# MALWARE ANALYSIS USING METHOD IDENTIFIERS

## A PROJECT REPORT

*Submitted by*

**ANAND PANDEY**       **(22BCY10122)**
**AYUSH MONGA**       **(22BCY10267)**
**REETI CHOPRA**       **(22BCY10128)**
**ABHIVYAKTI RATHOD**   **(22BCY10106)**
**AHMED AZAAN KHAN**   **(22BCY10245)**

*in partial fulfillment for the award of the degree*
*of*

## BACHELOR OF TECHNOLOGY

in

## COMPUTER SCIENCE AND ENGINEERING
### (Cyber Security and Digital Forensics)



www.vitbhopal.ac.in

## SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
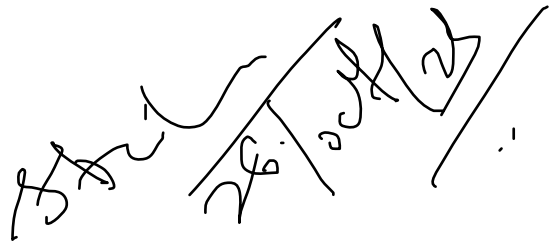
## VIT BHOPAL UNIVERSITY

## KOTHRIKALAN, SEHORE
## MADHYA PRADESH - 466114

OCT 23

# VIT BHOPAL UNIVERSITY, KOTHRI KALAN, SEHORE MADHYA PRADESH – 466114

## BONAFIDE CERTIFICATE

Certified that this project report titled **"METHOD IDENTIFIER"** is the bonafide work of **"REETI CHOPRA (22BCY10128), ABHIVYAKTI RATHOD (22BCY10106), AHMED AZAAN KHAN (22BCY10245), ANAND PANDEY (22BCY10122), and AYUSH MONGA (22BCY10267)"** who carried out the project work under my supervision. Certified furtherthat to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion to this or any other candidate.

**PROGRAM CHAIR**

**Dr. D. Saravanan**

Program Chair,

Division of Cyber Security and

Digital Forensics

School of Computer Science and

Engineering

VIT BHOPAL UNIVERSITY

**PROJECT SUPERVISOR**

**Dr. Hariharasitaraman S**

Supervisor,

Division of Cyber Security and

Digital Forensics

School of Computer Science and

Engineering

VIT BHOPAL UNIVERSITY

The Project Exhibition 1 Examination is held on_____.

# ACKNOWLEDGEMENT

# LIST OF ABBREVIATIONS

- **URL**- Uniform Resource Locator
- **DLL**- Dynamic Link Library
- **DNS**- Domain Name System
- **PE**- Portable Executor
- **ASCII**- American Standard Code for Information Interchange
- **SHA**- Secure Hash Algorithm
- **IOC**- Indicators of Compromise
- **APT**- Advance Persistent Threat
- **Exe-** Executable

# LIST OF FIGURES AND GRAPHS

# TABLE OF CONTENTS

# ABSTRACT

Malware is any malicious software that is harmful created by cybercriminals to steal data, harm, or destroy computers and computer systems.

The proposed name of the project is "Malware Analysis". The word "malware" is made up of the words "malicious" and "software." Any sort of software or code that is specifically created with the goal to hurt, damage, or allow unauthorized access to computer systems, networks, or userdata is referred to as malicious software.

This study focuses on the critical field of malware analysis, aiming to explore techniques and methodologies to understand different behaviors and characteristics of malicious software.
To conceal their malicious code and avoid being detected by security tools, malware authors frequently employ a variety of approaches. One typical tactic is to obscure their code by employing various identifiers and approaches.

Our study demonstrates how malware has developed from simple viruses to more complex varieties like ransomware. We dig into the significance of threat intelligence, code analysis, and behavioral analysis in defending against these dangers. We also stress the importance of cybersecurity experts working together to remain on top of changing malware threats. Method Identifiers are being used in the project for malware analysis.

Comments:

- The proposed name of the topic is clear.
- The objective is relevant.
- Includes research findings on the topic.

# CHAPTER 1

# PROJECT DESCRIPTION AND OUTLINE

## 1.1 Introduction

**Malware** is an abbreviated form of "malicious software." It is developed as harmful software that invades or corrupts one's computer network. The goal of malware is to cause havoc and steal information or resources for monetary gain or sheer sabotage intent. It is specifically designed to gain access to or damage a computer, usually without the knowledge of the owner. The term "malware" was reportedly first coined by Yisrael Radai in 1990. Yisrael Radai is an Israeli security expert who used the term in a Usenet newsgroup discussion. Since then, "malware" has become a widely accepted and commonly used term in the field of cybersecurity to refer to all types of malicious software. Now, over 25 million new types of malwares registered since the beginning of 2022 alone.

Classifying malware into distinct types or categories based on its characteristics, behavior, and intended purpose helps cybersecurity professionals, researchers, and antivirus software developers better understand, analyse, and respond to different types of malwares.

## 1.2 MOTIVATION FOR THE WORK

The motivation behind this project of ours is that there has always been a problem related to malicious software known as malware. So, it is important to note that while studying malware is essential for cybersecurity, it should always be done responsibly and within legal boundaries. Malware analysis is required to gain insights into how specific malware strains work, theirmethods and behaviours. The need to safeguard critical infrastructure has led to creating this project. So, let us quickly know more about the project in detail.

**1.3    Problem Statement**

As stated earlier in the introduction, method identifiers in malware analysis play an important role in understanding and analysing malicious software. These identifiers are essentially the techniques, patterns, or characteristics that can be used to identify and categorize malware.

So our project's aim is to perform analysis of the malware using method identifiers. The problem which we are facing is the ongoing threat of malware which is harmful for computers and networks. According to the recent article which tells how pervasive and serious danger has malware become in today's digital age.

Identifiers categorize and classify different types of malware behaviours. This classification allows security professionals to quickly understand the nature of a particular behaviour and its implications. Security teams can use identifiers to develop response strategies and mitigation techniques tailored to specific behaviours. This helps organizations proactively defend against known malware activities.

**1.4  Objective**

The objectives of the project are to:

- To obtain a dataset of malicious and non-malicious samples.

- To extract the features from the respective samples.

- To create a catalogue of known malicious behaviours.

- To analyse the extracted features and compare them with the catalogue of known malicious behaviours.

## 1.5 Scope

The scopes involved in the project are threat classification, malware behavioural analysis, vulnerability exploitation.

Basically, the overall process of this project can be viewed as below:

1. Collect the data

2. Extract features from the samples

3. Build a Malware Behavior Catalogue

4. Analyse the behavior of the extracted features

5. Test and evaluate

6. Alert and reporting

## 1.6 Report Organization

This thesis consists of five (5) chapters. Chapter 1 will discuss the introduction to the system which will explain the introduction, problem statement, objective, and scope. For Chapter 2, it will discuss the literature review, definition, and types of malwares. For Chapter 3, it will discuss the methodology and requirements of the project. For Chapter 4, it will discuss the design and implementation of the project. Chapter 5, it will show the results and discussion.

## SUMMARY

Our project focuses on malware analysis using method identifiers using static and synamic analysis. Now a day, internet becomes an essential part of the daily life of many people where as malware is designed to damage a computer system without user's informed consent.

# CHAPTER 2:
# RELATED WORK INVESTIGATION

## 2.1    Introduction

In this chapter, we are going to focus on discussing the results or findings based on the article, journals, or any other related reference material. Some original words from the reference material may be cited to enhance the review. The purpose of this chapter is to explain about the selected project. We focus on malware detection using a combination of static and dynamic analysis as well as signature-based approaches. Through our project, we aim to reveal the inner workingsof malware using this verification process. By using signature as a guide, we seek to contribute to ongoing efforts to improve cybersecurity, share our knowledge and protect our digital environment from malware threats.

Basically, it is divided into a few sub-sections as well. Those sub- section include some little explanation of basic concepts of the selected project, research of some already existing similar problem or solution done by others and the hardware, technique or method which will be applied or used in the selected project.

This chapter explains in detail the techniques or technologies which are suitable to be adapted into the project. This chapter contains information about the study of the project in general.

**LINKS:**

- **https://blough.ece.gatech.edu/research/papers/ccs21.pdf**

- **https://www.researchgate.net/publication/267777154_Malware_Analysis**

## 2.2 What is Malware Classification?

Malware classification is the process of categorizing malicious software (malware) into distinct types or categories based on its characteristics, behavior, and intended purpose. This classification helps cybersecurity professionals, researchers, and antivirus software developers better understand, analyze, and respond to different types of malware.
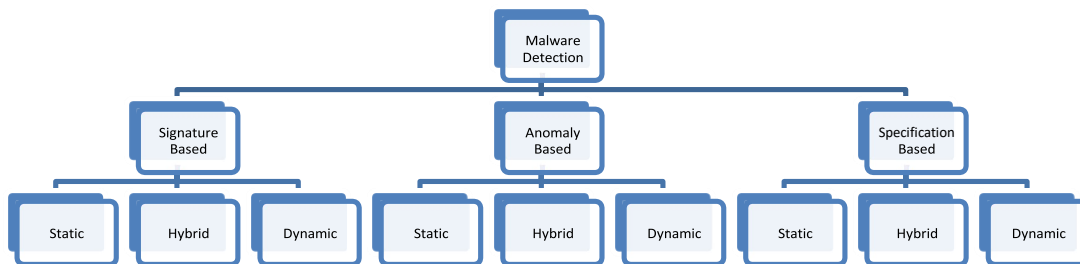


**Fig 1:    Malware Detection Flowchart**

Here are some common categories of malware:

- Worms: Worms are self-replicating malware that can spread across networks or the internet without the need for user interaction. They often exploit vulnerabilities to propagate.

- Viruses: Viruses are malicious programs that attach themselves to legitimate files or software. They can replicate and spread to other files or systems when the infected file is executed.

- Trojans: Trojans disguise themselves as legitimate software but have malicious functionality hidden within. They can perform a variety of harmful actions, such as data theft, remote control, or system damage.
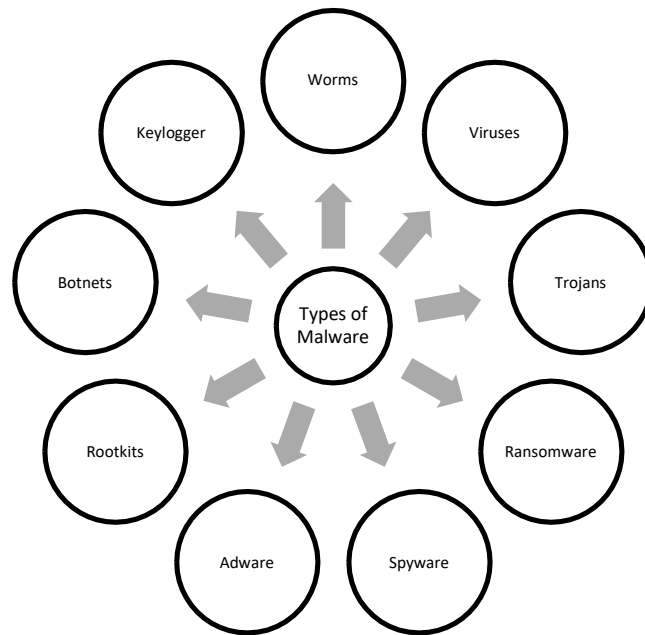
**Fig 2:    Malware Classification**

- Ransomware: Ransomware encrypts a victim's files or locks them out of their system, demanding a ransom payment in exchange for the decryption key. It's known for its extortion tactics.

- Spyware: Spyware is designed to secretly monitor and gather information about a user's activities, often without their consent. It can record keystrokes, capture screenshots, and more.

- Other: Adware, Rootkits, Botnets, Keyloggers, etc.

# CHAPTER: 3
# REQUIREMENT ARTIFACTS

## 3.1 Introduction

A system architecture is a conceptual model that defines the structure, behavior, and interactions of a system. It tells us about various components of a system and how they interact with each other. It is a blueprint for how the system will be built and deployed. The system architecture should be designed to meet the specific needs of the system and its users.

The system architecture is typically defined using a variety of diagrams and models. These diagrams and models show the different components of the system, how they are connected, and how they interact with each other. The system architecture may also include descriptions of the system's data flows, control flows, and performance characteristics.

The system architecture is important because it helps to ensure that the system is well-designed and that it will meet the needs of its users. It also helps to identify any potential problems with the system early on, before they become too difficult to fix.

## 3.2 Tools and Environments used

For malware analysis various tools, environments have been used as specified below:

1. ORACLE Virtual Box: VM Virtual Box is cross-platform virtualization software. It allows users to extend their existing computer to run multiple operating systems including Microsoft Windows, Mac OS X, Linux, and Oracle Solaris, at the same time.
   Designed for IT professionals and developers, Oracle VM Virtual Box is ideal for testing, developing, demonstrating,

   And deploying solutions across multiple platforms from one machine.

2. Operating System: Windows 10 is a computer operating system by Microsoft. It is part

of the Microsoft Windows family of operating systems. It was called Threshold when it  was being developed (made/coded). Windows 10 is  a Microsoft operating system for personal computers, tablets, embedded devices and internet of things devices.
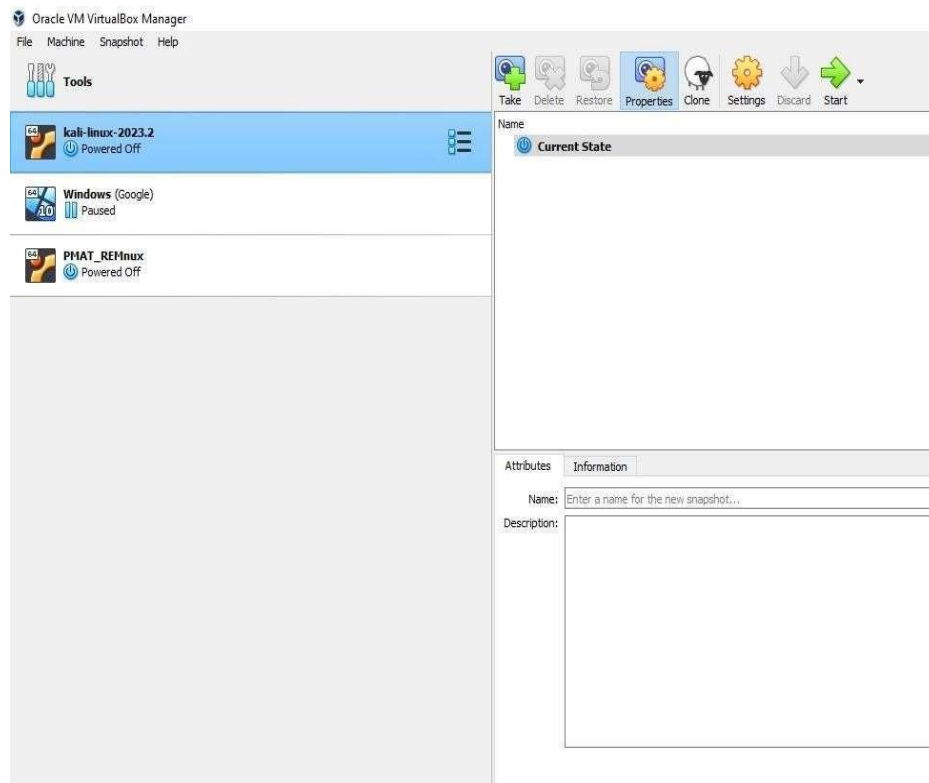


Fig 3: Installing Oracle VM and setting up Windows 10 through ISO file
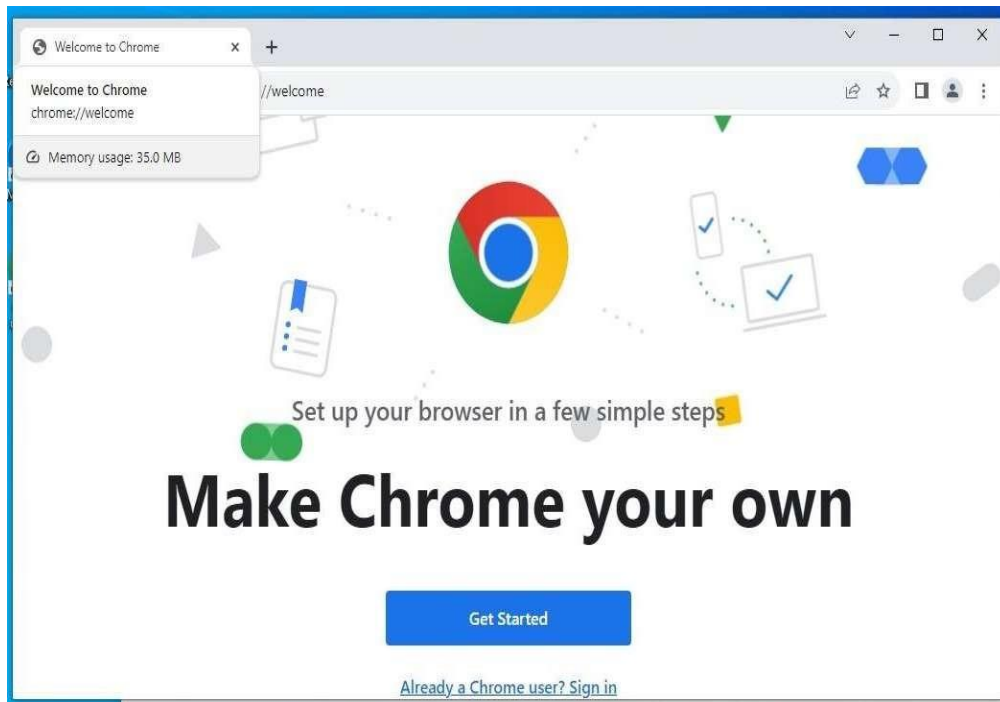
Fig : Setting up Chrome in Windows 10

3. Flare VM: FLARE VM - a collection of software installations scripts for Windows systems that allows you to easily setup and maintain a reverse engineering environment on a virtual machine (VM). FLARE VM was designed to solve the problem of reverse engineering tool curation and relies on two main technologies: Chocolatey and Boxstarter. Chocolatey is a Windows-based Nuget package management system, where a "package" is essentially a ZIP file containing PowerShell installation scripts that download and configure a specific tool. Boxstarter leverages Chocolatey packages to automate the installation of software and create repeatable, scripted Windows environments.
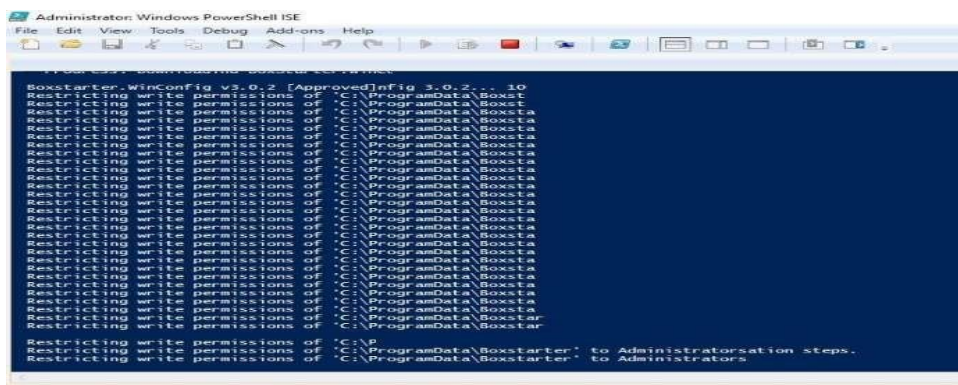
Fig 4: Installation of Flare VM on virtual Windows 10

REMnux : REMnux is a lightweight, Ubuntu-based Linux distribution for assisting malware analysts with reverse-engineering malicious software. It incorporates a number of tools for analysing malicious executables that run on Microsoft Windows, as well as browser-based malware, such as Flash programs and obfuscated JavaScript. The toolkit also include programs for analysing malicious documents, such PDF files, and utilities for reverse- engineering malware through memory forensics.



Fig 5: Setting up Renmux through ISO file

4. INetSim: INetSim is a free, Linux-based software suite that simulates common internet services. It's useful for analyzing the network behaviour of malware samples without connecting them to the Internet.
INetSim simulates services like: HTTP, DNS, SMTP.

   INetSim also has additional features, including:

   - Faketime

   - Connection redirection

   - IP-based redirection of arbitrary connections (tcp, udp and icmp)

5. Static Analysis:Static Analysis is the automated analysis of source code without executing the application.

6. Dynamic Analysis: When the  analysis is performed during program execution then it is known as Dynamic Analysis

# CHAPTER 4 :
# DESIGN METHODOLOGY AND ITS NOVELTY

## 4.1 Introduction

This chapter contains section 3.2 depicting the overall flowchart of the program which is the step-by-step analysis of the malware. Section 3.2 contains hardware and software requirements for "Malware Analysis".

This chapter contains the flow chart of the whole application. This chapter will explain every step involved in the formation of the application.
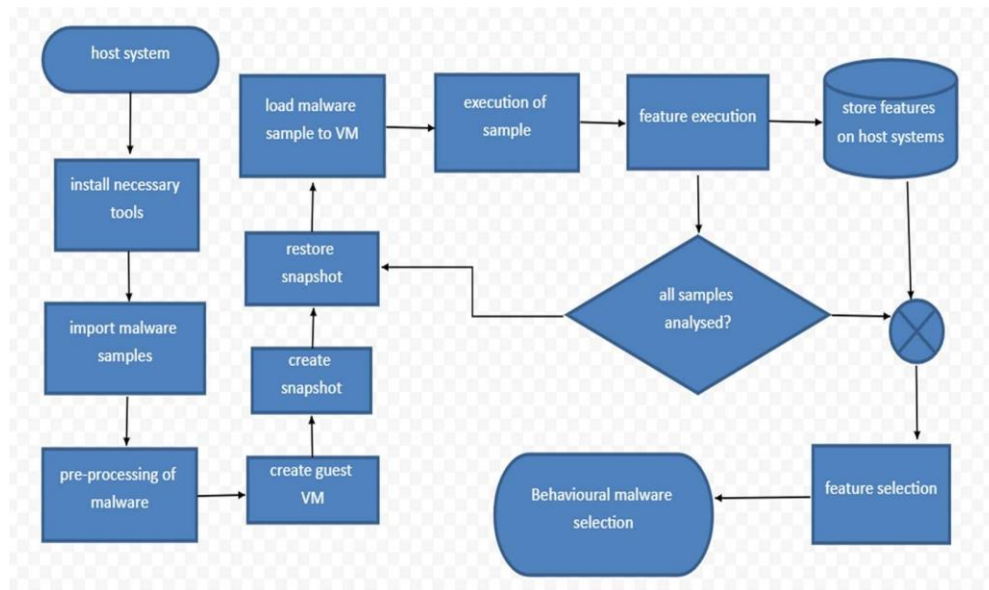


**Fig. 6: FLOWCHART FOR MALWARE ANALYSIS USING METHOD IDENTIFIERS**

**4.2 Requirements**

- **Setting up Windows 10 and Renmux on Oracle VirtualBox through iso file**

- **Setting on Renmux**

- **Installation of Flare-VM on virtual Windows 10**

- **Setting up Flare-VM**

- **Setting up INetSim**

- **Static Analysis**

- **Dynamic Analysis**

# CHAPTER-5
# TECHNICAL IMPLEMENTATION AND ANALYSIS

**STATIC ANALYSIS**

**5.1 Introduction**

A software testing technique named static analysis, commonly referred to as static code analysis or static program analysis, includes looking at a program's source code or compiled code without running it. Static analysis looks for grammatical errors, faults with the code's structure, security flaws, and compliance with coding standards to find potential problems and defects in the code.

Static analysis, often known as static code analysis, is a technique for troubleshooting computer programs that involves looking at the code without actually running the program. The procedure gives insight into the organization of the code and can help guarantee that it complies with best practices. Teams workingon software development and quality assurance use static analysis in this field.

Programmers and developers can benefit from the use of automated tools when performing static analysis. While validating the code, the software will scan the entire project for vulnerabilities.

As long as it's automated, the static analysis process is quite straightforward. Static analysis typically comes before software testing in the early stages of development. It happens during the creation phases in the DevOps development process.

After the code has been written, a static code analyzer should be used to review it. It will compare the code to predefined specified rules or defined coding rules from standards. The static code analyzer will have determined whether or not the code complies with the predetermined rules after it has run the code.It is crucial to go through and eliminate any false positives because the software may occasionally identify them. Once false positives are disregarded, developers can start to correct any obvious errors.

**5.2 Limitations of Static Malware Analysis**

**Generally, the source code of the malware samples is not easily available. It therefore reduces the applicable static analysis tecgniques for malware analysis to those who retrieve the information from the binary representation of the malware.**

**5.3 Advantages and Disadvantages of static analysis**

Static analysis has several advantages, including:

- It can assess every piece of code in an application, improving the quality of the code.

- Compared to manual code review, utilizing automated tools is faster.

- Static testing allows for more depth in code debugging when used with conventional testing techniques.

- Human error is less likely to occur with automated tools.

- This will improve online or application security by raising the possibility that flaws in the code will be discovered.

- In an offline development environment, it is possible.

Static analysis has significant limitations, though. Organizations should be mindful of the following, for instance:

- It is possible to find false positives.
- If there is a code defect, a tool might not be able to identify it.

## 5.4 Hashes Analysis:

Right click on the malware file and select HashMyFiles option and analyze the hash.

Copy the MD5 hash and run it on VirusTotal to know whether it has already been seen somewhere before.

**Important:**

• SHA-256

• MD5



Fig 7: Hashes Analysis

**5.5 String Analysis:**

In simple language, a string is an array of characters.

Right click on the file and select strings. The list of strings will be visible look for evident strings that could help in analysis.

At the end, we get static unicode strings these are sometimes some of the most telling strings out of all.



Fig 8: String Analysis

**5.6 PEview:**

Now we will continue our analysis through PEview.

Open PEview and open the malware file. PEview gives us the idea of how a portable executable will look like.

Select IMAGE_TIME_HEADER in IMAGE_NT_HEADERS, you could see various details related to file such as Time Date Stamp.

Select IMAGE_SECTION_HEADER .txt, compare the data of virtual size and size of raw data this gives us the information related to whether malware is packed or unpacked.

In SECTION .rdata select IMPORT Address Table, now you could see the API calls this portable executable is making note down the ones that are used in malicious activities.



(a)     PE look                                              (b) Time Date Stamp

(c)Comparing raw data and virtual size to identify    (d) Analysis of API calls

whether malware is packed or unpacked



Fig 10: A catalog of API calls that can be used maliciously. Malapi.io.

This catalog could help in analysis

Fig: All-important observation noted.

Analysis through pestudio:

Open pestudio and open the portable executable file.

PEStudio is a specialized software tool that is used for analyzing and auditing Windows executable files, commonly known as PE files and these files include various Windows applications, system files, and dynamic link libraries (DLLs). PEStudio is often used for security and software analysis purposes.

The red cross in flag section indicate that the following is used or can be used by malware.

**Screenshot 1: pestudio 9.55 — Malware Initial Assessment — www.winitor.com — [c:\users\vboxuser\desktop\malware.unknown.exe.malz]**

file  settings  about

Tree panel:
- c:\users\vboxuser\desktop\malware.unknown.ex
  - indicators (libraries > flag > name)
  - footprints (count > 12) *
  - virustotal (error)
  - dos-header (size > 64 bytes)
  - dos-stub (size > 184 bytes)
  - rich-header (tooling > Visual Studio)
  - file-header (executable > 32-bit)
  - optional-header (subsystem > console)
  - directories (stamp > Sep.2021)
  - sections (count > 5)
  - libraries (flag > 2) *
  - imports (flag > 52) *
  - exports (n/a)
  - thread-local-storage (n/a)
  - .NET (n/a)
  - resources (signature > manifest)
  - strings (count > 255)
  - debug (streams > 3)
  - manifest (level > asInvoker)
  - version (n/a)
  - certificate (n/a)
  - overlay (n/a)

| imports (52) | flag (9) | first-thunk-original (INT) | first-thunk (IAT) | hint | group (8) |
|---|---|---|---|---|---|
| InitializeSListHead | - | 0x00003EFA | 0x00003EFA | 867 (0x0363) | synchroniz... |
| GetCurrentProcessId | ✖ | 0x00003EB4 | 0x00003EB4 | 536 (0x0218) | reconnaissa... |
| IsProcessorFeaturePresent | - | 0x00003E7E | 0x00003E7E | 902 (0x0386) | reconnaissa... |
| IsDebuggerPresent | - | 0x00003F10 | 0x00003F10 | 895 (0x037F) | reconnaissa... |
| QueryPerformanceCounter | - | 0x00003E9A | 0x00003E9A | 1101 (0x044D) | reconnaissa... |
| URLDownloadToFileW | ✖ | 0x00003ADE | 0x00003ADE | 116 (0x0074) | network |
| InternetOpenW | ✖ | 0x00003B14 | 0x00003B14 | 201 (0x00C9) | network |
| InternetOpenUrlW | ✖ | 0x00003B00 | 0x00003B00 | 200 (0x00C8) | network |
| memset | - | 0x00003B64 | 0x00003B64 | 72 (0x0048) | memory |
| GetSystemTimeAsFileTime | - | 0x00003EE0 | 0x00003EE0 | 745 (0x02E9) | file |
| CreateProcessW | ✖ | 0x00003A44 | 0x00003A44 | 229 (0x00E5) | execution |
| GetCurrentThreadId | ✖ | 0x00003ECA | 0x00003ECA | 540 (0x021C) | execution |
| TerminateProcess | ✖ | 0x00003E6A | 0x00003E6A | 1420 (0x058C) | execution |
| GetCurrentProcess | ✖ | 0x00003E56 | 0x00003E56 | 535 (0x0217) | execution |
| ShellExecuteW | ✖ | 0x00003A64 | 0x00003A64 | 439 (0x01B7) | execution |
| UnhandledExceptionFilter | - | 0x00003E1C | 0x00003E1C | 1453 (0x05AD) | exception |
| SetUnhandledExceptionFilter | - | 0x00003E38 | 0x00003E38 | 1389 (0x056D) | exception |
| GetModuleFileNameW | - | 0x00003A20 | 0x00003A20 | 628 (0x0274) | dynamic-lit... |
| GetModuleHandleW | - | 0x00003F24 | 0x00003F24 | 632 (0x0278) | dynamic-lil... |
| CloseHandle | - | 0x00003A36 | 0x00003A36 | 134 (0x0086) | - |
| _Query_perf_frequency | - | 0x00003A80 | 0x00003A80 | 1425 (0x0591) | - |
| _Thrd_sleep | - | 0x00003A98 | 0x00003A98 | 1462 (0x05B6) | - |
| _Query_perf_counter | - | 0x00003AA6 | 0x00003AA6 | 1424 (0x0590) | - |
| _Xtime_get_ticks | - | 0x00003ABC | 0x00003ABC | 1484 (0x05CC) | - |
| _current_exception | - | 0x00003B30 | 0x00003B30 | 28 (0x001C) | - |
| _current_exception_context | - | 0x00003B46 | 0x00003B46 | 29 (0x001D) | - |
| _except_handler4_common | - | 0x00003B6E | 0x00003B6E | 33 (0x0035) | - |
| _p__commode | - | 0x00003D06 | 0x00003D06 | 1 (0x0001) | - |
| _stdio_common_vswprintf | - | 0x00003B9A | 0x00003B9A | 17 (0x0011) | - |
| _set_fmode | - | 0x00003C74 | 0x00003C74 | 84 (0x0054) | - |
| _c_exit | - | 0x00003CA8 | 0x00003CA8 | 22 (0x0016) | - |

sha256: 92730427321A1C4CCFC0D0580834DAEF98121EFA9BB8963DA332BFD6CF1FDA8A   cpu: 32-bit   file-type: executable   subsystem: console   entr...

4:19 AM  10/12/2023

---



**Screenshot 2: pestudio 9.55 — Malware Initial Assessment — www.winitor.com — [c:\users\vboxuser\desktop\malware.unknown.exe.malz]**

file  settings  about

Tree panel (same as above):
- c:\users\vboxuser\desktop\malware.unknown.ex
  - indicators (libraries > flag > name)
  - footprints (count > 12) *
  - virustotal (error)
  - dos-header (size > 64 bytes)
  - dos-stub (size > 184 bytes)
  - rich-header (tooling > Visual Studio)
  - file-header (executable > 32-bit)
  - optional-header (subsystem > console)
  - directories (stamp > Sep.2021)
  - sections (count > 5)
  - libraries (flag > 2) *
  - imports (flag > 52) *
  - exports (n/a)
  - thread-local-storage (n/a)
  - .NET (n/a)
  - resources (signature > manifest)
  - strings (count > 255)
  - debug (streams > 3)
  - manifest (level > asInvoker)
  - version (n/a)
  - certificate (n/a)
  - overlay (n/a)

| encoding (2) | size (bytes) | location | flag (9) | label (67) | group (8) | tech |
|---|---|---|---|---|---|---|
| ascii | 19 | .rdata | - | import | synchronization | |
| ascii | 25 | .rdata | - | import | reconnaissance | |
| ascii | 23 | .rdata | - | import | reconnaissance | |
| ascii | 19 | .rdata | ✖ | import | reconnaissance | T10: |
| ascii | 17 | .rdata | - | import | reconnaissance | T10: |
| ascii | 17 | .rdata | ✖ | import | network | - |
| ascii | 15 | .rdata | ✖ | import | network | - |
| ascii | 12 | .rdata | ✖ | import | network | - |
| ascii | 10 | .rdata | - | file | network | - |
| ascii | 11 | .rdata | - | file | network | - |
| ascii | 6 | .rdata | - | - | memory | - |
| ascii | 23 | .rdata | - | import | file | T11: |
| ascii | 13 | .rdata | ✖ | import | execution | T11: |
| ascii | 12 | .rdata | ✖ | import | execution | T11: |
| ascii | 17 | .rdata | ✖ | import | execution | T10: |
| ascii | 16 | .rdata | ✖ | import | execution | - |
| ascii | 18 | .rdata | ✖ | import | execution | T10: |
| ascii | 24 | .rdata | - | import | exception | - |
| ascii | 27 | .rdata | - | import | exception | - |
| ascii | 17 | .rdata | - | import | dynamic-library | - |
| ascii | 15 | .rdata | - | import | dynamic-library | - |
| unicode | 59 | .rdata | - | utility | - | T10: |
| unicode | 76 | .rdata | - | utility | - | T10 |
| unicode | 4 | .rdata | - | utility | - | - |
| unicode | 11 | .rdata | - | user-agent | - | - |
| unicode | 21 | .rdata | - | url-pattern | - | - |
| ascii | 11 | .rdata | - | import | - | - |
| ascii | 21 | .rdata | - | import | - | - |
| ascii | 11 | .rdata | - | import | - | - |
| ascii | 19 | .rdata | - | import | - | - |
| ascii | 16 | .rdata | - | import | - | - |

sha256: 92730427321A1C4CCFC0D0580834DAEF98121EFA9BB8963DA332BFD6CF1FDA8A   cpu: 32-bit   file-type: executable   subsystem: console   entr...

4:20 AM  10/12/2023

Difference in Packed and Unpacked Malware:

Packed and unpacked malware refer to two different states of malicious software, where "packing" is a technique used by malware authors to obfuscate their code and make it more difficult to detect, while "unpacking" is the process of reversing this technique to reveal the original, malicious code.

**Unpacked:**

**Packed:**

**DYNAMIC ANALYSIS**

**Introduction**

A software testing and analysis technique called dynamic analysis involves assessing a program's behavior while it is running or being used. Dynamic analysis focuses on the actual runtime behavior of the software as opposed to static analysis, which looks at the source code or produced code without running it. This method can identify problems that might not be seen during static analysis and offers insights into how the software acts under various circumstances.

Real-time data is used in dynamic analysis, often referred to as dynamic program analysis, to assess a technology or program. On a physical or virtual CPU, this analysis can be carried out. Instead of putting code offline, vulnerabilities and program behavior may be observed while the program is running, giving information about how it will behave in the real world.

There are basically two approaches for dynamic malware analysis which are as below:

- Firstly, analyzing the difference between two defined points , that is , in this approach comparison report states behavior of malware.

- Secondly, observing runtime behavior in which the malicious activities launched by a malicious application are monitored during runtime using a specialized tool.

**WORKING:**

Firstly analysing the malware on WannaCry ransomware. Then a URL request is found in static analysis. After it Wireshark results in URL request on running the malware and the filtered HyperText Transfer Protocol request on Wireshark. Procmon filtered by Process name for Ransomware.wannacry.ex.

Secondly, on running malware without internet simulation activated, malware pings for a count of 1 and deletes itself. While on running the malware with internet simulation activated, request to URL found in the strings output goes through when malware runs. So, the request to URL found in the strings output goes through when malware is processed. Processing it adds a file to public documents.
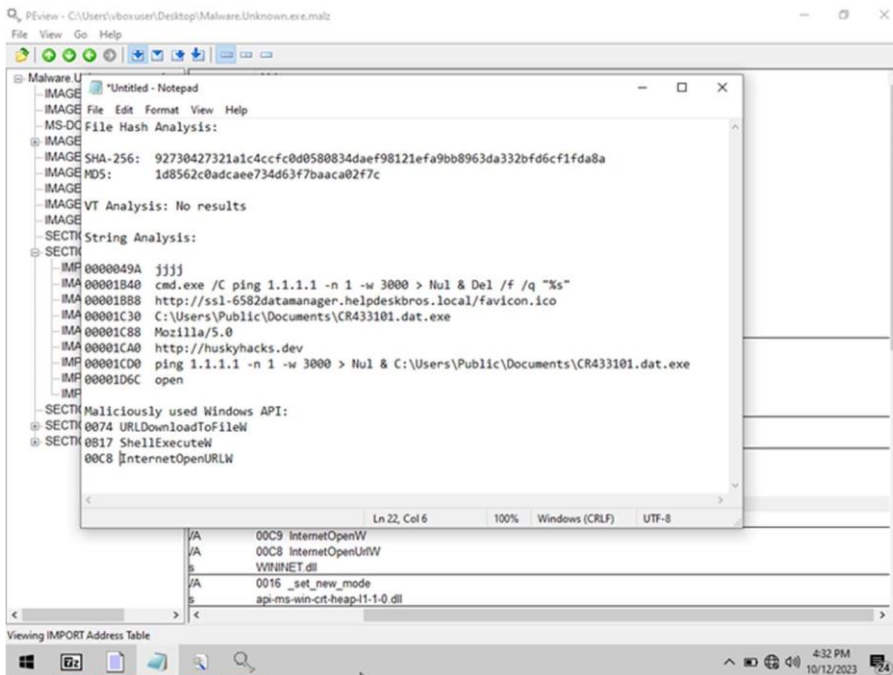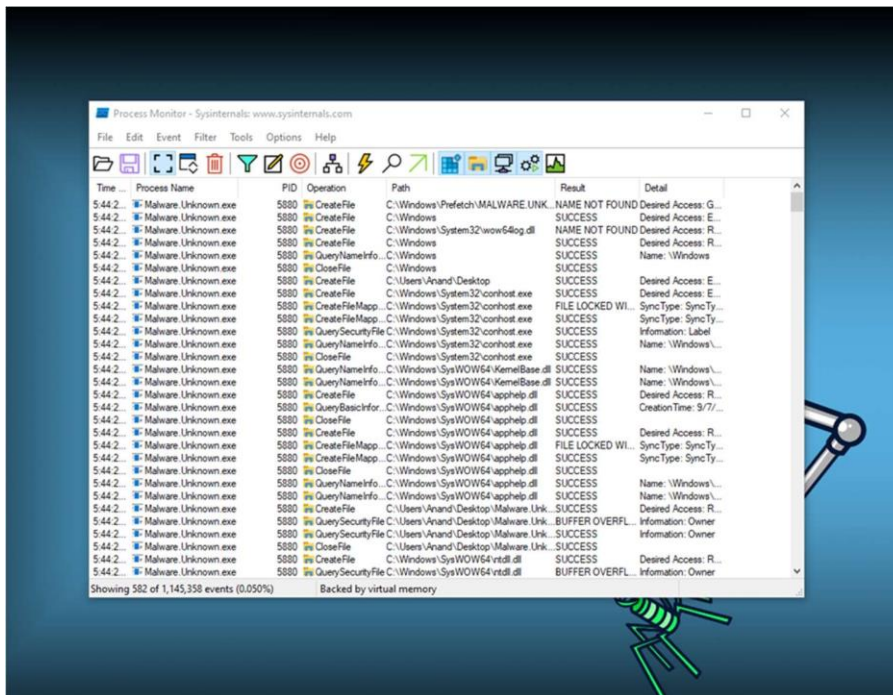
**Analysis on WannaCry ransomware**



Fig 11: URL request found in static analysis

Wireshark results in URL request on running the malware



```
Host: www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com\r\n
Cache-Control: no-cache\r\n
\r\n
[Full request URI: http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com/]
[HTTP request 1/1]
[Response in frame: 1328]
```

Flitered http request on wireshark.

Procmon filtered by process name for Ransomeware.wannacry.ex

**Analysis on an unknown malware sample**

On running malware without internet simulation activated,

Malware pings for a count of 1 and deletes itself,

On running malware with internet simulation activated, request to URL found in the strings output goes

through when malware is running.



Fig 12: Analysis on unknown malware sample

Request to URL found in the strings output goes through when malware is run

Running the malware adds a file to public documents,
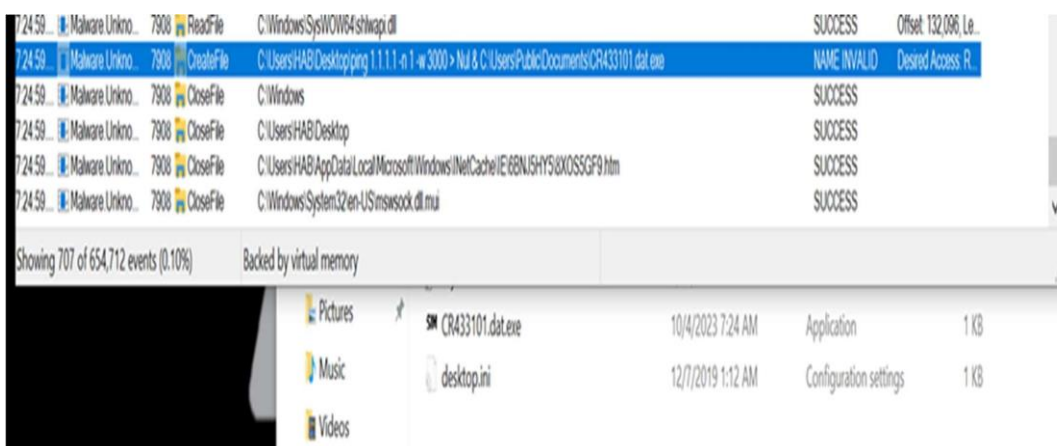
**Process flow of current malware sample:**

If URL exists:

- Download file

- Writes to disk

- Run favicon.ico

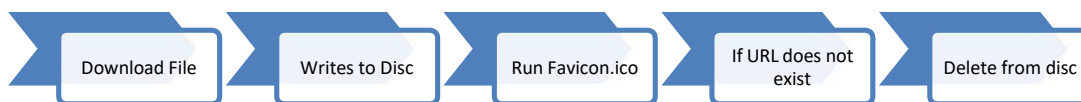- If URL does not exist:

- Delete from disk

| Download File | Writes to Disc | Run Favicon.ico | If URL does not exist | Delete from disc |

Fig:13 Process Flow

# CHAPTER-6:
# PROJECT OUTCOME AND APPLICABILITY

## 6.1 PROJECT OUTCOME
In this chapter, we present the outcomes and significance of our malware analysis project. The project's findings, impact assessment, threat intelligence, and mitigation strategies are discussed to provide a comprehensive overview of the analysis results.

## Analysis Results
- **Characteristics of a malware**: We identified the key characteristics of the analysed malware, including its code structure, obfuscation techniques, and persistence mechanisms.

- **Classification**: The malware was classified as a variant of a known threat family, providing valuable context for its behaviour and potential impact.

- **Behaviour and Capabilities**: Through dynamic analysis, we observed the malware's behaviour, including its attempts to communicate with external servers, its evasion tactics, and its ability to propagate through the network.

## Impact Assessment
Our assessment of the malware's impact revealed its potential threat level. The malware exhibited the capability to leak sensitive data, making it a significant concern for data security. Additionally, its evasive techniques could potentially bypass conventional security measures, necessitating proactive defences.

## Threat Intelligence
The analysis provided valuable threat intelligence in the form of indicators of compromise (IOCs) and behavioural patterns. These IOCs can be integrated into security systems to enhance threat detection and response capabilities

## Mitigation Strategies
Based on our analysis results, we recommend the following mitigation strategies:
- Update signatures and security software to detect and prevent the analysed malware.

- Patch vulnerabilities and perform system hardening to mitigate potential exploits.

- Enhance employee cybersecurity training to recognize and report suspicious activities.

## 6.2 APPLICABILITY
In this section, we explore how the project outcomes can be applied in various contexts within the field of cybersecurity and threat management.

## Practical Use Cases
The analysis findings have practical applications in the following scenarios:

- Incident Response: Our project findings are useful for incident response teams. When a similar malware attack occurs, they can use our insights to understand the threat quickly and take action to stop it before it causes significant damage.

- Network Security: Organizations can apply our insights to improve their network security. By configuring their firewalls and intrusion detection systems based on our findings, they can better detect and block similar malware activities, reducing the risk of breaches.

# CHAPTER 7:
# CONCLUSION

## 7.1 Conclusion

The report for the project "Malware Analysis using Method Identifiers" contains a systematic overview of Malware, their classification and the static and dynamicanalysis of different kinds of malware, the project's objectives, methodologies and outcomes. In addition to that, we conclude the context of malware analysis and its future scope in evolution of cyber security.

## 7.2 Project Contribution

Our contributions to the field of cybersecurity by this project are:

- We successfully identified and classified the analysed malware, gaining information about its threat level and potential impact.

- Our static analysis uncovered the malware's real-time behaviour, reveling it's impact and intention towards our system.

- Programmers and developers can benefit from the use of automated tools, while validating and looking at the source code without actually running it.

## 7.3 Future Scope

The objective of the project is to detect the different kinds of malwares and classify them as evasive malware detection is still challenging. For evasive malware detection efficient extraction techniques are required. Techniques such hashes and string analysis have been used to detect different malware behaviours.

- This project can be an inspiration for new programmers and developers for their malware analysis journey and can help them analyze or develop more challenging malicious softwares.

**7.4 References**

1. Nikam, U.V.; Deshmuh, V.M. Performance evaluation of machine learning classifiers in malware detection. In Proceedings of the 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballari, India, 23–24 April 2022; pp. 1–5.

2. Sharma, S.; Krishna, C.R.; Sahay, S.K. Detection of advanced malware by machine learning techniques. In Proceedings of the SoCTA 2017, Jhansi, India, 22–24 December 2017.

3. Chandrakala, D.; Sait, A.; Kiruthika, J.; Nivetha, R. Detection and classification of malware. In Proceedings of the 2021 International Conference on Advancementsin Electrical, Electronics, Communication, Computing and Automation (ICAECA), Coimbatore, India, 8–9 October 2021; pp. 1–3.

4. Zhao, K.; Zhang, D.; Su, X.; Li, W. Fest: A feature extraction and selection tool for android malware detection. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 714– 720.

5. Firdaus, A.; Anuar, N.B.; Karim, A.; Faizal, M.; Razak, A. Discovering optimal features using static analysis and a genetic search based method for Android malware detection. *Front. Inf. Technol. Electron. Eng.* **2018**, *19*, 712–736.

6. Anderson, B.; Storlie, C.; Lane, T. "Improving Malware Classification: Bridging the Static/Dynamic Gap. In Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence (AISec), Raleigh, NC, USA, 19 October 2012; pp. 3–14.