Research Report

Maldive
Malware Analysis Tool

Shane Doherty C00249279

Supervisor: Joseph Kehoe

South East Technological University

Started 17/10/22

Abstract

The threat that comes with malware continues to evolve as the security to prevent it moves alongside. For this to happen, new exploits in the underlying system must be investigated and understood. The goal of this project is to create a program that can effectively detect and analyse malicious programs. The method to which this will be performed is dynamic malware analysis, which extracts the executable code from a program. The malicious code will be executed, and will examine the steps that the program runs through in its process.

Through dynamic malware analysis, the program will be able to identify potential malicious activity and extract valuable information such as network traffic, memory values, and the instruction set used by the program.

Table of Contents

1. Introduction **4**
2. Malware **4**
2.1. Spyware 4
2.2. Ransomware 5
2.3. Scareware 6
3. Malware Analysis **7**
4. Portable Executable **7**
4.1. File version 8
5. Java Native Interface **9**
6. Virtual Memory **9**
6.1. Process Access Rights 10
6.2. Page Protection 11
6.3. Memory Basic Information 13
6.4. Querying 13
6.5. Read Memory 13
7. Dynamic Malware Analysis **14**
7.1. Analysis Techniques 14
7.1.1. Function call monitoring 15
7.1.2. Information flow tracking 15
7.2. Evasion 15
7.3. Analysis Environment 16
8. Existing Tools **16**
8.1. Wireshark 16
8.2. Regshot 17
8.3. EDB 17
9. Libraries **18**
9.1. Capstone 18
10. Operating System **19**
10.1. Linux 19
10.2. Windows 19
11. Conclusion **19**
12. References **20**
13. Additional Materials **21**

Table of Figures

Figure 1 Ransomware in operation (Akamai, n.d.) 6
Figure 2 Screenshot of WannaCry (Wikipedia, 2023) 7
Figure 3 Hex viewer of .exe file 8
Figure 4 Text views of .exe files 9
Figure 5 Generated .h file 10
Figure 6 Mapping virtual memory to physical memory (Abraham Silberschatz, n.d.) 11
Figure 7 Process Access Rights (Microsoft, n.d.) 12
Figure 8 Opening process with access right 12
Figure 9 Memory Protection Constants (Microsoft, n.d.) 13
Figure 10 Changing protection with protection constant 13
Figure 11 MEMORY_BASIC_INFORMATION structure (Microsoft, n.d.) 14
Figure 12 Query Function 14
Figure 13 Reading Process Memory 15
Figure 14 Intel Protection Ring (Wikipedia, 2022) 16
Figure 15 Wireshark GUI 18
Figure 16 Evan's Debugger (Teran, n.d.) 19

1. Introduction

This report contains documented research into dynamic malware analysis, with methods to analyse a program, along with methods that malicious code use to obscure and evade analysis. Given a thorough definition of different types of malware that the application is expected to handle, it should be clearly outlined what information is necessary to understand when going forward with the project.

2. Malware

Malicious software (malware) is the blanket term given to a program designed with nefarious intentions. This term is used to describe any type of malicious code, although there is a wide variety of ways this is done. Malware can be designed to steal personal or sensitive information, disrupt normal system operations, or allow unauthorised access to a system or network. Malware can be introduced to a system through various methods such as email attachments, downloads from untrusted sources, or infected software updates.

Once a system is infected, malware can carry out a variety of malicious activities including but not limited to, data theft, system hijacking, keylogging, ransomware attacks, and denial of service attacks. Malware can be designed to be stealthy and go undetected for long periods of time, making it difficult to identify and remove from a system.

The goal of this project is to provide tools to effectively scan a piece of software to determine if it is collecting sensitive information or otherwise being used to harm the users device.

2.1. Spyware

Spyware is a specific type of malware designed to gather information from a user's computer or device without their knowledge or consent. It can collect sensitive information such as browsing habits, keystrokes, passwords, and personal information that can be used for malicious purposes such as identity theft, fraud, and blackmail. This type of malware will attempt to steal sensitive information from the victim’s computer. The ways this is done can differ, with one method being using a keylogger to track which keys are being pressed, and sending the log to a server controlled by the attacker. As the keylogger can be enabled on the entire device, it does not need to hook onto a single process to be effective.

Spyware is often introduced to a system through downloads from untrusted sources, email attachments, or by exploiting vulnerabilities in software or operating systems. It can also be bundled with other software, often disguised as a legitimate program, and installed unknowingly by the user.

Once installed, spyware can run silently in the background, monitoring the user's activity and sending information back to a remote server. Some spyware can even take control of a

user's webcam or microphone, allowing for unauthorised monitoring of their physical surroundings.

2.2. Ransomware

Ransomware can be one of the most disastrous types of malware, considering attackers usually target high-value companies. The goal of ransomware is to encrypt vital data on a victim's system, then demand a payment for the encryption to be removed. As the encryption requires a key to be unencrypted, this is generally what the attacker will claim to trade for the ransom.

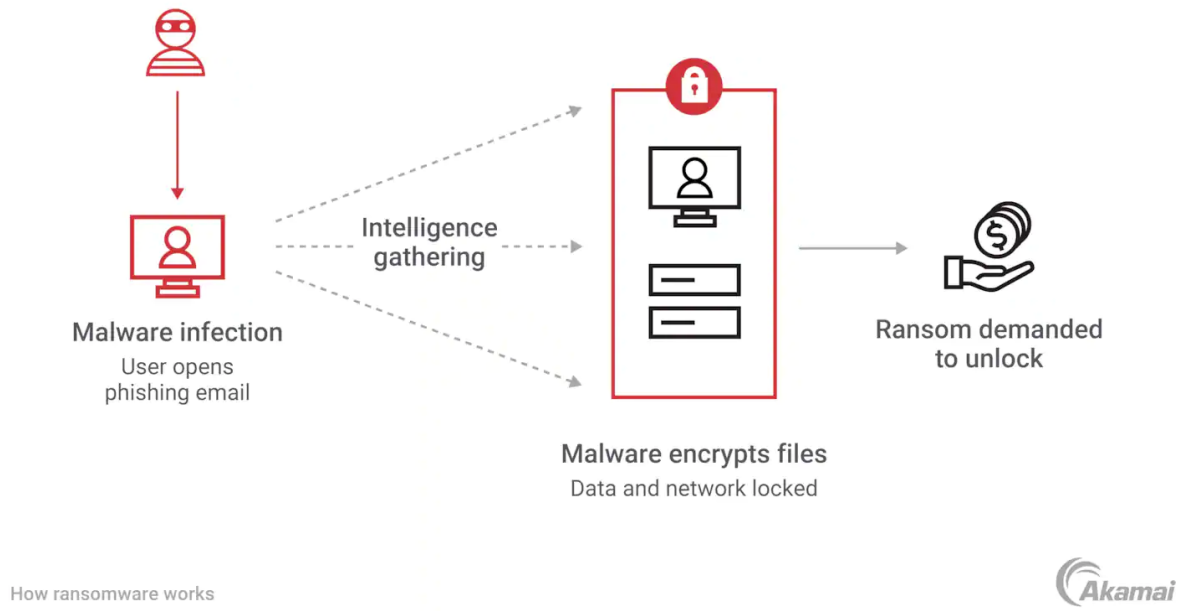


Figure 1 Ransomware in operation (Akamai, n.d.)

Ransomware can be spread through a variety of methods, including phishing emails, malicious downloads, and exploit kits. Once a system is infected, the ransomware will quickly begin to encrypt files and data on the victim's system, making them inaccessible to the user. The ransomware will then typically display a message demanding payment in exchange for the decryption key needed to unlock the encrypted files.

Attackers often demand payment of some form, most commonly in cryptocurrency such as Bitcoin, as it allows for anonymous transactions that are difficult to trace. However, even if a victim pays the ransom, there is no guarantee that the attacker will provide the decryption key needed to unlock the encrypted files. In some cases, even after payment has been made, the attacker may demand additional payments or refuse to provide the decryption key at all.

The impact of a successful ransomware attack can be severe, with potential consequences including the loss of critical data, downtime for businesses, and damage to reputation and customer trust. This makes ransomware a highly dangerous form of malware when targeting large companies in a cyber attack.

A popular example of ransomware is the *WannaCry* software, which encrypted many files on the victim's computer, demanding payment within a certain timeframe to retrieve a decryption key (Wikipedia, 2023). The timeframe was a short period, which was done to pressure the victim into sending the payment before analysis of the program could be completed. Should the payment not be sent in this period, the files were permanently locked and unable to be decrypted. This particular attack was particularly noteworthy due to the scale of which it was distributed, with over 230,000 computers being infected in a single day. This was achieved by using an infected system to distribute the malware over the network, compromising any other computer that was connected to it. While the amount of devices that were infected were high, the associated Bitcoin wallet showed that only 327 paid the ransom, totalling a payment amount of 51.62396539 BTC (US$130,634.77).

Figure 2 Screenshot of WannaCry (Wikipedia, 2023)

2.3. Scareware

Scareware can be very difficult to analyse and detect, considering it does not directly cause harm to the system. Instead, it will be disguised as a program such as anti-malware, then give false statistics to how infected a system is. The scareware will then ask for a payment for full access to the program, and to repair the "infected" system. As opposed to traditional malware, this method relies on social engineering to steal from the victim.

Scareware typically preys on a user's fear and lack of technical knowledge. It is designed to trick users into believing their system is infected with malware or other security threats, in order to convince them to pay for unnecessary or fake security software.

Scareware can be introduced to a system through a variety of methods, including pop-up ads, phishing emails, or by exploiting vulnerabilities in software or operating systems. Once installed, the scareware will often display a warning message or fake scan results, claiming that the user's system is infected with viruses, spyware, or other security threats. The scareware may then encourage the user to download or purchase a security software program to fix the supposed problem. However, the security software is often fake or ineffective, and may even install additional malware onto the user's system.

3. Malware Analysis

Analysing malware is an essential aspect of understanding how they operate. Without knowing exact details on how attackers take control of a system, it is impossible to develop ways of counteracting them. By analysing malware, security researchers can gain insights into how attackers operate and develop effective methods of preventing and detecting attacks.

4. Portable Executable

The Portable Executable (PE) format describes the layout of files for executables and dynamic-link libraries (DLL). It is used extensively within the Windows operating system and extracting information from executables using this format is important to gain information about malicious .exe files. The format can differ depending on if the file is an image (executable) or an object (non-executable such as .dll). Some sections within the file may not be used depending on this difference. For the purposes of this project, only images need to be considered. The following figure shows an outline of the beginning of a PE file, and some useful information that can be obtained.

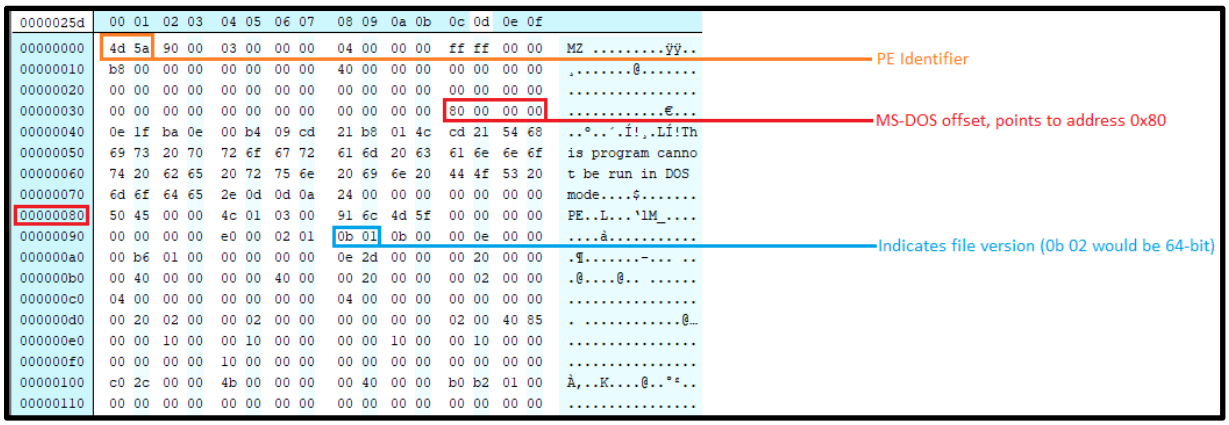


Figure 3 Hex viewer of .exe file

PE Identifier: The first two bytes of the file indicate that the file is a Portable Executable, and will always be 0x5a4d. This is used as an easy way to verify that the file can be read as intended.

MS-DOS Offset: At position 0x3c in every PE file, 4 bytes are used to identify the address where the MS-DOS stub ends. The stub is used to print a string if the file is run under DOS mode, but can be used for other purposes. This makes the MS-DOS stub a varying length size, and therefore the offset is used for programs to know where to begin reading values that follow. In the figure above, this offset is set to 0x80, as it is read in little-endian order. Values after this address have documented and fixed sizes.

File Version: The magic number determines whether an image is PE32 (32-bit, value of 0x010b) or PE32+ (64-bit, value of 0x020b). This can be assigned to a boolean as there are no other values possible. The importance of knowing the version is described below.

4.1. File version

An image may be either PE32 (32-bit) or PE32+ (64-bit). Determining which version an executable uses is the first step to reading the contents, as sections use different sizes of bytes to store information. This allows PE32+ files to accommodate 8 byte values, or 64 bits of information. There is a “quick and easy” way to determine if an executable is 32-bit or 64-bit, which involves opening it in a text editor and finding a character following the string “PE”. This method will work for most executables, however the documentation for the format uses the magic number as described above.

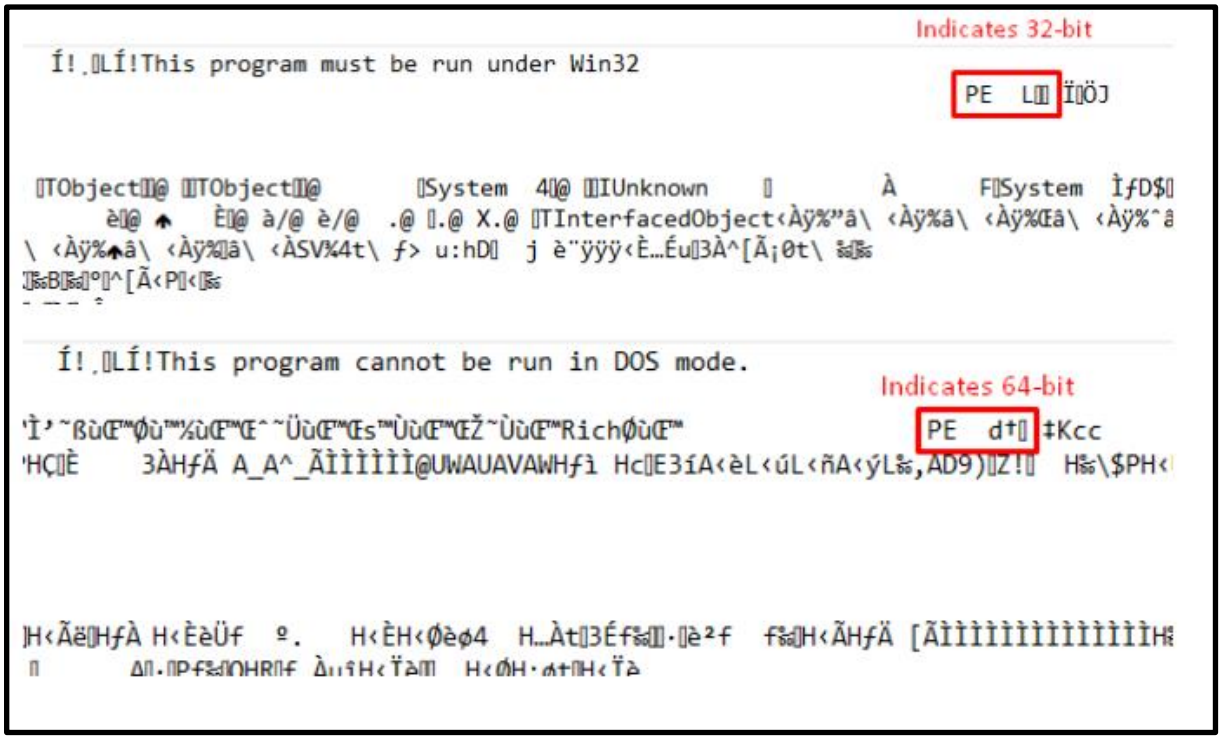


Figure 4 Text views of .exe files

The values shown actually indicate the text representation of the byte values for the machine type rather than the version, but are still used to quickly find the version as the other types are not commonly used.

5. Java Native Interface

Java Native Interface (JNI) allows for foreign functions from other languages such as C and C++ to be called and run within a Java program (Wikipedia, 2021). It is a supported interface that comes with Java, and can be initialised with the "*javac -h . FileName.java"* command when the Java Development Kit (JDK) is installed. This command will generate a *.h* header file which allows the foreign language to be linked to the Java program.

```
1   /* DO NOT EDIT THIS FILE - it is machine generated */
2   #include <jni.h>
3   /* Header for class dynamicAnalysis_VirtualMemory */
4
5   #ifndef _Included_dynamicAnalysis_VirtualMemory
6   #define _Included_dynamicAnalysis_VirtualMemory
7   #ifdef __cplusplus
8   extern "C" {
9   #endif
10  /*
11   * Class:     dynamicAnalysis_VirtualMemory
12   * Method:    scanProcess
13   * Signature: (I)[B
14   */
15  JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
16    (JNIEnv *, jobject, jint);
17
18  #ifdef __cplusplus
19  }
20  #endif
21  #endif
```

Figure 5 Generated .h file

With this header file, C and C++ functions can be created to be called within Java. When the functions are ready to be used, they must be compiled into a *.dll* file, which can then be loaded into Java.

6. Virtual Memory

In windows, any running process is allocated a virtual memory space (Wikipedia, 2020). This differs from a physical memory space in that not every section of the virtual memory needs to be stored in RAM at any one time, but instead can transfer some sections onto the hard disk if there is a shortage of RAM available. Doing this also allows for multiple processes to use the same memory address to reference different data, as virtual memory addresses are mapped to the physical addresses. This also serves as a security feature as processes cannot reference the virtual memory of other processes without directly calling the space.

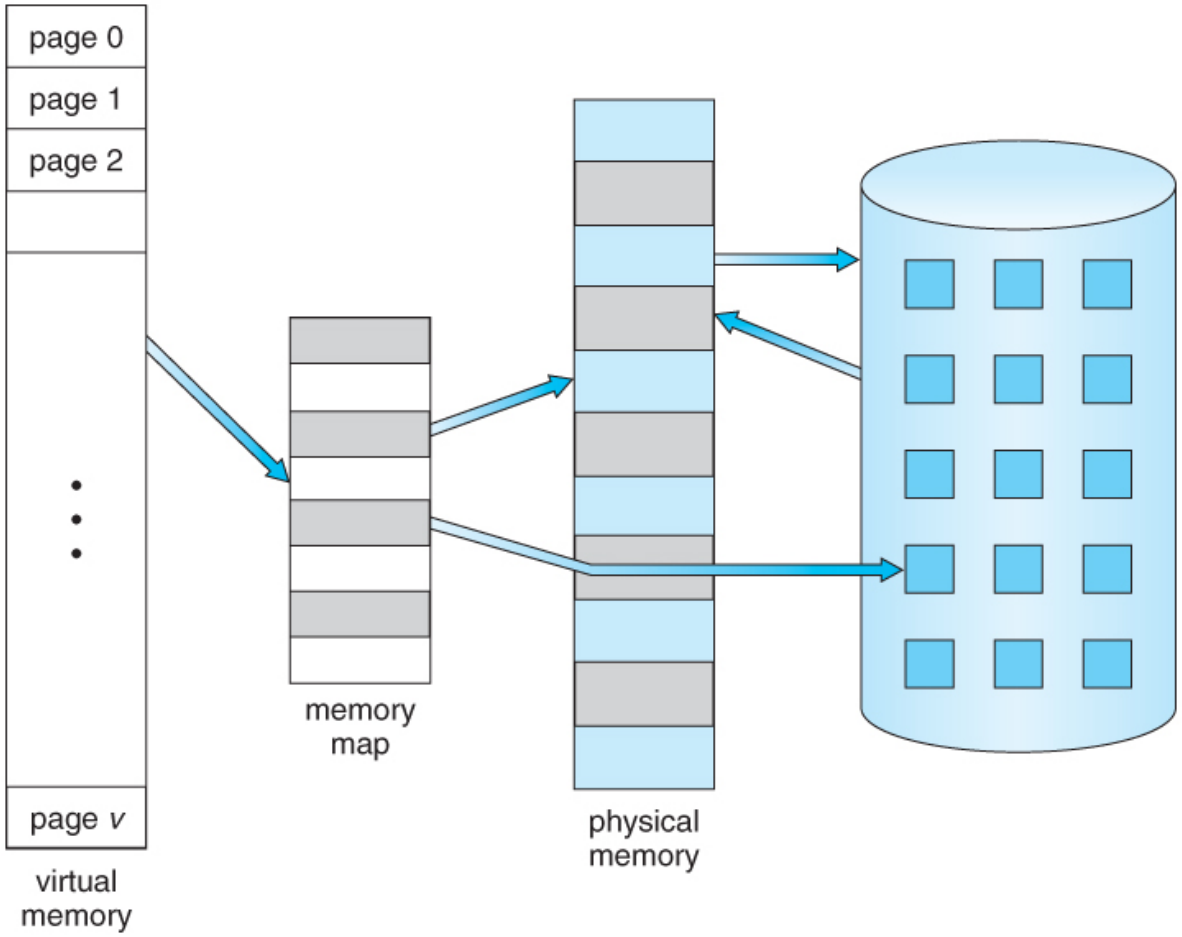


Figure 6 Mapping virtual memory to physical memory (Abraham Silberschatz, n.d.)

The virtual memory of a process can be used to analyse what a program is storing to be used at any point, such as URLs, strings, directories, etc. For this reason, accessing the virtual memory of a running process can be utilised to detect malicious actions by checking if sensitive information is being processes. The *windows.h* API for c++ gives the required functions for this to be possible. There are many challenges to be solved in order to be able to access the virtual memory space of a separate process, relating to how virtual memory is read, security features, PE version, etc.

6.1. Process Access Rights

To read and write into a virtual memory space, the process must have a sufficient access right to do so (Microsoft, n.d.). Initialising the process with the correct access right is required to be able to read the contents of the space. The figure below shows the different access rights that can be used, with the access right *PROCESS_ALL_ACCESS* being used in the project to read the entire space.

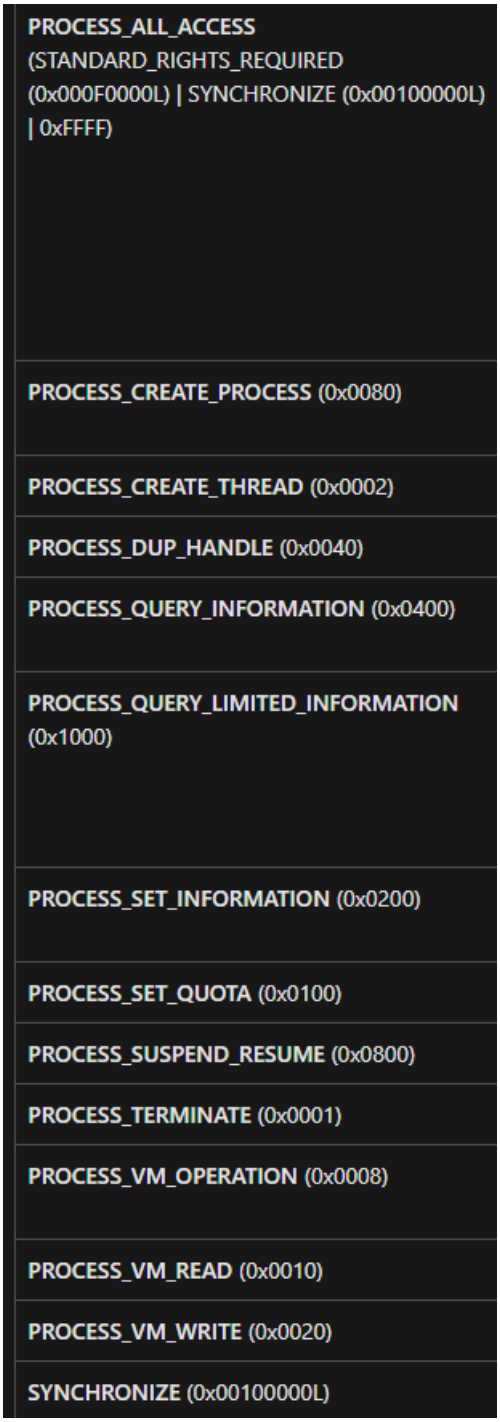


Figure 7 Process Access Rights (Microsoft, n.d.)

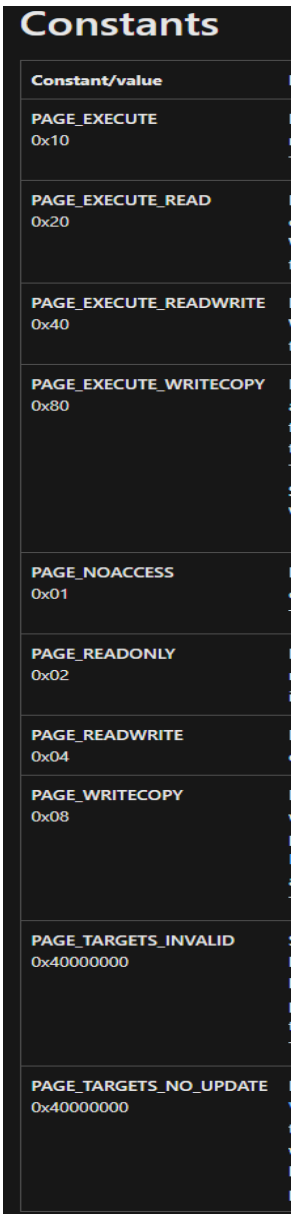
```
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (int)processId);
```

Figure 8 Opening process with access right

6.2. Page Protection

Once the process can be accessed, each section of virtual memory still needs to be read. The pages are separated into regions with identical attributes, each with their own protection

labelled *Memory Protection Constants* in the documentation (Microsoft, n.d.). To read the entire virtual memory space, each of these regions must be iterated through, changing the protection of each as this is done. The protection constant *PAGE_EXECUTE_READWRITE* is used for each to gain access to all pages. The starting address and the size of the region to be changed must also be specified, and the protection will change for every section within that region, for a total of *startingAddress* + *RegionSize* bytes changed.



Constants

Constant/value
PAGE_EXECUTE 0x10
PAGE_EXECUTE_READ 0x20
PAGE_EXECUTE_READWRITE 0x40
PAGE_EXECUTE_WRITECOPY 0x80
PAGE_NOACCESS 0x01
PAGE_READONLY 0x02
PAGE_READWRITE 0x04
PAGE_WRITECOPY 0x08
PAGE_TARGETS_INVALID 0x40000000
PAGE_TARGETS_NO_UPDATE 0x40000000

Figure 9 Memory Protection Constants (Microsoft, n.d.)

```
int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize, PAGE_EXECUTE_READWRITE, &oldprotect);
```

Figure 10 Changing protection with protection constant

The function above may fail, and will return a value of 0 if it does. This will happen if a region of memory has not been initialised, and must be skipped if so.

6.3. Memory Basic Information

The *MEMORY_BASIC_INFORMATION* (MBI) structure contains the information relating to a range of pages within the virtual memory space of a process (Microsoft, n.d.). This information contains the page protection mentioned above, along with the region size, base address, and other values.

```
typedef struct _MEMORY_BASIC_INFORMATION {
  PVOID  BaseAddress;
  PVOID  AllocationBase;
  DWORD  AllocationProtect;
  WORD   PartitionId;
  SIZE_T RegionSize;
  DWORD  State;
  DWORD  Protect;
  DWORD  Type;
} MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

Figure 11 MEMORY_BASIC_INFORMATION structure (Microsoft, n.d.)

The structure must be updated for each iteration through the range of pages by referencing it through querying, as mentioned below.

6.4. Querying

Once a section of data has the correct protection constant, the region can be queried (Microsoft, n.d.). This involves loading the relevant data into the MBI address, updating the region's size to the new region's attributes. An address is given to the function, which specifies the starting address to begin the query. Querying will return a value of 0 if it fails, which will happen if the end of the virtual memory space is reached, allowing for the function to be put in a while loop to iterate through the entire space as shown in the figure below.

```
while (VirtualQueryEx(hProcess, (LPVOID)addr, (PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi)))
```

Figure 12 Query Function

6.5. Read Memory

After the previous steps have been completed, the memory can finally be read. Each readable byte is placed into a character array that acts as a buffer for each region, then placed into a vector that stores the entire memory space. When the iterations have completed, the vector is converted to a JNI-friendly byte array to be passed back to the Java application. This byte array is the complete readable virtual memory space of the targeted process.

```
buffer = new unsigned char[signedSize];
ReadProcessMemory(hProcess, (LPCVOID)addr, buffer, mbi.RegionSize, &size);
for (int index = 0; index < mbi.RegionSize; index++)
{
        regionData.push_back(buffer[index]);
}
```

Figure 13 Reading Process Memory

7. Dynamic Malware Analysis

When attempting to detect malware, there are two common approaches; static analysis and dynamic analysis. Static analysis attempts to read the binary code of an executable in order to detect a malicious program. Dynamic programming, on the other hand, will execute the code and attempt to trace the actions of the program to determine if it is malicious. This method does not rely on signature detecting, but can implement signature tracking to skip redundant analysis.

Dynamic analysis can be a powerful tool in the fight against malware, as it allows analysts to observe the behaviour of a program in real-time and to detect previously unknown or zero-day malware. By monitoring the actions of a program during execution, analysts can identify suspicious activity such as attempts to modify system files, network traffic to suspicious domains, or unauthorised access to sensitive data.

In addition to detecting and analysing malware, dynamic analysis can also be used to develop and test new security measures. By simulating a malware attack in a controlled environment, analysts can evaluate the effectiveness of different security tools and techniques, and develop strategies for mitigating and preventing future attacks.

While dynamic analysis can be a powerful tool, it is not without its challenges. One of the main difficulties is that dynamic analysis can be time-consuming and resource-intensive, as it requires running the malware in a controlled environment and carefully monitoring its behaviour. Additionally, some malware may be designed to evade detection by detecting and avoiding dynamic analysis environments.

Despite these challenges, dynamic analysis remains an important tool in the fight against malware. By combining dynamic analysis with some elements of static analysis, analysts can develop a comprehensive approach to malware detection and analysis that is capable of detecting even the most sophisticated and elusive threats.

7.1. Analysis Techniques

Dynamic analysis relies on executing the code, then monitoring what the program does. There are two primary ways of achieving this; function call monitoring and information flow tracking (Aman, 2014).

7.1.1. Function call monitoring

As a program executes, it performs function calls. This is a section of reusable code that makes certain actions easier. At times, it is required to use these calls to communicate with the operating system. An analysis tool can use an operation called *hooking* to monitor these system calls. This is achieved by injecting code into the monitored program that is activated whenever a system call is triggered, allowing for the analysis tool to track which calls are performed.

7.1.2. Information flow tracking

This method aims to follow how a program uses and manipulates certain data throughout the system. The data that should be monitored is marked as *tainted* and allows the analysis tool to see how it is altered throughout execution.

7.2. Evasion

Malware employs several techniques in an attempt to make itself obscure to analysis. One of these techniques is to elevate its privileges to one that is above the tool being used (Or-Meir, 2019). The Intel processor uses a protection ring as seen in Figure 1, which states the privileges a section of code is granted. It comprises 4 sections, with ring 3 containing application code, and being the least privileged. Ring 3 and 2 are dedicated to device drivers, and ring 0 is the kernel section. For a malicious application, gaining access to the kernel ring means that it can run with very little restrictions. A common method of gaining access to the kernel ring is to get the user to install an infected kernel driver, giving complete access to the CPU and connected devices, such as a web camera. This can make it very difficult for an analysis tool to detect the malware, as it is generally loaded in the applications ring.

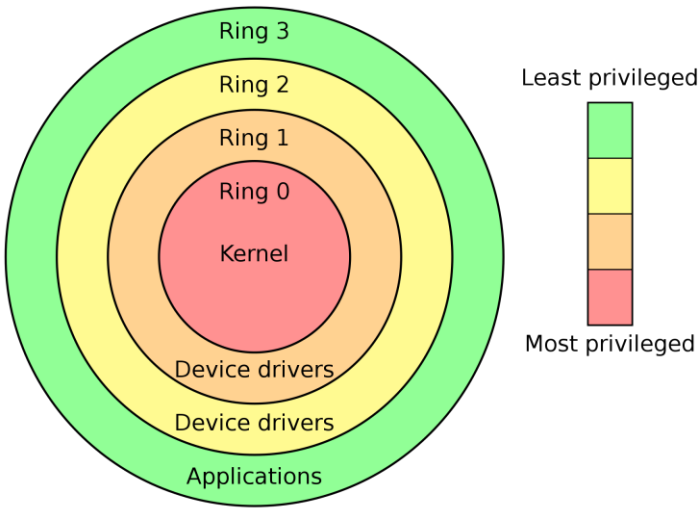


Figure 14 Intel Protection Ring (Wikipedia, 2022)

If the malware cannot gain access to elevated privileges, it relies on code obfuscation to avoid detection from an analysis tool. This attempts to hide its code and attack methods through a

variety of means. The primary way of doing this is to use polymorphism, which encrypts the code and adds difficulty in reading and executing the code.

7.3. Analysis Environment

As dynamic analysis requires running the malicious code to see its actions, there would be an extreme risk in performing analysis without taking precautions. The safest approach is to initialise a contained environment, such as a virtual machine or a sandbox. To prevent the malware from spreading through the network, internet access should also be limited. This provides a safe environment to perform analysis as any damage caused by the malware will only affect the contained environment, and not the host machine.

8. Existing Tools

8.1. Wireshark

Wireshark is a network analysis tool, which can be used in malware analysis. This tool allows analysers to see what is getting sent over a network. If a malicious program is sending data to a server from the machine, Wireshark can report this back to the user.
Using a Graphical User Interface (GUI), Wireshark shows a visual representation of the network information, as seen in Fig.2. This data can then be used to inspect the transferred packets and determine what is being sent to and from servers. The analysis tool could utilise this program for scanning if a piece of sensitive data is being checked on by another computer.

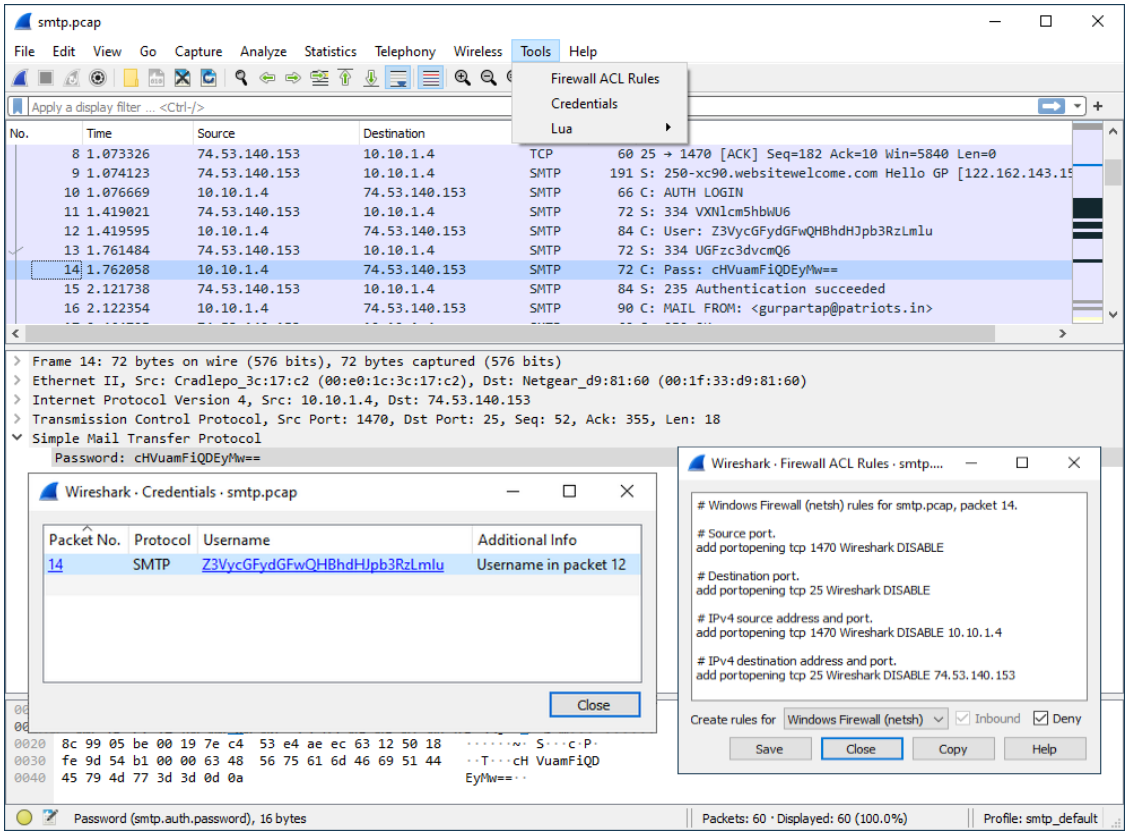


Figure 15 Wireshark GUI

8.2. Regshot

Regshot is a tool which will take a snapshot of registry data, and compare it to a snapshot taken at a later time. A comparison can be done to see which values were updated. Should the malware update these registry values during its execution, this data can be utilised to gain information on the actions it is taking.

8.3. EDB

Evan's Debugger (EDB) is a tool that is used to extract the assembly code from a program and read how it affects the stack values (Teran, n.d.). It is compatible with x86 instructions, and serves as a debugger in the sense of executing each line of code line-by-line, and updating values dynamically. This is an intended feature with the application of this project, with additional features to determine if the lines of code are malicious.

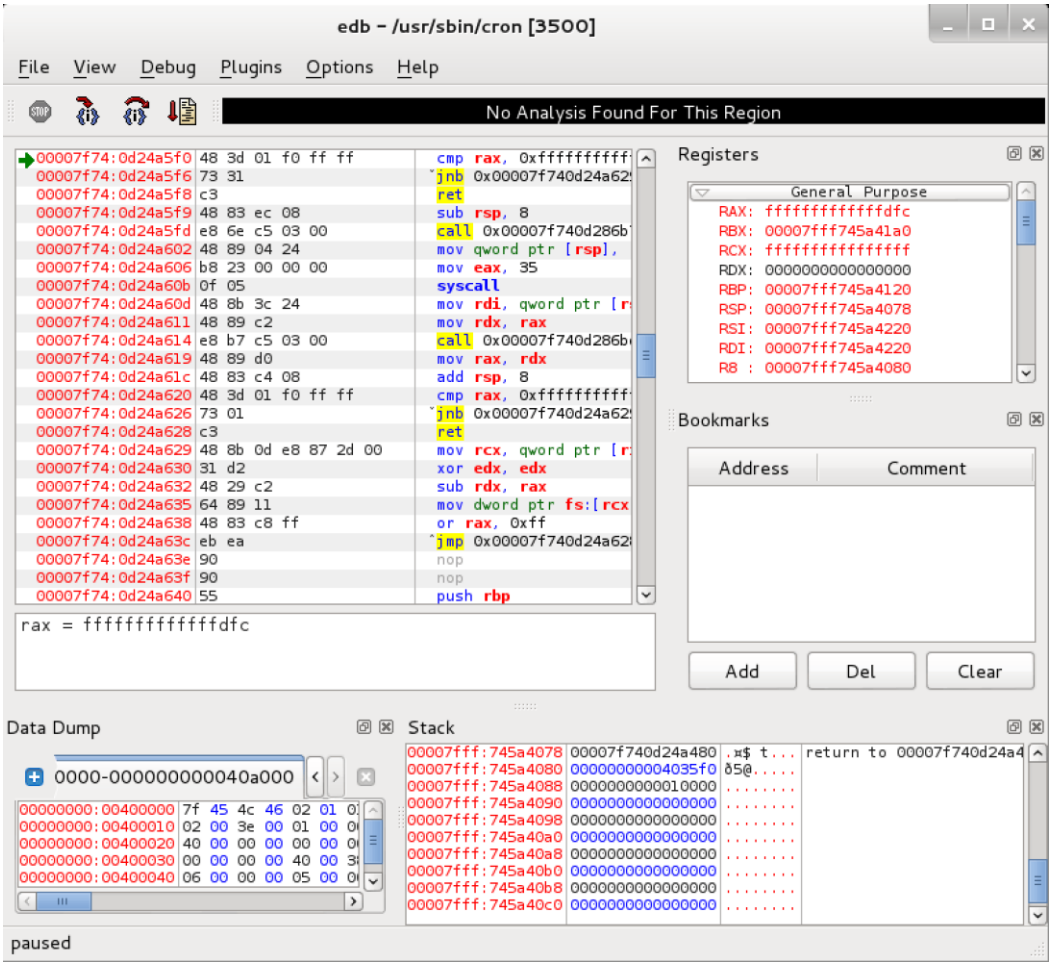


Figure 16 Evan's Debugger (Teran, n.d.)

9. Libraries

9.1. Capstone

Capstone (Capstone, n.d.) is a widely-used disassembly tool that enables analysts to extract the executable code from a binary file and to translate it into human-readable assembly language. The tool is open source and offers bindings for a wide range of programming languages, making it accessible to developers and analysts working in different environments.

Capstone supports a wide range of architectures, including x86, x86_64, ARM, MIPS, PowerPC, and others, allowing it to be used to analyse a variety of binary files. For the purposes of this project, the x86 and x86_64 instruction sets are specifically needed, as they are commonly used in PE files.

The output of Capstone's disassembly process is a detailed analysis of the binary code, including information about the instructions executed, the values of any registers used, and the memory addresses accessed. This information can be used to gain insights into the behaviour of a program, to identify specific vulnerabilities or malicious behaviour, and to develop targeted countermeasures.

10. Operating System

For the purposes of the project, two Operating Systems (OS) were considered, which were the Linux and Windows desktop operating systems. Other types of OS, such as Android and iOS, were not chosen as their respective screening process would detect most forms of malware. There are advantages to using both Operating Systems for the application.

10.1. Linux

With the open ended software commonly available on Linux, having a more in-depth control of processes is very important to an analysis tool. This would give access to more data that could be used to detect malicious code. This is also true for malware running on the platform, with the potential of inserting malicious code in the source code for the OS itself (Sharma, 2021).

10.2. Windows

In a survey performed in 2020 (Petrosyan, 2020), over 80% of all malware attacks occur on the Windows Operating System (OS). As Windows is the most commonly used desktop OS, it provides a large group of users for attackers to target. While there are more users with a mobile device than a desktop computer, both Apple and Android have a vetting process that can prevent most forms of malware.

With this information, the operating system of choice for this project will be Windows, as cross-platform compatibility is unlikely to be feasible in the given duration.

11. Conclusion

In conclusion, understanding how malware is distributed and the different ways it can affect a system is essential in creating a program to analyse it. Research into malware can never be considered complete as it is constantly changing and evolving, leading to an arms-race between attackers and analysers. However, it is vital to keep up to date with advances in vulnerability exploitation to invent ways of counteracting the exploits. The ambition with this project is that more people become aware of these dangers, and gain the required knowledge to be able to effectively defend themselves against the threats that come with moving into a technology-focused world.

Bibliography

Abraham Silberschatz, G. G. a. P. B. G., n.d. *Virtual Memory.* [Online]
Available at:
https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html

Akamai, n.d. *What is Ransomware?.* [Online]
Available at: https://www.akamai.com/our-thinking/cybersecurity/what-is-ransomware

Aman, W., 2014. *[1410.2131] A Framework for Analysis and Comparison of Dynamic Malware Analysis Tools.* [Online]
Available at: https://arxiv.org/abs/1410.2131
[Accessed 18 November 2022].

Anon., n.d. *Dynamic malware analysis in the modern era—A state of the art survey.* Or-Meir, O., Nissim, N., Elovici, Y. and Rokach, L., 2019. Dynamic malware analysis in the modern era—A state of the art survey. ACM Computing Surveys (CSUR), 52(5), pp.1-48.: s.n.

Anon., n.d. *Wireshark.* [Online]
Available at: https://www.wireshark.org

Capstone, n.d. *Capstone Dissasembly Tool.* [Online]
Available at: https://www.capstone-engine.org

Microsoft, n.d. *Memory Basic Information.* [Online]
Available at: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-memory_basic_information

Microsoft, n.d. *Memory Protection Constants.* [Online]
Available at: https://learn.microsoft.com/en-us/windows/win32/memory/memory-protection-constants

Microsoft, n.d. *Process Security and Access Rights.* [Online]
Available at: https://learn.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights

Microsoft, n.d. *VirtualQueryEx function.* [Online]
Available at: https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex

Or-Meir, O. N. N. E. Y. a. R., 2019. *Dynamic malware analysis in the modern era—A state of the art survey.* s.l.:ACM Computing Surveys.
Petrosyan, A., 2020. *Operating systems most affected by malware as of 1st quarter 2020.* [Online]
Available at: https://www.statista.com/statistics/680943/malware-os-distribution/

Sharma, A., 2021. *Bleeping Computer.* [Online]
Available at: https://www.bleepingcomputer.com/news/security/linux-bans-university-of-minnesota-for-committing-malicious-code/

Teran, E., n.d. *EDB Debugger.* [Online]
Available at: https://github.com/eteran/edb-debugger

Wikipedia, 2020. *Virtual Memory.* [Online]
Available at: https://en.wikipedia.org/wiki/Virtual_memory

Wikipedia, 2021. *Java Native Interface.* [Online]
Available at: https://en.wikipedia.org/wiki/Java_Native_Interface

Wikipedia, 2022. *Intel Protection Rings.* [Online]
Available at: https://en.wikipedia.org/wiki/Protection_ring

Wikipedia, 2023. *WannaCry ransomware attack.* [Online]
Available at: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

Additional Materials

Aslan, Ö.A. and Samet, R., 2020. A comprehensive review on malware detection approaches. *IEEE Access*, *8*, pp.6249-6271.

Souri, A. and Hosseini, R., 2018. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, *8*(1), pp.1-22.

Aslan, Ö. and Samet, R., 2017, October. Investigation of possibilities to detect malware using existing tools. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)* (pp. 1277-1284). IEEE.

Neugschwandtner, M., Platzer, C., Comparetti, P.M. and Bayer, U., 2010, July. Danubis–dynamic device driver analysis based on virtual machine introspection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 41-60). Springer, Berlin, Heidelberg.