGeeks, What is Hashing?, 2024)

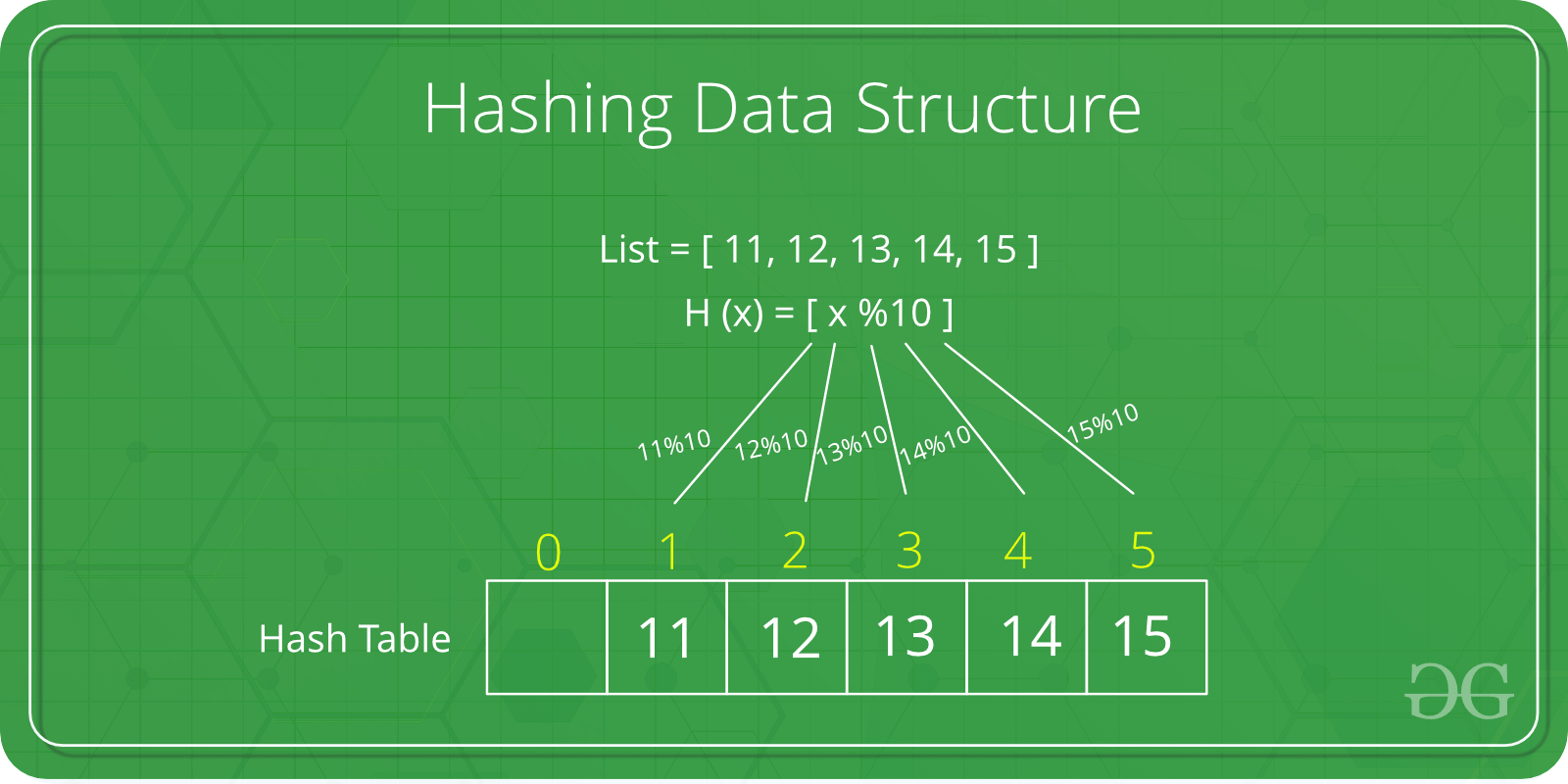
## **What is Hashing?**

Hashing is the process that involves the generation of a fixed-size output determined solely by the variable size of the input through the utilisation of mathematical functions or hash functions. (GeeksforGeeks, What is Hashing?, 2024)

Determines the data structure of an item by searching a location in storage determined by a given item within the same data structure. (GeeksforGeeks, What is Hashing?, 2024)

We will notice that programmatically hashing algorithms often utilise this process through the data structure objective achieved using arrays like the above example.

Arrays are the go-to data structure in programming for strictly keeping up with the vast volume of data transmitted across the Internet. (GeeksforGeeks, What is Hashing?, 2024)

However, the ability to conveniently and quickly access data is why array data storage structures are the popular choice for hashing algorithms. (GeeksforGeeks, What is Hashing?, 2024)

(GeeksforGeeks, 2024)

## **Components Of Hashing**

Believe it or not, hashing algorithms commonly possess these three vital components that are included in all the hashing algorithms that we will encounter that make hashing algorithms function. (GeeksforGeeks, What is Hashing?, 2024)

These are the following components stated below. (GeeksforGeeks, What is Hashing?, 2024)

The first component will commonly be the key, a string or integer of some kind, which is taken in as input into the hashing function. This method often finds the index to store an item with the overall data structure. (GeeksforGeeks, What is Hashing?, 2024)

The second component is a hash function, which is interesting as it intercepts the input key that will finalise and return the location of an element with our array, which we call the hash table.

The index, in this case, is referred to as the hash index. (GeeksforGeeks, What is Hashing?, 2024)

The last component is the hashing table, referring to the data structure responsible for mapping keys to their corresponding values, all handled by the hash function. (GeeksforGeeks, What is Hashing?, 2024)

Diagram of a diagram of a function

Description automatically generatedThese hashes store the data in a narrow order. (GeeksforGeeks, What is Hashing?, 2024)

(GeeksforGeeks, 2024)

## **Wrap it up**

The above hope successfully showed how these hashing algorithms adhere to the essential parts that define a hashing algorithm.

However, as stated, we will be utilising tools as malware researchers.

Besides, the algorithms we will utilise frequently are MD5 hash and SHA Hashing, typically 256-bit, with the lowest at 128-bit and the higher-end as high as 512-bit.

Because we have mentioned the typical use before in some of the previous chapters highlighting the primary use case that we are concerned about in malware analysis, it would be interesting to highlight only the primary but practical to have an insight into it on a basic programmatical level let us move on to talking about Signatures or the fingerprinting process.

# **Signatures**

As we mentioned in the first chapter, "Introduction to Malware Analysis," we mentioned that Malware Analysis utilises Static Analysis when we want to analyse the assembly instructions, looking for the functions and sections containing elements of the overall code structure of a file with the blueprint we call Portable Executables that are utilised by Dynamic Link Libraries and Executable Files. (Honig M. S., Malware Analysis Primer, 2012)

(Mentioned before in the "Static Analysis" chapter) We mention all these in the "Static Analysis" chapter, including linking libraries, functions, ELF and much more. (Honig M. S., Malware Analysis Primer, 2012)

Remember, in Dynamic Analysis, we are running the malware and utilising specific tools to check processes, services, files, and registry keys, using a tool like Procmon to observe a malware sample's purpose, origin, and functionality. (Honig M. S., Malware Analysis Primer, 2012)

## **When Analysing Malware**

We will start analysing a suspected binary, which helps us detect the network, measure it, and ultimately contain it to prevent further damage. (Honig M. S., Malware Analysis Primer, 2012)

From here, we can identify the files that require extensive examination, leading to the development of host-based and network signatures that we will define below. (Honig M. S., Malware Analysis Primer, 2012)

## **Host-based Signatures**

Here, we focus on indicators that reveal malicious code on a victim's computer, starting with the critical threats such as searching for modification and creation of each of the files and specific alterations to the registry, as all possible changes correlated by the presence of malware, these will differ and highlight the how and what the malware can do on the system, this is what makes this technique adaptable, reliable, and a fluid approach. (Honig M. S., Malware Analysis Primer, 2012)

Host-based signatures are desirable when dealing with malware that transfigures and valuable when dealing with malware that is erased from the hard drive. (Honig M. S., Malware Analysis Primer, 2012)

## **Network Signatures**

We specialise in monitoring network traffic to identify the presence of malicious code. (Honig M. S., Malware Analysis Primer, 2012)

We know that network signatures can occasionally be crafted without needing malware analysis. (Honig M. S., Malware Analysis Primer, 2012)

Even though malware analysis will often aid in creating signatures, it is generally very effective in enhancing the detection rate, typically resulting in fewer false positives. (Honig M. S., Malware Analysis Primer, 2012)

## **Wrap it up**

So here are the process and criteria of malware signatures that are very useful for determining the malware group and type we are dealing with by the method of which determining the migration can be achieved through the capturing of these host-based signatures and, finally, when dynamically analysing, we can determine the even the network signature of malware samples.

Now that we have clarified and gone into signatures let us navigate to the file entropy chapter in the next chapter.

# **File Entropy**

Now we will start by determining what we mean when we say File Entropy, with some exciting insight not just providing a background into File Entropy, the man, the myth, the legend behind the recommendation utilised in digital compression known as Shannon Entropy developed by Claude E. Shannon derived from the study of thermodynamics reapplied to the world of cryptography.(Hartman, 2013)

File Entropy is the metric utilised to determine the randomness or disorder of a digital file.

When determining the distinction between an encrypted file and a file that is determined to be genuinely random with greater than fifty per cent probability, it is said that the threat actor has the upper hand.(Hartman, 2013)

A threat actor could exploit this advantage and potentially break the encryption. (Hartman, 2013)

The upper hand derives from the concept that applies to the mathematical testing of encryption algorithms. (Hartman, 2013)

However, applications in the real world, such as files containing randomise, have no value being utilised in a file system.(Hartman, 2013)

Therefore, it is highly probable that files with high entropy are encrypted or compressed.(Hartman, 2013)

When Claude .E. Shannon paved the way for digital communications, Shannon was determined to figure out the theoretical limit for a digital file, attempting to push the boundaries of the maximum amount of digital compression that a given digital file can be pushed. (Hartman, 2013)

Shannon Entropy works as a mathematical statistical probability formula that states the following. (community, 2024)

If only probabilities are given (pk), then given the Shannon Entropy calculation states that "H = -sum(pk \* log(pk))". (community, 2024)

Next, it determines that if relative entropy is indeed computed, then it should present in near identical format to (pk) known as discrete distribution, along with each axis slicing of (pk) element (i) is the possibly un-normalised probability of an event (i). (community, 2024)

Relative entropy equals "D = sum(pk \* log(pk / qk))". (community, 2024)

The quantity identified is defined as "Kullback-Leibler divergence". (community, 2024)

If all else fails, this condition readjusts and equalises both "(pk) (qk), only if they do not sum to 1". (community, 2024)

## **Packed Files**

What we mean by packing is concealing the contents of files by Malware Authors to prevent their malware or malicious files from being successfully analysed by taking advantage of the mechanism referred to as compression, usually utilising some data compression algorithm. (Honig M. S., Packers And Unpacking, 2012)

These are achieved through the use of packer programs that may utilise encryption on the executable contained within it, achieved through implementation of a combination of ant-reverse-engineering, anti-disassembly, ant-debugging and even anti-VM all examples of evasion techniques that a Malware Author may employ to make their packer program more difficult to breakdown and study its contents on the binary and assembly instructional, we expect to find to in the anatomy of Packer program or packer files. (Honig M. S., Packers And Unpacking, 2012)

The following section will review the reconstruction of a packer program's content and structure for study to find the import table and reveal the functions contained within the obfuscated file. So, the goal is to reconstruct the import section or table to analyse the functions and libraries contained within it, which can reveal the program's import information. (Honig M. S., Packers And Unpacking, 2012)

## **The Unpacking Stub**

Before discussing the process, we go through what is required to enable us to statically read a packed file, starting with the Unpacking Stub, the nonpacked executables that the Operating System runs. (Honig M. S., Packers And Unpacking, 2012)

When dealing with packed programs, it is crucial to highlight The Unpacking Stub that the Operating System runs, The Unpacking Stub running the initial program, and the Code entry point for our executable points to the unpacked stub rather than the original code. (Honig M. S., Packers And Unpacking, 2012)

The original program will be typically stored in either one or more extra sections within the file. A malware analyst can view the unpacked stub to piece together the different parts of it to provide clarification, which is essential when unpacking the executable. (Honig M. S., Packers And Unpacking, 2012)

An exciting thing to note is that the unpacked stub is usually small because it does not supply anything to the main operational program. Its purpose is typically to unpack the initial executable. (Honig M. S., Packers And Unpacking, 2012)

If attempting this by performing static analysis on the packed program, we usually encounter that we are investigating the stub and not the initial program. (Honig M. S., Packers And Unpacking, 2012)

A diagram of a packing algorithm

Description automatically generated

(Infosec, Packed-Malware1.jpg, 2020)

## **Unpacked Files**

A diagram of a unpacking process

Description automatically generatedBelow is a way to visually illustrate the process and adequately highlight how unpacking is undertaken. (Honig M. S., Packers And Unpacking, 2012)

(Infosec, Packed-Malware2.jpg, 2020)

**There are three main ways of tackling the unpacking process listed below** (Honig M. S., Packers And Unpacking, 2012)

1. The first and most used technique is instructing The Unpacked Stub to be important to use the functions "GetProcAddress", "LoadLibrary*",* and no other imported functions. (Honig M. S., Packers And Unpacking, 2012)

Once The Unpacked Stub unpacks the initial executable. (Honig M. S., Packers And Unpacking, 2012)

After this reads the initial import information, The Unpacked Stub will call "LoadLibrary" (Honig M. S., Packers And Unpacking, 2012) individually for every library before we can run the DLL into memory. (Honig M. S., Packers And Unpacking, 2012)

Then, we can now utilise "GetProcAddress*"* to retrieve the address for every function. (Honig M. S., Packers And Unpacking, 2012)

1. The stealthiest approach is to store only one import function from every DLL we encounter, which is reserved in the initial import table. (Honig M. S., Packers And Unpacking, 2012)

We will uncover only one method from a given imported library during the investigation.

An investigation will still unveil all imported libraries. (Honig M. S., Packers And Unpacking, 2012)

It is not just a more straightforward approach and implementation from the packer's point of view than our previous approach. (Honig M. S., Packers And Unpacking, 2012)

The libraries do not need to run the Unpacked Stub. (Honig M. S., Packers And Unpacking, 2012)

Nevertheless, there is still a requirement from The Unpacked Stub to finalise a large portion of the functions. (Honig M. S., Packers And Unpacking, 2012)

1. The last technique is a more straightforward method that keeps the initial import table altogether to leverage the Windows loader to run the DLLs along with the functions. (Honig M. S., Packers And Unpacking, 2012)

We can see why this approach is the most straightforward and laziest as we do not even require the use of the Unpacking Stub for unveiling the imports as well as when statically investigating the packer program to disclose all the initial imported functions as these functions will be stored in clear text within the executable. (Honig M. S., Packers And Unpacking, 2012)

Compression can be technically achieved, but it is still highly discouraged when using this technique. (Honig M. S., Packers And Unpacking, 2012)

Now that we have a good understanding of File Entropy, packed files and how we go about unpacking a file or program, we are going to move on to the next chapter, where we will see the prime candidates that perform either similarly known or used tools by Malware Analysts such as PE Explorer and Dependency Walker tools come to mind but let us see some exciting libraries, GitHub projects containing a variety of different tools and toolkits and some have the existence of both types mentioned here. (Honig M. S., Packers And Unpacking, 2012)

# **Similar Products / Tools**

Start by introducing each of the similar products and tools we discovered while doing our research, starting with the Yara tool. (Revision, 2022)

Yara is a tool aimed at but not exclusively at helping malware researchers identify and classify malware samples. (Revision, 2022)

Yara utilises creating signatures or fingerprinting of certain malware families by creating descriptions. (Revision, 2022)

Descriptions in Yara are the basis of textual and binary patterns. (Revision, 2022)

Each description may consist of descriptions or are known in Yara as rules, the set of strings and Boolean expressions that determine the logic for Yara's classification system. (Revision, 2022)

Yara Rules are robust and can be complicated, utilising a combination of wildcards, case-insensitive strings, regular expressions, special operators, and a wide range of other features that show a taste of the possible capabilities of Yara. (Revision, 2022)

We will move on to the following tool: Pyew, a Python tool for analysing malware. (joxeankoret, 2024)

Pyew's capabilities include its support for hexadecimal viewing and disassembly of Portable Executables and Executable and Linked File format. (joxeankoret, 2024)

Pyev performs code analysis and lets us write scripts utilising an API to perform different types of analysis. (joxeankoret, 2024)

Finally, onto our last tool, Scapy, is a sophisticated networking Swiss army knife with a wide range of capabilities like tools like tcpdump, tshark, or even Nmap. (Philippe Biondi & Scapy Community, 2024)

Scapy has the unique capability of enabling complete control over the intercepted network data. (Philippe Biondi & Scapy Community, 2024)

It easily integrates with little issues in case additional capabilities are incorporated into the tool without syntaxial limitations. (Philippe Biondi & Scapy Community, 2024)

Nevertheless, adding features and functionality where the only limitation is that we cannot breach the tool's scope, such as if our tool is a port scanner, cannot and should be able to start performing any or all the shared network attacks. (Philippe Biondi & Scapy Community, 2024)

# **Guided Walkthrough**

Here in this chapter, like many things in life, the only way to give anyone a feel for what the process of static analysis might initially be from at least a theoretical point of view.

We can go into detail about the specificities and any nitty-gritty details of Static analysis, but why talk about Static Analysis?

It is better to show a practical example from a real-world example of a famous ransomware program called the WannaCry Ransomware Program from the Practical Malware Analysis Lab.

It can be found through a GitHub project or search in any web browser.

We are looking at Lab 10, which we will find in the follower once we have uncompressed the zip file.

Before we do anything, we will make sure to turn off any antivirus programs and firewalls temporarily.

Otherwise, it will flag the compressed file as malicious and prevent us from downloading it to our local machines.

Another thing is that we are probably best to ensure we have set the FlareVM Malware lab in a Windows 10 Virtual Machine to ensure it is isolated from the outside world after we ensure the internet connection is disabled, as we do not want the malware to spread as this will happen if we are conducting dynamic analysis.

However, we should do it if, strictly speaking, we can use either Windows or Linux.

Both will work on solving this problem. This lab might have different approaches because it depends on the tools utilised in these operating systems.

Nevertheless, in static analysis, as we are statically investigating and examining our malware sample within a tool, we do not need to worry about the connection to the Internet being disabled.

However, it is no harm to do it at the same time.

Let us start by opening our FlareVM environment and navigating to the Lab 10 folder within the Binary Collections folder from our extracted folder, Practical Malware Analysis Labs.

From there, looking the Lab10.3.exe file and make sure to right-click, and we should then be able to see the below.

A screenshot of a computer

Description automatically generated

We can also identify that in this lab 10-03.exe that we will examine next, we can see what appears to be a text file on our desktop.

Here is a text file that contains the hashes generated for this file.

A screenshot of a computer

Description automatically generatedIf it is useful when using the tool VirusTotal.

We can scan and see what threat-level information we can get back, but we do not strictly need to do so for this demonstration.

Here are the hashes generated, but strictly speaking, they should have changed the file extension if we plan to verify with VirusTotal that our hashes were not incorrectly generated.

Next, the objective of this lab is to find the kill switch based on the kill switch that was meant to stop the ransomware in real life.

Nevertheless, we want to select strings, where we will try to see if the kill switch will be a string type.

A screenshot of a computer

Description automatically generated

As we can see, we have a list of string literals and the memory addresses for each string above.

We want to apply the " http " criteria to identify URL-encoded strings, like the ASCII Strings above.

Now that we know what we are looking for, the moment of truth is that we should be able to find a match below.

A screenshot of a computer

Description automatically generated

Eureka! We have the kill switch, as highlighted in the above image.

The Kill Switch: "<http://www.malwareanalysisbook.com/ad.html>"

# **Python**

## **Pleading the case for Python**

We need to start by disclosing that Python yes is an example of a high-level programming language meaning it requires an interpreter as well as a complier to compile our code into binary or machine code due to how far away it is from machine code or the closer to system language, but that is all stuff you can look up but the thing about Python is it also the ideal programming language that was designed towards being beginner friendly, that is why a lot of beginners to programming, will start out likely being introduced to Python rather than, Java or C++ and how a lot of the functions, methods, logic is designed to be self-explanatory, and closer to English grammatical than any other scripting language you will come across, but do not let this language being beginner friendly fool you, this language is one of the most powerful among the other higher programming languages including both the ones we mentioned at the start and the ones we did not mention, there is a reason this language is used across multiple different fields and yes it is also a highly recommended programming language other than, C, C++ or even the versions of Assembly code you will encounter like x86 Assembly, it is the most sought after programming language which tells you a lot.

Another advantage is its endless libraries, including data science libraries and libraries for artificial intelligence, machine learning, computer science, software development, finance, and mathematics, as well as libraries that cover areas in the vast field of IT Security and Cyber Security.

## **Python (Malware Analysis) Libraries**

Some of the libraries proved very useful, especially those listed below.

**Note: When we get onto the "Angr" Chapter, I will introduce the binary analysis tool, an essential element for the malware report generation tool.**

1. **String Module** developed for Python for manipulating strings in a variety of different ways, from analysing the array of characters that make up a string to utilising prebuilt functions such as printable to ensure all characters are of ASCII format if we are looking for ASCII strings in particular or maybe essential functions like uppercasing, lowercasing strings and the casting data of one type to another quickly and many more functions that come part and parcel as part of this library.

However, it will help identify strings and list the types of strings found in a malicious file, such as URL encoded strings, Base64 encoded strings and decoded Base64 String. (Foundation, string — Common string operations, 2024)

1. **VirusTotal-Python** developed and made this Python library for calling the API straightforward and more manageable to process our requests, significantly implementing a rate limit to ensure we do not exceed the token limit or send more calls than we need for management reasons, which means that we still have to implement the basic code to integrate the calling of this API so fall VirusTotal's Instructions and usually best to follow the API documentation and this even applies for technologies or especially programming libraries and code. (Foundation, virustotal-python, 2024)
2. **Hashlib** is a library that is the go-to for interfacing, generating and manipulating cryptographic hashing algorithms.

We are more interested in the SHA and MD5 Hashing algorithms.

MD5 or Message Digest Algorithm Version 5, Secure Hashing Algorithm 128 bit, and the highly recommended SHA-256 cryptographic hashing algorithm.

We use this library to generate our file hashes to prove file integrity.

This library has a lot of functionality and methods to integrate the range of possibilities that almost seem endless, but it is an instrumental library. (Foundation, hashlib — Secure hashes and message digests, 2024)

1. **Urllib** is a package designed to fetch URLs and interfaces, handles and deals with URL encoded strings, so we can parse URLs, open and read, and create rules or exceptions to combat and debug any possible unexpected errors.

We will use this package, module, or library for our VirusTotal API function, which involves utilising many different networking protocols to send and receive requests for API calling and the capturing and possibly decoding URL encoded strings if the decoded URL strings provided are helpful. (GeekforGeeks, Python Urllib Module, 2021)

1. **Regex Expression or RegEx** is a package used to create, manipulate, and interface with regular expressions to develop search patterns using the sequence of characters. As we can see, this is another way to apply our search patterns to even strings. (Foundation, re — Regular expression operations, 2024)

Hence, this could be another way to apply and create search patterns when identifying and storing our ASCII, URL encoded, and base64 encoded strings with the aid of Angr, including its binary loading from a malicious file, working similarly to the String Module. (Foundation, re — Regular expression operations, 2024)

1. **Base64 Library** integrates, interfaces, manipulates, and performs functions such as encoding and decoding base64 encoded strings.

We can utilise this library in the malware report generation tool along with Angr, Hashlib, Urllib, and String or RegEx. (Foundation, base64 — Base16, Base32, Base64, Base85 Data Encodings, 2024)

1. **NumPy Module** is a primary package in Python that is utilised for its scientific computing that uses a large set of multi-dimensional array objects that contains an extensive collection of arrays using a combination of fast operations such as mathematical, logical, sorting, selecting and manipulating that utilised basic statistics, linear algebra and even the packaged is designed in such that through adding some already complied coding utilising the programming language C, that is programmed with vectorisation in mind this will enable us to have fewer bugs due to the consistency of the code.

However, adding far fewer lines of code means less chance of facing many bugs or difficulty in finding the bugs that do occur. (team N. , 2024)

1. **SciPy Module** is a library that houses all the statistical and mathematical algorithms, some of which are our file entropy from the statuses within the statistics function.

This library is built on top of the NumPy library, so we must import both when coding and instructing the calculation programming for the file entropy element. (community, 2024)

1. **System Module (Sys)** is a module for modifying, manipulating, and intercepting data and directly interfacing with the Python interpreter using this library. (GeekforGeeks, Python sys Module, 2023)
2. **Maths Module** is built into one of Python's many default libraries.

It is designed to provide ease by allowing us to utilise its various mathematical calculations gauged towards basic examples of each mathematical function, everything from basic arithmetic such as addition, subtraction, multiplication and division. (GeeksforGeeks, Python Math Module, 2023)

1. **OS Module** is one of the default libraries used in Python, without which we would be unable to interact with handling files on our system through Python.

Other exciting operating system information could be displayed, but we will primarily use this for modification. (GeekforGeeks, Python sys Module, 2023)

1. **Matplotlib Module** is an extensive library that houses static, animation, and visualisation in Python.

This library is typically used for creating and plotting graphs for visualising metrics data, which helps provide either a visual flare to a Python project or, in our case, utilising it as a benchmark. (team T. M., 2024)

1. **Markdown module** is a Python implementation of John Gruber's GitHub Markdown Project. It is almost entirely compliant with the reference implementation.

However, a few minor known issues encountered when implementing the Continuous Integration Pipeline package are the go-to for generating the same structure that similar markdown templates implement, such as html, which is known as a markdown language. (Foundation, Markdown 3.7, 2024)

1. **The markdown-pdf module** is a Python open-source module that enables the creation of PDF files converted and generated from a given markdown format.

This module accomplishes this because it utilises a range of functions from two Python libraries, including markdown-it-py, for conversion from markdown format to HTML web page.

The last library is PyMuPDF, which converts HTML templates or web pages to PDF formats. (Foundation, markdown-pdf 1.3, 2024)

# **Angr (Open-Source Binary Analysis Toolkit)**

To start this, we are going to introduce this Binary Investigation toolkit, which is an open-source project that is surprisingly extensive and can look intimidating when examining its Angr Documentation Article that is presented as standard web pages, but as we will find the format and acting as a comprehensive online book.

Let us start by providing some background on Angr, as shown below.

## **What is Angr?**

Angr defines itself as a multi-constructive binary analysis toolkit, one of the few tools with a multi-architectural path due to its backing for popular binary architectures. (Stephens, 2016) (Shoshitaishvili Y. a., 2015) (Shoshitaishvili Y. a., 2016) (Gedam, Introduction - angr documentation, 2024)

We will encounter Microsoft Windows Environments, which Angr supports, one of two popular operating systems used for general purposes or environmentally aimed desktop systems. (Gedam, Introduction - angr documentation, 2024) (Stephens, 2016) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015)

These binary formats and files that we quickly find are the Executables going by the extension (.exe), also goes by Executable Files, Dynamic Link Libraries or known by the abbreviated extension DLLs and finally, Portable Executables or (PE) file format that serves as the template layout for both DLLs and EXEs in Windows. (Gedam, Introduction - angr documentation, 2024) (Stephens, 2016) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015)

Next, it has the backing of a Macintosh environment with the binary file format utilised known as MAC-O binary for Macintosh, which, if we do not already know, as we will see in the next chapter, 'Similar Products', we will begin to notice that allot of the GitHub tool projects we mentioned per our findings, do not support Macintosh binary file format incorporated into its static analysis investigations. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015) (Stephens, 2016)

Lastly, it has the backing of Executable and Link Format or ELF, the binary file format backed by many popularly and widely used Linux Distributions. (Gedam, Introduction - angr documentation, 2024) (Stephens, 2016) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015)

## **What is the function of Angr?**

Before highlighting Angr's primary desirable trait is the utilisation of its binary search algorithms, before highlighting some these lets talk about what we are required to as the first prerequisite that we will have to verify is complete, this is known as The Loader involving the following process of the process of adequately loading binaries within Angr, specifically loading a given binary loaded within the project meaning that objects in Angr like project and when investigating the binary code, we utilise the multiple strategies for analysing the assembly instructions, that relate to the locations of registries within memory all utilised by the binary loader, which is known by another name of the Concrete Loader Engine or by the acronym CLE , which allows Angr this capability of transforming or interrupting the both the binary code and the assembly instructions utilising the Abstract Syntax Tree or known by the acronym AST that enables the representation for symbolic expressions and constraints, that allows the manipulation of symbolic variables that provides the framework for the binary analysis technique of Symbolic Execution in order to achieve a flawless operation, this primarily serves as a dynamic symbolic execution combined with both reverse engineering and static analysis in terms of analysis strategies when tackling and analysing binary files. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2015) (Shoshitaishvili Y. a., 2016) (Stephens, 2016)

The other analysis, as they are known in Angr, is as follows: Simulation Managers ( that is responsible for enabling execution and simulation through easily interfacing with the states that enable access to sections, segments, registry information, memory address locations and finally, blocks that allow us to view disassembled code in units of basic blocks. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2015) (Shoshitaishvili Y. a., 2016) (Stephens, 2016)

The analysis technique known as CFG or Control Flow Graph (Fast) constructs a dynamic hierarchical diagram that represents the Control Flow Graph of our binary code, and the CFG Extensive analysis technique statically constructs a Control Flow Graph of our binary code. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2015) (Shoshitaishvili Y. a., 2016) (Stephens, 2016)

We will encounter another thing in Angr that, like the first binary search strategy we mentioned, will call the memory address of all utilising objects, whether using the factory object and its library of functions to analyse our binary file and reconstruct its code. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015) (Stephens, 2016)

Angr is very powerful. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015) (Stephens, 2016)

Even though it has a comprehensive guide to its toolkit and overall functionality, it is interesting to see the future goals for Angr and any additional features or capabilities to improve or enable Angr capability it does not already possess. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015) (Stephens, 2016)

Let us move on to the next chapter, which introduces the legendary and widely used Web Framework known as Flask. (Gedam, Core Concepts - angr documentation, 2024) (Shoshitaishvili Y. a., 2016) (Shoshitaishvili Y. a., 2015) (Stephens, 2016)

# **Flask (Web-Framework)**

This chapter starts with an introduction to the web framework Flask, discussing why we are utilising Flask, which will be a good framework for creating web applications to add an element to this project to enhance the capabilities and future potential further.

## **Pleading the case for Flask Web (Minimalistic) Framework**

Flask is used to create web applications using Python, which is imported and used in a Python programming environment with a command line interpreter such as pip or IPython. (GeeksforGeeks, 2024)

Flask has built-in capabilities for implementing and setting up a basic registration system, handling cookies, sessions, and more. (GeeksforGeeks, 2024)

It has many libraries like Django, but a lot of the libraries are straightforward, have clear instructions, and have an endless number of tutorials. (GeeksforGeeks, 2024)

These libraries include SQL Alchemy for creating and setting up an SQL database, a Jinga2 template engine for dynamic website creation rendering, and Bcrypt, which uses the Blow Fish Hashing algorithm for hashing passwords. (GeeksforGeeks, 2024)

We also include a broad set of different databases that can be used, such as a library exclusively for SQLite and another library utilised for MongoDB. (GeeksforGeeks, 2024)

The Flask Web Framework is built from Python and has an easier learner curve than popular web frameworks like Django. (GeeksforGeeks, 2024)

Due to how less complex and lightweight Flask is with its minimalistic approach, this makes it the stronger candidate. (GeeksforGeeks, 2024)

Even with the strong incorporation of IT security and its limitless potential of Django, Flask is still the ideal candidate for the malware report generation tool we are currently planning and constructing and what dedicates the better candidate the use case for the web framework which in our case it is not currently best seeking Django over Flask for at least what we want to utilise it for as part of our overall project. (GeeksforGeeks, 2024)

The target audience is gauged as beginner web developers or, at minimum, having a solid foundational grasp of HTML or Hyper Text Makeup Language, which is the essential backbone providing a basic layout that all webpages utilise for rendering in any web browser. (GeeksforGeeks, 2024)

As well as familiarity with Python, which we covered in the previous chapter "Python". (GeeksforGeeks, 2024)

Another thing to note is a familiarity with JSON Web Token and JSON requests for user authentication, which includes basic web protocols such as 80 or 443 and the default most used services and the ports utilised, and knowledge of a primary database hosting for storing our data. (GeeksforGeeks, 2024)

Now, we want to use it not only for Flask user authentication but also for Flask integration with extensions such as flask-sessions, flask-security, and flask-login, flash WT Forms for streaming and efficiency increasing for management of web forms, and finally, even flask-mail that utilises the simple mail transfer protocol for the handling of emails over the Internet. (GeeksforGeeks, 2024) (GeeksforGeeks, 2024)

Nevertheless, as we can notice, Flask, as said before, is designed to be customisable, meaning it does not have all these features like Django's ready out-of-box libraries or modules already built-in, which is why this has a steep learning curve when compared to Flask as Flask is designed for smaller or medium-sized projects does not mean it cannot be used in large projects but explains why with Django we need to understand it is at first glance intimidating architecture before we are even able to start developing projects which is why it is used typically for larger projects. (GeeksforGeeks, 2024)

It is time to wrap it all up.

Unfortunately, as the next chapter suggests, we have finally come to the end of this report.

In conclusion, we will finalise and summarise, hopefully clarifying the tools, libraries, and concepts we have finalised.

Determining the tools we are incorporating into our Malware Analysis Report Generation product resulting from our Malware Analysis Report Generation Project.

# **Conclusion**

We will finalise what we have learned through this document and determine the tools we will use in the final product, along with some libraries and tools we will utilise in the Malware Analyse Report Generation Tool.

We should go with Angr over Pyev or even IDA Python for our core project backbone because Angr is open source and has many capabilities are some of the reasons why it is prime target.

For the libraries and coding language we are going to implement, Python, and the libraries we plan to implement are the String Module, Urllib, virus-total-python, Hashlib, NumPy, SciPy, Base64, System, Maths, OS, Markdown, Markdown-to-pdf, and Matplotlib module.

These are to perform the File Entropy, string extraction, the filename and filetype, file hashing, determining if the file is likely packed or not based on the threshold or the lower or higher the score is below the given threshold for encrypted or packed or compressed to predict the likelihood of a given file being obfuscated or not.

We will communicate with the VirusTotal API, utilising the virus-total-python module to accomplish this task.

We are utilising the SciPy and NumPy modules to accomplish this task for the file entropy.

Hashing will be accomplished by using the Hashlib Module.

We will also integrate Flask, and some of the libraries mentioned will be incorporated, such as SQL Alchemy, to create our database.

Finally, we will utilise it to build a malware classifier.

We will leverage a tool called YARA that can be imported and integrates well with Python, enabling us to set up our own classification rules for flagging different categories of malware we may encounter.

# **References**

CNET. (2012, November 20). *PE Viewer for Windows - free download and software reviews - CNET Download*. doi:3000-2352\_4-10966763

CNET. (2013, August 9). *PEBrowse Professional (64-bit) for Windows - Free download and software reviews - CNET Download*, 4.0. Retrieved November 10, 2024, from CNET: https://download.cnet.com/pebrowse-professional-64-bit/3000-2218\_4-75176627.html

community, T. S. (2024, January 1). *entropy — SciPy v1.14.1 Manual*, 1.14.1. (T. S. community, Editor) Retrieved November 10, 2024, from SciPy document: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html

Foundation, P. S. (2024, November 9). *base64 — Base16, Base32, Base64, Base85 Data Encodings*. Retrieved November 11, 2024, from Python documentation: https://docs.python.org/3/library/base64.html

Foundation, P. S. (2024, November 9). *hashlib — Secure hashes and message digests*, 3.13.0. (P. S. Foundation, Editor) Retrieved November 10, 2024, from Python documentation: https://docs.python.org/3/library/hashlib.html

Foundation, P. S. (2024, August 16). *Markdown 3.7*, 3.7. (P. S. Foundation, Editor) Retrieved November 7, 2024, from PyPi: https://pypi.org/project/Markdown/

Foundation, P. S. (2024, June 11). *markdown-pdf 1.3*, 1.3. (P. S. Foundation, Editor) Retrieved November 7, 2024, from PyPi: https://pypi.org/project/markdown-pdf/

Foundation, P. S. (2024, November 9). *re — Regular expression operations*, 3.13.0. (P. S. 2., Editor) Retrieved November 10, 2024, from Python documentation: https://docs.python.org/3/library/re.html

Foundation, P. S. (2024, November 9). *string — Common string operations*, 3.13.0. Retrieved November 11, 2024, from Python documentation: https://docs.python.org/3/library/string.html

Foundation, P. S. (2024, May 26). *virustotal-python*, 1.1.0. (P. S. Foundation, Editor) Retrieved November 10, 2024, from PyPI: https://pypi.org/project/virustotal-python/

Gedam, P. a. (2024, January 1). *Core Concepts - angr documentation*. (P. Gedam, Editor) Retrieved November 10, 2024, from Angr documentation: https://docs.angr.io/en/latest/core-concepts/toplevel.html

Gedam, P. a. (2024, January 1). *Introduction - angr documentation*. (P. Gedam, Editor) Retrieved November 10, 2024, from Angr Documentation: https://docs.angr.io/en/latest/quickstart.html#citing-angr

GeekforGeeks. (2021, October 13). *Python Urllib Module*. Retrieved November 11, 2024, from GeekforGeeks: https://www.geeksforgeeks.org/python-urllib-module/

GeekforGeeks. (2023, November 18). *Python sys Module*. Retrieved November 11, 2024, from GeekforGeeks: https://www.geeksforgeeks.org/python-sys-module/

GeeksforGeeks. (2023, December 21). *Python Math Module*. Retrieved November 11, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/python-math-module/

GeeksforGeeks. (2024, February 24). ComponentsofHashing-660x342.png. Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305). Retrieved November 10, 2024, from https://media.geeksforgeeks.org/wp-content/uploads/20220701080941/ComponentsofHashing-660x342.png

GeeksforGeeks. (2024, October 5). *Flask Tutorial*. Retrieved November 10, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/flask-tutorial/

GeeksforGeeks. (2024, February 26). HashingDataStructure-min.png. Sovereign Corporate Tower, Sector- 136,, Noida, Uttar Pradesh (201305). doi:/20200609180838/

GeeksforGeeks. (2024, August 1). *OS Module in Python with Examples*. Retrieved November 11, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/os-module-python-examples/

GeeksforGeeks. (2024, February 26). *What is Hashing?* Retrieved November 08, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/what-is-hashing/

GitBook. (2024, October 10). *How to create a plugin in IDAPython?* Retrieved November 10, 2024, from hex-ray / Docs: https://docs.hex-rays.com/developer-guide/idapython/idapython-getting-started

Hartman, K. G. (2013, May 18). Calculate File Entropy. Retrieved November 10, 2024, from https://kennethghartman.com/blog/calculate-file-entropy/

Hiddenillusion H. (2024, January 1). *GitHub - hiddenillusion/AnalyzePE: Wraps around various tools and provides some additional checks/information to produce acentralised report of a PE file.* Retrieved November 10, 2024, from GitHub: https://github.com/hiddenillusion/AnalyzePE

Honig, M. S. (2012). Basic Static Techniques. In S. Honig, *Practical Malware Analysis* (pp. 13-15). San Francisco, California , America: William Pollock. Retrieved November 11, 2024

Honig, M. S. (2012). Malware Analysis Primer. In M. S. Honig, *Practical Malware Analysis* (pp. 1-2). San Francisco, California, America: William Pollock. Retrieved November 11, 2024

Honig, M. S. (2012). Malware Behaviour. In S. Honig, *Practical Malware Analysis* (pp. 231-235). San Francisco, California , America: William Pollock. Retrieved November 10, 2024

Honig, M. S. (2012). Packers And Unpacking. In S. Honig, *Practical Malware Analysis* (pp. 383-385). San Francisco, California , America: William Pollock.

Infosec, A. (2020, January 8). Packed-Malware1.jpg. Tirumala Shivpuri Colony, Yapral, Secunderabad. doi:5000062

Infosec, A. (2020, January 08). Packed-Malware2.jpg. Tirumala Shivpuri Colony, Yapral, Secunderabad. Retrieved November 09, 2024, from https://www.arridae.com/assets/img/blog-img/Packed-Malware2.jpg

Johnson, A. (2023, November 19). *Resource Hacker*, Version 5.2.7. Retrieved November 08, 2024, from Angus Johnson Website: https://www.angusj.com/resourcehacker/

joxeankoret. (2024, January 1). *Malware Analysis*. Retrieved November 7, 2024, from GitHub: https://github.com/joxeankoret/pyew/wiki/MalwareAnalysis

Mattick, L. L. (2023, July 18). *Intro - Putting the "You" in CPU*. (H. Club, Editor) Retrieved November 10, 2024, from Putting the "You" in CPU: https://cpu.land/

Philippe Biondi & Scapy Community. (2024, January 1). *Introduction — Scapy 2.6.1 documentation*, 2.6.1. (P. Biondi, Editor) Retrieved November 10, 2024, from Scapy Read the Doc: https://scapy.readthedocs.io/en/stable/introduction.html

Revision, V. (Ed.). (2022, January 1). *Welcome to YARA's documentation! — yara 4.4.0 documentation*, 4.4.0. Retrieved November 10, 2024, from Yara Read the Doc: https://yara.readthedocs.io/en/stable/index.html

Sarang S. Babu. (2023, May 7). *Using Python for Malware Analysis — A Beginners Guide*. (I. Write-ups, Editor) Retrieved November 07, 2024, from Medium: https://infosecwriteups.com/using-python-for-malware-analysis-a-beginners-guide-8432377df2c4

Security, R. (2023, January 1). Malware analysis image. Viale Marco Polo,11900154, Roma, Italy. Retrieved November 7, 2023, from https://www.redfrogsecurity.it/red-site/wp-content/uploads/2023/11/Malware\_Analysis.jpg

Shoshitaishvili, Y. a. (2015, January 1). Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. *NDSS*. Retrieved November 10, 2024

Shoshitaishvili, Y. a. (2016, January 1). SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. *IEEE Symposium on Security and Privacy*. Retrieved November 10, 2024

SOFTPEDIA. (2018, April 24). *PEID 0.95 - Download, review, screenshots*, 0.95. Retrieved November 9, 2024, from SOFTPEDIA: https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

Software, C. (2024, January 1). *Dependency Walker 2.2.6000.9*, 2.2.6000.9. Retrieved November 10, 2024, from Chocolatey Software: https://community.chocolatey.org/packages/dependencywalker

Software, H. (2024, January 1). *PE.Explorer. Resource Editor. Disassembler.* Retrieved November 8, 2024, from Heaventools Software: https://www.pe-explorer.com/

Stephens, N. a. (2016, January 1). Driller: Augmenting Fuzzing Through Selective Symbolic Execution. *NDSS*. Retrieved November 10, 2024

Team, C. G. (2024, January 1). *GitHub - CyberGrandChallenge/samples: DARPA Cyber Grand Challenge Sample Challenges*. (T. Vidas, Editor) Retrieved November 10, 2024, from GitHub: https://github.com/CyberGrandChallenge/samples

team, N. (2024, August 18). *NumPy*, 2.1. (N. team, Editor) Retrieved November 10, 2024, from NumPy: https://numpy.org/

team, T. M. (2024, January 1). *Matplotlib —Visualisation with Python*. (T. M. team, Editor) Retrieved November 10, 2024, from Matplotlib: https://matplotlib.org/